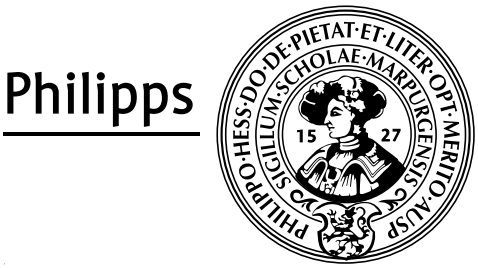
Verbundoperationen über Datenströmen und deren Optimierung unter Verwendung dynamischer Metadaten



Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften
(Dr. rer. nat.)

dem Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
vorgelegt von

Michael Cammert
aus Gießen

Marburg an der Lahn 2014

Vom Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg (Hochschulkennziffer: 1180)
als Dissertation angenommen am 12.12.2014.

Vorgelegt von Diplom-Informatiker Michael Cammert, geboren in Gießen.

Erstgutachter:	Prof. Dr. Bernhard Seeger
Zweitgutachter:	Prof. Dr. Bernd Freisleben
Weitere Mitglieder der Prüfungskommission:	Prof. Dr. Gabriele Taentzer
	Prof. Dr. Manfred Sommer
Tag der Einreichung:	27.10.2014
Tag der mündlichen Prüfung (Disputation):	12.12.2014

Zusammenfassung

Technischer Fortschritt und Vernetzung haben zu einem enormen Zuwachs an kontinuierlich anfallenden Datenströmen geführt, aus denen immer schneller wertvolle Informationen abgeleitet werden sollen. Zur Verarbeitung von kontinuierlichen Anfragen über Datenströme etablieren sich daher Datenstrommanagementsysteme, die hauptsächlich im Hauptspeicher operieren und im Gegensatz zu Datenbanksystemen auf die Verarbeitung kontinuierlicher Anfragen über Datenströmen zugeschnitten sind. Bei der Implementierung dieser neuen Systeme hat sich die Übernahme des Konzeptes der relationalen Operatorgraphen aus der Welt der Datenbanken bewährt, wobei diese allerdings statt bedarfsgesteuert nun kontinuierlich und datengetrieben ausgewertet werden. Durch Zuweisen von Gültigkeitsintervallen zu den Datenstromelementen kann dabei das Problem gelöst werden, mit endlichen Ressourcen potentiell unbegrenzte Datenströme zu verarbeiten.

Die Mächtigkeit solcher Systeme hängt wesentlich von der Verfügbarkeit effizienter wohldefinierter Techniken zur Verknüpfung von Informationen aus verschiedenen Datenströmen ab. Ziel dieser Arbeit ist es daher, das für Datenbanken bewährte Konzept des relationalen Verbundes auf die datengetriebene Datenstromverarbeitung mit Gültigkeitsintervallen zu übertragen.

Dazu wird die Semantik der Verbundoperation für Datenströme mittels der Schnappschuss-Reduzierung formal auf die des Verbundes in der erweiterten relationalen Algebra zurückgeführt. Es werden mehrere Verbundalgorithmen vorgestellt und gezeigt, dass diese die gewünschte Semantik haben. Durch eine konsequente Parametrisierung der Verfahren bezüglich der Datenstrukturen zur Statusverwaltung werden verschiedenste Typen von Verbundprädikaten effizient unterstützt. Bewährte Techniken der Verbundberechnung mittels verschachtelter Schleifen, Hashing und Indexierung werden dabei für die Datenstromverarbeitung adaptiert. Mit dem temporalen Progressive-Merge-Join wird zudem ein Verfahren vorgestellt, dass den Verbund über Datenströmen mittels einer wertbasierten Sortierung berechnet. Zudem werden für die Verfahren verschiedenste Optimierungen vorgeschlagen, darunter für alle Verfahren die Verallgemeinerung auf mehr als zwei Datenströme.

Zur Ermöglichung der automatischen Auswahl und Parametrisierung der Implementierungen anhand ihres prognostizierten Ressourcenverbrauchs werden diese in ein detailliertes Kostenmodell eingebettet. Da bestimmte Metadaten bezüglich der zu verarbeitenden Datenströmen bei der Registrierung kontinuierlicher Anfragen oftmals nicht vorliegen und sich zudem während der langen Laufzeit der Anfragen ändern können, ist es wichtig, jederzeit detaillierte Informationen bezüglich der Datenströme und des Systemverhaltens erheben und gegebenenfalls durch Anpassungen an der Verarbeitungsstrategie darauf reagieren können. Ein wesentlicher Teil der Arbeit ist daher der Fragestellung gewidmet, wie in einem Datenstrommanagementsystem dynamische Metadaten erhoben werden können. Dazu wird ein benutzerfreundliches Rahmenwerk vorgestellt, das es ermöglicht, dynamische Metadaten konsistent und effizient zu erheben. In Experimenten wird die dynamische Metadatenerhebung untersucht und auch für die Evaluation der vorgestellten Verbundoperationen eingesetzt. Zudem wird eine Technik vorgestellt, mit der kontinuierliche Anfragen zur Laufzeit restrukturiert werden können, und deren Anwendbarkeit für die Verbundoptimierung aufgezeigt.

Abstract

Due to technological progress there has been an enormous increase of the number of continuous data streams from which valuable information has to be derived as fast as possible. Therefore, data stream management systems have emerged as a new technology to process continuous queries over data streams. In contrast to databases they primarily operate in memory and are optimized for processing continuous queries over data streams. During the development of this new kind of systems taking over the concept of using relational operator graphs from database theory has proven to be of value if executing them data driven instead of demand driven. Assigning validity intervals to the elements of the data streams solves the problem of processing potentially unbounded data streams while using bounded resources.

The capabilities of such systems considerably depend on the availability of efficient and well defined techniques for combining information from different data streams. The objective of this thesis therefore is to transfer the proven concept of the relational join to the data driven data stream processing using validity intervals.

For this purpose the semantic of the join operation for data streams is derived from the one of the extended relational algebra using the concept of snapshot-reducibility. Several join algorithms are presented and proven to comply with this semantic. The consequent usage of parameterization of the techniques with respect to the data structures used for storing the status allows supporting a large variety of different join predicates. Well known techniques of join processing using nested loops, hashing or indexing are adapted for data stream processing. Additionally, the Temporal Progressive-Merge-Join is introduced as an algorithm which allows to derive the join by using value based sorting of the data stream elements. Additionally, several optimizations are proposed, including the generalization of all presented algorithms for more than two input streams.

To enable the automated choice and parameterization of the concrete implementations with respect to a forecast of their resource usage the techniques are embedded in a detailed cost model. Often, several kinds of metadata concerning the data streams to process are not available at registration time of the continuous queries and additionally may change during their long-lasting execution time. Therefore, it is important to be able to gather detailed information regarding the data streams and the system state at any time and to be able to adapt the processing strategy accordingly if necessary. A main part of the thesis therefore deals with the question how to provide dynamic metadata within a data stream management system. For that purpose a user-friendly framework is presented which allows to efficiently obtain consistent dynamic metadata. This process is investigated in several experiments and also used to evaluate the presented join techniques. Additionally, an approach allowing to restructure continuous queries at runtime is presented and it is shown that this approach is applicable for join processing.

Danksagungen

Ich möchte mich bei Herrn Prof. Dr. Bernhard Seeger für die Möglichkeit, in seiner Arbeitsgruppe Datenbanksysteme im Rahmen des Projektes *Anfrageverarbeitung aktiver Datenströme* an einem hochinteressanten und praxisrelevanten Thema zu forschen, seine Geduld und die Betreuung dieser Arbeit bedanken. Dem Fachbereich Mathematik und Informatik der Philipps-Universität Marburg und der Deutschen Forschungsgemeinschaft[1] gilt mein Dank für die Förderung dieses Forschungsvorhabens. Dr. Christoph Heinz, Dr. Jürgen Krämer, Dr. Tobias Riemenschneider, Sonny Vaupel und allen anderen Mitgliedern der Arbeitsgruppe möchte ich für die tolle Zusammenarbeit danken. Herrn Prof. Dr. Bernd Freisleben danke ich für die Erstellung des Zweitgutachtens.

Meinen Arbeitskollegen, insbesondere Dr. Christoph Heinz, Dr. Jürgen Krämer, Dr. Ulrich Post, Dr. Tobias Riemenschneider und Daniel Schäfer, sowie meinen Vorgesetzten Dr. Mark Horsburgh, Dr. Johannes Viegener und Dr. Walter Waterfeld gilt mein Dank für die Unterstützung der Fortführung des Promotionsvorhabens auch nach meiner Zeit an der Universität.

Mein Dank gilt auch meiner Familie, meinen Freunden und Bekannten, die mich stets ermutigt und unterstützt haben, insbesondere Dr. Andreas Klein für das Korrekturlesen dieser Arbeit. Meinem Onkel Erich Bloh (†) und meinem Lehrer Friedel Klein (†) möchte ich dafür danken, dass sie mich schon früh für die Informatik begeistert haben. Mein besonderer Dank gilt meinen Eltern Lieselotte Cammert und Herbert Cammert (†), die mir während meiner gesamten Ausbildung stets Rückhalt und Unterstützung waren.

[1] Projekt Nummern SE-553/4-1 und SE-553/4-3

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Teil I.

Einleitung

1. Einleitung

Das persistente Speichern von Daten und der Zugriff darauf ist eine Standardaufgabe bei der Softwareentwicklung, die in der Softwarearchitektur üblicherweise abstrahiert und an ein *Datenbankmanagementsystem* (DBMS) delegiert wird. Dies gilt sogar auf vielen Plattformen mit eingeschränkten Möglichkeiten, beispielsweise Smartphones. Es steht eine große Auswahl sowohl an kommerziellen, als auch an freien DBMS mit verschiedenen Eigenschaften und Spezialisierungen zur Verfügung. Den wichtigsten Typ stellen aber nach wie vor relationale DBMS mit der Anfragesprache *Structured Query Language* (SQL) dar. In der Informatik ist die Forschung bezüglich Datenbanksystemen als wichtige Disziplin etabliert.

Neben den Fortschritten in den Jahrzehnten der Forschung und Entwicklung im Bereich Datenbanksysteme sind im selben Zeitraum insbesondere auch die Möglichkeiten der zugrunde liegenden Hardware enorm gewachsen. Dies ermöglichte Anwendungen, die zuvor undenkbar gewesen wären. Andererseits führte der Fortschritt aber auch dazu, dass die Anforderungen bezüglich der Menge der zu speichernden und zu verarbeitenden Daten enorm wuchsen. Dieser Effekt wurde durch die zunehmende Vernetzung von Computern, insbesondere den enormen Erfolg des Internets, noch deutlich verstärkt. Gleichzeitig wuchs mit der zunehmenden Durchdringung vieler Lebensbereiche durch die Elektronik die Zahl der Geräte, die Daten liefern können, extrem an. Dies war natürlich nur möglich, da als Nebeneffekt die Kosten für das Erfassen vieler Arten von Daten rapide sanken. Während zu Beginn der Nutzung von DBMS Daten noch gezielt für die persistente Speicherung manuell digitalisiert wurden, fallen heute so viele Daten digital an, dass sich insbesondere die Frage stellt, welche es sich noch zu persistieren lohnt.

Im Laufe der Zeit wuchsen zusätzlich auch noch die Ansprüche der Benutzer, die von Anwendungsprogrammen und damit den zu Grunde liegenden DBMS schnelle Antwortzeiten auch bei komplizierten Operationen erwarten. Anwendungsprogramme stellen dazu oft immer wieder die gleichen Anfragen an ein DBMS, um Antworten möglichst bezüglich des aktuellsten Datenbestandes liefern zu können. Anwender erwarten dabei immer kürzere Zyklen und möchten permanent auf dem Laufenden sein.

Zu Anfang des 21. Jahrhunderts stellte sich angesichts dieser Entwicklungen in der Datenbankforschung die Frage, ob die Paradigmen der DBMS noch für alle gewünschten Anwendungen das Mittel der Wahl darstellen. DBMS speichern Daten zunächst, bevor dann bei Bedarf Anfragen bezüglich dieser Daten gestellt werden können. Die neuen Anforderungen gehen jedoch in eine andere Richtung.

- Die ständige Auswertung der immer gleichen Anfrage konvergiert mit immer kleiner werdendem Anfrageintervall zu einem neuen Typ von Anfrage, der *kontinuierlichen Anfrage*. Die Anforderung ist es, über einen längeren Zeitraum durchgängig das Ergebnis zu einer vorgegebenen Anfrage zu liefern.

- Im Falle kontinuierlicher Anfragen fällt der Anfragezeitpunkt als Auslöser für die Auswertung weg. Als Alternative bietet es sich an, das Eintreffen neuer Informationen als Anlass für weitere Berechnungen herzunehmen. Das sich so ergebende Konzept ist die *datengetriebene Verarbeitung*.
- Eine wesentliche Eigenschaft von DBMS ist die persistente Speicherung der Daten, was mit der Nutzung nichtflüchtiger Medien einhergeht. Ist es aber das Ziel, aus neu eintreffenden Daten möglichst schnell Informationen zu gewinnen, während die kompletten Ursprungsdaten längerfristig zumeist von geringerer Bedeutung sind, bietet es sich an, primär flüchtigen aber schnellen Speicher zu benutzen und *Hauptspeichersysteme* zu entwickeln.

Dies stellt einen Paradigmenwechsel dar, bei dem die Rollen von Daten und Anfragen vertauscht werden. Während bei DBMS die Daten persistent und die Ad-hoc-Anfragen flüchtig sind, sind in diesem Modell die kontinuierlichen Anfragen langlebig und die Daten werden nur flüchtig gehalten. Zur Umsetzung dieses Paradigmenwechsels wird daher eine neue Art von Systemen benötigt. Da die Daten bei diesem Konzept kontinuierlich in das System strömen und die Ergebnisse heraus, hat sich für die Eingaben und Ausgaben der Name *Datenstrom* etabliert. Die Systeme, sie darüber operieren, werden entsprechend *Datenstrommanagementsysteme* (DSMS) genannt. Bezüglich deren Entwicklung hat sich die Datenstromverarbeitung als Teildisziplin der Datenbankforschung etabliert.

1.1. Datenstromverarbeitung

Bei der Datenstromverarbeitung möchte man nun nicht die Annahme treffen, dass die zu verarbeitenden Ströme zwangsweise irgendwann enden. Würde man diese Einschränkung treffen, so könnte man dies natürlich ausnutzen, um Verarbeitungsschritte erst nach dem vollständigen Eintreffen eines Stromes durchzuführen. Dies widerspräche aber dem Ziel, Anfragen kontinuierlich zu beantworten. Das Konzept unbegrenzte Datenströme mit einem Hauptspeichersystem, also durchaus begrenztem Speicher zu verarbeiten, stellt nun zumindest dann eine Herausforderung dar, wenn man Anfragen unterstützen möchte, die verschiedene Elemente der Datenströme in Beziehung zueinander setzen, die also nicht auf den einzelnen Elementen separat ausgewertet werden können. Denn dann müssen Wege gefunden werden um zu verhindern, dass der Speicherverbrauch mit zunehmender Laufzeit der Anfragen immer weiter anwächst. Wichtig bei der Suche nach Lösungen ist dabei, dass diese nicht im Konflikt zu dem Ziel stehen sollten, eine solide Semantik bezüglich der zu produzierenden Anfrageergebnisse bereitzustellen. Das zufällige Löschen von Daten sollte also beispielsweise nicht als zielführend angesehen werden. Als wesentlich sinnvoller hat sich die Hinzunahme der Annahme erwiesen, dass Elemente der Datenströme nur dann gemeinsam zu Ergebnissen beitragen sollen, wenn diese in einem definierten zeitlichen Kontext zueinander stehen. Der Faktor Zeit wird dabei zu einem essentiellen Teil der Semantik.

Betrachtet man das Ziel datengetriebener Verarbeitung lediglich bezüglich des Gesamtsystems, so könnte man es auch erreichen, indem man die eintreffenden Elemente der Datenströme jeweils einzeln in ein DBMS speichert und nach jedem Einfügen die

kontinuierlichen Anfragen neu ausführt. Diese Vorgehensweise verschenkt jedoch naiv die Vorteile, die sich aus der Tatsache ziehen lassen, dass die Anfragen kontinuierlich beantwortet werden sollen. Setzt man einen Menschen in einen leeren Raum mit einer Tür und bittet ihn, kontinuierlich die Zahl der anderen Personen in diesem Raum zu berichten, so wird er mit hoher Wahrscheinlichkeit die Tür beobachten und seine Antwort lediglich bezüglich der eintreffenden und ausgehenden Personen anpassen. War seine vorherige Antwort 41 und trifft eine weitere Person ein, so wird er nicht alle Person im Raum zählen, sondern umgehend 41 + 1 = 42 als neues Ergebnis nennen. Überträgt man dieses Beispiel auf die Datenstromverarbeitung, so erkennt man, dass hier der Datenstrom beobachtet wird und Informationen über die bisherigen Geschehnisse mit neu eingetroffenen Daten zu einem neuen Ergebnis verknüpft werden. Die Gesamtheit der bisher eingetroffenen Daten muss hingegen nicht immer wieder betrachtet werden. Ein DSMS sollte also nicht ziellos alle eingetroffenen Daten speichern, sondern lediglich die Informationen, die zur Beantwortung der registrierten kontinuierlichen Anfragen benötigt werden.

DSMS sollten die Ergebnisse bezüglich der kontinuierlichen Anfragen natürlich möglichst zeitnah produzieren, nachdem alle zur Determinierung der jeweiligen Ergebnisse beitragenden Elemente der Eingabedatenströme eingetroffen sind. Um dies zu erreichen bietet es sich an, alle durch ein neu eintreffendes Element möglichen Verarbeitungsschritte umgehend auszulösen. Dies kann durch die konsequente Fortsetzung des Konzeptes der datengetriebenen Verarbeitung auch innerhalb des Systems erreicht werden. Ein Element wird dazu geeignet weitergeleitet, löst Verarbeitungsschritte aus, die dann gegebenenfalls weitere Daten erzeugen, die wieder Verarbeitungsschritte auslösen, und so weiter. Während dieses Vorgangs berechnete Anfrageergebnisse werden dann umgehend an registrierte Nutzer der Anfrage aktiv weitergeleitet. Die Anfrageergebnisse bilden so aktive Ausgabedatenströme. Der komplette Pfad bestehend aus Eingaben, dem System und den Ausgaben arbeitet somit aktiv datengetrieben.

1.1.1. Abgrenzung zu Hauptspeicherdatenbanken

Auf Grund technischer Fortschritte und der Verfügbarkeit von Betriebssystemen und Software, die mit 64 Bit adressieren, haben in den letzten Jahren sogenannte Hauptspeicherdatenbanken an Popularität gewonnen. Der Name ist dabei teilweise irreführend, da im Allgemeinen die Anforderung der persistenten Speicherung der Daten keinesfalls aufgegeben wurde. Vielmehr werden die Daten lediglich so weit wie möglich jederzeit im Hauptspeicher gehalten. Dort werden sie zudem in vielen Fällen ausschließlich oder redundant zusätzlich in einer Anordnung gehalten, die für das effizientere Ausführen analytischer Anfragen optimiert ist. Bei tabellarischen Daten, die in DBMS klassisch zeilenorientiert gespeichert werden, eignet sich eine spaltenorientierte Speicherung nämlich erheblich besser zur Berechnung von Aggregaten. Neben echten Neuentwicklungen wurden und werden auch viele etablierte DBMS nachträglich um eine solche Hauptspeicherschicht erweitert.

Durch die auf diese Weise verringerten Anfragezeiten werden solche Systeme vielfach als Echtzeitsysteme bezeichnet und stehen in Konkurrenz zu DSMS. Von denen für ein DSMS formulierten Paradigmenwechseln sind sie jedoch lediglich den Weg in den Hauptspeicher gegangen, und selbst den auf Grund der zusätzlich zumeist nach wie vor

stattfindenden Persistierung nur halbherzig. Weder das Konzept der kontinuierlichen Anfragen, noch das der datengetriebenen Verarbeitung ist in solchen Systemen konsequent umgesetzt. Übertragen auf die Analogie aus 1.1 zählen solche Systeme also immer wieder alle Personen im Raum.

1.1.2. Anwendungsbeispiele

Mit der Entstehung eines neuen Typs von Standardsystemen verbinden viele Benutzer die Erwartung, dass damit auch zuvor unmögliche Anwendungsfälle möglich wären. Rein technisch gesehen ist dies natürlich falsch, denn jeder solche Anwendungsfall hätte auch mit einer Spezialsoftware ermöglicht werden können, als Teil deren Entwicklung die Funktionalität des Systems als Komponente programmiert worden wäre. Da diese Komponente dabei speziell für den Anwendungsfall optimiert werden könnte, wäre das Ergebnis insgesamt in vielen Fällen sogar performanter. So können spezielle Datenströme natürlich auch mit darauf zugeschnitten entwickelten Anwendungen verarbeitet werden.

Dennoch ist die Erwartungshaltung keinesfalls falsch, dass neue Standardsysteme neue Anwendungen ermöglichen. Denn zieht man auch die Aspekte Kosten, Zeitaufwand und Komplexität der Anwendungsentwicklung sowie die Erweiterbarkeit und Wartbarkeit des Ergebnisses in Betracht, so trifft sie zumeist zu. Da ein Standardsystem für viele Anwendungen zum Einsatz kommen kann, kann es oft in viel höherer Qualität und mit besseren Features entwickelt werden und kostet dennoch weniger als eine Spezialentwicklung. Zudem stellt ihre Außenschnittstelle eine natürliche Abstraktion her, die der Architektur der darauf basierenden Anwendung oft zugutekommt. Und die Verfügbarkeit des Standardsystems verringert die Entwicklungszeit der darauf basierenden Anwendung.

Entsprechend gab es Anwendungsprogramme, die Datenströme verarbeiten, schon vor der Entwicklung von DSMS. Beispielsweise werden Finanzdaten schon länger zum auf Algorithmen basierenden automatischen Handel an den Börsen eingesetzt. Die enorme Höhe der dabei zu erzielenden Gewinne erlaubt für diesen Anwendungsfall die Entwicklung von darauf spezialisierter Software. Der gleichzeitige starke Konkurrenzdruck, insbesondere bezüglich der Reaktionszeiten, ist sogar ein echtes Argument, um in diesem Segment nach wie vor spezialisierte Komponenten einer Standardsoftware vorzuziehen. Dennoch kommen auch für diesen Anwendungsfall mittlerweile vielfach DSMS zum Einsatz, da diese kurze Entwicklungszeiten und geringere Kosten bei der Umsetzung neuer Handelsstrategien ermöglichen.

DSMS finden heute auf verschiedensten weiteren Gebieten Anwendung, beispielsweise auf den folgenden:

Betrugserkennung Betrug im elektronischen Handel und Zahlungsverkehr richtet hohen Schaden an und schädigt zudem den Ruf entsprechender Plattformen. Dennoch erwarten die Kunden schnelle und effiziente Zahlungsflüsse und Lieferungen. Daher ist es wichtig, Betrugsversuche schnell und möglichst noch vor dem Vollzug aufzudecken und zu verhindern. DSMS erlauben den schnellen Abgleich von Transaktionsströmen und helfen so Betrugsversuche abzuwehren.

Marketing Marketing ist ein so großer Markt, dass die Kunden sich vielfach davon belästigt fühlen. Die Akzeptanz erhöht sich jedoch oftmals, wenn Angebote auf ihre

konkrete Situation und Bedürfnisse zugeschnitten sind. DSMS ermöglichen durch die Analyse von Positionsdaten und anderer Informationen über das Kundenverhalten zielgerichtete Angebote.

Kundenzufriedenheit Neben der Kundengewinnung ist auch die Kundenbindung für Unternehmen von enormer Bedeutung, auch da negative Mundpropaganda sehr schädlich sein kann. DSMS ermöglichen es beispielsweise, eine zeitliche Häufung von Problemen eines Kunden zu erkennen und rechtzeitig gegenzusteuern, um diesen nicht zu verärgern.

Geschäftsprozessüberwachung Die Effizienz ihrer Geschäftsprozesse ist essenziell für die Profitabilität von Unternehmen. DSMS ermöglichen durch Überwachung verschiedener elektronisch erfasster Prozessschritte die schnelle Erkennung von und Reaktion auf Probleme in der Prozessabwicklung.

Logistik Durch die Überwachung von Positionsdatenströmen und Anforderungen ermöglichen DSMS schnelle Reaktionen zum Umlenken von Waren und Transportmitteln in der Logistik.

Computersystemüberwachung Da in Computersystemen alle Daten ohnehin digital anfallen, kann deren Analyse mittels eines DSMS vielfach zusätzliche Möglichkeiten zur Steuerung und Überwachung liefern.[HFS12]

Diesen Anwendung ist gemein, dass eine Fülle von Einzelereignissen überwacht und das Vorliegen von komplizierteren Ereignissen wie Betrugsfällen oder Geschäftschancen schnell erkannt und gegebenenfalls automatisch darauf reagiert werden soll. Die Datenstromverarbeitung wird daher auch vielfach als *Complex Event Processing* bezeichnet.

1.2. Einordnung

Obwohl die mangelnde Einsetzbarkeit von DBMS für die Verarbeitung kontinuierlicher Anfragen über Datenströmen ja gerade erst zur Entwicklung von DSMS geführt hat, ähneln sich die Problemstellungen bei der Entwicklung beider Arten von Systemen natürlich doch wesentlich. Daher bietet es sich an, sich bei der Entwicklung von DSMS von bewährten Konzepten und Techniken aus Jahrzehnten der Datenbankforschung inspirieren zu lassen.

Eine solches Konzept aus Datenbanken ist die Beschleunigung von Anfragen durch das Anlegen eines Index. Die Idee zur Übertragung dieses Konzeptes in die Welt der Datenstromverarbeitung ist es nun, dem Paradigmenwechsel bezüglich der Persistenz von Daten oder Anfragen folgend einen Index über die kontinuierlichen Anfragen aufzubauen und dann die Elemente der Datenströme als Punktanfragen gegen diesen Index zu betrachten.

1.2.1. Verarbeitungsansatz

Der mit Abstand am weitesten verbreitete Ansatz ist es jedoch, den kompletten Prozess der Anfrageverarbeitung in relationalen DBMS zu betrachten und die einzelnen Schritte

dann wenn nötig für die Verwendung in der Datenstromverarbeitung zu modifizieren.

- Als erster Schritt wird eine Anfrage in einem DBMS zunächst analysiert und aus den verwendeten Anfragesprache, zumeist SQL, in eine systeminterne Repräsentation, zumeist einen logischen Operatorgraphen, überführt. Die einzelnen Operatoren dieses Graphen repräsentieren abstrakte Verarbeitungsschritte.

  Bei der Übertragung dieses Konzeptes auf DSMS ist es erstrebenswert, die Anfragesprache möglichst ähnlich zu der für ein DBMS zu gestalten, um erfahrenen Datenbanknutzern den Einstieg in die Datenstromverarbeitung zu erleichtern. Dazu ist es natürlich erforderlich, dass neben der Syntax der Sprache auch deren Semantik so weit wie möglich erhalten bleibt. Die Überführung in eine analoge logische Repräsentation bietet sich also an.

- Da die logische Repräsentation als Ergebnis der Strukturanalyse einer gegebenenfalls deskriptiven Anfragebeschreibung in der Regel nicht die optimale Strategie zur Ausführung der Anfrage beschreibt, folgt als nächstes ein Optimierungsschritt. Dabei wird der logische Operatorgraph unter Beibehaltung seiner Semantik umstrukturiert, wobei auch Kombinationen von Operatoren gegen andere ausgetauscht werden können. Gegebenenfalls werden auch Zusatzinformationen für die Nachfolgeschritte ergänzt. Für seine Aufgaben nutzt der Optimierer üblicherweise Daten über die zu verarbeitenden Daten, sogenannte *Metadaten*.

  Ein solcher Optimierungsschritt macht natürlich auch für die Datenstromverarbeitung Sinn. Die Schwierigkeiten dabei liegen darin, dass die Kriterien zur Bewertung der verschiedenen Möglichkeiten auf die geänderten Erfordernisse angepasst werden müssen. Zudem liegen über Datenströme, die erst noch eintreffen müssen, in der Regel erheblich weniger Informationen vor, als DSMS diese in Form von Metadaten bezüglich der Datenbank zur Verfügung haben.

- Als nächster Schritt folgt die Übersetzung des logischen Anfragegraphen in einen physischen Anfragegraphen, der nun aus konkreten Operatoren besteht, die jeweils einen bestimmten Verarbeitungsschritt konkret durchführen können. Zu einem logischen Operator kann es dabei eine Vielzahl von physischen Übersetzungszielen geben. Die Entscheidung, welches davon gewählt wird, ist meist bereits Teil des Optimierungschrittes.

  Für die Datenstromverarbeitung adaptiert werden muss dieser Schritt dahingehend, dass die verwendeten Bausteine, die im nächsten Schritt die Anfrage ausführen, andere sein müssen, nämlich solche die Datenströme verarbeiten können.

- Der letzte Schritt ist die Anfrageausführung. In DBMS werden die Ergebnisse dabei üblicherweise schrittweise konsumiert. Die Operatoren kommunizieren hierbei zumeist über eine sogenannte *Open-Next-Close-Schnittstelle* (ONC). Dabei wird das Konzept der bedarfsgesteuerten Ausführung auf die Operatoren übertragen: Jeder Operator fordert die Daten, die er benötigt, erst bei Bedarf an. Wird beispielsweise ein Filteroperator nach dem nächsten Ergebnis gefragt, so fordert dieser bei seiner Quelle nur so lange Datensätze an, bis einer die Filterbedingung erfüllt.

Dieser wird dann zurückgeliefert. Bei diesem Konzept geben die Operatoren die Abarbeitungsreihenfolge vor.

Da Datenströme nicht bei Bedarf gelesen werden können, sollte die Kommunikation zwischen den Operatoren hier anders ablaufen. Die Analogie besteht nun darin, dass wieder das Konzept bezüglich der gesamten Anfrage, nämlich die datengetriebene Verarbeitung, auf die Operatoren übertragen wird. Trifft also beispielsweise ein Datenstromelement bei einem Filteroperator ein, so prüft er es bezüglich der Filterbedingung. Ist diese erfüllt, so leitet er das Element aktiv an den Folgeoperator weiter. Bei diesem Konzept geben die Daten die Abarbeitungsreihenfolge vor.

1.2.2. Elementrepräsentation

Bei der vorstehenden Diskussion der Übertragung der Konzepte der Anfrageverarbeitung noch unberücksichtigt blieb die Fragestellung, wie das Konzept umgesetzt wird, durch das Einbeziehen des Faktors Zeit das Problem zu lösen, mit endlichen Ressourcen potentiell unbeschränkte Datenströme zu verarbeiten. Dies wird zumeist dadurch erreicht, dass man zwischen den physischen Operatoren nicht mehr nur Datensätze austauscht, sondern diese zusätzlich mit Informationen bezüglich ihrer Gültigkeit versieht.

Dafür haben sich im Wesentlichen zwei Ansätze etabliert: Entweder werden der Start und das Ende der Gültigkeit eines Datensatzes durch separate Datenstromelemente angezeigt, oder die Information wird in einem einzelnen Element bestehend aus einem Datensatz und einem halboffenen Gültigkeitsintervall zusammengefasst. Der zweite Ansatz wird als *Intervallansatz* bezeichnet.

1.3. Motivation

Die Möglichkeiten eines nach dem im vorherigen Abschnitt erläuterten Vorgehen entworfenen DSMS hängen nun davon ab, wie gründlich und vollständig die Übertragung von Komponenten erfolgt und welche Erweiterungen vorgenommen werden.

Viele einfache Umsetzungen des Konzeptes beschränken sich insbesondere bezüglich der unterstützten Operatoren. Beliebt ist beispielsweise die Kombination der unären Operatoren Selektion (Filter), Projektion und Aggregation. Diese Kombination hat den besonderen Vorteil, dass jeder der Operatoren pro einströmendem Element maximal ein Ausgabeelement erzeugt. Solche Einschränkungen vereinfachen nicht nur die Bereitstellung der Operatoren enorm, sondern auch andere Arbeiten wie die Bereitstellung einer Anfragesprache und die Optimierung. Allerdings sind solche Systeme auch bezüglich der damit umsetzbaren Anwendungen entscheidend limitiert. Die wirklich interessanten Anwendungen erfordern nämlich fast immer die Kombination von Informationen aus verschiedenen Datenströmen und damit mindestens die Verwendung binärer Operatoren.

Betrachtet man die üblichen binären Operatoren für DBMS, so kommt dem *Verbund* (englisch: *Join*) eine besondere Bedeutung zu. Dabei handelt es sich aus Sicht der relationalen Algebra, die die Grundlage der Semantik der meisten Datenbanksysteme bildet, um eine abgeleitete Operation, nämlich die Kombination des binären Operators Kreuzprodukt (auch kartesisches Produkt genannt) mit den unären Operatoren Selektion

und optional noch Projektion. Umgangssprachlich formuliert lautet die Funktionalität dabei so: Bilde alle Paare aus je einem Datensatz der beiden Eingaben, bestimme davon diejenigen, die eine vorgegebene Bedingung erfüllen, und liefere von diesen einen vorgegebenen Teil der Information. Diese Formulierung lässt sich unter geeigneter Beachtung der Gültigkeitsinformationen direkt für die Datenstromverarbeitung übernehmen. Der Vorteil der Kombination der drei zu Grunde liegenden Operationen in einen einzigen Operator liegt nun darin, nicht erst alle Paare bilden zu müssen, sondern direkt solche zu bestimmen, die die Bedingung erfüllen.

Im Sinne des Complex Event Processing erkennt die Verbundoperation also anhand eines vorgegebenen Kriteriums interessante Kombinationen aus einfachen Ereignissen und leitet daraus ein komplexes Ereignis ab. Genau diese Möglichkeit zur Kombination von Informationen aus verschiedenen Datenströmen ermöglicht erst die Mehrzahl der interessanten Anwendungsfälle.

Nimmt man zum Verbund noch eine stark verwandte Operation, den sogenannten äußeren Verbund, hinzu, so kann man mit Hilfe dieser Verbundoperationen und den unären Operatoren der relationalen Algebra deren binäre Operatoren ableiten. Dies ist ein weiteres starkes Indiz für die Bedeutung der Verbundoperation für die Möglichkeiten eines DSMS.

Ein wichtiger Schritt bei der Entwicklung eines DSMS ist also die Bereitstellung einer Implementierung der Verbundoperation. Analog zu DBMS empfiehlt es sich aber, darüber hinaus für verschiedene Spezialfälle gleich eine Vielzahl von Verbundimplementierungen zur Verfügung zu haben, um die Effizienz zu erhöhen. Um diese jedoch geeignet einsetzen zu können, muss der Optimierer für den Übersetzer eine möglichst gute Auswahl bezüglich der für eine konkrete Anfrage zu verwendenden Implementierung treffen können. Dafür benötigt der Optimierer geeignete Metadaten. Da diese im Falle eines DSMS bei der Installation einer Anfrage oft nur eingeschränkt verfügbar sind, ist es wichtig, während der Laufzeit der kontinuierlichen Anfrage weitere Metadaten zu erheben und auf dieser Basis die Ausführung weiter zu optimieren. Die Erhebung solcher dynamischer Metadaten und deren Einsatz in Techniken zur Reoptimierung spielen also neben effizienten Implementierungen eine wesentliche Rolle für die effiziente Ausführung von Verbundoperationen in der Datenstromverarbeitung.

1.4. Zielsetzung

Ziel dieser Arbeit ist es nun, die Verbundoperation in Datenstrommanagementsystemen, die datengetriebene Operatorgraphen und den Intervallansatz verwenden, effizient zu unterstützen.

Ziel der Arbeit ist also ausdrücklich nicht, ohne Kontext irgendeinen Spezialfall einer Verbundoperation zu definieren und für das derart konstruierte Problem eine spezielle Lösung anzubieten. Vielmehr soll für ein DSMS die Möglichkeit geschaffen werden, das gesamte breite Spektrum an Verbundoperationen geeignet zu unterstützen. Die Semantik des Verbundes soll dabei solide aus der bewährten Semantik von Verbundoperationen aus relationalen Datenbanksystemen hergeleitet werden.

Neben der Bereitstellung effizienter Implementierungen der Verbundoperation soll

insbesondere auch die nötige Infrastruktur zu deren Einsatz geliefert werden. Dies beinhaltet Kostenmodelle zur Bewertung von möglichen Ausführungsplänen, Techniken zum Austausch der Implementierungen zur Laufzeit, äußere Verbünde und eine Reihe von Optimierungen. Insbesondere umfasst dies jedoch auch Techniken zur Beobachtung des Verhaltens eines DSMS zur Laufzeit, also zur Bestimmung von dynamischen Metadaten. Auf Grund deren essentieller Bedeutung für den effizienten Einsatz der Verbundoperation ist den dynamischen Metadaten ein eigener Teil dieser Arbeit gewidmet.

Ziel der Arbeit ist dabei auch, die praktische Umsetzbarkeit der entwickelten Lösungen mit Hilfe der *Public Infrastructure for Processing and Exploring Streams* (PIPES), des DSMS-Forschungsprototypen der Philipps-Universität Marburg, aufzuzeigen und diese experimentell zu evaluieren.

1.5. Veröffentlichungen

Grundlegende Ergebnisse der zentralen Themen dieser Arbeit wurden bereits auf Konferenzen veröffentlicht.

- Michael Cammert, Christoph Heinz, Jürgen Krämer und Bernhard Seeger. Sortierbasierte Joins über Datenströmen. In *Proceedings of the Conference on Database Systems for Business, Technology, and the Web (BTW)*, 365–384, 2005.
- Michael Cammert, Jürgen Krämer und Bernhard Seeger. Dynamic Metadata Management for Scalable Stream Processing Systems. In *Proceedings of the First International Workshop on Scalable Stream Processing Systems (SSPS)*, 2007.

Zusätzlich wurden in diversen Publikationen Ergebnisse veröffentlicht, die im Zusammenhang mit dieser Arbeit entstanden und auf denen diese aufbaut.

- Michael Cammert, Christoph Heinz, Jürgen Krämer, Martin Schneider und Bernhard Seeger. A Status Report on XXL – a Software Infrastructure for Efficient Query Processing. *IEEE Data Engineering Bulletin*, 26(2):12–18, 2003.
- Michael Cammert, Christoph Heinz, Jürgen Krämer und Bernhard Seeger. Datenströme im Kontext des Verkehrsmanagements. In *Mobilität und Informationssysteme, Workshop des GI-Arbeitskreises „Mobile Datenbanken und Informationssysteme“*, 2003.
- Michael Cammert, Christoph Heinz, Jürgen Krämer und Bernhard Seeger. Anfrageverarbeitung auf Datenströmen. *Datenbank Spektrum*, 11:5–13, 2004.
- Michael Cammert, Christoph Heinz, Jürgen Krämer, Tobias Riemenschneider, Maxim Schwarzkopf, Bernhard Seeger und Alexander Zeiss. Stream Processing in Production-to-Business Software. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 168–169, 2006.
- Michael Cammert, Jürgen Krämer, Bernhard Seeger und Sonny Vaupel. An Approach to Adaptive Memory Management in Data Stream Systems. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 137–139, 2006.

- Jürgen Krämer, Yin Yang, Michael Cammert, Bernhard Seeger und Dimitris Papadias. Dynamic Plan Migration for Snapshot-Equivalent Continuous Queries in Data Stream Systems. In *Proceedings of the International Conference on Extending Data Base Technology (EDBT) Workshops*, 497–516, 2006.
- Michael Cammert, Christoph Heinz, Jürgen Krämer, Bernhard Seeger, Sonny Vaupel und Udo Wolske. Flexible Multi-Threaded Scheduling for Continuous Queries over Data Streams. In *Proceedings of the First International Workshop on Scalable Stream Processing Systems (SSPS)*, 2007.
- Michael Cammert, Jürgen Krämer, Bernhard Seeger und Sonny Vaupel. A Cost-Based Approach to Adaptive Resource Management in Data Stream Systems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2007.

1.6. Gliederung der Arbeit

Der Rest dieser Arbeit ist wie folgt aufgebaut:

- Im anschließenden Kapitel 2 dieses Einleitungsteiles werden die in diesem Kapitel zunächst informell beschriebenen Grundlagen aus dem Gebiet der Datenstromverarbeitung, die für diese Arbeit relevant sind, formal definiert sowie diskutiert.
- Dynamische Metadaten spielen für die Nutzung der Verbundoperation eine wichtige Rolle. Da sie aber auch unabhängig davon und darüber hinaus von wichtiger Bedeutung für die Datenstromverarbeitung sind, wird das Thema Metadaten zunächst separat in Teil II behandelt. Darin wird ausführlich auf die effiziente Erfassung und Verarbeitung dynamischer Metadaten eingegangen.
- Teil III ist dann dem zentralen Thema Verbundoperationen gewidmet. Deren Semantik wird formal hergeleitet, bevor eine Reihe von parametrisierbaren Implementierungen vorgestellt wird. Zudem werden umfassend Techniken zur Optimierung und Reoptimierung zur Laufzeit dargelegt.
- In Teil IV werden dann Experimente zu den in den vorstehenden Teilen beschriebenen Lösungen diskutiert.
- Teil V betrachtet dann zu dieser Arbeit verwandte Arbeiten aus der Literatur.
- Abschließend liefert Teil VI eine Diskussion der Ergebnisse dieser Arbeit und gibt einen Ausblick auf mögliche darauf aufbauende Forschungsziele.

2. Grundlagen

Während der Fokus von Kapitel 1 auf der Einführung in das Thema Datenstromverarbeitung und die Motivation dieser Arbeit liegt, dient dieses Kapitel der formellen Definition der für diese Arbeit relevanten Grundlagen.

Einige der grundlegenden Definitionen in dieser Arbeit orientieren sich dabei an denen in [KS05, Krä07, CKSV08, KS09], weichen jedoch an einigen Stellen auch leicht ab.

2.1. Elemente

Obwohl relationale Datenbanksysteme einen extrem hohen Marktanteil haben, gibt es auch andere sinnvolle Datenmodelle für DBMS. Dementsprechend möchte man auch mit DSMS relationale Tupel verarbeiten können, sich aber nicht zwingend auf solche festlegen. Daher werden für die Elemente lediglich ein Typ und eine Menge von möglichen Werten dieses Typs als Grundlage verlangt.

Definition 2.1 (Elementtyp und Wertebereich der Elemente)**.** Zu einem *Elementtyp* $\mathcal{T}$ sei $\Omega_{\mathcal{T}}$ die *Menge der Elemente* vom Typ $\mathcal{T}$.

2.2. Zeit

Der Faktor Zeit spielt in DSMS eine entscheidende Rolle. Da Computer Zeit nur mit einer endlichen Genauigkeit erfassen können, bietet es sich an, den verwendeten Zeitpunkten eine diskrete Menge zu Grunde zu legen. Da die Zeit fortschreitet, sollte die Menge geordnet sein; um Abstände von Zeitpunkten berechnen zu können, benötigt man zudem eine Additionsoperation. Es bietet sich daher an, als Zeitdomäne einen zu $\mathbb{Z}$ isomorphen Ring zu verwenden.

Definition 2.2 (Zeitdomäne)**.** Sei $\mathbb{T} \equiv \mathbb{Z}$ eine *Zeitdomäne*.

Es handelt sich hierbei um logische Zeitpunkte, die nicht zwingend in Bezug zur Verarbeitungszeit im System stehen. Damit soll insbesondere erreicht werden, dass die Ergebnisse der Datenstromverarbeitung unabhängig von der Verarbeitungszeit sind, also beispielsweise nicht durch Verzögerungen beim Eintreffen von Elementen beeinflusst werden.

2.3. Datenströme

Von essentieller Bedeutung für ein DSMS ist natürlich die Definition, was ein Datenstrom ist, da diese Gegenstand der Verarbeitung sind. Bei der Definition ist es entscheidend,

dass mit ihrer Hilfe zwei Fragen beantwortet können werden müssen: Was soll das DSMS berechnen können und wie soll es das erreichen?

2.3.1. Logische Datenströme

Für die Frage nach dem *Was* ist eine Definition eines Datenstroms hilfreich, die einem Menschen ein einfaches Verständnis des Datenstroms ermöglicht. Ob diese Darstellung effizient in einem Computer gespeichert und verarbeitet werden kann, stellt für diese logische Sicht auf einen Datenstrom hingegen keine Rolle. Sie dient alleine der Definition der Semantik.

Ein Datenstrom soll nun Elemente eines Elementtyps $\mathcal{T}$ enthalten können, die nur zu einem Teil der möglichen Zeitpunkte $\mathbb{T}$ gültig sind. Zudem soll es möglich sein, dass ein Element zum selben Zeitpunkt mehrmals vorhanden ist. Ein logischer Datenstrom ist nun eine Menge, die für Elemente und Zeitpunkte angibt, wie oft das Element zu diesem Zeitpunkt gültig, also im Strom enthalten ist.

Definition 2.3 (Logischer Datenstrom)**.** Sei $S^l \subseteq \Omega_{\mathcal{T}} \times \mathbb{T} \times \mathbb{N}^+$, $a : \Omega_{\mathcal{T}} \times \mathbb{T} \to S^l$, $(e,t) \to (e,t,n)$ eine Abbildung und habe $\{n|(e,t,n) \in S^l\}$ ein Maximum. Dann ist S^l ein *logischer Datenstrom* vom Typ $\mathcal{T}$. Die Menge aller logischen Datenströme vom Typ $\mathcal{T}$ sei $\mathbb{S}^l_{\mathcal{T}}$.

Die Bedingung der Existenz der Abbildung a sorgt dafür, dass zu jedem Paar aus Element und Zeitpunkt nur maximal eine Anzahl existiert. Ist diese Anzahl Null, so ist kein Element im logischen Datenstrom enthalten. Zudem ist gefordert, dass es eine maximale Häufigkeit gibt.

Um zwei Datenströme als verschieden ansehen zu können, müsste es einen Zeitpunkt t und ein Element e geben, so dass das e bei t in einem der Ströme häufiger gültig ist als im anderen. Dementsprechend sind zwei logische Datenströme äquivalent, wenn zu jedem Zeitpunkt und für jedes Element die Gültigkeitsanzahl gleich ist. Die Darstellung eines Datenstroms als logischer Datenstrom ist dementsprechend aufgrund der Abbildungsbedingung eindeutig.

2.3.2. Eingabedatenströme

Für die Frage nach dem *Wie* ist nun zunächst noch zu klären, in welcher Form die Eingabedatenströme beim System eintreffen sollen. Ein Datenstrom ist nun naturgemäß zunächst eine Folge von Daten. Zur Formalisierung werden diese Elemente wieder auf einen gemeinsamen Typ $\mathcal{T}$ festgelegt. Für den zeitlichen Bezug wird nun noch verlangt, dass jedes solche Element mit einem Zeitstempel aus der Zeitdomäne $\mathbb{T}$ versehen ist. Die Forderung eines solchen Zeitstempels ist in der Datenstromforschung allgemein akzeptiert.[GÖ03b, ABW06, ACC+03, ACG+04, SQR03]

Zusätzlich sollen im Eingabedatenstrom die Zeitstempel monoton steigend sein. Dadurch wird verhindert, dass jederzeit wieder ältere Elemente als die zuletzt eingetroffenen nachfolgen können.

Definition 2.4 (Eingabedatenstrom)**.** Eine Folge a mit $a_i := (e_i, t_i) \in \Omega_{\mathcal{T}} \times \mathbb{T}$ ist ein *Eingabedatenstrom* S^r vom Typ $\mathcal{T}$, falls $\forall i,j \in \mathbb{N} : i < j \Rightarrow t_i \leq t_j$. Die Menge aller Eingabeströme vom Typ $\mathcal{T}$ sei $\mathbb{S}^r_{\mathcal{T}}$.

Da die Reihenfolge von Elementen mit gleichem Zeitstempel nicht vorgegeben wird, ist diese Darstellung nicht eindeutig, das heißt zwei Eingabeströme, die dieselben Elemente enthalten, können sich in der Abfolge ihrer Elemente unterscheiden.

Natürlich soll ein DSMS auch andere Arten von Eingabeströmen verarbeiten können, um auf möglichst viele Probleme und in möglichst vielen Systemumgebungen anwendbar zu sein. Der allgemeinste Datenstrom ist dabei einfach eine Abfolge von Werten. Identifiziert man für diese einen gemeinsamen Obertyp, so ist die erste Komponente eines Eingabestroms gefunden. Nun muss noch ein geeigneter Zeitstempel zu jedem Element gefunden werden. Gibt es keine sinnvolle Variante, diesen aus den jeweiligen Elementen zu berechnen, die ja auch temporale Informationen beinhalten könnten, so besteht immer noch die Möglichkeit, als Zeitstempel den Zeitpunkt des Eintreffens des Elements beim DSMS zu verwenden. In diesem Fall erfolgt hier ein Übergang von physischer Zeit in logische Zeit. Die reale Systemzeit wird also nur bei der Vergabe der Zeitstempel benutzt, die darauffolgende Verarbeitung im System betrachtet den Zeitstempel als logischen Zeitstempel, wodurch die Verarbeitungslogik von der Systemzeit unabhängig bleibt.

2.3.3. Physische Datenströme

Die eigentliche Frage nach dem *Wie* ist nun die Frage nach der internen physischen Repräsentation der Datenströme im realen System. Ihr kommt eine fundamentale Bedeutung zu, da sie die Implementierung aller Operatoren beeinflusst und damit grundlegend die Performanz des Systems mitbestimmt.

Im Unterschied zum logischen Datenstrom, bei dem für jedes Element für jeden Zeitpunkt ein einzelnes Tupel existiert, gilt es, bei den physischen Strömen die Information über mehrere Zeitpunkte zu bündeln, um so die Zahl der zu verarbeitenden Stromelemente zu reduzieren. Dies ist möglich, da Werte üblicherweise über einen Zeitraum von mehreren aufeinander folgenden Zeitpunkten hinweg im logischen Strom enthalten sind. Einen solchen Zeitraum der Gültigkeit eines Elements e im Strom kann man als halboffenes Zeitintervall $[t_s, t_e)$ mit $t_s, t_e \in \mathbb{T}$ repräsentieren, wobei t_s der erste der aufeinanderfolgenden Zeitpunkte ist, an denen das Element gültig ist, und t_e entsprechend der erste Zeitpunkt danach, an dem es nicht mehr gültig ist.

Beim sogenannten *Positiv/Negativ-Ansatz*[ABW06, GHM+07] wird so ein Zeitraum derart abgebildet, dass zu Beginn des Zeitraums ein Stromelement $(e, t_s, +)$ und direkt nach dessen Ende ein Stromelement $(e, t_s, -)$ gesendet wird. Diese sind als positiv beziehungsweise negativ gekennzeichnet und signalisieren so, dass die Gültigkeit des Elements beginnt beziehungsweise endet. Dieser Ansatz ist zwar weit verbreitet, hat jedoch den Nachteil, dass jedes Element immer zweimal gesendet werden muss, wenn es für einen Zeitraum gültig ist.

Für PIPES wurde daher der *Intervallansatz* entwickelt und gewählt, bei dem ein Element direkt mit einem Gültigkeitsintervall übertragen wird, also als $(e, [t_s, t_e))$. Dabei wird t_s als *Startzeitstempel* und t_e als *Endzeitstempel* bezeichnet. Zusätzlich sollen die Startzeitstempel der Zeitintervalle monoton steigend eintreffen. Dadurch wird verhindert, dass jederzeit wieder ältere Elemente als die zuletzt eingetroffenen nachfolgen können.

Definition 2.5 (Physischer Datenstrom)**.** Sei $I_{\mathbb{T}} := \{[t_s, t_e) | t_s, t_e \in \mathbb{T} \wedge t_s < t_e\}$ die Menge

der rechts halboffenen Zeitintervalle über der Zeitdomäne $\mathbb{T}$. Eine Folge b mit $b_i := (e_i, [t_{s_i}, t_{e_i})) \in \Omega_{\mathcal{T}} \times I_{\mathbb{T}}$ ist ein *physischer Datenstrom* S^p vom Typ $\mathcal{T}$, falls $\forall i, j \in \mathbb{N} : i < j \Rightarrow t_{s_i} \leq t_{s_j}$ und $\forall t \in \mathbb{T} : |\{(e_i, [t_{s_i}, t_{e_i})) \in b \mid t_{s_i} = t\}| \in \mathbb{N}$. Die Menge aller physischer Datenströme vom Typ $\mathcal{T}$ sei $\mathbb{S}^p_{\mathcal{T}}$.

Sei $\bar{\omega} : \mathbb{S}^p_{\mathcal{T}} \to \wp((\Omega_{\mathcal{T}} \times I_{\mathbb{T}}) \to \mathbb{N})$ definiert durch $\bar{\omega}((e_1, e_2, \ldots)) := f$ mit $f(e) = |\{i \mid e_i = e\}|$. Sei $\omega : \wp((\Omega_{\mathcal{T}} \times I_{\mathbb{T}}) \to \mathbb{N}) \to \mathbb{S}^p_{\mathcal{T}}$ eine Abbildung mit $\bar{\omega}(\omega(x)) = x$.

Die Funktionen $\bar{\omega}$ und ω dienen in dieser Arbeit als Hilfsfunktionen, die es erlauben, physische Datenströme durch Angabe der Vielfachheiten ihrer Stromelemente (e, i) anzugeben. Mit Hilfe der Funktion ω erhält man dann aus einer solchen Vielfachheitsfunktion eine mögliche Anordnung der Stromelemente als physischer Datenstrom; die Stromelemente werden also nach Startzeitstempeln geordnet. Welche der gegebenenfalls mehreren möglichen Reihenfolgen gewählt wird, bleibt dabei offen. Da zu einem Zeitpunkt t nur die Gültigkeit einer endlichen Zahl von Stromelementen beginnen kann, existiert allerdings immer mindestens eine geeignete Funktion ω.

Gegenüber dem Positiv/Negativ-Ansatz wird bei der hier verwendeten Repräsentation physischer Datenströme die gleiche Information unter Einsparung der zweiten Übertragung von e sowie ohne + und − geliefert. Die wesentlichen Vorteile des Ansatzes werden jedoch erst bei der Verarbeitung der Datenströme deutlich. Eine Analyse der Vor- und Nachteile der beiden Ansätze, welche die Überlegenheit des Intervallansatzes aufzeigt, findet sich in [Krä07]. In [GHM+07] wird ein dem Intervallansatz mit Heartbeats (siehe 10.2.2) ähnlicher Ansatz als Optimierung des Positiv/Negativ-Ansatzes entwickelt.

Im Gegensatz zu den logischen Datenströmen fehlt bei den physischen Strömen auch die Vielfachheit der Werte zu einem Zeitpunkt. Diese wird dadurch ausgedrückt, dass ein Element mehrmals für identische Zeitpunkte in einem Strom vorkommen kann.

Wie schon bei den Eingabedatenströmen ist schon auf Grund der fehlenden Ordnungsvorgabe für Elemente mit gleichem Startzeitstempel die Darstellung äquivalenter physischer Datenströme nicht eindeutig. Hinzu kommt aber noch der Effekt, dass ein Zeitintervall getrennt beziehungsweise zusammengefügt werden kann. Das Stromelement $(a, [t_s, t_e))$ ist also für $t_s < t_t < t_e$ semantisch äquivalent zu den beiden Stromelementen $(a, [t_s, t_t))$ und $(a, [t_t, t_e))$, da beide Darstellungen ausdrücken, dass das Element a vom Zeitpunkt t_s einschließlich bis zum Zeitpunkt t_e ausschließlich gültig ist.

Die Abbildung $\bar{\omega}$ liefert zu einem physischen Datenstrom die Multimenge der darin enthaltenen Stromelemente. Umgekehrt dient die Abbildung ω dazu, eine Multimenge von physischen Stromelementen derart zu einer Folge anzuordnen, dass die Ordnungsbedingung auf den Startzeitstempeln erfüllt ist und somit ein physischer Datenstrom bestehend aus genau diesen Stromelementen erzeugt wird.

2.3.4. Umwandlung der Datenströme

Die verschiedenen Varianten von Datenströmen lassen sich nun ineinander umrechnen. Dabei sollen in allen Darstellungen eines Stroms zu jedem Zeitpunkt dieselben Elemente in derselben Anzahl gültig sein.

Eingabedatenstrom in physischen Datenstrom

Für die Eingabedatenströme muss dabei noch festgelegt werden, wie der Zeitstempel bezüglich der Gültigkeit ausgelegt werden soll. Da jedem Element exakt ein Zeitstempel zugewiesen wird, bietet es sich an, das Element auch nur zu diesem Zeitpunkt als gültig zu betrachten. Damit ergibt sich als Umwandlungsfunktion $\varphi^{r \mapsto p} : \mathbb{S}^r_{\mathcal{T}} \to \mathbb{S}^p_{\mathcal{T}}$:

$$\varphi^{r \mapsto p}(i \to (e_i, t_i)) = i \to (e_i, [t_i, t_i + 1))$$

Physischer Datenstrom in logischen Datenstrom

Um aus einem physischen einen logischen Datenstrom zu machen, muss für jedes Element und jeden Zeitpunkt die Zahl der Gültigkeiten bestimmt werden. Damit ergibt sich als Umwandlungsfunktion $\varphi^{p \mapsto l} : \mathbb{S}^p_{\mathcal{T}} \to \mathbb{S}^l_{\mathcal{T}}$:

$$\begin{aligned}\varphi^{p \mapsto l}(S^p) = \{(e, t, n) \in \Omega_{\mathcal{T}} \times T \times \mathbb{N}^+ \mid \\ n = \ |\{i \to (e_i, [t_{s_i}, t_{e_i})) \in S^p \mid e = e_i \wedge t \in [t_{s_i}, t_{e_i})\}| \ \}\end{aligned}$$

Logischer Datenstrom in Eingabedatenstrom

Bei der Umwandlung eines logischen Datenstroms in einen Eingabestrom wird schließlich ein Element mit n-facher Gültigkeit zum Zeitpunkt t in n einzelne Stromelemente umgesetzt. Damit ergibt sich als rekursive Umwandlungsfunktion $\varphi^{l \mapsto r} : \mathbb{S}^l_{\mathcal{T}} \to \mathbb{S}^r_{\mathcal{T}}$:

$$\varphi^{l \mapsto r}(S^l) = \begin{cases} \emptyset, S^l = \emptyset \\ \underbrace{(e, t), \ldots, (e, t)}_{n-mal}, S^l = \{(e, t, n)\} \\ \varphi^{l \mapsto r}(\{(e, t, n)\}) \circ \varphi^{l \mapsto r}(S^l \backslash \{(e, t, n)\}), (e, t, n) \in S^l, t = min\{t' | (e, t', n) \in S^l\} \end{cases}$$

Eines der Elemente mit dem minimalen Zeitstempel im logischen Strom und Vielfachheit n bildet also die ersten n Folgenglieder des Eingabestroms. Danach folgt die Umwandlung des restlichen Stroms entsprechend.

Weitere Umwandlungen

Aus den drei angegebenen Umwandlungen lassen sich die fehlenden Umwandlungen zwischen den anderen Stromdarstellungen durch Verkettung ableiten.

$$\begin{aligned}\varphi^{r \mapsto l} &= \varphi^{p \mapsto l} \circ \varphi^{r \mapsto p} \\ \varphi^{p \mapsto r} &= \varphi^{l \mapsto r} \circ \varphi^{p \mapsto l} \\ \varphi^{l \mapsto p} &= \varphi^{r \mapsto p} \circ \varphi^{l \mapsto r}\end{aligned}$$

Da die Darstellungen eines Datenstroms als Eingabestrom oder physischer Datenstrom nicht eindeutig sind, sind die angegebenen Umwandlungen nur eine mögliche

Variante. In der Praxis ist für physische Datenströme immer eine Minimierung der Zahl der Stromelemente anzustreben, im Falle nicht-endlicher Ströme bezüglich endlicher Teilströme.[1]

2.3.5. Äquivalenz von Datenströmen

Mit Hilfe der Umwandlungen und der Eindeutigkeitseigenschaft der logischen Datenströme kann nun die Äquivalenz zweier Datenströme formal definiert werden.

Definition 2.6 (Äquivalenz von logischen Datenströmen)**.** Zwei logische Datenströme $S_1^l, S_2^l \in \mathbb{S}_{\mathcal{T}}^l$ sind äquivalent, wenn $S_1^l = S_2^l$. Als Schreibweise dafür wird $S_1^l \equiv S_2^l$ verwendet.

Demnach sind zwei logische Datenströme äquivalent, wenn sie gleich sind.

Definition 2.7 (Äquivalenz von physischen Datenströmen)**.** Zwei physische Datenströme $S_1^p, S_2^p \in \mathbb{S}_{\mathcal{T}}^p$ sind äquivalent, wenn $\varphi^{p\mapsto l}(S_1^p) \equiv \varphi^{p\mapsto l}(S_2^p)$. Als Schreibweise dafür wird $S_1^p \equiv S_2^p$ verwendet.

Physische Datenströme sind also äquivalent, wenn ihre logischen Darstellungen gleich sind. Sie können sich somit bezüglich der Stromelemente unterscheiden und trotzdem äquivalent sein. Für sie wird daher zusätzlich eine verschärfte Form der Äquivalenz eingeführt, die bis auf Umordnung der Stromelemente die Gleichheit der Ströme fordert, also dass sie jedes physische Element gleich oft enthalten.

Definition 2.8 (Physische Äquivalenz von physischen Datenströmen)**.** Zwei physische Datenströme $S_1, S_2, \in \mathbb{S}_{\mathcal{T}}^p$ sind physisch äquivalent, wenn $\bar{\omega}(S_1) = \bar{\omega}(S_2)$ gilt. Als Schreibweise dafür wird $S_1 \equiv^p S_2$ verwendet.

2.3.6. Beispiel

Wählt man zum Beispiel $\mathcal{T} := \mathbb{N}$ als Schema (und somit $\Omega_{\mathcal{T}}$ als $\mathbb{N}$) und $\mathbb{T} := \mathbb{Z}$, so erhält man Ströme von natürlichen Zahlen mit ganzzahligen Zeitstempeln. Gegeben seien nun folgende Datenströme:

$$S^l := \{(42, 1000, 1), (42, 1001, 2), (42, 1002, 1)\}$$

$$S_1^p := (42, [1000, 1002)), (42, [1001, 1003))$$

$$S_2^p := (42, [1000, 1001)), (42, [1001, 1002)), (42, [1001, 1003))$$

$$S_3^p := (42, [1000, 1001)), (42, [1001, 1003)), (42, [1001, 1002))$$

S^l bildet einen logischen Datenstrom, da für jeden Zeitpunkt $(1000, 1001, 1002)$ und jedes Element (42) die Gültigkeit eindeutig und nicht größer als 2 ist.

S_1^p, S_2^p und S_3^p sind äquivalent, da ihre logische Darstellung jeweils S^l ergibt: $\varphi^{p\mapsto l}(S_1^p) = \varphi^{p\mapsto l}(S_2^p) = \varphi^{p\mapsto l}(S_3^p) = S^l$. Dabei sind allerdings nur S_2^p und S_3^p als Folgen gleicher Elemente

[1] Zu einem endlichen physischen Teilstrom mit m Elementen lässt sich dabei in $O(m)$ ein minimaler äquivalenter physischer Strom berechnen. Bei der Verwendung physischer Ströme mit Vielfachheit wäre dieses Problem hingegen NP-hart.

auch physisch äquivalent, während S_1^p schon daher nicht physisch äquivalent zu diesen sein kann, weil er eine andere Anzahl von Elementen hat.

2.4. Datenstromalgebra

2.4.1. Motivation

Während ein DBMS schon alleine durch die persistente Speicherung der eingefügten Daten eine Aufgabe erfüllt, selbst wenn die Daten nur unverändert wieder ausgelesen werden, erhält ein DSMS seine Daseinsberechtigung erst durch die Verarbeitung der Datenströme, also indem es andere Datenströme als Ausgabe liefert, als es als Eingabe erhalten hat. Bei der Definition der Semantik der Verarbeitung bietet sich wie bei vielen Fragestellungen der DSMS-Entwicklung ein Blick auf die Erfahrungen aus dem Bereich relationaler DBMS an. Auf diese Art werden nicht nur sinnvolle Lösungen weiter verwendet, sondern auch den Benutzern vertraute Konzepte verwendet, was die Akzeptanz der Lösungen fördert.

Grundlage der Semantik relationaler Datenbanksysteme ist die erweiterte relationale Algebra[DGK82, GUW00], einer Weiterentwicklung der relationalen Algebra[Cod70]. Den Anker der Algebra bilden dabei Relationen, die man mit Multimengen von Tupeln über einem relationalen Schema identifizieren kann. Über den Relationen sind dann verschiedenste Operationen definiert, bezüglich derer die Algebra abgeschlossen ist. Neben einer Basis an Grundoperationen ist eine Vielzahl von abgeleiteten Operationen üblich, zu denen auch alle Varianten des Verbundes (Joins) zählen. Sowohl die Grundoperationen als auch die abgeleiteten Operationen lassen sich dabei auf die Verarbeitung von Multimengen über nichtrelationalen Schemata erweitern.

Die Datenströme haben nun mit den Zeitinformationen und der diesbezüglichen Ordnung zwei Eigenschaften, die in relationalen Datenbanken keine entsprechende Rolle spielen. Betrachtet man jedoch nur diejenigen Elemente, die in einem Datenstrom zu einem festen Zeitpunkt t gültig sind, so erhält man eine Multimenge ohne ausgezeichnete zeitlich Information. Möchte man nun den Ausgabestrom einer Operation über Datenströmen definieren, so kann man dies erreichen, indem man für jeden Zeitpunkt t die im Ausgabestrom gültigen Elemente festlegt. Wählt man diese als die Multimenge, die man erhält, wenn man einen Operator O über Multimengen auf die Mengen anwendet, die man erhält, wenn man die zum Zeitpunkt t in den zu verarbeitenden Strömen gültigen Elemente betrachtet, so hat man aus dem Operator O einen Datenstromoperator abgeleitet. Die betrachteten Multimengen, die zu einem Zeitpunkt t gültig sind, bezeichnet man dabei als *Schnappschuss* und den abgeleiteten Operator als *Schnappschuss-reduzierbar* auf den ursprünglichen Operator.

Golab und Özsu betrachten diese Art des Zurückführens der Semantik auf relationale Anfragen als allgemein akzeptiert:[GÖ05]

> „Let $Q(\tau)$ be the answer of a continuous query Q at time τ. ... consider the following generally accepted definition of continuous query semantics. Definition 1. At any time τ , $Q(\tau)$ must be equal to the output of a corresponding one-time relational query whose inputs are the current states of the streams, sliding windows, and relations referenced in Q.“

Dennoch gibt es, insbesondere auch im Bereich der Verbundberechnung, eine Vielzahl von Semantiken, die davon abweichen.

2.4.2. Operatoren

Zunächst wird der Begriff des Operators definiert.

Definition 2.9 (Operatoren)**.** Seien $\mathcal{T}_1, \ldots, \mathcal{T}_n$ und $\mathcal{T}$ Typen. Eine Abbildung $op^l : \mathbb{S}^l_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}_n} \to \mathbb{S}^l_{\mathcal{T}}$ ist ein n-stelliger *logischer Operator*, eine Abbildung $op^p : \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n} \to \mathbb{S}^p_{\mathcal{T}}$ ist ein n-stelliger *physischer Operator*.

2.4.3. Operatorgraphen

Während in Datenbankmanagementsystemen die Anfragen üblicherweise getrennt voneinander ausgeführt werden, bietet es sich angesichts der kontinuierlichen Anfrageausführung ins Datenstrommanagementsystemen an, mehrfach auftretende Anfragen und Teilanfragen nur einmal auszuführen. Dies ist einer der Gründe dafür, dass die Operatoren im Allgemeinen keinen Wald, also eine Ansammlung von Anfragebäumen, sondern vielmehr einen Graphen mit mehreren Quellen und Senken bilden.

Definition 2.10 (Operatorgraphen)**.** Ein Graph aus Operatoren über Multimengen mit n Eingängen vom Typ $\mathcal{T}^e_1, \ldots, \mathcal{T}^e_n$ und m Ausgängen vom Typ $\mathcal{T}^a_1, \ldots, \mathcal{T}^a_m$ stellt eine Abbildung $g^m : \wp(\Omega_{\mathcal{T}^e_1} \to \mathbb{N}) \times \ldots \times \wp(\Omega_{\mathcal{T}^e_n} \to \mathbb{N}) \to \wp(\Omega_{\mathcal{T}^a_1} \to \mathbb{N}) \times \ldots \times \wp(\Omega_{\mathcal{T}^a_m} \to \mathbb{N})$ dar und wird *Operatorgraph über Multimengen* genannt.

Ein Graph aus logischen Operatoren mit n Eingängen vom Typ $\mathcal{T}^e_1, \ldots, \mathcal{T}^e_n$ und m Ausgängen vom Typ $\mathcal{T}^a_1, \ldots, \mathcal{T}^a_m$ stellt eine Abbildung $g^l : \mathbb{S}^l_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}^e_n} \to \mathbb{S}^l_{\mathcal{T}^a_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}^a_m}$ dar und wird *logischer Operatorgraph* genannt.

Ein Graph aus physischen Operatoren mit n Eingängen vom Typ $\mathcal{T}^e_1, \ldots, \mathcal{T}^e_n$ und m Ausgängen vom Typ $\mathcal{T}^a_1, \ldots, \mathcal{T}^a_m$ stellt eine Abbildung $g^p : \mathbb{S}^p_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}^e_n} \to \mathbb{S}^p_{\mathcal{T}^a_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}^a_m}$ dar und wird *physischer Operatorgraph* genannt.

Seien $\pi^m_i : \wp(\Omega_{\mathcal{T}_1} \to \mathbb{N}) \times \ldots \times \wp(\Omega_{\mathcal{T}_k} \to \mathbb{N}) \to \wp(\Omega_{\mathcal{T}_i} \to \mathbb{N})$, $\pi^l_i : \mathbb{S}^l_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}_k} \to \mathbb{S}^l_{\mathcal{T}_i}$ und $\pi^p_i : \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_k} \to \mathbb{S}^p_{\mathcal{T}_i}$ die Projektionen eines Tupels von Multimengen sowie logischen beziehungsweise physischen Strömen auf die jeweils i-te Komponente.

Für solche Anfragegraphen, deren Semantik sich aus der der enthaltenen Operatoren ergibt, wird nun ebenfalls der Äquivalenzbegriff definiert. Daraus ergibt sich durch den Spezialfall von Anfragegraphen mit nur einem Knoten auch unmittelbar der Äquivalenzbegriff für Operatoren.

Definition 2.11 (Äquivalenz von Operatorgraphen)**.** Seien $\mathcal{T}^e_1, \ldots, \mathcal{T}^e_n$ und $\mathcal{T}^a_1, \ldots, \mathcal{T}^a_n$ Typen.

$(S^l_1, \ldots, S^l_n)$ und $(\bar{S}^l_1, \ldots, \bar{S}^l_n) \in \mathbb{S}^l_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}^e_n}$ sind äquivalent ($\equiv$), wenn $\forall i \in \{1, \ldots, n\} : S^l_i \equiv \bar{S}^l_i$ ist.

$(S^p_1, \ldots, S^p_n)$ und $(\bar{S}^p_1, \ldots, \bar{S}^p_n) \in \mathbb{S}^p_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}^e_n}$ sind äquivalent ($\equiv$), wenn $\forall i \in \{1, \ldots, n\} : S^p_i \equiv \bar{S}^p_i$ ist.

$(S_1^p, \ldots, S_n^p)$ und $(\bar{S}_1^p, \ldots, \bar{S}_n^p) \in \mathbb{S}^p_{\mathcal{T}_1^e} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n^e}$ sind physisch äquivalent ($\equiv^p$), wenn $\forall i \in \{1, \ldots, n\} : S_i^p \equiv^p \bar{S}_i^p$ ist.

Zwei Operatorgraphen für Multimengen $g^m, \bar{g}^m : \wp(\Omega_{\mathcal{T}_1^e} \to \mathbb{N}) \times \ldots \times \wp(\Omega_{\mathcal{T}_n^e} \to \mathbb{N}) \to \wp(\Omega_{\mathcal{T}_1^a} \to \mathbb{N}) \times \ldots \times \wp(\Omega_{\mathcal{T}_m^a} \to \mathbb{N})$ sind äquivalent ($\equiv$), wenn $\forall (M_1, \ldots, M_n), (\bar{M}_1, \ldots, \bar{M}_n) \in \wp(\Omega_{\mathcal{T}_1^e} \to \mathbb{N}) \times \ldots \times \wp(\Omega_{\mathcal{T}_n^e} \to \mathbb{N}) : (M_1, \ldots, M_n) = (\bar{M}_1, \ldots, \bar{M}_n) \Rightarrow g^m(M_1, \ldots, M_n) = \bar{g}^m(\bar{M}_1, \ldots, \bar{M}_n)$.

Zwei logische Operatorgraphen $g^l, \bar{g}^l : \mathbb{S}^l_{\mathcal{T}_1^e} \times \ldots \times \mathbb{S}^l_{\mathcal{T}_n^e} \to \mathbb{S}^l_{\mathcal{T}_1^a} \times \ldots \times \mathbb{S}^l_{\mathcal{T}_m^a}$ sind äquivalent ($\equiv$), wenn $\forall (S_1^l, \ldots, S_n^l), (\bar{S}_1^l, \ldots, \bar{S}_n^l) \in \mathbb{S}^l_{\mathcal{T}_1^a} \times \ldots \times \mathbb{S}^l_{\mathcal{T}_n^a} : (S_1^l, \ldots, S_n^l) \equiv (\bar{S}_1^l, \ldots, \bar{S}_n^l) \Rightarrow g^l(S_1^l, \ldots, S_n^l) \equiv \bar{g}^l(\bar{S}_1^l, \ldots, \bar{S}_n^l)$.

Zwei physische Operatorgraphen $g^p, \bar{g}^p : \mathbb{S}^p_{\mathcal{T}_1^e} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n^e} \to \mathbb{S}^p_{\mathcal{T}_1^a} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_m^a}$ sind äquivalent ($\equiv$), wenn $\forall (S_1^p, \ldots, S_n^p), (\bar{S}_1^p, \ldots, \bar{S}_n^p) \in \mathbb{S}^p_{\mathcal{T}_1^e} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n^e} : (S_1^p, \ldots, S_n^p) \equiv (\bar{S}_1^p, \ldots, \bar{S}_n^p) \Rightarrow g^p(S_1^p, \ldots, S_n^p) \equiv \bar{g}^p(\bar{S}_1^p, \ldots, \bar{S}_n^p)$.

Zwei physische Operatorgraphen $g^p, \bar{g}^p : \mathbb{S}^p_{\mathcal{T}_1^e} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n^e} \to \mathbb{S}^p_{\mathcal{T}_1^a} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_m^a}$ sind physisch äquivalent ($\equiv^p$), wenn sie äquivalent sind und $\forall (S_1^p, \ldots, S_n^p), (\bar{S}_1^p, \ldots, \bar{S}_n^p) \in \mathbb{S}^p_{\mathcal{T}_1^e} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n^e} : (S_1^p, \ldots, S_n^p) \equiv^p (\bar{S}_1^p, \ldots, \bar{S}_n^p) \Rightarrow g^p(S_1^p, \ldots, S_n^p) \equiv^p \bar{g}^p(\bar{S}_1^p, \ldots, \bar{S}_n^p)$.

Die Forderung der Äquivalenz als eine der Voraussetzungen für physische Äquivalenz ist nötig, da sonst die physische Äquivalenz zweier physischer Operatorgraphen nicht die Äquivalenz implizieren würde. Dies wird dadurch deutlich, dass ein physischer Operator nicht einmal zu sich selber äquivalent sein muss, wenn er keinem logischen Operator entspricht. Dann kann er nämlich für verschiedene physische Eingabeströme, die jedoch identische logische Darstellungen haben, verschiedene Ergebnisse produzieren, die nicht zwingend äquivalent sind. So sind beispielsweise die physischen Ströme b mit $b_1 = (1, [100, 200)), b_2 = (2, [100, 200))$ und c mit $c_1 = (2, [100, 200)), c_2 = (1, [100, 200))$ äquivalent. Bildet ein physischer Operator diese Ströme auf das zuerst eingetroffene Element ab, also b auf den Ausgabestrom $\bar{b}$ mit $\bar{b}_1 = (1, [100, 200))$, c auf $\bar{c}$ mit $\bar{c}_1 = (2, [100, 200))$, und bildet er zudem alle anderen Eingaben auf den leeren Ausgabestrom ab, so erfüllt er zwar trivialerweise die Bedingung, physisch äquivalente Eingabeströme auf das physisch gleiche Ergebnis abzubilden wie er selber, bildet sie aber nicht alle auf äquivalente Ausgabeströme ab.

2.4.4. Implementierung

Der Zweck physischer Operatoren und Operatorgraphen ist es, das Ergebnis logischer Operatoren beziehungsweise logischer Operatorgraphen effizienter berechnen zu können, da sie statt auf einzelnen Zeiteinheiten mit Gültigkeitsintervallen arbeiten. Die Semantik der Operatoren wird üblicherweise auf den logischen Datenströmen definiert. Die der physischen Operatoren wird dann darauf zurückgeführt, indem gefordert wird, dass das gleiche Ergebnis entstehen muss, wenn man zuerst auf die entsprechenden logischen Ströme abbildet und dann den logischen Operator anwendet und wenn man erst den physischen Operator anwendet und dann erst die Ausgabe auf einen logischen Strom abbildet. Für Operatorgraphen wird komponentenweise analog vorgegangen.

Definition 2.12 (Implementierung)**.** Ein n-stelliger physischer Operator $op^p : \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n} \to \mathbb{S}^p_{\mathcal{T}}$ *implementiert* einen logischen Operator $op^l : \mathbb{S}^l_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}_n} \to \mathbb{S}^l_{\mathcal{T}}$, wenn $\forall S^p_1 \in \mathbb{S}^p_{\mathcal{T}_1}, \ldots, S^p_n \in \mathbb{S}^p_{\mathcal{T}_n} : \varphi^{p \mapsto l}(op^p(S^p_1, \ldots, S^p_n)) = op^l(\varphi^{p \mapsto l}(S^p_1), \ldots, \varphi^{p \mapsto l}(S^p_n))$ gilt.

Ein physischer Operatorgraph $g^p : \mathbb{S}^p_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}^e_n} \to \mathbb{S}^p_{\mathcal{T}^a_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}^a_m}$ *implementiert* einen logischen Operatorgraphen $g^l : \mathbb{S}^l_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}^e_n} \to \mathbb{S}^l_{\mathcal{T}^a_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}^a_m}$, wenn $\forall i \in \{1, \ldots, m\}\ \forall S^p_1 \in \mathbb{S}^p_{\mathcal{T}_1}, \ldots, S^p_n \in \mathbb{S}^p_{\mathcal{T}_n} : \varphi^{p \mapsto l}(\pi^p_i(g^p(S^p_1, \ldots, S^p_n))) = \pi^l_i(g^l(\varphi^{p \mapsto l}(S^p_1), \ldots, \varphi^{p \mapsto l}(S^p_n)))$ gilt.

Während bei einem logischen Operator durch die zu verarbeitenden Datenströme der resultierende logische Datenstrom eindeutig festgelegt wird, ist dies bei einem physischen Operator, der diesen implementiert, nicht zwingend der Fall, da $\varphi^{p \mapsto l}$ nicht injektiv ist. Allerdings sind alle erlaubten Ergebnisse äquivalent.

Die Äquivalenz von Operatorgraphen lässt sich nun von logischen Operatorgraphen auf ihre Implementierungen übertragen, wie das folgende Lemma zeigt:

Lemma 2.1. *Seien $g^p, \hat{g}^p : \mathbb{S}^p_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}^e_n} \to \mathbb{S}^p_{\mathcal{T}^a_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}^a_m}$ physische Operatorgraphen, $g^l, \hat{g}^l : \mathbb{S}^l_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}^e_n} \to \mathbb{S}^l_{\mathcal{T}^a_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}^a_m}$ logische Operatorgraphen, g^p implementiere g^l und $\hat{g}^p$ implementiere $\hat{g}^l$. Wenn g^l und $\hat{g}^l$ äquivalent sind, dann sind auch g^p und $\hat{g}^p$ äquivalent und implementieren beide sowohl g^l als auch $\hat{g}^l$.*

Beweis. Seien $(S^p_1, \ldots, S^p_n), (\bar{S}^p_1, \ldots, \bar{S}^p_n) \in \mathbb{S}^p_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}^e_n}$ physische Datenströme mit $(S^p_1, \ldots, S^p_n) \equiv (\bar{S}^p_1, \ldots, \bar{S}^p_n)$.

Damit gilt $(\varphi^{p \mapsto l}(S^p_1), \ldots, \varphi^{p \mapsto l}(S^p_n)) \equiv (\varphi^{p \mapsto l}(\bar{S}^p_1), \ldots, \varphi^{p \mapsto l}(\bar{S}^p_n))$ nach Definitionen 2.7 und 2.11.

Sei $i \in \{1, \ldots, m\}$. Dann gilt für den jeweils i-ten Ausgabestrom der physischen Graphen, dass sie äquivalent sind:

$$\begin{aligned}
\varphi^{p \mapsto l}(\pi^p_i(g^p(S^p_1, \ldots, S^p_n))) &= \pi^l_i(\varphi^{p \mapsto l}(g^p(S^p_1, \ldots, S^p_n))) \\
&= \pi^l_i(g^l(\varphi^{p \mapsto l}(S^p_1), \ldots, \varphi^{p \mapsto l}(S^p_n))) \\
&\equiv \pi^l_i(g^l(\varphi^{p \mapsto l}(\bar{S}^p_1), \ldots, \varphi^{p \mapsto l}(\bar{S}^p_n))) \\
&\equiv \pi^l_i(\hat{g}^l(\varphi^{p \mapsto l}(\bar{S}^p_1), \ldots, \varphi^{p \mapsto l}(\bar{S}^p_n))) \\
&= \pi^l_i(\varphi^{p \mapsto l}(\hat{g}^p(\bar{S}^p_1, \ldots, \bar{S}^p_n))) \\
&= \varphi^{p \mapsto l}(\pi^p_i(\hat{g}^p(\bar{S}^p_1, \ldots, \bar{S}^p_n)))
\end{aligned}$$

Die Implementierungseigenschaft folgt aus der Tatsache, dass g^l und $\hat{g}^l$ auf Grund der Definition der Äquivalenz logischer Datenströme für gleiche Eingaben gleiche Ausgaben produzieren. □

Mit Hilfe dieses Lemmas kann nun der Aufbau von Implementierungen logischer Operatorgraphen aus Implementierungen logischer Operatoren erfolgen.

Korollar 2.1. *Ersetzt man in einem logischen Operatorgraphen g^l alle logischen Operatoren durch physische Implementierungen, so erhält man unabhängig von der konkreten Wahl jeweils äquivalente physische Operatorgraphen, die g^l implementieren.*

Beweis. Die Behauptung folgt mit Lemma 2.1 per struktureller Induktion über den Aufbau des Operatorgraphen. □

2.4.5. Schnappschuss-Reduzierung

Die Semantik der logischen Operatoren kann nun wiederum wie anfangs motiviert auf die von Operatoren über Multimengen, also insbesondere auf Operatoren der erweiterten relationalen Algebra, zurückgeführt werden. Ähnliche Betrachtungen für temporale Datenbanken wurden bereits in [Gad86] angestellt.

Definition 2.13 (Schnappschuss-Reduzierung)**.** Für einen Typ $\mathcal{T}$ sei $\xi : \mathbb{S}^l_{\mathcal{T}} \times \Omega_{\mathcal{T}} \times \mathbb{T}$ definiert durch $\xi(S^l, e, t) = n$, falls $(e, t, n) \in S^l, \xi(S^l, e, t) = 0$, sonst.

Die Abbildung $\tau : (\mathbb{S}^l_{\mathcal{T}} \times \mathbb{T}) \to \wp(\Omega_{\mathcal{T}} \to \mathbb{N})$ mit $\tau(S^l, t) := (f : \Omega_{\mathcal{T}} \to \mathbb{N}, f(e) = \xi(S^l, e, t))$ wird als *Schnappschussabbildung* bezeichnet.

Ein logischer Operator $op^l : \mathbb{S}^l_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}_n} \to \mathbb{S}^l_{\mathcal{T}}$ heißt *Schnappschuss-reduzierbar* auf einen Operator $op^m : \wp(\Omega_{\mathcal{T}_1} \to \mathbb{N}) \times \ldots \times \wp(\Omega_{\mathcal{T}_n} \to \mathbb{N}) \to \wp(\Omega_{\mathcal{T}} \to \mathbb{N})$ über Multimengen, falls

$$\forall t \in \mathbb{T}\ \forall S^l_1 \in \mathbb{S}^l_{\mathcal{T}_1}, \ldots, S^l_n \in \mathbb{S}^l_{\mathcal{T}_n} : \tau(op^l(S^l_1, \ldots, S^l_n), t) = op^m(\tau(S^l_1, t), \ldots, \tau(S^l_n, t))$$

Ein logischer Operatorgraph $g^l : \mathbb{S}^l_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}^e_n} \to \mathbb{S}^l_{\mathcal{T}^a_1} \times \ldots \times \mathbb{S}^l_{\mathcal{T}^a_m}$ heißt *Schnappschuss-reduzierbar* auf einen Operatorgraphen $g^m : \wp(\Omega_{\mathcal{T}^e_1} \to \mathbb{N}) \times \ldots \times \wp(\Omega_{\mathcal{T}^e_n} \to \mathbb{N}) \to \wp(\Omega_{\mathcal{T}^a_1} \to \mathbb{N}) \times \ldots \times \wp(\Omega_{\mathcal{T}^a_m} \to \mathbb{N})$ über Multimengen, falls

$$\begin{gathered}\forall i \in \{1, \ldots, m\}\ \forall t \in \mathbb{T}\ \forall S^l_1 \in \mathbb{S}^l_{\mathcal{T}_1}, \ldots, S^l_n \in \mathbb{S}^l_{\mathcal{T}_n} : \\ \tau(\pi^l_i(g^l(S^l_1, \ldots, S^l_n), t)) = \pi_i(g^m(\tau(S^l_1, t), \ldots, \tau(S^l_n, t)))\end{gathered}$$

Betrachtet man für ein $t \in \mathbb{T}$ die Abbildung $\tau_t : \mathbb{S}^l_{\mathcal{T}} \to \wp(\Omega_{\mathcal{T}} \to \mathbb{N})$ mit $\tau_t(S^l) := \tau(S^l, t)$, so fordert die Bedingung, dass τ_t mit der Operatoranwendung kommutiert, wie Gleichung 2.1 zeigt. Das entspricht dem Ziel, dass für jeden Zeitpunkt t die Anwendung des Datenstromoperators die gleichen Ergebnisse für diesen Zeitpunkt liefern soll, die der entsprechende Operator auf den zu diesem Zeitpunkt in den Eingabeströmen gültigen Multimengen berechnet hätte.

$$\begin{array}{ccc} S^l_1, \ldots, S^l_n & \xrightarrow{\tau_t} & R_1, \ldots, R_n \\ {\scriptstyle op^l}\downarrow & & {\scriptstyle op^m}\downarrow \\ S^l & \xrightarrow{\tau_t} & R \end{array} \tag{2.1}$$

Gleichung 2.2 zeigt, wie sich das Konzept auf Operatorgraphen mit mehreren Ausgabeströmen beziehungsweise Ausgabemultimengen überträgt.

$$\begin{array}{ccc} S_1^l, \ldots, S_n^l & \xrightarrow{\tau_t} & R_1, \ldots, R_n \\ g^l \downarrow & & g^m \downarrow \\ \bar{S}_1^l, \ldots, \bar{S}_m^l & \xrightarrow{\tau_t} & \bar{R_1}, \ldots, \bar{R_m} \end{array} \tag{2.2}$$

Analog zu Lemma 2.1 lässt sich nun die Äquivalenz von Operatorgraphen über Multimengen auf darauf Schnappschuss-reduzierbare logische Operatorgraphen übertragen:

Lemma 2.2. *Seinen $g^l, \hat{g}^l : \mathbb{S}^l_{\mathcal{T}_1^e} \times \ldots \times \mathbb{S}^l_{\mathcal{T}_n^e} \to \mathbb{S}^l_{\mathcal{T}_1^a} \times \ldots \times \mathbb{S}^l_{\mathcal{T}_m^a}$ logische Operatorgraphen, $g^m, \bar{g}^m : \wp(\Omega_{\mathcal{T}_1^e} \to \mathbb{N}) \times \ldots \times \wp(\Omega_{\mathcal{T}_n^e} \to \mathbb{N}) \to \wp(\Omega_{\mathcal{T}_1^a} \to \mathbb{N}) \times \ldots \times \wp(\Omega_{\mathcal{T}_m^a} \to \mathbb{N})$ Operatorgraphen für Multimengen, g^l Schnappschuss-reduzierbar auf g^m und $\hat{g}^l$ Schnappschuss-reduzierbar auf $\hat{g}^m$. Wenn g^m und $\hat{g}^m$ äquivalent sind, dann sind auch g^l und $\hat{g}^l$ äquivalent und Schnappschuss-reduzierbar auf sowohl g^m als auch $\hat{g}^m$.*

Beweis. Seien $(S_1^l, \ldots, S_n^l), (\bar{S}_1^l, \ldots, \bar{S}_n^l) \in \mathbb{S}^l_{\mathcal{T}_1^e} \times \ldots \times \mathbb{S}^l_{\mathcal{T}_n^e}$ mit $(S_1^l, \ldots, S_n^l) \equiv (\bar{S}_1^l, \ldots, \bar{S}_n^l)$ logische Datenströme.

Nach Definitionen 2.7 und 2.11 gilt damit $\forall i \in \{1, \ldots, n\} : S_i^l = \bar{S}_i^l$.

Sei $i \in \{1, \ldots, m\}$ und $t \in \mathbb{T}$. Dann gilt für den jeweils i-ten Ausgabestrom der logischen Graphen, dass sie bezüglich t äquivalent sind:

$$\begin{aligned} \tau(\pi_i^l(g^l(S_1^l, \ldots, S_n^l), t)) &= \pi_i(g^m(\tau(S_1^l, t), \ldots, \tau(S_n^l, t))) \\ &= \pi_i(g^m(\tau(\bar{S}_1^l, t), \ldots, \tau(\bar{S}_n^l, t))) \\ &= \pi_i(\hat{g}^m(\tau(\bar{S}_1^l, t), \ldots, \tau(\bar{S}_n^l, t))) \\ &= \tau(\pi_i^l(\hat{g}^l(\bar{S}_1^l, \ldots, \bar{S}_n^l), t)) \end{aligned}$$

Somit ist der i-te Ausgabestrom für beliebiges $t \in \mathbb{T}$ auf $\Omega_{\mathcal{T}} \times \{t\} \times \mathbb{N}$ gleich, also auch auf $\Omega_{\mathcal{T}} \times \mathbb{T} \times \mathbb{N}^+$. Die Schnappschuss-Reduzierbarkeit folgt aus der Tatsache, dass g^m und $\hat{g}^m$ auf Grund der Definition der Äquivalenz von Operatorgraphen für Multimengen für gleiche Eingaben gleiche Ausgaben produzieren. □

Analog zu Korollar 2.1 kann nun der Aufbau logischer Operatorgraphen auf Basis der Schnappschuss-Reduzierung erfolgen.

Korollar 2.2. *Ersetzt man in einem Operatorgraphen für Multimengen g^m alle logischen Operatoren durch darauf Schnappschuss-reduzierbare logische Operatoren, so erhält man unabhängig von der konkreten Wahl jeweils äquivalente logische Operatorgraphen, die Schnappschuss-reduzierbar auf g^m sind.*

Beweis. Die Behauptung folgt mit Lemma 2.2 per struktureller Induktion über den Aufbau des Operatorgraphen. □

2.4.6. Anwendung

Die Menge der logischen Datenströme bildet zusammen mit logischen Operatoren eine logische Datenstromalgebra. Analog bildet die Menge der physischen Datenströme zusam-

men mit physischen Operatoren eine physische Operatoralgebra. Über die Schnappschuss-Reduzierung induziert die erweiterte relationale Algebra als Spezialfall der Algebra über Multimengen eine entsprechende logische relationale Datenstromalgebra, die wiederum eine physische relationale Datenstromalgebra als Implementierung induziert. Diese Mechanismen werden in Kapitel 8 für die Verbundoperation detailliert betrachtet.

Die Bedeutung dieser Übertragung der relationalen Algebra liegt darin, dass diese als Basis relationaler DBMS umfassend untersucht wurde. Die Theorie zur Optimierung der Anfrageverarbeitung von DBMS liefert dabei insbesondere umfassende Transformationsregeln zur Ableitung äquivalenter relationaler Operatorgraphen. Diese lassen sich als Spezialfall von Korollar 2.2 auf die logische Datenstromalgebra übertragen. Dadurch werden wesentliche Grundlagen für die logische Optimierung in DSMS gelegt, die in [Rie08] näher untersucht wird. Und analog dazu kann die Auswahl physischer relationaler Operatoren zur Ausführung von Anfragepläne mittels Korollar 2.1 von DBMS auf DSMS übertragen werden. Einen umfassenden Überblick über den Übergang von relationalen Operatoren bis hin zu physischen Datenstromoperatoren über Datenströmen mit Gültigkeitsintervallen liefert [Krä07].

Zudem eröffnet die Schnappschuss-Reduzierung einen äußerst eleganten Weg, eine Anfragesprache für DSMS zu definieren, die auf der sinnvollen und erprobten relationalen Algebra basiert. Die als Standard für DBMS etablierte *Structured Query Language* (SQL), deren Semantik über Abbildung in die erweiterte relationale Algebra klar definiert ist, kann mittels Schnappschuss-Reduzierung unmittelbar auf DSMS Anfragen übertragen werden, wo sie dann in logische Anfragepläne über relationalen Stromtypen übersetzt, wie vorstehend erläutert optimiert und mittels entsprechender physischer Operatoren ausgeführt werden kann. Dieses Vorgehen wird in [KS05, Krä07] für Datenströmen mit Gültigkeitsintervallen umfassend untersucht. [ABW06] definiert analog unter dem Namen *Continuous Query Language* (CQL) eine Anfragesprache für den Positiv/Negativ-Ansatz.

Das Ergebnis einer SQL-Anfrage q über Datenströmen erhält man somit, indem man die logischen Datenströme $S_1, \ldots, S_n$ für jeden Zeitpunkt auf Relationen $R_1, \ldots, R_n$ projiziert und darauf die modifizierte SQL-Anfrage q_{R_i/S_i} anwendet, bei der die Ströme durch die Relationen substituiert wurden. Das Ergebnis von q bilden nun Datenströme $\bar{S}^l_1, \ldots, \bar{S}^l_m$, die bei entsprechender Projektion die Ergebnisrelationen $\bar{R}_1, \ldots, \bar{R}_m$ der Auswertung von q_{R_i/S_i} ergeben:

$$\begin{array}{ccc} S^l_1, \ldots, S^l_n & \xrightarrow{\tau_t} & R_1, \ldots, R_n \\ {\scriptstyle q}\downarrow & & \downarrow{\scriptstyle q_{R_i/S_i}} \\ \bar{S}^l_1, \ldots, \bar{S}^l_m & \xrightarrow{\tau_t} & \bar{R}_1, \ldots, \bar{R}_m \end{array} \tag{2.3}$$

Zusätzlich zur Übertragung des klassischen SQL aus der Datenbankwelt in die Datenstromwelt werden oft auch Spracherweiterungen definiert, die den beim Übergang zur Stromverarbeitung hinzukommenden temporalen Aspekt berücksichtigen, insbesondere Techniken zur Spezifikation der Gültigkeit der Stromelemente. Dazu zählen insbesondere verschiedenste Arten von Zeitfenstern.

Teil II.

Metadaten

3. Einleitung

Unter *Metadaten* versteht man, wie der Name schon sagt, Daten über Daten. Welche Daten dabei in einem Kontext als originäre Daten, und welche als Metadaten zu verstehen sind, ist dabei gleichwohl oft eine Frage der Betrachtungsweise. Zum Beispiel versteht man den Inhalt der Dateien auf einer Festplatte üblicherweise als Daten, während Informationen über die Dateien wie Länge und Ablagesektor eher als Metadaten verstanden werden. Andererseits werden diese auch auf der Festplatte gespeichert und sind zum Beispiel für ein Defragmentierungsprogramm die entscheidenden Daten, während sich dieses für den Inhalt der Dateien nicht interessiert. Ähnlich verhält es sich mit relationalen Datenbankmanagementsystemen, bei denen die von den Benutzern in den Tabellen gespeicherten Informationen in Form der Datenbank als die eigentlichen Daten angesehen werden, während zum Beispiel Schemainformationen als Metadaten betrachtet werden. Andererseits werden aber auch diese zumeist in einer Tabelle gespeichert und stellen Eingabedaten etwa für den Optimierer dar.

Das Beispiel der Schemainformationen zeigt zudem, welche essentielle Bedeutung Metadaten für ein DBMS haben. Ohne sie wäre ein sinnvoller Zugriff auf die in den Tabellen gespeicherten Informationen kaum möglich. Andere Metadaten wie zum Beispiel statistische Informationen über die Verteilung der Werte einzelner Attribute sind zwar für ein DBMS nicht zwingend erforderlich, verbessern aber die Möglichkeiten des Optimierers und erhöhen somit die Performanz des Systems.

Da bei der Entwicklung von DSMS aufgrund der engen Verwandtschaft und guten Erfahrungen mit dieser Vorgehensweise zumeist auf bewährte Techniken aus dem Bereich der DBMS zurückgegriffen wird, drängt es sich geradezu auf, auch in DSMS Metadaten einzusetzen. Bei der Umsetzung stellt sich dabei analog zur Übertragung anderer Konzepte aus DBMS in DSMS zunächst die Frage, welche gemeinsamen und unterschiedlichen Anforderungen bestehen. Während ähnliche Anforderungen zumeist durch analoge Lösungen bedient werden können, bedingen unterschiedliche Anforderungen die Entwicklung neuer, innovativer Lösungen.

Nach diesem einführenden Kapitel erläutert Kapitel 4 die in dieser Arbeit verwendete Architektur zur Verwaltung von Metadaten. In Kapitel 5 werden die Mechanismen zur Aktualisierung der Metadaten diskutiert, bevor Kapitel 6 verschiedene Aspekte zur Implementierung des Metadaten-Rahmenwerkes beleuchtet.

In diesem Kapitel skizziert Abschnitt 3.1 zunächst Metadaten in DBMS, bevor in 3.2 die Anforderungen an Metadaten in DSMS erläutert werden. In 3.3 wird dann eine Unterscheidung in statische und dynamische Metadaten getroffen. Den Abschluss der Einleitung bildet in 3.4 eine Diskussion der Idee eines Metadatenstromsystems.

3.1. Metadaten in DBMS

Die klassischen Metadaten einer relationalen Datenbank beziehen sich auf die Schemata ihrer Tabellen. Sie beinhalten beispielsweise die Namen der Tabellen, deren Spaltenzahl und deren Größe. Für die Spalten werden üblicherweise die Tabelle, zu der sie gehören, und ihr Typ als Metadaten erfasst. Diese Aufteilung erlaubt wiederum die einfache Speicherung der Metadaten in relationalen Tabellen. Eine solche wurde bereits 1985 von Edgar Frank Codd, dem Pionier relationaler Datenbanken, in der vierten seiner Regeln für relationale Datenbanksysteme[Cod85b, Cod85a] indirekt gefordert:

> „The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data."

Die von der Regel geforderte Zugriffsmöglichkeit auf die Metadaten auf demselben logischen Level wie auf normale Daten legt natürlich nahe, die Metadaten ebenfalls in Tabellen zu organisieren.

Weitere wichtige Metadaten eines DBMS sind Informationen über deren Nutzer und Nutzergruppen, Rechte, erstellte Sichten und Indizes. Diesen und den Schemainformationen ist gemein, dass sie nur relativ seltenen Änderungen unterworfen sind, was nochmals für ihre Verwaltung in Datenbanktabellen spricht. Ähnliches gilt für übliche Metadaten eines Data Warehouses, die üblicherweise Informationen über die Quellsysteme und das Laden daraus umfassen.

Eine wichtige Klasse von Metadaten in relationalen DBMS bilden zudem diejenigen, die vom Optimierer verwendet werden. Dazu zählen, neben der bereits erwähnten Größe der Tabellen und den Schemainformationen sowie zum Beispiel Integritätsbedingungen, insbesondere auch Statistiken über die Werteverteilungen in den Tabellen und Spalten. Zum Beispiel kann die Ergebnisgröße einer Selektion über einer Tabelle anhand der Selektionsbedingung und den Werteverteilungen der Spalten der Tabelle geschätzt werden. Diese allerdings können sich mit jeder Änderungsoperation in der Datenbank ebenfalls ändern. Dabei kann eine Aktualisierung der Statistiken nach Änderungen an einem Tupel bereits das Lesen der kompletten Tabelle erfordern, beispielsweise um die Zahl verschiedener Werte innerhalb einer Spalte zu ermitteln. Da es somit zu aufwendig wäre, die Statistiken immer aktuell zu halten, und da übliche Kostenmodelle ohnehin approximativ arbeiten, begnügt man sich in der Regel damit, die Statistiken nur in gewissen Abständen oder bei geringer Systemlast zu aktualisieren.

Insbesondere ist bei einem DBMS davon auszugehen, dass zur Optimierung einer Anfrage die Metadaten zwar vor der Ausführung der Anfrage durch den Optimierer zugegriffen werden, während der zeitlich begrenzten Laufzeit der Anfrage allerdings nicht zum Zwecke der Optimierung aktualisiert werden. Allerdings können bei der Ausführung der Anfrage bemerkte Diskrepanzen zwischen Wirklichkeit und Statistiken zu deren Aktualisierung genutzt werden.[SLMK01] Zudem existieren Ansätze, die versuchen, Anfragen zur Laufzeit zu reoptimieren, falls sich ein anfangs gewählter Anfrageplan als ungünstig erweist.[KD98, MRS+04, LSM+07]

3.2. Anforderungen an Metadaten in DSMS

Ein großer Teil der Metadaten aus DBMS finden auch in DSMS in ähnlicher Weise Anwendung. So werden analog zu den Schemainformationen für Tabellen in DBMS in DSMS Schemainformationen für die Datenquellen benötigt. Auch hier beschreiben diese die Eigenschaften der zu verarbeitenden Daten, sind also klassische Metadaten. Auch die Änderungscharakteristik unterscheidet sich nicht wesentlich von der in DBMS. Neue Datenquellen werden analog zu Tabellen eher selten hinzugefügt und ändern auch ihr Schema fast nie. Auch bei den Metadaten bezüglich Nutzern, Nutzergruppen und Rechten ergeben sich keine grundlegenden Unterschiede.

Deutlich anders stellt sich die Situation bei denjenigen Metadaten dar, die bei der Optimierung von Anfragen zum Einsatz kommen. Während Anfragen in DBMS eine relativ kurze Laufzeit haben und die Optimierer wie in 3.1 erläutert daher nur vor deren Ausführung auf Metadaten zurückgreifen, führen DSMS kontinuierliche Anfragen aus, deren Laufzeit prinzipiell unbegrenzt ist. Da einige für die Optimierung von Anfragen wichtige Eigenschaften der Datenquellen, beispielsweise die Rate, mit der diese Daten liefern, sich über die Zeit ändern können, sollten DSMS auf derartige Änderungen reagieren können. Daraus ergeben sich gleich mehrere Anforderungen an die Bereitstellung von Metadaten.

Zunächst sollten die für die Optimierung benutzten Metadaten während der Laufzeit der Anfragen aktualisiert und für neue Optimierungen genutzt werden. Dies erfordert eine kontinuierliche oder zumindest regelmäßige Überwachung der optimierungsrelevanten Eigenschaften der Eingabeströme, eben auch zur Laufzeit der Anfragen.

Analog zu DBMS ergibt sich jedoch auch bei DSMS das Problem, dass eine Optimierung aufgrund von Metadaten der Eingaben keinesfalls zu optimalen Ergebnissen führen muss, weil die dazu verwendeten Verfahren grundlegende Fehlannahmen bezüglich der Vorgänge innerhalb der Anfrage machen können. Besagt zum Beispiel die Statistik einer Unternehmensdatenbank, dass 2% der Beschäftigten schwanger und 50% männlich sind, so würde ein Optimierer aufgrund mangelnden Weltwissens und üblicherweise ebenfalls fehlender Statistiken über Korrelationen annehmen müssen, eine Anfrage nach männlichen schwangeren Angestellten werde 1% der Belegschaft ergeben. Eine Korrektur an der Eingabestatistik, dass der Anteil der schwangeren Beschäftigten von 2% auf 3% gestiegen ist, würde den absoluten Fehler bei der Schätzung der Ergebnisgröße in diesem Fall sogar erhöhen. Erkennen und beheben lässt sich das Problem jedoch durch eine Beobachtung der Vorgänge innerhalb der laufenden Anfrage selbst. Diese sollten daher auch als Metadaten zur Verfügung gestellt werden.

Allerdings stellt sich die Frage, inwieweit es sich bei Informationen über die Vorgänge innerhalb der Anfragen überhaupt um Metadaten handelt. Ähnlich könnte man jedoch zum Beispiel auch bezüglich der Nutzer und Rechte argumentieren; es handelt sich ja um Daten über die Nutzer und nicht über die Daten. Andererseits beschreiben sie die Zugriffsorganisation der Daten, sind also doch Daten bezüglich der Daten. Zudem stellen die Anfragen in einem DSMS zugleich auch wieder Datenquellen dar. Informationen über die Eigenschaften der Anfragen sind somit zugleich auch wieder Daten bezüglich der Eigenschaften der von ihnen gelieferten Daten beziehungsweise bezüglich der Art der Lieferung. Aus diesem Grund werden in dieser Arbeit auch sämtliche Informationen

über die internen Abläufe der Operatoren als Metadaten betrachtet.

Zwar wurden die Metadaten der Operatoren hier aufgrund der bestehenden Analogie zu DBMS anhand der Optimierungsproblematik motiviert, doch kommen diese in DSMS noch auf weiteren, teilweise verwandten Gebieten zum Einsatz. Während ein DBMS vornehmlich die Laufzeit der Anfragen zu minimieren sucht, spielt dieses Kriterium aufgrund der kontinuierlichen Anfragen in DSMS naturgemäß keine Rolle. Stattdessen muss dieses mit den Ressourcen, insbesondere CPU-Zeit und Speicher, so haushalten, dass alle gleichzeitig laufenden Anfragen damit ausgeführt werden können.[CKSV06, CKSV08] Dazu ist es enorm wichtig, den aktuellen Ressourcenverbrauch innerhalb des Systems messen zu können. Dies stellt also eine weitere Anforderung an die Metadaten dar. Neben den Systemkomponenten finden solche Metadaten aber auch menschliche Abnehmer. Systemadministratoren oder sogar Anwender können mit ihrer Hilfe einen Eindruck vom Ressourcenverbrauch des Systems oder ihrer Anfragen gewinnen und geeignet darauf reagieren. Und Wissenschaftler können mit Hilfe dieser Metadaten prüfen, ob sich ihre Annahmen über die Ressourcenentwicklung mit der Wirklichkeit decken. Viele Messungen in dieser Arbeit wären zum Beispiel ohne die entsprechenden Metadaten nur unter erheblichem Mehraufwand möglich gewesen. Die somit sehr heterogene Gesamtheit aller, die auf Metadaten zugreifen, wird in der Folge als *Metadatennutzer* bezeichnet.

3.3. Statische und dynamische Metadaten

Prinzipiell lassen sich die Metadaten innerhalb eines DSMS in zwei Klassen einteilen. Diejenigen, die sich nach ihrer einmaligen Festlegung nicht mehr ändern, und diejenigen, die im Laufe der Zeit Änderungen unterworfen sein können.

3.3.1. Statische Metadaten

Die unveränderlichen Metadaten, die zumeist schon bei der Erstellung einer Komponente, insbesondere eines Knotens des Operatorgraphen, feststehen, werden in dieser Arbeit als *statische Metadaten* bezeichnet. Darunter fallen bei den Knoten des Operatorgraphen zum Beispiel die Schemainformationen und die Größe der Ein- und Ausgabeobjekte. Statische Metadaten werden üblicherweise bereits bei der Erzeugung der Komponenten bestimmt.

3.3.2. Dynamische Metadaten

Die verbleibenden Metadaten, die sich im Laufe der Zeit ändern können, werden als *dynamische Metadaten* bezeichnet. Dabei spielt es keine Rolle, wie häufig sich die Metadaten wirklich ändern oder ob dies regelmäßig geschieht. Typische Beispiele für dynamische Metadaten sind die Eingaberate und die Selektivität des Filteroperators. Dennoch kann die Eingaberate eines konkreten Filteroperators natürlich über seine gesamte Laufzeit konstant bleiben.

3.4. Metadatenstromsysteme

Da Metadaten in relationalen DBMS üblicherweise mit den Mitteln des DBMS selber verwaltet und zugegriffen werden, also wie Daten behandelt werden, stellt sich natürlich die Frage, ob die dynamischen Metadaten eines DSMS nicht auch mit den Mitteln des DSMS verarbeitet werden sollten. Die Antwort darauf sollte vom konkreten Fall abhängig gemacht werden.

Zur Verarbeitung und Nutzung weniger, einfacher Metadaten wäre diese Vorgehensweise eher übertrieben. Denn auch die zur Verarbeitung der Metadaten benutzten zusätzlichen Quellen, Operatoren und Senken würden Ressourcen verbrauchen und sogar ihrerseits wieder Metadaten zur Verfügung stellen. Diese könnten natürlich wieder als Datenströme verwaltet werden; unendlich fortsetzen lässt sich dieser Prozess aber nicht. Stattdessen müssen dann die Metadaten entweder ignoriert oder mit einfacheren Mechanismen als dem DSMS selber verarbeitet werden. In vielen Fällen reichen diese Mechanismen aber schon für diejenigen Metadaten aus, die von den Knoten des Operatorgraphen verursacht werden, welche die originären Daten verarbeiten.

Für komplexere Auswertungen über den Metadaten, insbesondere in der Forschung über DSMS, bietet die Verarbeitung der Metadaten mit Hilfe eines DSMS, also die Benutzung eines *Metadatenstromsystemes*, interessante Möglichkeiten. Daher sollte ein Metadatenrahmenwerk immer die nötigen Schnittstellen bereitstellen, um ein solches aufbauen zu können. Diese Arbeit widmet sich daher auch dieser Fragestellung. Essentiell dafür ist aber zunächst die Diskussion der Mechanismen, die benötigt werden, um die Metadaten eines DSMS zu erfassen und kontinuierlich zu analysieren. Darauf aufbauend können dann die verschiedenen Arten der Weiterverarbeitung diskutiert werden, darunter die der Bereitstellung von Metadaten als Datenquellen. Die Entscheidung für einen oder mehrere der Zugriffswege, ob ein Metadatennutzer also Spezialanwendungen oder ein DSMS zur Verarbeitung der Metadaten einsetzt, ist analog zur entsprechenden Entscheidung für die Verarbeitung von Daten zu treffen.

4. Architektur

Dieses Kapitel behandelt die grundsätzliche Architektur anhand der Fragen, wie die Metadaten in einem DSMS gehalten (4.1) und zugegriffen (4.2) werden können.

4.1. Speicherung

Während eine wesentliche Eigenschaft der DBMS die Persistenz der Daten ist und schon aufgrund dieser Tatsache die Speicherung der Daten und Metadaten primär auf dem Externspeicher erfolgt, sind DSMS aus Performanzgründen primär als Hauptspeichersysteme ausgelegt. Ein teilweiser Verlust von Daten und (Zwischen-)ergebnissen wird dabei zunächst in Kauf genommen. Gehen also Instanzen von Operatoren bei einem Hauptspeicherverlust verloren, so kann sicherlich auch ein Verlust der sie beschreibenden Metadaten hingenommen werden. Eine persistente Speicherung aller Metadaten ist somit nicht notwendig und sollte aus Performanzgründen vermieden werden.

4.1.1. Komponenten mit Metadaten

Bei der Wahl eines geeigneten Speicherverfahrens für die Metadaten im Hauptspeicher ist nun zunächst die Frage interessant, wo innerhalb eines DSMS überhaupt Metadaten auftreten beziehungsweise benötigt werden. Die Schemainformationen werden zunächst bezüglich der Datenquellen benötigt, sind jedoch aufgrund der Tatsache, dass auch Operatoren wieder Datenquellen darstellen, auch bei diesen sinnvoll. Für Anfragen der Benutzer wiederum werden die Antworten in Form von Datensenken bereitgestellt, wobei die Zuordnung zum anfragenden Benutzer ein sinnvolles Metadatum ist. Diese einfachen Beispiele zeigen schon, dass jeder Knoten des Operatorgraphen eines DSMS sinnvoller Träger von Metadaten ist. Ein weiteres Beispiel ist der Speicherverbrauch, den das System bezüglich aller Knoten des Operatorgraphen ermitteln können soll. Dies gilt jedoch prinzipiell auch für alle anderen Komponenten des Systems, da diese ja auch Speicher benötigen. Will man den Speicherbedarf der anderen Mechanismen mit denselben Mechanismen überwachen wie den der Knoten des Operatorgraphen, so werden damit alle Komponenten des DSMS zu Trägern von Metadaten.

4.1.2. Verwaltung der Metadaten

Da die Aufgabe der Verwaltung der Metadaten in einem DSMS somit eine Querschnittsaufgabe ist, bieten sich dafür im Wesentlichen zwei Lösungen an. Zum einen könnte eine zentrale Komponente die Verwaltung aller Metadaten übernehmen, zum anderen kann man die Metadaten direkt in den jeweiligen Komponenten verwalten, die sie bereitstellen.

Für PIPES wurde die zweite Lösung gewählt, die Metadaten werden also von den jeweiligen Komponenten selber verwaltet. Diese Entscheidung hat mehrere Gründe. Zunächst setzt die Entwicklung von PIPES die von XXL fort, welches die Metadaten der passiven Operatoren ebenfalls in den Operatoren selber verwaltet. Bei der objektorientierten Entwicklung von XXL und PIPES spielt dabei auch das Geheimnisprinzip eine Rolle. Die Nutzer der Metadaten brauchen keine Kontrolle darüber, wie diese bestimmt werden. Insbesondere benötigt die Berechnung bestimmter Metadaten aber tiefgreifenden Zugang zu den Interna zum Beispiel der Operatorimplementierungen. Daher muss die Bereitstellung der Metadaten ohnehin bei der Implementierung der Komponenten mit berücksichtigt werden. Obwohl die Generierung der Metadaten somit innerhalb der Komponenten erfolgen muss, könnte man sie jedoch trotzdem zentral speichern. Da Benutzer der Metadaten einer Komponente aber sowieso Zugriff zu der Komponente haben, können die diese auch direkt bei den Komponenten anfordern.

4.2. Zugriff

Um den Nutzern der Metadaten trotz der dezentralen Verwaltung einen einheitlichen Zugriff auf die Metadaten zu gewähren, implementieren die Komponenten von XXL (und somit auch PIPES) einheitliche Schnittstellen. Über die Schnittstelle `MetaDataManageable` können Nutzer von den Komponenten eine Instanz von `CompositeMetaData` erhalten, aus der dann die einzelnen Metadaten der Komponente bezogen werden können. Der Zugriff erfolgt dabei analog zu einer `Map` in Java über Objekte als Schlüssel, wobei `CompositeMetaData` intern eine `ConcurrentHashMap` benutzt, um konkurrierende nebenläufige Zugriffe zu ermöglichen.

4.2.1. Problematik der Zusatzaufwandes

Bei aller Notwendigkeit von Metadaten und trotz der zahlreichen Vorteile, die ein möglichst vielfältiges Angebot verschiedener Metadaten bietet, muss auch berücksichtigt werden, dass die Bereitstellung bestimmter Metadaten selber mit der Nutzung von Systemressourcen einhergeht. So erfordert die Bestimmung der Selektivität eines Filteroperators, dass mitgezählt wird, wie viele Elemente der Operator verarbeitet und für wie viele davon das Filterprädikat erfüllt ist. Auch wenn das Aktualisieren einzelner Zähler keinen großen Aufwand darstellt, summiert sich dieser über verschiedenste Zähler und Komponenten doch auf. Allgemein sollte es daher vermieden werden, aktuell nicht benötigte Metadaten fortlaufend bereitzustellen. Beispielsweise sind bei der Erforschung des Verhaltens verschiedener Operatorimplementierungen Metadaten bezüglich des internen Aufrufverhaltens von Interesse, zum Beispiel die Aufrufhäufigkeit der Hashfunktion bei hash–basierten Operatoren. Im Produktivbetrieb eines DSMS spielt dieses Interesse jedoch zunächst keine Rolle mehr. Dennoch sollte die Möglichkeit, entsprechende Metadaten abzurufen, nicht entfernt werden. Zum einen ist der Programmcode zur Bereitstellung der Metadaten oft eng mit dem Programmcode der Komponenten verwoben und somit nicht einfach zu entfernen. Zum anderen können sie bei Problemen mit dem DSMS für Administratoren oder Entwickler ein wichtiges Hilfsmittel bei der

Analyse sein. Da Performanz aber ein Schlüsselaspekt von DSMS ist, sollte der durch die Metadaten verursachte Overhead grundsätzlich so gering wie sinnvoll möglich gehalten werden.

4.2.2. Bereitstellung

Um also den Overhead niedrig zu halten, bietet es sich an, nur diejenigen Metadaten zu berechnen, die auch wirklich benutzt werden. Leider lassen sich aber nicht alle Metadaten erst im Falle eines Zugriffes bedarfsgesteuert berechnen. Möchte ein Metadatennutzer beispielsweise die Selektivität des Filteroperators abfragen, so ist eine umgehende Antwort nur dann möglich, wenn bereits zuvor die Zahl der eintreffenden und die der das Filterprädikat erfüllenden Elemente protokolliert wurden. Eine bedarfgesteuerte Berechnung der Metadaten kommt daher nur für einen Teil der Metadaten in Betracht (siehe auch 5.2.1).

Um also nicht alle Metadaten immer vorhalten zu müssen und trotzdem bereits beim ersten Zugriff sinnvolle Daten liefern zu können, bietet es sich als Lösung an, den Metadatennutzern aufzuerlegen, diejenigen Metadaten zu abonnieren, die sie in der Folge zugreifen möchten. Diese Lösung wurde in PIPES für die dynamischen Metadaten gewählt.

Anforderung

Vor dem ersten Zugriff auf ein dynamisches Metadatum einer Komponente muss ein Metadatennutzer bei deren `MetaDataManagement` mit dem Befehl `include` kundtun, dass das Metadatum bereitgestellt werden soll. Daraufhin wird dann ein sogenannter `MetaDataHandler` erzeugt und im `CompositeMetaData` der Komponente hinterlegt. Zudem werden in der Komponente die notwendigen Vorkehrungen getroffen, um dem `MetaDataHandler` zu ermöglichen, das Metadatum korrekt zu berechnen. Über ihn kann dieses fortan zugegriffen werden.

Fordert ein weiterer Nutzer ein schon von einem anderen Nutzer angefordertes Metadatum nochmals an, so wird vermieden, dass ein weiterer `MetaDataHandler` für dasselbe Metadatum erstellt wird. Zum einen würde das nämlich unnötigen Overhead erzeugen. Zum anderen sollen aber auch beide Nutzer eine konsistente Sicht auf die Metadaten haben, was am einfachsten zu gewährleisten ist, wenn beide auf denselben `MetaDataHandler` zugreifen. Auf diese Weise wird zugleich das Synchronisationsproblem gelöst, dass dadurch entsteht, dass verschiedene Nutzer aus verschiedenen Threads heraus auf das Metadatum zugreifen könnten. Dieses Problem besteht im Falle statischer Metadaten nicht, da diese sich ja nicht ändern.

Freigabe

Benötigt ein Metadatennutzer ein von ihm angefordertes Metadatum nicht mehr länger, so muss er es mit Hilfe des Befehls `exclude` wieder freigeben. Der zugehörige `MetaDataHandler` darf aber erst dann entfernt werden, wenn alle Anforderungen wieder freigegeben wurden. Dazu hält PIPES jeweils einen Zähler der aktiven Anforderungen.

Erst wenn dieser null erreicht, wird der `MetaDataHandler` entfernt. Zudem werden die für ihn notwendigen Veränderungen an der Komponente rückgängig gemacht. Dadurch entfällt der durch die Bereitstellung des Metadatums verursachte Overhead also fortan wieder. `Exclude` verhält sich also dual zu `include`.

Für die Metadatennutzer ist es wichtig zu beachten, dass sie die benötigten Metadaten auf jeden Fall anfordern, bevor sie sie benutzen, auch wenn diese bereits von anderen Nutzern angefordert wurden. Anderenfalls könnten die Zugriffe ins Leere gehen, sobald die anderen Nutzer das Metadatum freigegeben haben.

4.2.3. Abhängigkeiten

Natürliche Abhängigkeiten

Angesichts der Vielfalt an möglichen Metadaten kommt es häufig vor, dass Metadaten auf Basis anderer Metadaten berechnet werden können. Zum Beispiel kann der Quotient aus Ein- und Ausgaberate eines Operators, ein beim Scheduling gerne verwendetes Kriterium [BBDM03], natürlich leicht aus der Ein- und der Ausgaberate des Operators berechnet werden. Werden diese ohnehin bereits als Metadaten benutzt, so brauchen sie nur gelesen und der Quotient daraus gebildet zu werden. Aber auch wenn sie nicht ohnehin in Benutzung sind, ist diese Art der Berechnung zu empfehlen. Denn anderenfalls müssten in der Operatorimplementierung verschiedene Wege zur Berechnung ein und desselben Metadatums implementiert werden. Der damit verbundene Aufwand würde sich in den meisten Fällen nicht nur durch redundante Berechnungen nachteilig auf die Performanz auswirken, sondern auch die Implementierung erschweren, da auch Wartbarkeit und Qualität des Quelltextes leiden würden.

Künstliche Abhängigkeiten

Zudem treten auch Fälle auf, in denen sich Aufgaben zur Berechnung verschiedener Metadaten zwar überschneiden, wobei das gemeinsam genutzte Zwischenergebnis aber selber nicht unmittelbar als Metadatum von Interesse ist. Trotzdem kann es als Metadatum definiert werden, wobei als Nutzer dann nur der Operator selber auftritt, und zwar zur Berechnung der darauf aufbauenden Metadaten.

Knotenübergreifende Abhängigkeiten

Knoten im Operatorgraphen können ihre Metadaten nicht in allen Fällen aus ihrem internen Status und ihren eigenen Metadaten berechnen. Soll beispielsweise ein Mapping-Operator seine Ausgaberate schätzen, so ist diese trivial zu berechnen, sie entspricht nämlich der geschätzten Ausgaberate des stromaufwärts im Operatorgraphen gelegenen Operators, also dem gleichen Metadatum, nur eines anderen Operators. Es besteht also eine Abhängigkeit zwischen Metadaten verschiedener Operatoren. Derartige Inter-Operator-Abhängigkeiten können nicht nur bezüglich der Quellen, sondern auch der Senken eines Knotens im Operatorgraphen bestehen. So hängen zum Beispiel die QoS-Anforderungen an einen Operator von denen an seine Senken ab.

PIPES erlaubt daher die Definition von Abhängigkeiten eines Metadatums sowohl von explizit angegebenen, als auch von allen seinen Quellen oder Senken, was insbesondere bei Operatoren mit variabler Zahl von Quellen, wie zum Beispiel der Vereinigung, hilfreich ist. Die Definition von Abhängigkeiten bezüglich nicht benachbarter Knoten im Operatorgraphen wird nicht unterstützt. So ist gewährleistet, dass die Entwickler neuer Operatoren auch die durch diese vorgenommenen Operationen auf den Daten in den Metadaten reflektieren können.

Dynamische Abhängigkeiten

Die bisher diskutierten Abhängigkeiten sind statische Eigenschaften der jeweiligen Knoten des Operatorgraphen, also für alle Instanzen der jeweiligen Operatoren gleich und zum Zeitpunkt der Implementierung bekannt. Allerdings gibt es auch Konstellationen, bei denen sich eine dynamische Bestimmung von Abhängigkeiten zur Laufzeit empfiehlt. Kann zum Beispiel ein Metadatum A sowohl auf Grundlage eines Metadatums B als auch eines anderen Metadatums C berechnet werden, wobei die Berechnung von B und C jeweils hohen Aufwand verursacht, so sollte es vermieden werden, A statisch von entweder B oder C abhängig zu machen. Vielmehr sollte diese Entscheidung bei einer Anforderung von A zur Laufzeit davon abhängig getroffen werden, ob B oder C schon vorher angefordert wurden, also ohnehin zur Verfügung stehen.

Der Mechanismus der *dynamischen Abhängigkeiten* ermöglicht es zudem, zur Laufzeit neue Typen von dynamischen Metadaten zu generieren und für diese dennoch ihre Abhängigkeiten zu registrieren. Damit werden auch für diese Metadaten die automatischen Mechanismen benutzt, die auf den Abhängigkeiten basieren.

Abhängigkeitsgraph

Die Gesamtheit aus Abhängigkeiten zwischen Metadaten innerhalb eines und zwischen Knoten des Operatorgraphen kann man sich als gerichteten Graphen mit den Metadaten als Knoten und den Abhängigkeiten als Kanten vorstellen. Dieser muss kreisfrei, aber nicht zusammenhängend sein. Er wird in dieser Arbeit als Abhängigkeitsgraph bezeichnet.

4.2.4. Automatische Anforderung

Möchte ein Metadatennutzer auf ein Metadatum zugreifen, so muss es dieses wie in 4.2.2 erläutert zunächst einmal anfordern. Basiert die Berechnung des Metadatums jedoch auf anderen Metadaten, so müssen diese natürlich auch zur Verfügung stehen. Ist dies noch nicht der Fall, so müssen sie ebenfalls angefordert werden. Diese Aufgabe sollte aber nicht dem Nutzer der Metadaten auferlegt werden, denn dies würde die Nutzung unangemessen verkomplizieren. Insbesondere benötigte der Benutzer Kenntnisse über die Abhängigkeiten der Metadaten untereinander, also über Details der Implementierung, die er auf Grund des Geheimnisprinzips gar nicht kennen sollte. Daher soll der Benutzer nur das Metadatum anfordern müssen, dass er selber benötigt. Der Prozess der Anforderung zugrundeliegender Metadaten muss dann automatisch erfolgen.

Grundlage für das automatische Einbinden sind die im vorangegangenen Abschnitt 4.2.3 erläuterten Abhängigkeiten. Werden diese bei der Implementierung der Knoten des Operatorgraphen für dessen Metadaten definiert, so kann bei der Anforderung eines Metadatums sichergestellt werden, dass auch alle benötigten zugrundeliegenden Metadaten zur Verfügung stehen. Zu diesem Zweck werden bei der Erstellung eines `MetaDataHandler` zu einem Metadatum alle Metadaten angefordert, von denen dieses abhängt. Diese Anforderungen werden erst dann freigegeben, wenn das anfordernde Metadatum selber komplett freigegeben wurde, sein Anforderungszähler also null erreicht. Da sich die Anforderungen rekursiv durch den Abhängigkeitsgraphen fortsetzen, leistet diese einfache Strategie genau das Gewünschte. Bei der ersten Anforderung eines Metadatums werden alle dafür direkt oder transitiv benötigten Metadaten angefordert. Die Rekursion bricht aber überall da ab, wo die benötigten Metadaten bereits zur Verfügung stehen. Insbesondere wird also bei weiteren Anforderungen eines Metadatums nicht nochmals der Abhängigkeitsgraph rekursiv durchlaufen.

Um bei der Implementierung eines Operators auf Metadaten anderer Knoten sinnvoll zugreifen zu können, müssen diese natürlich über einheitliche Schlüssel zugreifbar sein. Diese sind daher trotz der dezentralen Struktur des Metadaten-Rahmenwerks in PIPES in zentralen Schnittstellen definiert.

Trotzdem ist es natürlich möglich, dass Operatoren einige Metadaten nicht unterstützen. Daher können Anforderungen an Metadaten aus benachbarten Knoten des Operatorgraphen gegebenenfalls ins Leere laufen. Um diesem Problem zu begegnen ist der Anforderungsprozess in PIPES so ausgelegt, dass er zurückmeldet, ob die Anforderung erfolgreich war. Anderenfalls erfährt der Nutzer daher, dass er das Metadatum nicht nutzen darf. Da er das Metadatum in diesem Fall auch nicht wieder freigeben kann und darf, ist der rekursive Anforderungsprozess so ausgelegt, dass im Falle einer fehlgeschlagenen Anforderung an einer beliebigen Stelle alle bis dahin erfolgreich getätigten Anforderungen wieder freigegeben werden. Erfolglose Anforderungen haben somit keine bleibenden Auswirkungen auf die Metadaten.

5. Aktualisierung

Nachdem im vorangegangenen Kapitel die Frage behandelt wurde, wie Metadaten angefordert und dann zugegriffen werden können, widmet sich dieses Kapitel der Frage, wie die bei dynamischen Metadaten, die sich ja dadurch auszeichnen, dass sie sich mit der Zeit ändern können, notwendige Aktualisierung erfolgen kann. Zunächst werden in 5.1 die Anforderungen an den Aktualisierungsprozess dargelegt und in 5.2.1 die naheliegende Lösung der bedarfsgesteuerten Aktualisierung beim Zugriff erläutert. Diese kann jedoch in vielen Fällen zu Problemen führen, die in den folgenden Unterabschnitten diskutiert und für die alternative Aktualisierungskonzepte als Lösungen vorgestellt werden. In 5.3 wird dann deren Zusammenspiel diskutiert.

5.1. Anforderungen

In 4.2.1 wurde schon diskutiert, dass Metadaten nur dann ein effektives Mittel zur Optimierung des Ressourcenverbrauchs eines DSMS sein können, wenn die Bereitstellung der Metadaten nicht selber zu viele Ressourcen verbraucht. Durch die Verpflichtung zur Anforderung benötigter Metadaten wird daher der durch aktuell nicht benötigte Metadaten verursachte Overhead minimiert. Aber auch der Overhead, der durch die bei dynamischen Metadaten notwendige Aktualisierung fortlaufend besteht, sollte möglichst gering gehalten werden.

Beim Abwägen des Aufwands für die Aktualisierung der Metadaten sollte natürlich auch bedacht werden, dass diese nur ein Nebenprodukt des DSMS sind und hauptsächlich intern verwendet werden. Ähnlich wie die Nutzer des DSMS QoS-Anforderungen an die Qualität der Ergebnisse stellen können, sollte also auch betrachtet werden, welche Qualität der Metadaten benötigt wird, damit diese ihre Aufgaben erfüllen können, und welche Kompromisse dabei gemacht werden können. Der Grad der Prioritätensetzung zwischen Qualität der Metadaten und Einsparung von Ressourcen bei deren Produktion sollte dabei soweit möglich ein Parameter des Systems sein.

Bei einer weiteren Anforderung an die Aktualisierung der Metadaten sollten jedoch keine Kompromisse gemacht werden: Die von einem Metadaten-Benutzer abgerufenen Metadaten sollten nicht dadurch beeinflusst werden, ob und wann weitere Benutzer dieselben Metadaten nutzen. Dieses *Isolationskriterium* stellt sicher, dass die Metadaten allein vom DSMS determiniert werden, und nicht von den Metadatennutzern. Anderenfalls könnte beispielsweise der Optimierer, der Metadaten benutzt, bei seinen Entscheidungen dadurch beeinflusst werden, dass ein Administrator zu Analysezwecken auf Metadaten zugreift. Die dann gegebenenfalls anderen Optimierungsentscheidungen würden den Administrator einen Zustand des Systems beobachten lassen, der ohne diese Beobachtung gar nicht eingetreten wäre.

5.2. Aktualisierungsmechanismen

Wie in 4.2.2 beschrieben erfolgt der Zugriff auf die dynamischen Metadaten in PIPES über so genannte `MetaDataHandler`. Für die in den folgenden Abschnitten vorgestellten Arten der Aktualisierung gibt es daher jeweils eine entsprechende `MetaDataHandler`-Implementierung.

5.2.1. Bedarfsgesteuerte Aktualisierung

Die nahe liegende Lösung zur Minimierung der zur Bestimmung der Metadaten nötigen Berechnungen ist es natürlich, diese nur bei Bedarf, also dann, wenn sie zugegriffen werden, zu berechnen. Bei dieser *bedarfsgesteuerten Aktualisierung* berechnet also der entsprechende *bedarfsgesteuerte* `MetaDataHandler` den aktuellen Wert des Metadatums erst dann, wenn lesend auf ihn zugegriffen wird. Dies ist in vielen Fällen auch effektiv möglich. Zum Beispiel kann der aktuelle Speicherverbrauch des Aggregationsoperators einfach berechnet werden, indem die aktuelle Größe der internen Datenstruktur des Operators, welche diese ohnehin stets angeben kann, mit der Größe der Objekte darin, also einem statischen Metadatum, multipliziert wird.

Wie schon in 4.2.2 erläutert lässt es sich oft nicht vermeiden, schon vor dem ersten und auch zwischen den Zugriffen auf die Metadaten bereits Informationen vorzuhalten, um die Metadaten beim Zugriff berechnen zu können. Manchmal reicht es dann aber bereits aus, diese Informationen als Metadatum zu liefern, was dann problemlos mit einem bedarfsgesteuerten `MetaDataHandler` geschehen kann.

5.2.2. Periodische Aktualisierung

Um die Eingaberate eines Operators als Metadatum zur Verfügung stellen zu können, zählt der Operator die eintreffenden Elemente mit. Um daraus eine Rate berechnen zu können, muss dieser Zähler nun in Beziehung zu einer Zeitdauer gesetzt werden. Bei Verwendung eines bedarfsgesteuerten `MetaDataHandler`s müsste also der Zähler gelesen und zu einer Zeit in Beziehung gesetzt werden. Lässt man den Zähler immer weiter laufen, so würde er sich zeitlich auf die Zeitspanne seit dem Start des Zählers beziehen. Dieser Startzeitpunkt wäre wiederum der Zeitpunkt, zu dem das Metadatum angefordert wurde. Dies hätte jedoch gleich zwei wesentliche Nachteile. Zum einen würde sich die so berechnete Rate immer auf die gesamte Zeit seit der Anforderung beziehen. Von dem Metadatum Eingaberate erwartet man aber eigentlich, dass es sich auf einen aktuellen Zeitraum bezieht. Zum anderen wäre das Isolationskriterium verletzt, da sich der Wert bei mehreren Anforderungen immer auf die Zeit seit der ersten Anforderung bezöge, und nicht auf die Zeit seit der Anforderung des Metadatums Eingaberate durch den jeweils aktuellen Nutzer.

Als Lösung könnte man nun den Zähler bei jedem Zugriff auf das Metadatum wieder auf null zurücksetzen und als Bezugszeitraum jeweils die Zeit seit dem letzten Zugriff wählen. Wäre das Metadatum also vor zwei Sekunden zum letzten Mal abgefragt worden und wären seither zehn Elemente eingetroffen, so würde man fünf pro Sekunde als aktuelle Eingaberate liefern. Diese Vorgehensweise würde bei einem einzelnen Nutzer des

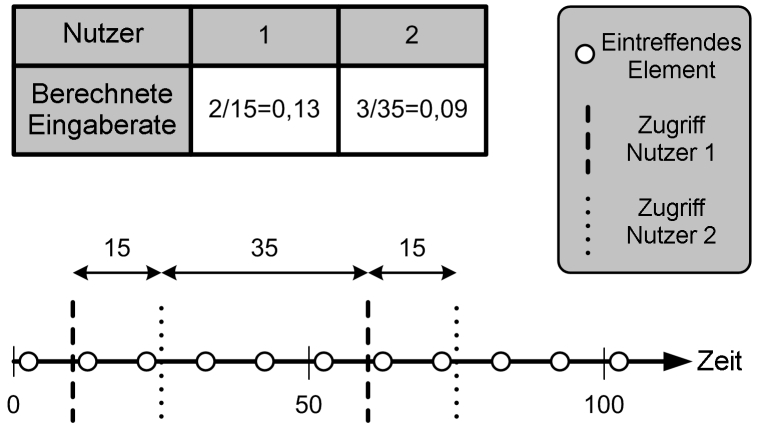


Abbildung 5.1.: Probleme bei konkurrierender Metadatenaktualisierung zum Zugriffszeitpunkt

Metadatums sogar Sinn machen. Leider verletzt sie aber ebenfalls das Isolationskriterium, wie das Beispiel[CKS07] aus Abbildung 5.1 zeigt. Dort wird ein Operator, bei dem Elemente genau alle 10 Zeiteinheiten eintreffen, also mit einer Rate von 0,1 pro Zeiteinheit, von zwei Metadatennutzern beobachtet. Diese greifen beide exakt alle 50 Zeiteinheiten auf das Metadatum Eingaberate zu, der erste immer 15 Zeiteinheiten, nachdem es der zweite getan hat. Auf diese Art berechnet der erste immer auf Basis von zwei Elementen in 15 Zeiteinheiten eine Rate von 0,13, während der zweite eine Rate von 0,09 berechnet, da in seinem Fall seit dem letzten Zurücksetzen des Zählers jeweils 35 Zeiteinheiten vergangen und drei Elemente eingetroffen sind. Obwohl die Elemente also exakt periodisch eintreffen und die Abfragen ebenfalls periodisch erfolgen, erhalten beide Nutzer verschiedene und zudem falsche Werte des Metadatums.

Diese Beispiele zeigen, dass die Aktualisierung von Metadaten wie der Eingaberate immer wieder aktuell, aber unabhängig vom Zugriffsverhalten der Metadatennutzer geschehen sollte. Damit scheidet die Benutzung des bedarfsgesteuerten `MetaDataHandlers` aber aus. Mit dem *periodischen* `MetaDataHandler` wird stattdessen ein `MetaDataHandler` zur *periodischen Aktualisierung* eingeführt, der in festen Zeitabständen den Wert seines Metadatums neu berechnet. Dieser wird dann zwischengespeichert und bis zur nächsten Aktualisierung des `MetaDataHandlers` bei allen Zugriffen als Wert des Metadatums geliefert.

Diese Vorgehensweise erfüllt nun die verschiedenen Anforderungen. Alle Zugriffe liefern den jeweils zuletzt bei der periodischen Aktualisierung vorberechneten Wert. Dieser ist unabhängig von den anderen Zugriffen, das Isolationskriterium ist also erfüllt. Dieser Wert bezieht sich zudem immer auf Zeitintervalle gleicher Länge, wodurch seine Semantik klar definiert ist. Zudem skaliert der Ansatz sehr gut, da weitere Nutzer des Metadatums keine zusätzlichen Berechnungen bewirken, sondern nur den vorberechneten Wert auslesen. Wird pro Vorberechnung der Wert von mindestens einem Metadatennutzer ausgelesen, so fallen zudem auch keine unnötigen Berechnungen an. Als nachteilig könnte man jedoch ansehen, dass der zurückgelieferte Wert sich im ungünstigsten Fall auf einen Zeitraum bezieht, der zum Zeitpunkt des Auslesens schon um seine Länge vergangen

ist. Dies kann man aber angesichts der Vorteile und der Tatsache, dass ein geeigneter Kompromiss zwischen Kosten der Metadaten und deren Qualität getroffen werden muss, in Kauf nehmen.

Die geeignete Wahl des Aktualisierungsintervalls für ein Metadatum mit periodischer Aktualisierung hängt neben der Art des Metadatums auch davon ab, wie viel Systemlast für das Berechnen von Metadaten verwendet werden soll. Höhere Aktualisierungsraten erlauben zwar zumeist ein genaueres Erfassen der Metadaten, erzeugen aber auch höhere Kosten. In PIPES wurde für die meisten Metadaten ein Aktualisierungsintervall von einer halben Sekunde gewählt. Zusätzlich wird sichergestellt, dass kein Aktualisierungsintervall kleiner als 100 Millisekunden verwendet wird.

5.2.3. Ausgelöste Aktualisierung

Basiert die Berechnung eines periodisch zu aktualisierenden `MetaDataHandlers` auf dem Wert eines anderen ebenfalls periodischen `MetaDataHandlers`, so können sich dabei verschiedene Probleme ergeben. Dies gilt insbesondere, falls sich die Zeitabstände zwischen den Aktualisierungen unterscheiden.

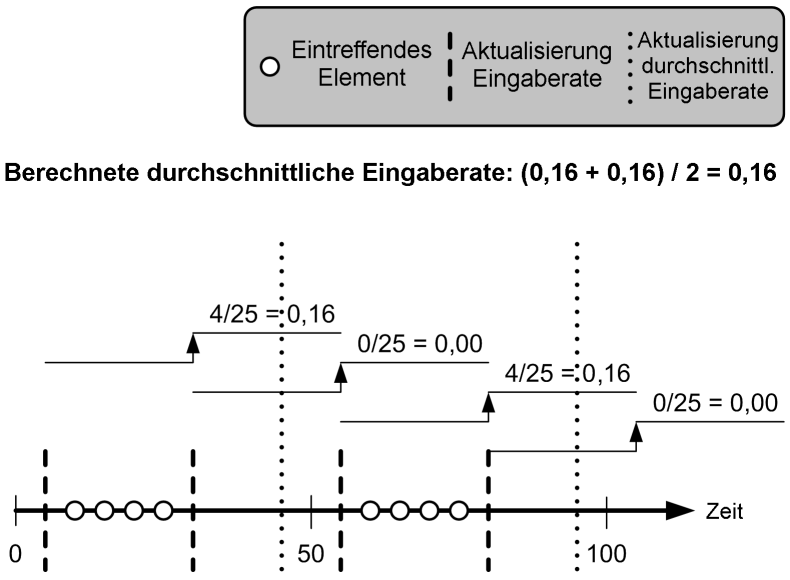


Abbildung 5.2.: Problem bei seltenerer Aktualisierung mit asynchronem Aktualisierungsintervall

Abbildung 5.2 zeigt als Beispiel[CKS07] die Aktualisierung der durchschnittlichen Eingaberate eines Operators. Dabei treffen die Elemente in Schüben zu je vier ein, auf die eine Pause folgt. Das Metadatum Eingaberate wird alle 25 Zeiteinheiten aktualisiert und erfasst diese Schübe im Beispiel genau. Das darauf basierend berechnete Metadatum durchschnittliche Eingaberate wird im Beispiel nur alle 50 Zeiteinheiten aktualisiert und erfasst jeweils nur die Schübe an Eingabedaten, während es die Werte des Metadatums Eingaberate, welche den Pausen entsprechen, nicht erfasst. Somit wird als durchschnittliche Eingaberate 0,16 Elemente pro Zeiteinheit erfasst, obwohl diese in Wirklichkeit nur

0,08 ist, da nur acht Elemente pro 100 Zeiteinheiten eintreffen.

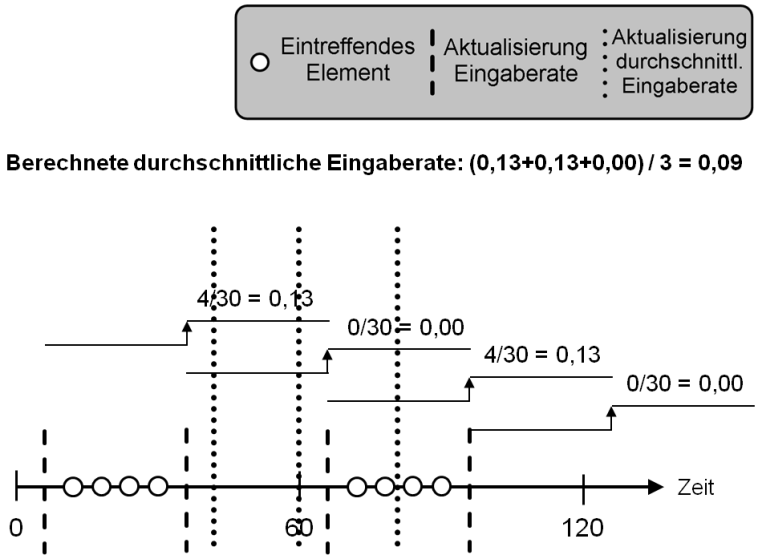


Abbildung 5.3.: Problem bei häufigerer Aktualisierung mit asynchronem Aktualisierungsintervall

Abbildung 5.3 zeigt ein Beispiel für den Fall, in dem das abhängige Metadatum durchschnittliche Eingaberate öfter periodisch aktualisiert wird als das Metadatum Eingaberate, auf dem es basiert. Analog zum vorangegangenen Beispiel treffen die Elemente wieder in Schüben ein, die vom Metadatum Eingaberate, dass periodisch nach jeweils 30 Zeiteinheiten aktualisiert wird, erfasst werden. Das Metadatum durchschnittliche Eingaberate hingegen wird periodisch nach jeweils 20 Zeiteinheiten neu berechnet und erfasst dabei zweimal die Schübe, aber nur einmal die Pause bei der Eingabe. Somit findet eine falsche Gewichtung der gemessenen Eingaberaten bei der Berechnung der durchschnittlichen Eingaberate statt, die korrekterweise 4/60=0,07 sein müsste.

Diese Beispiele zeigen, dass die Aktualisierungsperiode des Metadatums durchschnittliche Eingaberate an die des Metadatums Eingaberate sinnvoll gekoppelt sein sollte. Bei gleicher, aber phasenverschobener Periode, wenn also beide mit der gleichen Periode, aber zu verschiedenen Zeitpunkten aktualisiert werden, würde das Metadatum durchschnittliche Eingaberate unnötig verspätet aktualisiert. Seine Aktualisierung sollte also eine direkte Folge der periodischen Aktualisierung der Eingaberate sein.

Für solche bedingten Aktualisierungen gibt es daher den Mechanismus der *ausgelösten Aktualisierung* und entsprechend einen *ausgelösten* `MetaDataHandler`. Dieser hält, wie der periodische `MetaDataHandler`, stets einen vorberechneten Wert des von ihm zur Verfügung zu stellenden Metadatums vor, den er bei Zugriffen auf dieses liefert. Allerdings erfolgt dessen Aktualisierung nicht periodisch, sondern muss ausgelöst werden.

Aufgrund des Ziels, den durch die Metadaten verursachten Overhead so gering wie möglich zu halten, sollte dieses Auslösen natürlich nur dann geschehen, wenn es nötig ist. Da der ausgelöste `MetaDataHandler` immer den vorberechneten Wert zurückliefert,

muss diese Vorberechnung immer dann erfolgen, wenn sich die Berechnungsgrundlagen des Metadatums geändert haben. Diese Berechnungsgrundlagen können Teile des Status des Knotens des Operatorgraphen oder andere Metadaten sein. Für beide Fälle gibt es entsprechende Mechanismen in PIPES. Bei Änderungen am Status eines Knoten, die einen ausgelösten `MetaDataHandler` betreffen, muss im Code explizit eine Auslösung aufgerufen werden.

Auslösen abhängiger Metadaten

Der andere Fall hingegen kann automatisch behandelt werden. Denn wenn ein Metadatum als Berechnungsgrundlage ein anderes benötigt, so ist dies ja bereits für den Mechanismus der Anforderung als Abhängigkeit (siehe 4.2.3) erfasst. Ändert sich also der Wert eines Metadatums, so können automatisch alle davon abhängigen ausgelösten `MetaDataHandler` ausgelöst werden, die dann eine Neuberechnung ihres Metadatums durchführen.

Dieses automatische Auslösen kann sich dann rekursiv fortsetzen, wenn von dem ausgelösten Metadatum ein weiteres Metadatum abhängt, für das ein ausgelöster `MetaDataHandler` benutzt wird. Die Auslösevorgänge setzen sich dabei im Abhängigkeitsgraphen (siehe 4.2.3) fort, und zwar entgegengesetzt der Kanten, welche die Abhängigkeiten darstellen. Kehrt man im Abhängigkeitsgraphen alle Kanten um und entfernt dann diejenigen, die nicht in einem Metadatum mit ausgelöster Aktualisierung enden, so erhält man den *Auslösegraphen*. Da dieser mehrere Wege von einem Knoten zu einem anderen enthalten kann, ist es möglich, dass innerhalb eines rekursiven Aktualisierungsvorganges ein Metadatum mehrmals besucht wird. Diesem Problem kann zur Verhinderung unnötiger Berechnungen durch Bestimmung und topologische Sortierung [Las61] der zu aktualisierenden Knoten und eine entsprechende Auslösereihenfolge gelöst werden.

Aufgrund der in 4.2.3 erläuterten knotenübergreifenden Abhängigkeiten setzen sich auch die Auslösevorgänge knotenübergreifend fort. Da Änderungen an Metadaten sowohl Quellen, als auch Senken betreffen können, kann ein geändertes Metadatum theoretisch Änderungen an Metadaten in allen Knoten des Operatorgraphen auslösen. In der Praxis sind aber zumeist allenfalls die Knoten stromaufwärts oder -abwärts des auslösenden Knotens betroffen.

5.3. Anwendung der Aktualisierungsmechanismen

Für ein Metadatum steht nun nicht direkt fest, welches der richtige Aktualisierungsmechanismus dafür ist. Diese Entscheidung muss vielmehr bei der konkreten Implementierung getroffen werden und hängt unter anderem auch davon ab, welche anderen Metadaten implementiert werden und welches Zugriffsverhalten auf die Metadaten erwartet wird. Die Wahl des Mechanismus muss dabei natürlich immer die in 3.2 formulierten Anforderungen an die Metadaten im Auge behalten. Die folgenden Betrachtungen können daher keine exakten Regeln, sondern nur Leitlinien dafür sein, welcher Aktualisierungsmechanismus bei einer konkreten Implementierung von Metadaten in einem neuen Operator

für welche Metadaten zu verwenden ist.

Die bedarfsgesteuerte Aktualisierung verursacht als einzige der drei vorgestellten Aktualisierungsarten pro Zugriff neuen Berechnungsaufwand. Sie sollte daher vorwiegend bei Metadaten Verwendung finden, deren Berechnung wenig Aufwand erfordert und bei denen eine Vorberechnung daher nicht nötig ist. Dies trifft unter anderem in folgenden Fällen zu:

- Bei Metadaten, die unmittelbar oder sehr einfach aus dem Status des Knotens des Operatorgraphen, zu dem sie gehören, berechnet werden können. Ein Beispiel dafür ist die aktuelle Zahl der Senken eines Operators.
- Bei Metadaten, die unter Verwendung anderer Metadaten direkt oder sehr leicht berechenbar sind, zum Beispiel durch unveränderte Weitergabe eines Metadatums einer Quelle oder Senke.
- Bei Metadaten, die sich aus einer Kombination der vorstehenden Grundlagen berechnen lassen. Dies trifft zum Beispiel in vielen Fällen auf den Speicherverbrauch eines Operators zu, der sich oft aus der aktuellen Größe der Statusstrukturen und der Objektgröße der darin gespeicherten Elemente, einem anderen Metadatum, berechnen lässt.

Allerdings macht die bedarfsgesteuerte Aktualisierung auch bei Metadaten Sinn, deren Berechnung teuer ist, falls das Metadatum zwar oft langfristig angefordert ist, aber selten zugegriffen wird. In diesen Fällen würde eine Vorberechnung des Metadatums unnötigen Aufwand verursachen. Stattdessen wird das Metadatum erst bei Bedarf ermittelt, woher auch die Benennung dieses Aktualisierungsmechanismus rührt.

Umgekehrt bieten sich die anderen Aktualisierungsmechanismen, die beim Zugriff jeweils einen vorberechneten Wert zurückliefern, dann an, wenn das Metadatum häufiger zugegriffen wird, als sich sein Wert ändert, da dann Berechnungsaufwand eingespart wird. Die ausgelöste Aktualisierung stellt insbesondere sicher, dass nur dann Berechnungen für ein Metadatum angestellt werden, wenn dessen Berechnungsgrundlage sich geändert hat. Da die Aktualisierungen im Operatorgraphen durch die ausgelöste Aktualisierung weitergereicht werden, kann diese Berechnungsgrundlage dabei auch aus anderen Metadaten in beliebigen Knoten des Operatorgraphen bestehen.

Als Einsatzgebiet für den periodischen Aktualisierungsmechanismus könnte man zunächst alle Metadaten sehen, die in einem Zusammenhang mit über die Zeit gesammelten Werten stehen. Da das Isolationskriterium eine Abhängigkeit von der Zugriffszeit verbietet, muss die Aktualisierung des Metadatums durch das Rahmenwerk für die Metadaten selbst geschehen. In 5.2.2 wurde dies am Beispiel der Eingaberate eines Operators erläutert. Durch die potentiell unbegrenzte Laufzeit von Anfragen in einem DSMS spielen derartige Raten als Metadaten eine wichtige Rolle, da die Erfassung absoluter Zahlen, wie zum Beispiel der Zahl der bisher von einem Operator verarbeiteten Elemente, keine geeignete Aussagekraft über den aktuellen Zustand des Systems hat. Im Zusammenspiel der Aktualisierungsmechanismen ergibt sich nun aber, dass nicht nur die Metadaten periodisch aktualisiert werden, die selber durch einen periodischen `MetaDataHandler` verwaltet werden, sondern indirekt auch alle Metadaten mit ausgelöster Aktualisierung,

die von einem solchen abhängen, da diese jedes Mal automatisch mit ausgelöst werden (siehe 5.2.3). Wie das Beispiel der durchschnittlichen Eingaberate am Anfang von Abschnitt 5.2.3 zeigt, ist dieses Vorgehen auch zu empfehlen, da es sich als ungünstig erweist, wenn im Falle einer Abhängigkeit zwischen zwei Metadaten beide unabhängig durch einen periodischen `MetaDataHandler` aktualisiert werden.

Voneinander abhängige periodische Aktualisierung benötigende Metadaten, die im Abhängigkeitsgraphen eine Zusammenhangskomponente bilden, sollten daher so implementiert werden, dass nur ein periodischer `MetaDataHandler` zum Einsatz kommt. Damit dieser dann aber alle Knoten der Zusammenhangskomponente mit aktualisiert, muss der von ihm entgegen der gerichteten Kanten erreichbare Teilgraph diese Zusammenhangskomponente komplett abdecken. Da in einem zusammenhängenden gerichteten Graphen nicht immer ein Knoten existiert, von dem aus alle anderen Knoten entlang der gerichteten Kanten erreichbar sind, ist dies nicht immer möglich. Allerdings kann dieses Problem stets durch Hinzufügen eines weiteren Knotens mit Kanten zu einer (möglichst minimalen) Menge von Knoten, von denen alle anderen Knoten erreichbar sind, behoben werden. Wählt man für diesen Knoten dann den periodischen und für die anderen den ausgelösten Aktualisierungsmechanismus, so bildet die periodisch zu aktualisierende Zusammenhangskomponente des Abhängigkeitsgraphen auch eine Zusammenhangskomponente im Auslösegraphen, in der alle Knoten von dem mit periodischer Aktualisierung aus erreichbar sind.

Der bei dieser Konstruktion gegebenenfalls benötigte zusätzliche Knoten entspricht dabei einem zusätzlichen Metadatum, das einen periodischen `MetaDataHandler` verwendet und dessen Sinn alleine darin besteht, die Aktualisierung anderer Metadaten periodisch auszulösen. Ein solches Metadatum wird in PIPES in jedem Knoten des Operatorgraphen automatisch zur Verfügung gestellt. Es verursacht keinen unnötigen Overhead, da es durch die automatische Anforderung erst dann aktiviert wird, wenn ein davon abhängiges Metadatum angefordert wird.

5.3.1. Beispiel

Folgendes in Abbildung 5.4 veranschaulichte Beispiel zeigt das operatorübergreifende Zusammenspiel verschiedener Aktualisierungsmechanismen. Es demonstriert, wie die Abschätzung von Aufwänden anhand des in 10.1 vorgestellten Kostenmodells mit Hilfe dynamischer Metadaten zu Laufzeit erfolgen kann.

Ein wesentlicher Faktor für die Kosten eines Verbundes über Zeitfenstern sind die Längen dieser Fenster. Diese können bei Bedarf zur Laufzeit verändert werden (siehe dazu auch 10.1.5 und 14.2). Solche Reoptimierungen sind Aufgabe des Systemoptimierers, der dazu seinerseits auf Metadaten zurückgreift und diese gegebenenfalls zeitweise anfordert. Da viele Reoptimierungen sich jedoch erst mit Verzögerung auszahlen und zuvor sogar Zusatzkosten verursachen können, finden sie nicht mit hoher Frequenz statt. Daher sollten die zugrundeliegenden Metadaten nur im Falle von Änderungen aktualisiert werden.

Dazu wird die *Fenstergröße* als bedarfsgesteuertes Metadatum abgebildet. Bei Modifikationen wird operatorintern das ausgelöste Metadatum *Erwartete Gültigkeit* aktualisiert, welches ein wesentlicher Parameter des Kostenmodells ist. Der darüber liegende Ver-

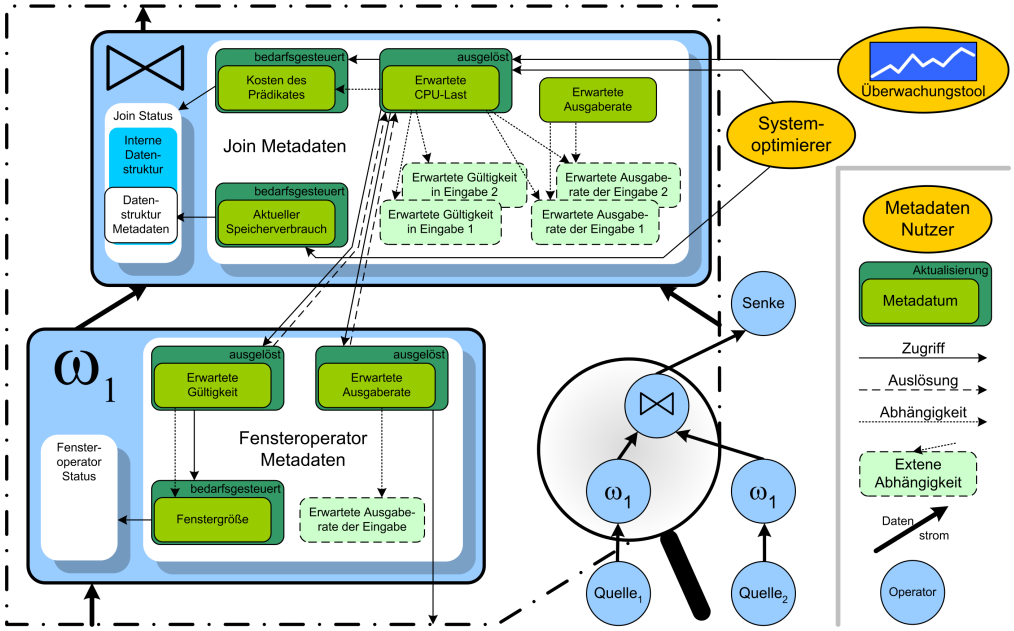


Abbildung 5.4.: Beispiel für Abhängigkeit und Aktualisierung von Metadaten

bundoperator verwendet dieses Metadatum als operatorübergreifende Abhängigkeit für sein ausgelöstes Metadatum *Erwartete CPU-Last*, das somit ebenfalls aktualisiert wird. Der Optimierer kann nun auf diese aktualisierte Kostenschätzung zugreifen, ohne sie selber vornehmen zu müssen. Und auch anderen Metadatennutzer wie Überwachungstools steht nun die aktualisierte Schätzung zur Verfügung.

Anhand des Metadatums *Erwartete Ausgaberate* des Fensteroperators kann man sehen, dass operatorübergreifend ausgelöste Aktualisierungen ihrerseits erneut operatorübergreifend wirken können. Auf diese Art können sich Aktualisierungen durch den Operatorgraphen fortpflanzen.

Abbildung 5.4 zeigt auch, wie Metadaten der Datenstrukturen des Verbundes in die Berechnung der Metadaten des gesamten Operators eingehen. So hängt der Speicherverbrauch des Verbundoperators wesentlich von dem seiner Datenstrukturen ab und wird daher unter deren Einbeziehung bedarfsgesteuert aktualisiert.

6. Implementierungsaspekte

Nach den generellen Überlegungen zu Metadaten in DSMS wird in diesem Kapitel auf Aspekte der Implementierung des Rahmenwerkes für die Metadaten in PIPES näher eingegangen. Dazu werden zunächst in 6.1 kurz die Anforderungen an die Implementierung diskutiert. Dann wird zunächst in 6.2 die Benutzung vorhandener und dann in 6.3 die Implementierung neuer Metadaten erläutert.

6.1. Anforderungen

Schon DBMS nutzen ihre Metadaten nicht nur intern, sondern stellen Sie auch über Schnittstellen den Nutzern des Systems zur Verfügung. Relationale Datenbankmanagementsysteme erlauben dazu zumeist unter anderem bestimmten Nutzern Zugriff auf die Tabellen, in denen sie ihre Metadaten verwalten. Zusätzlich können Metadaten wie Schemainformationen aber auch über andere Schnittstellen, wie zum Beispiel ODBC oder JDBC zugegriffen werden. Durch die Analogien von DSMS zu DBMS wird daher auch von einem DSMS eine derartige Bereitstellung erwartet. Die Metadaten sollen von außen also einfach und intuitiv zugreifbar sein. Da das DSMS selbst dennoch der umfangreichste Nutzer der eigenen Metadaten bleibt, kommt ein derartiger Zugriff natürlich auch dessen Entwicklung zugute.

PIPES als Bibliothek zur Erforschung und Entwicklung von DSMS muss jedoch mehr leisten als ein reines System.[KS04] Zwar sollte auch bei der Implementierung eines Systems die interne Implementierung hohen Standards genügen. Allerdings ist der Kreis der Programmierer, die an dieser Implementierung arbeiten, bei einem System enger begrenzt. Eine Bibliothek hingegen kann von einem weiteren Kreis von Programmierern weiterentwickelt werden. Gerade im immer noch vergleichsweise jungen Gebiet der Datenstromverarbeitung besteht dazu zudem weitreichendes Potenzial. Die Implementierung des Metadatenrahmenwerks von PIPES wurde daher so gestaltet, dass insbesondere auch die einfache Erweiterbarkeit gewährleistet ist. Dies gilt sowohl für die Erweiterung bestehender Komponenten um zusätzliche Metadaten, als auch um die Bereitstellung von Metadaten in neu geschaffenen Komponenten.

6.2. Benutzung

In diesem Abschnitt wird zunächst erläutert, wie sich die Benutzung der Metadaten für den Programmierer darstellt.

6.2.1. Zugriff auf Metadaten

Der Zugriff auf Metadaten gestaltet sich sehr einfach. Dabei ist nur die in 4.2.2 erläuterte Bedingung zu beachten, dass ein dynamisches Metadatum vor dem ersten Zugriff angefordert und nach dem letzten wieder freigegeben werden muss.

```
final Object metaDatum =
  TemporalFilterMetaDataManagement.SELECTIVITY_MEASUREMENT;

filter.getMetaDataManagement().include(metaDatum);
MetaDataHandler<Double> handler =
  (MetaDataHandler<Double>) filter.getMetaData().get(metaDatum);

for (int i=0; i<10; i++) {
  Thread.sleep(1000);
  System.out.println(handler.getMetaData());
}

filter.getMetaDataManagement().exclude(metaDatum);
```

Abbildung 6.1.: Einfaches Beispiel für Zugriff auf ein Metadatum

Das Beispiel in Abbildung 6.1 zeigt, wie das Metadatum gemessene Selektivität eines Filteroperators (Variable `filter`) auf die Konsole ausgegeben werden kann. Dazu wird zur Erhöhung der Lesbarkeit zunächst die passende Schlüsselkonstante für das Metadatum in einer Konstanten `metaDatum` gespeichert. Mit diesem Schlüssel wird dann das Metadatum beim `MetaDataManagement` des Filteroperators angefordert (`include`). Direkt danach kann der `MetaDataHandler` für das Metadatum über den Schlüssel vom Operator bezogen werden, da dessen Existenz durch die vorangegangene Anforderung sichergestellt ist. Über den `handler` kann nun das Metadatum beliebig zugegriffen werden; im Beispiel geschieht dies in einer Schleife zehnmal im Abstand von jeweils einer Sekunde. Abschließend wird das Metadatum wieder freigegeben (`exclude`).

6.2.2. Aggregierte Metadaten

In der Praxis kommt es häufig vor, dass dynamische Metadaten des DSMS nicht direkt verarbeitet, sondern vorher durch Aggregation weiter verdichtet werden sollen. Zu diesem Zweck könnten die Metadaten als Datenstrom zugegriffen und dann mit Hilfe des Aggregationsoperators aggregiert werden. Dieses Vorgehen ist allerdings eine der Anwendungen eines Metadatenstromsystems (siehe 3.4), die eher übertriebenen Aufwand verursacht. Damit Metadatennutzer die Aggregation dennoch nicht selber übernehmen müssen, stellt PIPES einfache Mechanismen bereit, um Metadaten in aggregierter Form anfordern und zugreifen zu können.

Die aggregierten Metadaten werden dabei wiederum als Metadaten zur Verfügung gestellt. Dabei besteht logisch eine natürliche Abhängigkeit der Aggregate von dem

jeweilig zu Grunde liegenden Metadaten. Dabei gibt es eine Vielzahl verschiedener Aggregate, die wiederum auf verschiedenste Metadaten anwendbar sind. Zum Beispiel sind die typischen mathematischen Aggregate wie Maximum oder Mittelwert auf numerische Metadaten anwendbar. Ein wichtiger Aspekt ist hierbei, dass die Erweiterung auf neue Metadaten und weitere Aggregate einfach möglich sein sollte. Daher gilt es zu vermeiden, dass die Abhängigkeiten jedes Aggregates von den Metadaten, auf die es anwendbar ist, jeweils explizit definiert werden müssen. Analog gilt für die zum Zugriff auf die Metadaten benötigten Schlüssel, dass nicht für die Vielzahl an Kombinationen von Aggregaten und Metadaten jeweils explizite Schlüssel definiert werden sollen.

Aus diesen Gründen stellt PIPES einen speziellen Typ von Metadatenschlüsseln für aggregierte Metadaten zur Verfügung. Unter Angabe des Schlüssels des zugrunde liegendem Metadatums und der Aggregatfunktion kann ein Schlüssel dieses Typs erzeugt werden. Die Schlüssel der Aggregate werden damit bei Bedarf dynamisch erzeugt und müssen nicht explizit vordefiniert sein. Der Anforderungsmechanismus von PIPES erkennt nun Schlüssel dieses Aggregationstyps und behandelt sie geeignet. Dazu wird wie in 4.2.3 erläutert dynamisch die Abhängigkeit des Aggregates vom zugrunde liegenden Metadatum registriert. Für das Aggregat wird ein ausgelöster `MetaDataHandler` erzeugt. Dieser wird durch die Abhängigkeit jeweils ausgelöst, wenn sich der Wert des Metadatums ändert. Er wendet dann die Aggregatsfunktion auf das Metadatum an und liefert den neuen Aggregatswert als Metadatum. Dieses kann vom Benutzer wie gewohnt über den schon bei der Anforderung benutzten Schlüssel, in diesem Fall also den erzeugten Aggregatschlüssel, zugegriffen werden. Auch die Freigabe erfolgt wie gewohnt über diesen Schlüssel, wobei dann die dynamisch erzeugte Abhängigkeit wieder hinfällig wird.

Greifen mehrere Metadatennutzer auf diese Weise auf dasselbe aggregierte Metadatum zu und würde dabei der Mechanismus zur Redundanzvermeidung aktiv, so erhielten sie dieselben Aggregatwerte. Da die Aggregation aber mit der ersten Anforderung des Aggregats beginnt, würde dies eine Verletzung des Isolationskriteriums darstellen, da weitere Nutzer Aggregate erhalten, die Ereignisse vor der weiteren Anforderung betreffen. Andererseits ist die Berechnung mehrerer Versionen desselben Aggregates ab verschiedenen Startzeitpunkten oftmals unnötiger Aufwand, da für die jeweilige Aufgabe auch ein gemeinsames Aggregat genügen würde. Daher ermöglicht PIPES beide Varianten, indem es dem Metadatennutzer erlaubt, bei der Erstellung des Schlüssels des Aggregattyps anzugeben, ob er ein eigenes Aggregat benötigt oder bereit ist, ein gegebenenfalls bereits bestehendes zu nutzen.

6.2.3. Metadatum als Datenquelle

Um wie in 3.4 motiviert Metadaten wieder als Datenströme verarbeiten zu können, bietet PIPES die Möglichkeit, dynamische Metadaten als Datenquelle zu nutzen. Um eine solche *Metadatenquelle* zu erhalten, benötigt ein Nutzer lediglich, wie vom normalen Zugriff auf die Metadaten gewohnt, den zugehörigen Schlüssel, und muss zusätzlich nur den Knoten oder das `MetaDataManagement` angeben, von dem das Metadatum bezogen werden kann. Abbildung 6.2 zeigt ein einfaches Beispiel, bei dem analog zum Beispiel für den normalen Zugriff auf ein Metadatum die gemessene Selektivität eines Filteroperators (`filter`)

benutzt wird.

```
Source<TemporalObject<Object>> selectivityMetaDataSource =
  new MetaDataSource(
    TemporalFilterMetaDataManagement.SELECTIVITY_MEASUREMENT,
    filter
  );
```

Abbildung 6.2.: Einfaches Beispiel für ein Metadatum als Datenquelle

Die Metadatenquelle tritt bezüglich der Metadaten wie ein Metadatennutzer auf. Beim Erstellen der Metadatenquelle wird das Metadatum daher angefordert und beim Schließen wieder freigegeben, da es fortan nicht mehr benötigt wird.

Bezüglich der Zeitpunkte, zu denen die Metadatenquelle das Metadatum liefert, gibt es verschiedene Möglichkeiten. Möchte der Nutzer der Quelle, dass diese regelmäßig den jeweils aktuellen Wert das Metadatum liefert, so kann er einen Zeittakt vorgeben, in dem dann jeweils der aktuelle Wert des Metadatums abgefragt und aktiv von der Quelle geliefert wird. Diese liefert also das Metadatum mit konstanter Datenrate.

Allerdings gibt es auch eine Vielzahl an dynamischen Metadaten, die sich selten oder unregelmäßig ändern. In diesen Fällen verursacht das regelmäßige Senden des Metadatums zumeist nur unnötige Last im Metadatensystem. Wird ein Metadatum von einem ausgelösten `MetaDataHandler` verwaltet, so bietet es sich daher als Alternative an, die Metadatenquelle das Metadatum nur dann senden zu lassen, wenn der Handler ausgelöst und das Metadatum dadurch aktualisiert wurde. Dies gilt in ähnlicher Weise auch für diejenigen ausgelösten `MetaDataHandler`, die ihrerseits direkt oder transitiv von einem periodischen `MetaDataHandler` abhängen und somit periodisch ausgelöst werden. In diesem Fall wird gewährleistet, dass das geänderte Metadatum direkt nach der Aktualisierung von der Metadatenquelle geliefert wird. Analog können auch Metadaten periodischer `MetaDataHandler` direkt nach der periodischen Aktualisierung von der Metadatenquelle gesendet werden.

Zur Umsetzung dieser Steuerung der Metadatenquelle durch die `MetaDataHandler` bietet sich der ohnehin vorhandene Mechanismus der Abhängigkeiten und Aktualisierungen an. Eine solche *datengetriebene Metadatenquelle* registriert dazu dynamisch eine spezielle Variante eines ausgelösten `MetaDataHandler`s als abhängig von dem Metadatum, das sie liefern soll. Dieser `MetaDataHandler` wird damit automatisch bei Änderungen ausgelöst und bewirkt dann das Senden des aktuellen Wertes durch die Metadatenquelle. Da sich Metadaten mit bedarfsgesteuerter Aktualisierung (5.2.1) niemals ohne aktives Auslesen ändern, eignet sich dieses Vorgehen für sie nicht. Die datengetriebene Version der Metadatenquelle kann daher nicht auf bedarfsgesteuerte `MetaDataHandler` angewendet werden. Die periodische Version der Metadatenquelle ist hingegen auf alle Typen von `MetaDataHandler`n anwendbar, zum Beispiel um periodische Metadaten zwar periodisch, aber mit erheblich verminderter Rate ins Metadatensystem einzuspeisen.

Zeitstempel

Eine wichtige Rolle bei der Funktion und Semantik eines DSMS spielt die Vergabe der Zeitstempel beziehungsweise Gültigkeitsintervalle an den Quellen. Kann sie nicht aufgrund bereits in den Daten vorhandener temporaler Information erfolgen, so wird zumeist der physische Zeitpunkt des Eintreffens als Zeitstempel vergeben. Beispielsweise würden Sensormesswerte einer Fabrik derart bereits von den Sensoren, den Rechnern, die Sensoren auslesen, oder auch erst vom DSMS beim Eintreffen der Sensorwerte über ein Netzwerk mit Zeitstempeln versehen. In weiteren DSMS werden die einmal vergebenen Zeitstempel beziehungsweise die daraus generierten Zeitintervalle dann als logische Intervalle behandelt. So wird gewährleistet, dass die Semantik des DSMS unabhängig von den internen Verarbeitungsmechanismen ist.

Dementsprechend muss auch für ein Metadatenstromsystem entschieden werden, welche Zeitstempel beziehungsweise Gültigkeitsintervalle an den Metadatenquellen vergeben werden. Dabei wäre es natürlich ein interessanter Ansatz, diese bezüglich der logischen Zeitstempel des DSMS zu vergeben, aus dem die Metadaten stammen. Dies würde sich jedoch sehr schwierig gestalten. Zum einen liefern auch Komponenten des DSMS Metadaten, die selber gar keine Daten verarbeiten, mit deren logischen Zeitstempeln man ihre Metadaten in Bezug setzen könnte. Zum anderen liefern auch die Operatoren, die generell Daten verarbeiten, Metadaten auch dann, wenn sie aktuell keine Daten zu verarbeiten haben. Auch in diesem Falle wäre unklar, wie der Bezug zur logischen Systemzeit hergestellt werden sollte.

Ein einfach zu verwirklichender Ansatz ist es hingegen, als Zeitstempel in den Metadatenquellen die Systemzeit zu verwenden. Die Quelle stempelt dazu einfach jedes Element entsprechend, direkt nachdem sie das Metadatum gelesen hat. Dabei geht allerdings der Bezug zu den logischen Zeitstempeln der Daten verloren. Stellt das Metadatensystem beispielsweise einen hohen Speicherverbrauch zu einem logischen Zeitpunkt des Metadatensystems fest, so fehlt der Bezug zum logischen Zeitpunkt des DSMS, zu dem diese Last verursacht wurde, und damit gegebenenfalls zu dem Ereignis in der vom DSMS behandelten Anwendung, das diese Last verursacht hat. Dennoch ist die Verwendung der Systemzeit semantisch sinnvoll und konsequent. Genau wie das DSMS reale Zeiten des von ihm beobachteten Systems, zum Beispiel einer Fabrik, als logische Zeitpunkte verwendet, kann das Metadatensystem reale Systemzeiten des von ihm beobachteten Systems, nämlich des DSMS, verwenden.

6.3. Implementierung neuer Metadaten

Dieser Abschnitt widmet sich verschiedenen Aspekten der Implementierung des Metadatenrahmenwerks in PIPES. Dazu werden die dazu verwendeten Mechanismen anhand der Frage diskutiert, wie die Implementierung von neuen Metadaten in bestehenden und neuen Komponenten von statten geht und wie diese durch das Rahmenwerk unterstützt wird. Daher wird zunächst kurz skizziert, welche Schritte nötig sind, um ein neues Metadatum zur Verfügung stellen zu können, bevor detaillierter auf verschiedene Aspekte eingegangen wird.

6.3.1. Prinzipielles Vorgehen

In der Softwareentwicklung sollten vor der Implementierung immer eine sorgfältige Analyse des Problems und ein geeigneter Entwurf einer Lösung stehen.

Analyse und Entwurf

Bezüglich eines neuen Metadatums empfiehlt es sich bei der Analyse zunächst auf folgende Fragen besonderen Wert zu legen:

1. Welche Informationen werden benötigt, um das Metadatum berechnen zu können?
2. Welche dieser Informationen lassen sich aus bereits zur Verfügung stehenden Metadaten gewinnen?
3. Welche der verbleibenden Informationen können aus dem Status der Komponente gewonnen werden?
4. Welche Informationen müssen zusätzlich gesammelt werden?

Beim Entwurf sind insbesondere folgende Fragestellungen von Bedeutung:

- Wie können die Informationen zu 3. effizient gewonnen werden?
- Welche Änderungen und Ergänzungen sind nötig, um die Informationen zu 4. zu gewinnen?
- Welcher der in 5.2 erläuterten Aktualisierungsmechanismen eignet sich dabei am besten für das Metadatum?

Zudem sollte noch die Frage gestellt werden, ob zur Berechnung des Metadatums ein Teil der Informationen hilfreich wäre, die für ein anderes Metadatum gesammelt werden, das selber aber nicht genutzt werden kann, da daraus diese Teilinformation nicht mehr rekonstruiert werden kann. In diesem Fall sollte die Teilinformation als künstliches Metadatum abgespalten und wie in 4.2.3 beschrieben eine künstliche Abhängigkeit des bestehenden Metadatums davon eingeführt werden. Danach kann das neue Metadatum ebenso davon abhängig gemacht werden, um Implementierungsaufwand zu sparen und zudem Overhead zu vermeiden, wenn gleichzeitig das bestehende und das neue Metadatum genutzt werden.

Implementierung

Bei der Implementierung des Metadatums sind dann im Wesentlichen folgende Schritte durchzuführen:

- Hinzufügen von Datenstrukturen und Code zum Sammeln der in Punkt 4. der Analyse identifizierten Informationen.
- Registrierung in 1. gefundenen Abhängigkeiten.

- Implementierung einer Funktion zur Berechnung des Wertes des Metadatum auf Basis aller Informationen.
- Implementierung der bei erster Anforderung und letzter Freigabe des Metadatums nötigen Handlungen, darunter insbesondere die Aktivierung und Deaktivierung des Sammelns zusätzlicher Informationen und die Bereitstellung des `MetaDataHandler`s zum gewählten Aktualisierungsmechanismus.

Diese Schritte werden in den folgenden Abschnitten jeweils im Kontext der Diskussion derjenigen Aspekte des Metadatenrahmenwerks diskutiert, auf die sie sich beziehen.

6.3.2. MetaDataManagement

Da das Bereitstellen von Metadaten eine Querschnittsaufgabe der Komponenten in PIPES ist, erfolgt es in den jeweiligen Komponenten. Um dennoch eine so weit wie mögliche Trennung des dafür nötigen Codes von dem für die Funktionalität der Klasse zu ermöglichen, werden die Metadaten einer Komponente jeweils von einer Implementierung der Schnittstelle `MetaDataManagement` verwaltet. Soll eine Komponente Metadaten zur Verfügung stellen, so muss sie daher lediglich eine Instanz einer Implementierung von `MetaDataManagement` liefern, die für sie zuständig ist.

Diese Implementierungen von `MetaDataManagement` werden zumeist in Form von inneren Klassen der Klassen vorgenommen, deren Metadaten sie verwalten. Somit ist der Code zwar klar von dem für die Funktionalität der Komponenten abgegrenzt, hat aber dennoch vollen Zugriff auf dessen Felder und Methoden.

Vererbung

Viele Komponenten in PIPES entstehen durch Erweiterung anderer, bereits bestehender Komponenten, also durch Vererbung. Dies gilt insbesondere für Knoten des Operatorgraphen, da PIPES für Quellen, Operatoren und Senken umfangreiche abstrakte Vorimplementierungen bereitstellt, auf denen praktisch alle Implementierungen aufbauen. Diese Vorimplementierungen stellen dabei auch bereits viele wichtige Metadaten bereit, zum Beispiel Ein- oder Ausgaberaten. Mit den inneren `MetaDataManagement`-Klassen erben Implementierungen auch diese Metadaten. Allerdings ist es dabei zumeist nötig, auch diese innere Klasse mit einer neuen inneren Klasse zu beerben; zum einen, um zusätzliche Metadaten bereitzustellen, zum anderen aber auch, um die Berechnung bestehender Metadaten anzupassen. Benutzt beispielsweise eine Spezialisierung einer Implementierung eines Operators zusätzliche Datenstrukturen, so muss der zusätzliche Speicherverbrauch in das Metadatum aktueller Speicherverbrauch eingerechnet werden.

6.3.3. Verwaltung der Abhängigkeiten

Verwaltung statischer Anhängigkeiten

Die gerichteten Kanten des Abhängigkeitsgraphen der Metadaten (siehe 4.2.3) werden zur Laufzeit in beide Richtungen benutzt. Bei Anforderung und Freigabe muss zu einem

Metadatum jeweils bestimmt werden, von welchen anderen dieses abhängt. Umgekehrt muss bei der Aktualisierung zu einem Metadatum bestimmt werden, welche Metadaten von diesem abhängen, damit diese aktualisiert werden können.

Daher muss die Bestimmung sowohl der von einem Knoten des Abhängigkeitsgraphen entlang der Kanten erreichbaren, als auch der entgegengesetzt der Kanten erreichbaren Knoten effizient und konsistent möglich sein. Um dies zu gewährleisten, stellt PIPES eine entsprechende zentrale Komponente zur Verwaltung der statischen Abhängigkeiten bereit. Bei dieser werden die Abhängigkeiten beim Laden der Klassen, bei denen statische Abhängigkeiten der Metadaten vorliegen, registriert. Die erfolgt im statischen Initialisierungsteil der Klasse. Abbildung 6.3 zeigt am Beispiel der abstrakten Vorimplementierung `AbstractPipe` für Operatoren, wie das Metadatum *Verhältnis aus Ein- und Ausgaberate* als abhängig von den Metadaten *Eingaberate* und *Ausgaberate* deklariert wird.

```
MetaDataDependencies.addLocalMetaDataDependencies(
  AbstractPipe.class,
  AbstractPipe.AbstractPipeMetaDataManagement.INPUT_OUTPUT_RATIO,
  AbstractPipe.AbstractPipeMetaDataManagement.INPUT_RATE,
  AbstractPipe.AbstractPipeMetaDataManagement.OUTPUT_RATE
);
```

Abbildung 6.3.: Beispiel für die Deklaration einer statischen Abhängigkeit

Verwaltung dynamischer Anhängigkeiten

Um bei der Verwendung dynamischer Abhängigkeiten eine möglichst große Flexibilität zu wahren, werden diese nicht zentral verwaltet, sondern über Methoden des jeweiligen `MetaDataManagements` (siehe 6.3.2) jeweils neu berechnet. Eine Methode muss dazu zu einem Metadatenschlüssel die Schlüssel aller abhängigen Metadaten liefern, eine zweite invers dazu zu einem Schlüssel eines Metadatums die Schlüssel aller Metadaten, von denen dieses abhängt.

Bei der Benutzung der Abhängigkeiten bildet das Metadatenrahmenwerk von PIPES automatisch die Vereinigung aus statischen und dynamischen Abhängigkeiten. Ist also nur ein Teil der Abhängigkeiten eines Metadatums dynamisch, so kann der statische Teil über den einfacheren Mechanismus für statische Metadaten definiert werden.

Abhängigkeiten für neue Metadaten

Die Registrierung der Abhängigkeiten bei der Implementierung eines neuen Metadatums gestaltet sich somit extrem einfach, falls nur statische Abhängigkeiten vorliegen. Aber auch im Falle dynamischer Abhängigkeiten nimmt das Metadatenrahmenwerk von PIPES dem Programmierer viele Aufgaben ab. Lediglich die eigentliche dynamische Abhängigkeitsfunktion und deren Umkehrfunktion müssen in den jeweiligen Methoden zur Berechnung der dynamischen Abhängigkeiten implementiert werden.

6.3.4. Gewinnung der Metadaten

Können zur Berechnung eines Metadatums Informationen verwendet werden, die ohnehin im Status der Komponente vorhanden sind, so stellt sich die Benutzung der Informationen einfach dar, da die `MetaDataManagement`-Klassen als innere Klassen direkten Zugriff auf den Status haben. Die Gewinnung zusätzlicher Informationen durch Eingriff in den Teil des Quellcodes, der für die Funktionalität der Komponente verantwortlich ist, gestaltet sich hingegen problematischer. Denn um den durch die Metadaten verursachten Overhead so gering wie möglich zu halten, ist dabei sowohl darauf zu achten, dass die Informationsgewinnung im Falle einer Anforderung des Metadatums möglichst effizient geschieht, als auch darauf, dass der Overhead bei nicht angefordertem Metadatum so gering wie möglich oder gar nicht gegeben ist. Da zwischen diesen beiden sogar noch eine Abhängigkeit besteht, sollte beim Entwurf noch berücksichtigt werden, ob eine häufige Nutzung des Metadatums zu erwarten ist. In diesem Fall sollte der Aufwand zur Informationsgewinnung und Berechnung besonders optimiert werden, anderenfalls sollte insbesondere Overhead vermieden werden, der auch schon während Nichtbenutzung des Metadatums eintritt.

Die einfachste Art, für die Bestimmung von Metadaten benötigten Code nur dann auszuführen, wenn er wirklich benötigt wird, ist, diesen Code mit einem booleschen Flag abzusichern und dieses dann zu kippen, wenn der entsprechende `MetaDataHandler` erzeugt beziehungsweise wieder entfernt wird, also bei der ersten Anforderung und bei der entsprechenden Freigabe des Metadatums.

Die generischen Komponenten von PIPES werden in vielen Fällen mit diversen Funktionen und Prädikaten parametrisiert, wobei deren Aufrufhäufigkeiten oft gute Kostenmaße für die Komponente darstellen. Beispielsweise ist bei hash–basierten Implementierungen von Interesse, wie oft die Hashfunktion aufgerufen wird. Da eine solche Kostenmessung jedoch im Normalbetrieb eines DSMS nicht die Regel sein wird, sollte hier der Overhead bei nicht angefordertem Metadatum möglichst gering sein. Dies kann unter Anwendung des Decorator Patterns[GHJV95] elegant erreicht werden. Dazu wird bei nicht angefordertem Metadatum die Funktion beziehungsweise das Prädikat normal verwendet. Bei einer Anforderung wird dann bei der Erzeugung des `MetaDataHandlers` die Funktion beziehungsweise das Prädikat durch einen Dekorator ersetzt, der sie beziehungsweise es dekoriert. Als zusätzliche Funktionalität ergänzt der Dekorator diejenigen Berechnungen, die zur Bestimmung des Metadatums benötigt werden.

Beispielsweise arbeitet der Filteroperator unverändert, solange keine Metadaten mit Bezug zum Filterprädikat, wie zum Beispiel dessen Kosten oder Selektivität, angefordert wurden. Diese hängen nämlich alle von einem künstlichen Metadatum ab, das die Dekoration des Filterprädikats übernimmt. Sobald eines davon angefordert wird, wird dann durch die Abhängigkeit automatisch das Filterprädikat dekoriert. Der Dekorator zählt die Zahl der Aufrufe und die Zahl der Fälle, in denen das Resultat wahr war. Damit kann nun das Metadatum berechnet werden. Zwar protokolliert der Dekorator jetzt gegebenenfalls auch Informationen, die für das angeforderte Metadatum gar nicht benötigt werden. Dies wird aber in Kauf genommen, um geschachtelte Dekorationen zu verhindern, die unangemessenen Mehraufwand verursachen könnten.

6.3.5. Bereitstellung

Durch die automatische Erkennung mehrfacher Anforderungen eines dynamischen Metadatums muss bei Anforderungen beziehungsweise Freigaben nur dann eine neue Bereitstellung des Metadatums erfolgen, wenn dieses angefordert wird, aber noch nicht angefordert war, beziehungsweise wenn die letzte aktive Anforderung freigegeben wird. In diesen beiden Fällen ruft das Metadatenrahmenwerk jeweils ein dafür vorgesehene Methode des `MetaDataManagements` auf, die dann die erforderlichen Schritte vornehmen muss. Diese beiden Methoden, `addMetaData` und `removeMetaData`, müssen daher bei der Implementierung eines neuen Metadatums geeignet erweitert werden.

In der Methode `addMetaData` müssen dabei zwei wesentliche Aufgaben erledigt werden. Zum einen müssen die Änderungen am Status der Komponente vorgenommen werden, die bewirken, dass die speziell für dieses Metadatum nötigen Informationen gesammelt werden. Zu diesen Änderungen zählt etwa das Kippen von Flags, die entsprechende Programmteile abgrenzen, oder das Dekorieren von Funktionen und Prädikaten.

Zum anderen muss der `MetaDataHandler` erzeugt und zum `CompositeMetaData` hinzugefügt werden, der fortan das Metadatum bereitstellt. Für die in 5.2 vorgestellten Aktualisierungsmechanismen gibt es in PIPES Vorimplementierungen, die nur noch geeignet parametrisiert werden müssen, und zwar mit einer Funktion, die einen neuen Wert des Metadatums berechnet. Implementiert als anonyme innere Klassen des `MetaDataHandlers`, haben diese dabei vollen Zugriff auf alle nötigen Informationen, neben dem Status der Komponenten auch auf deren andere Metadaten. Die Anforderung der davon benötigten erfolgt auf Basis der Abhängigkeiten automatisch.

Die Implementierung der dazu inversen Methode `removeMetaData` gestaltet sich zumeist sehr einfach. Hier muss nur der `MetaDataHandler` wieder entfernt und der ursprüngliche Zustand der Komponente wieder hergestellt werden.

Insgesamt muss bei der Implementierung eines neuen Metadatums also nur der Code neu geschrieben werden, der zwingend notwendig ist, nämlich derjenige zum Gewinnen der zusätzlich benötigten Informationen aus der Komponente, der zur Vermeidung von Redundanz bei Nichtbenutzung und der zur Berechnung des Metadatums aus diesen Informationen.

6.3.6. Synchronisation

Wie in 1.2.1 erläutert ist die datengetriebene Verarbeitung ein Kernparadigma der Datenstromverarbeitung. Die Knoten des Operatorgraphen werden in ihrer Verarbeitung daher nur dann aktiv, wenn ein neues Datum bei ihnen eintrifft. In einem effizienten DSMS treiben dabei mehrere Threads nebenläufig oder parallel die Daten durch den Operatorgraphen. Strategien zur effizienten Aufteilung des Operatorgraphen unter den Threads stellen dabei ein interessantes Forschungsgebiet dar.[CHK+07]

Die datengetriebene Verarbeitung in einem Operator kann dabei regelmäßig geschehen, muss aber im Allgemeinen nicht. Da, wie in 5.2.2 erläutert, die Aktualisierung bestimmter Metadaten auf periodischer Basis geschehen sollte, sollte diese Aufgabe somit Threads obliegen, die nicht für die eigentliche Datenverarbeitung zuständig sind. Weil die Kapazität des Systems aber vorwiegend für die Datenverarbeitung genutzt werden soll,

bündelt PIPES die Aufgabe der Metadatenaktualisierung in einem kleinen Threadpool, der üblicherweise aus nur einem Thread besteht. Die `MetaDataHandler` mit periodischer Aktualisierung werden in der Vorimplementierung automatisch für die Aktualisierung durch diesen Pool angemeldet.

Über die Threads, aus denen Metadatennutzer auf die Metadaten zugreifen, können nicht einmal derartige Einschränkungen gemacht werden. Insgesamt ist es daher erforderlich, die Aktivitäten der Komponenten, die zur Aktualisierung ihrer Metadaten und die zum Zugriff auf die Metadaten, voneinander abzuschirmen. Dabei sollte so viel wie nötig, aber so wenig wie möglich abgeschirmt werden, um Verzögerungen und Overhead durch die Synchronisation so gering wie möglich zu halten. Beispielsweise werden die Flags, die bestimmen, ob im Code der Komponente zusätzliche für die Metadaten benötigte Berechnungen durchgeführt werden, nur durch threadsicheren Zugriff auf das entsprechende Feld abgesichert. Erst bei Ausführung des Codes wird dieser durch echte Synchronisation abgeschirmt.

Das folgende Beispiel verdeutlicht, dass schon einfache Zugriffe auf den Zustand eines Operators eine Synchronisation mit der datengetriebenen Verarbeitung erfordern. Verarbeitet ein Operator gerade ein Element und wird dabei etwas in eine Statusstruktur des Operators eingefügt, so würde eine schlichte Abfrage der Größe der Datenstruktur durch das Metadatenrahmenwerk während der laufenden Einfügeoperation bereits eine Ausnahme auslösen.

Das Beispiel zeigt zudem, dass dem Synchronisationsaspekt immer gesonderte Aufmerksamkeit gewidmet werden muss. Denn obwohl ein Metadatum durch Zugriff auf ohnehin vorhandene Informationen berechnet werden kann und seine Implementierung damit keinen Eingriff in den für die Funktionalität einer Komponente zuständigen Code erfordert, kann es nötig werden, diesen zum Zwecke der Synchronisation zu verändern.

6.3.7. Metadaten der Datenstrukturen

Auf Grund des generischen Designs von PIPES sind viele Komponenten auch bezüglich der verwendeten Datenstrukturen parametrisiert. Zum Beispiel können viele Operatoren verschiedene Varianten von `SweepArea`s einsetzen (siehe auch 9.2). Da diese Strukturen in vielen Fällen einen wesentlichen Einfluss auf die Eigenschaften der Komponenten, wie zum Beispiel CPU-Kosten oder Speicherverbrauch haben, haben sie auch grundlegenden Einfluss auf die Metadaten. Nach dem Geheimnisprinzip hat dabei ein Operator nicht zwingend Einblick in die Interna der benutzten Strukturen. Diese müssen jedoch im Gegenzug als Teil ihrer Schnittstellen auch die Informationen bereitstellen, die für die Metadaten benötigt werden.

In PIPES sind derartige Datenstrukturen daher ebenfalls Anbieter von Metadaten. Da sie jedoch nie allein auftreten, sondern immer von anderen Metadatenanbietern benutzt werden, findet hier eine Variante der Metadatenmechanismen mit verringertem Funktionsumfang Anwendung. Diese erlaubt keine Abhängigkeiten von anderen Komponenten und keine eigene periodische Aktualisierung der Metadaten. Diese wird vielmehr von der benutzenden Komponente übernommen. Dies gewährleistet, dass die Aktualisierung der Metadaten der Datenstrukturen synchron mit der der benutzenden Komponente geschieht.

Die Operatoren in PIPES können nun die Metadaten ihrer Datenstrukturen als eigene Metadaten weiterleiten. Auf diese Art kann ein Metadatennutzer Metadaten über die Datenstrukturen abfragen, ohne direkten Zugriff auf diese haben zu müssen.

Teil III.

Verbundoperationen

7. Einleitung

In theoretischen Herleitungen wird die *Verbundoperation* (englisch *Join*) zumeist aus den Operatoren Kreuzprodukt (auch kartesisches Produkt genannt), Filter und optional Projektion abgeleitet. So gesehen ist die Verbundoperation lediglich eine Kurzschreibweise. Überträgt man diese Sichtweise auf die praktische Umsetzung, so sind Verbundoperationen lediglich optionale Optimierungen, die zur Berechnung des korrekten Ergebnisses nicht benötigt werden. Bei näherer Betrachtung in der Theorie, beispielsweise bezüglich Fragen der Komplexität, insbesondere aber in der Praxis zeigt sich jedoch, dass diese Optimierung in vielen Fällen keinesfalls optional, sondern für die Praktikabilität der Ausführung essentiell ist. Dies gilt sowohl im Kontext von Datenbanksystemen, als auch für die Datenstromverarbeitung.

Dieser Kernteil der Arbeit ist komplett dieser Verbundoperation gewidmet. Sie wird dabei nicht aus anderen Datenstromoperatoren, sondern aus ihrem Gegenstück in der Datenbankwelt hergeleitet. Entscheidend ist jedoch die Bereitstellung effizienter Implementierungen, da die Operation erst dadurch ihre bedeutende Aufgabe erfüllen kann, der Kombination aus Grundoperationen überlegen zu sein. Worin genau sich eine solche Überlegenheit ausdrücken sollte, wird in diesem Kapitel in einer Anforderungsanalyse besprochen. Die folgenden Kapitel dieses Teils der Arbeit liefen dann dazu passende praktische Lösungen und diskutieren zusätzliche Optimierungen für die vorgestellten Verfahren.

Nach diesem einführenden Kapitel wird in Kapitel 8 das theoretische Fundament für diesen Teil gelegt. In Kapitel 9 wird dann die praktische Implementierung der Verbundoperation über Datenströmen behandelt. Diesbezügliche Optimierungen werden in Kapitel 10 besprochen. Abschließend wird in Kapitel 11 mit dem Temporalen Progressive-Merge-Join eine sortierbasierte Alternative zu den in Kapitel 9 behandelten Verfahren hergeleitet und diskutiert.

In diesem Kapitel werden in Abschnitt 7.1 zunächst Anforderungen an Verbundoperationen in DBMS diskutiert, bevor die davon abweichenden Anforderungen in DSMS in Abschnitt 7.2 grundlegend formuliert werden. In 7.3 wird dann noch erläutert, welche Bedeutung die in Teil II thematisierten Metadaten für die Verbundberechnung in DSMS haben.

7.1. Verbundoperationen in DBMS

Auf Grund ihrer Bedeutung ist die Verbundoperation Gegenstand umfangreicher Forschungsarbeiten im Bereich der Datenbanksysteme (siehe dazu auch Kapitel 17). Das mit weitem Abstand dominierende Ziel dieser Arbeiten ist es dabei, für vorgegebene Parameter des Verbundes dessen komplettes Ergebnis in möglichst kurzer Zeit zu berechnen. Ein

weiteres verbreitetes Ziel ähnelt diesem zudem noch, nämlich die Bereitstellung möglichst großer Teile des Ergebnisses in Abhängigkeit von der benötigten Verarbeitungszeit.

Diese Ziele sind gleich aus mehreren Gründen sinnvoll. Die Berechnung der Verbundoperation ist Teil der Anfrageverarbeitung, und die Konsumenten der Ergebnisse, seien es direkt oder mittelbar Mensch oder Maschine, wünschen im Allgemeinen eine möglichst kurze Wartezeit zwischen Anfrage und Antwort. Und auch wenn man die Berechnung nicht isoliert betrachtet, sondern im Kontext weiterer Anfragen oder Änderungsoperationen, ist es normalerweise erstrebenswert, wenn die Anfrageausführung schnell zu einem Ende kommt. Denn ist die Anfrage beendet und das Ergebnis ausgeliefert, so nimmt sie auch keine Ressourcen des Systems mehr in Anspruch.

Inwieweit und wie sich diese Ziele erreichen lassen, hängt dabei natürlich essentiell von den zu Grunde liegenden Daten, insbesondere aber auch vom Verbundprädikat ab. Für verschiedene Typen von Prädikaten wurden daher verschiedene Verbundtechniken entwickelt, die in 9.1.2 näher diskutiert werden. Sie lassen sich grob in Verfahren basierend auf verschachtelten Schleifen, Klassifizierung, Indexierung und Sortierung einordnen.

Für verschiedene Spezialfälle von Daten gibt es zusätzlich noch charakteristische Arten von Prädikaten. Beispielsweise werden bezüglich Daten mit Geoinformationen häufig räumliche Schnitte angefragt. Für solchen wichtigen Kombinationen wurden daher eine Reihe von darauf spezialisierten Verbundimplementierungen entwickelt.

Die Anfrageausführung in einem DBMS erfolgt dann zumeist in mehreren Schritten. Zunächst wird die Anfrage vom Optimierer des DBMS analysiert und unter Einbeziehung verfügbarer Metadaten ein möglichst optimaler Ausführungsgraph gewählt. Dabei wird auch entschieden, ob ein Verbundoperator verwendet werden und welche Implementierung zum Einsatz kommen soll. Danach wird dieser Ausführungsgraph installiert und schnellstmöglich abgearbeitet.

Um das Ziel der schnellen Ausführung zu erreichen ist es dabei erforderlich, die Zahl der Ausführungsschritte zu minimieren, insbesondere natürlich die derjenigen, die besonders viel Zeit benötigen. Bei DBMS sind dabei an erster Stelle Externspeicherzugriffe zu nennen, da diese oft um Größenordnungen länger dauern als Zugriffe auf den Hauptspeicher. Zugriffe auf den Cache des Prozessors sind noch günstiger. Daneben gilt es natürlich auch, die Zahl der vom Prozessor auszuführenden Operationen zu minimieren. Da diese sehr vielfältig sind, wird sich zu Analysezwecken zumeist auf besonders charakteristische Operationen konzentriert. Dazu zählen im Falle von Verbundoperationen üblicherweise insbesondere Vergleichsoperationen.

Zur Optimierung mehrerer Anfragen in DBMS bietet es sich vielfach an, diese sequentiell auszuführen, da dann zur Ausführung jeder einzelnen alle Ressourcen des Systems zur Verfügung stehen. Zudem werden nachteilige Konflikte bei paralleler Ausführung vermieden, beispielsweise das Repositionieren des Lesekopfes von Magnetspeichermedien oder Konflikte bei der Nutzung der verschiedenen Stufen des Caches des Prozessors. Werden Anfragen und Änderungsoperationen parallel ausgeführt, gilt es zusätzlich noch zu vermeiden, dass die zur Vermeidung von Inkonsistenzen eingesetzten Locks länger als nötig gehalten werden.

Ziel bei der Entwicklung von Verbundalgorithmen für DBMS ist es nun, dem Optimierer möglichst gute Auswahloptionen zur Verfügung zu stellen, um die genannten Ziele zu erreichen.

7.2. Anforderungen an Verbundoperationen in DSMS

Bei der Übertragung des Konzeptes der Verbundoperation auf DSMS stellt sich nun natürlich die Frage, ob und wie sich die Optimierungsziele ebenfalls übertragen oder wie sie modifiziert werden müssen.

Schon beim Ziel der kurzen Ausführungszeit zeigt sich jedoch, dass dieses im Kontext der kontinuierlichen Anfragen in DSMS weniger Sinn macht, da die Laufzeit im Wesentlichen durch die Zeitspanne zwischen Registrierung und Deregistrierung einer Anfrage bestimmt wird. So lange dabei die Elemente, die aus den Eingabeströmen eintreffen, schnell genug verarbeitet werden können, endet die Laufzeit der Anfrage dann bei der Deregistrierung. Nur wenn noch Elemente nachzuverarbeiten sind, dauert dies länger. Eine wichtige Anforderung an ein stabiles DSMS ist es daher, dass das System nicht dauerhaft mehr CPU-Last erzeugt, als zur Verfügung steht. Die Anforderung, dass eine Operatorimplementierung diese Ressource möglichst wenig belasten sollte, überträgt sich also durchaus von DBMS auf DSMS.

Das Ziel der Minimierung der Externspeicherzugriffe verliert hingegen stark an Bedeutung, da DSMS im Wesentlichen als Hauptspeichersysteme entworfen werden. Erst wenn das Auslagern von Teilen des Status unvermeidlich ist, gewinnt dieses Ziel wieder an Relevanz. Ein wichtiges Ziel ist es jedoch, genau dies zu verhindern. Operatoren sollten also ihren Status nicht unnötig aufblähen und nicht mehr benötigte Informationen zeitnah bereinigen.

Da DSMS kontinuierliche Anfragen ausführen, kommt auch eine sequentielle Anfrageausführung nicht mehr in Frage. Vielmehr werden alle Anfragen nebenläufig verarbeitet und müssen sich daher die Ressourcen des Systems teilen.

Eine detailliertere Diskussion der Optimierungsziele für DSMS folgt in 10.1.1.

Beim Design von Verbundalgorithmen für DSMS stellt sich nun die Frage, wie diese Ziele erreicht werden können. Dabei ist es trotz der merklichen Unterschiede natürlich sinnvoll, die aus DBMS bewährten Techniken auf ihre Verwendbarkeit in DSMS hin zu untersuchen und angepasste Konzepte zu erarbeiten. In dieser Arbeit werden dabei in Kapitel 9 Verfahren entwickelt, die bezüglich der Behandlung der Werte Anleihen bei den bekannten Verfahren mit verschachtelten Schleifen, Klassifizierung und Indexierung nehmen und zugleich bezüglich der Gültigkeitsintervalle von sortierbasierten Verfahren inspiriert sind. Zur Komplettierung der Analogien wird in Kapitel 11 noch ein Algorithmus vorgestellt, der mit wertbasierter Sortierung operiert.

7.3. Bedeutung von dynamischen Metadaten

Wie erläutert werden die Optimierungsentscheidung bei Anfragen in DBMS zumeist vor deren Ausführung endgültig getroffen und nur in Ausnahmefällen revidiert, da der zusätzlich für Reoptimierungen zu investierende Zeitaufwand zumeist nicht mehr durch die zusätzlich eingesparte Zeit aufgeholt werden kann.

Im Falle von DSMS muss der Optimierer seine Entscheidungen bei der Installation einer Anfrage zumeist auf noch weniger Informationen gründen als ein Optimierer eines DBMS, da diesem häufig zumindest grundlegende Metadaten bezüglich der Daten in den

Tabellen und beispielsweise deren Verteilung vorliegen. In DSMS ist dies im Allgemeinen nicht möglich, da die Werte der Datenströme erst nach der Optimierungsentscheidung entstehen. Unter der berechtigten Annahme, dass Datenströme aber in vielen Fällen einer gewissen Charakteristik folgen, kann die zumeist lange Laufzeit kontinuierlicher Anfragen aber gegebenenfalls genutzt werden, um Optimierungsentscheidungen zu revidieren. Insbesondere dazu ist es aber essentiell, detaillierte Metadaten über die Datenströme und das Verhalten der Operatoren zur Laufzeit zu sammeln. Dies gilt insbesondere auch für Verbundoperatoren.[EEVK14] Die Metadaten können nicht nur für die Reoptimierung, sondern auch für die initiale Optimierung hinzukommender Anfragen genutzt werden. Die Umsetzung der Reoptimierung im System wird in 10.1.3 erläutert.

Neben dem Einsatz zur Optimierung spielen dynamische Metadaten aber natürlich auch zur Beobachtung und Analyse von Verbundoperationen eine essentielle Rolle. Die mit ihrer Hilfe erlangten Erkenntnisse haben die in dieser Arbeit vorgestellten Algorithmen stark beeinflusst. Das in Teil II vorgestellte Rahmenwerk kam dabei umfassend zum Einsatz, auch für die in Kapitel 14 besprochenen Experimente.

8. Grundlagen

Bevor es in den folgenden Kapiteln um die Umsetzung geht, werden in diesem zunächst die theoretischen Grundlagen für die Berechnung des Verbundes über Datenströmen gelegt. Wie bereits motiviert soll die Definition von Verbundoperationen über Datenströmen auf die der erweiterten relationalen Algebra zurückgeführt werden, die wiederum die theoretische Grundlage der Verbundoperation in Datenbankmanagementsystemen darstellt.

In 8.1 wird zunächst mit der relationalen Algebra das Fundament gelegt. Aufbauend darauf wird dann in 8.2 die Verbundoperation zunächst klassisch definiert. 8.3 verallgemeinert diese Definition unter Verzicht auf die Verwendung von Schemata auf Multimengen. In 8.4 wird dann daraus die Definition der Verbundes über Datenströmen mit Hilfe der Schnappschuss–Reduzierung formal hergeleitet, und zwar zunächst für logische und dann für physische Datenströme. Zusätzlich wird gezeigt, dass die so definierten Verbünde Verallgemeinerungen des Verbundes in der erweiterten relationalen Algebra und dem über Multimengen darstellen. Zusätzlich wird in 8.5 noch der äußere Verbund für Datenströme definiert.

8.1. Erweiterte relationale Algebra

In der Literatur wird auch heute noch vielfach die relationale Algebra, die über Mengen operiert, zur Erklärung der Grundlage der Datenbankoperationen genutzt. In der Praxis erlauben relationale Datenbanksysteme allerdings Duplikate von Tupeln. Daher findet in dieser Arbeit die erweiterte relationale Algebra Anwendung, die über Multimengen operiert.

Da sich auch deren Definition und zugehörige Schreibweisen in der Literatur unterscheiden, werden hier zunächst die dafür benötigten Definitionen angegeben.

Definition 8.1 (Erweiterte relationale Algebra)**.** Sei $\mathbb{A}$ eine Menge von Attributen, $\mathbb{V}$ eine Menge von Wertebereichen und $dom_{\mathbb{A}} : \mathbb{A} \to \mathbb{V}$ eine Funktion, die einem Attribut einen Wertebereich zuordnet. Sei $RS := \bigcup_{n=0}^{\infty} \mathbb{A}^n \setminus \{(\ldots, A, \ldots, A, \ldots) \mid A \in \mathbb{A}\}$ die Menge aller Schemata. Die Abbildung $dom : RS \to \bigcup_{n=0}^{\infty} \mathbb{V}^n$ mit $dom((a_1, \ldots, a_n)) := dom_{\mathbb{A}}(a_1) \times \ldots \times dom_{\mathbb{A}}(a_n)$ ordne einem Schema einen Wertebereich zu.

Ein Relation R wird definiert durch ein Paar $R = (RS_R, V_R)$ bestehend aus einem Schema $RS_R \in RS$ und einer Vielfachheitsfunktion $V_R : dom(RS_R) \to \mathbb{N}$. Sei Rel die Menge aller Relationen.

Die Relationen bilden nun zusammen mit den Operationen die erweiterte relationale Algebra. Die Operationen, die für diese Arbeit von Bedeutung sind, werden im Folgenden definiert.

Definition 8.2 (Grundoperationen der erweiterten relationalen Algebra)**.**

- Umbenennung ρ: $\rho_{Y_1 \leftarrow X_1,\ldots,Y_m \leftarrow X_m} : Rel \rightarrow Rel$ mit $\rho_{Y_1 \leftarrow X_1,\ldots,Y_m \leftarrow X_m}((RS_R, V_R)) = (RS_S, V_S)$, falls $RS_R = (A_1, \ldots, A_n)$, $RS_S = (B_1, \ldots, B_n)$, $\forall i \in \{1, \ldots, n\} : (A_i = B_i \wedge \nexists j \in \{1, \ldots, m\} : A_i = X_j) \vee (\exists j \in \{1, \ldots, m\} : (A_i = X_j \wedge B_i = Y_j))$, $dom(RS_R) = dom(RS_S)$ und $V_R = V_S$.
- Kreuzprodukt $\times$: $\times : Rel \times Rel \rightarrow Rel$ mit $\times((RS_R, V_R), (RS_S, V_S)) = (RS_T, V_T)$, falls $\neg\exists A : RS_R = (\ldots, A, \ldots) \wedge RS_S = (\ldots, A, \ldots)$, $RS_T = RS_R \circ RS_S$ und $V_T(x \circ y) = V_R(x) \cdot V_S(y)$
- Vereinigung $\cup$: $\cup : Rel \times Rel \rightarrow Rel$ mit $\cup((RS_R, V_R), (RS_S, V_S)) = (RS_T, V_T)$, falls $RS_R = RS_S = RS_T$ und $V_T(x) = V_R(x) + V_S(x)$
- Selektion σ: $\sigma_P : Rel \rightarrow Rel$ mit $\sigma_P((RS_R, V_R)) = (RS_S, V_S)$, falls $RS_S = RS_R$, P ein Prädikat über $dom(RS_R)$ und $V_S(x) = V_R(x)$, falls $P(x)$, und $V_S(x) = 0$, sonst.
- Projektion π: $\pi_{B_1,\ldots,B_m} : Rel \rightarrow Rel$ mit $\pi_{B_1,\ldots,B_m}((RS_R, V_R)) = (RS_S, V_S)$, falls $RS_R = (A_1, ..., A_n)$, $RS_S = (B_1, ..., B_m)$ und $\exists i_1, \ldots, i_m : A_{i_1} = B_1 \wedge \ldots \wedge A_{i_m} = B_m \wedge V_S(b_1, \ldots, b_m) = \sum_{(a_1,\ldots,a_n) \in dom(RS_R),\ a_{i_1}=b_1,\ldots,a_{i_n}=b_n} V_R(a_1, \ldots, a_n)$

8.2. Relationale Verbünde

Bezüglich der (erweiterten) relationalen Algebra werden unter dem Begriff *Verbundoperation* (*Join*) in der Regel eine Menge von Operatoren zusammengefasst, die sich in verschiedener Weise aus einer Kombination der elementaren Operatoren Umbenennung, Kreuzprodukt, Selektion und Projektion in dieser Reihenfolge ergeben.

Definition 8.3 (Verbundoperation der erweiterten relationalen Algebra)**.** Seien $A = \{A_1, \ldots, A_n\}$, $B = \{B_1, \ldots, B_m\}$, $C = \{C_1, \ldots, C_l\} := A \cap B$ und D Attributmengen mit $|D| = |C|$ und $A \cap D = B \cap D = \emptyset$ und sei $\rho_{\mathbb{A}} : C \rightarrow D$ bijektiv. Seien $R = ((A_1, \ldots, A_n), V_R)$ und $S = ((B_1, \ldots, B_m), V_S)$ Relationen. Sei P ein Prädikat über $dom(A) \circ dom(B)$ und sei $E = \{E_1, \ldots, E_k\} \subseteq A \cup B \cup D$. Dann ist die Verbundoperation $\bowtie_{rel}$ der erweiterten relationalen Algebra definiert durch

$$\bowtie_P^{rel}(R, S) := \pi_{E_1,\ldots,E_k}(\sigma_P(\times(R, \rho_{\rho_{\mathbb{A}}(C_1) \leftarrow C_1,\ldots,\rho_{\mathbb{A}}(C_l) \leftarrow C_l}(S))))$$

Die Umbenennung erfüllt dabei die Aufgabe, die für das Kreuzprodukt benötigte Disjunktheit der Eingabeschemata zu gewährleisten. Das Kreuzprodukt bildet alle möglichen Paare aus den Eingaben und das Prädikat filtert dann die gewünschten Paare heraus. Abschließend entfernt die Projektion nicht benötigte Attribute des Ergebnisses.

Für spezielle Arten von Eingaberelationen R und S sowie durch geeignete Wahl des Prädikates P und der Menge der Ergebnisattribute E erhält man wichtige Spezialfälle des Verbundes, die in Tabelle 8.1 Bezug nehmend auf die Notation aus Definition 8.3 aufgelistet sind. Das Verbundprädikat hat dabei wesentlichen Einfluss auf die Komplexität des Problems.[CCKN01]

Bezeichnung	Voraus-setzungen	P	E
Kreuzprodukt	$C = \emptyset$	*wahr*	$A \cup B$
Theta-Verbund	$C = \emptyset$	$A_i \theta B_j$	$A \cup B$
Gleichverbund	$C = \emptyset$	$A_{i_1} = B_{j_1}, \ldots, A_{i_p} = B_{j_p}$	$A \cup B$
Natürlicher Verbund		$C_1 = \rho_{\mathbb{A}}(C_1), \ldots, C_l = \rho_{\mathbb{A}}(C_l)$	$A \cup B$
Linker Semi-Verbund		$C_1 = \rho_{\mathbb{A}}(C_1), \ldots, C_l = \rho_{\mathbb{A}}(C_l)$	A
Rechter Semi-Verbund		$C_1 = \rho_{\mathbb{A}}(C_1), \ldots, C_l = \rho_{\mathbb{A}}(C_l)$	B
Linker Anti-Verbund		$\neg(C_1 = \rho_{\mathbb{A}}(C_1), \ldots, C_l = \rho_{\mathbb{A}}(C_l))$	A
Rechter Anti-Verbund		$\neg(C_1 = \rho_{\mathbb{A}}(C_1), \ldots, C_l = \rho_{\mathbb{A}}(C_l))$	B
Band-Verbund	$C = \emptyset$	$\lvert A_i - B_j \rvert < \epsilon$	$A \cup B$

Tabelle 8.1.: Wichtige Spezialfälle der relationalen Verbundoperation

8.3. Allgemeiner Verbund

Die Idee der Verbundoperation kann man nun von Relationen über einem Schema auf allgemeine Multimengen übertragen. Die Umbenennung als Spezialität der relationalen Algebra wird damit hinfällig. Das Kreuzprodukt ist auch auf allgemeine Multimengen anwendbar und bildet Paare aus Elementen der zugrunde liegenden Multimengen. Die Vielfachheit eines Paares ergibt sich dabei als Produkt der Vielfachheiten seiner Komponenten, was allen möglichen Paarbildungen aus den jeweiligen Einzelelementen entspricht. Das Verbund-Prädikat muss daher auf solchen Paaren anwendbar sein. Die Paare, für die das Prädikat erfüllt ist, könnte man als Ausgabe betrachten. Um aber eine echte Verallgemeinerung der relationalen Verbundoperation zu erhalten, muss eine Möglichkeit gegeben sein, nur den Teil der Information aus den Paaren zu erhalten, der benötigt wird. Dazu wird eine abschließende Ergebnisfunktion eingesetzt, die aus den Paaren Elemente der Ergebnisdomäne erzeugt.

Definition 8.4 (Verbundoperation auf Multimengen)**.** Seinen A, B und C Mengen und $R : A \to \mathbb{N}$ und $S : B \to \mathbb{N}$ Multimengen. Sei P ein Prädikat, genannt *Verbundprädikat*, über $A \times B$ und $r : A \times B \to C$ eine Funktion, genannt *Ergebnisfunktion*. Dann ist die allgemeine Verbundoperation $\bowtie$ definiert durch

$$\bowtie_P (R, S) := T \text{ mit } T : C \to \mathbb{N}, T(c) = \sum_{a \in A, b \in B : P(a,b) \wedge c = r(a,b)} R(a) \cdot S(b)$$

Die Verbundoperation der erweiterten relationalen Algebra ist zwar kein echter Spezialfall der Verbundoperation auf Multimengen, da diese keine Schemata als Teil der Ein- und Ausgabe berücksichtigt. Die wesentliche Berechnung, nämlich die der Vielfachheitsfunktion der Ergebnisrelation, wird jedoch von der Verbundoperation auf Multimengen geleistet, wie das folgende Lemma zeigt.

Lemma 8.1. *Die Vielfachheitsfunktion des Ergebnisses der Verbundoperation der erweiterten relationalen Algebra kann mit der Verbundoperation auf Multimengen berechnet werden.*

Beweis. Seien $A = \{A_1, \ldots, A_n\}$, $B = \{B_1, \ldots, B_m\}$, $C = \{C_1, \ldots, C_l\} := A \cap B$ und D Attributmengen mit $|D| = |C|$ und $A \cap D = B \cap D = \emptyset$ und sei $\rho_{\mathbb{A}} : C \to D$ bijektiv. Seien $R = ((A_1, \ldots, A_n), V_R)$ und $S = ((B_1, \ldots, B_m), V_S)$ Relationen. Sei P ein Prädikat über $dom(A) \circ dom(B)$ und sei $E = \{E_1, \ldots, E_k\} \subseteq A \cup B \cup D$.

Dann sind $A' := dom(A)$, $B' := dom(B)$ und $C := dom(E)$ Mengen und $V'_R : A' \to \mathbb{N} : V'_R((a_1, \ldots, a_n)) := V_R(a_1, \ldots, a_n)$ und $V'_S : B' \to \mathbb{N} : V'_S((b_1, \ldots, b_m)) := V_S(b_1, \ldots, b_m)$ Multimengen. Sei P' definiert durch $P'((a_1, \ldots, a_n), (b_1, \ldots, b_m)) := P(a_1, \ldots, a_n, b_1, \ldots, b_m)$ und die Ergebnisfunktion r' durch $I := \{i | A_i \in E\}$, $J := \{j | (B_j \in (E \backslash A)) \vee (B_j \in A \wedge \rho_{\mathbb{A}}(B_j) \in E)\}$ und $r'((a_1, \ldots, a_n), (b_1, \ldots, b_m)) := (a_{min(I)}, \ldots, a_{max(I)}, b_{min(J)}, \ldots, b_{max(J)})$.

Damit gilt $\forall x \in dom(RS_{\bowtie_{rel}(R,S)})$:

$$
\begin{aligned}
&\bowtie_{P',r'} (R', S')(x) = \\
&\sum_{a \in A', b \in B' : P'(a,b) \wedge x = r'(a,b)} V'_R(a) \cdot V'_S(b) = \\
&\sum_{\substack{(a_1,\ldots,a_n,b_1,\ldots,b_m) \in dom(\times(R, \rho_{\rho_{\mathbb{A}}(C_1) \leftarrow C_1, \ldots, \rho_{\mathbb{A}}(C_l) \leftarrow C_l}(S))), \\ x = r'((a_1,\ldots,a_n),(b_1,\ldots,b_m)) \wedge P(a_1,\ldots,a_n,b_1,\ldots,b_m)}} V_R(a_1, \ldots, a_n) \cdot V_S(b_1, \ldots, b_m) = \\
&\sum_{\substack{(a_1,\ldots,a_n,b_1,\ldots,b_m) \in dom(\times(R, \rho_{\rho_{\mathbb{A}}(C_1) \leftarrow C_1, \ldots, \rho_{\mathbb{A}}(C_l) \leftarrow C_l}(S))), \\ x = r'((a_1,\ldots,a_n),(b_1,\ldots,b_m)) \wedge P(a_1,\ldots,a_n,b_1,\ldots,b_m)}} V_R(a_1, \ldots, a_n) \cdot V_{\rho_{\rho_{\mathbb{A}}(C_1) \leftarrow C_1, \ldots, \rho_{\mathbb{A}}(C_l) \leftarrow C_l}(S)}(b_1, \ldots, b_m) = \\
&\sum_{\substack{(a_1,\ldots,a_n,b_1,\ldots,b_m) \in dom(\times(R, \rho_{\rho_{\mathbb{A}}(C_1) \leftarrow C_1, \ldots, \rho_{\mathbb{A}}(C_l) \leftarrow C_l}(S))), \\ x = r'((a_1,\ldots,a_n),(b_1,\ldots,b_m)) \wedge P(a_1,\ldots,a_n,b_1,\ldots,b_m)}} V_{\times(R, \rho_{\rho_{\mathbb{A}}(C_1) \leftarrow C_1, \ldots, \rho_{\mathbb{A}}(C_l) \leftarrow C_l}(S))}(a_1, \ldots, a_n, b_1, \ldots, b_m) = \\
&\sum_{\substack{(a_1,\ldots,a_n,b_1,\ldots,b_m) \in dom(\sigma_P(\times(R, \rho_{\rho_{\mathbb{A}}(C_1) \leftarrow C_1, \ldots, \rho_{\mathbb{A}}(C_l) \leftarrow C_l}(S)))), \\ a_{min(I)} = x_1, \ldots, a_{max(I)} = x_{|I|}, b_{min(J)} = x_{|I|+1}, \ldots, b_{max(J)} = x_{|I|+|J|}}} V_{\sigma_P(\times(R, \rho_{\rho_{\mathbb{A}}(C_1) \leftarrow C_1, \ldots, \rho_{\mathbb{A}}(C_l) \leftarrow C_l}(S)))}(a_1, \ldots, a_n, b_1, \ldots, b_m) = \\
&V_{\pi_{E_1,\ldots,E_k}(\sigma_P(\times(R, \rho_{\rho_{\mathbb{A}}(C_1) \leftarrow C_1, \ldots, \rho_{\mathbb{A}}(C_l) \leftarrow C_l}(S))))}(x) = \\
&V_{\bowtie_P^{rel}(R,S)}(x)
\end{aligned}
$$

□

8.4. Verbundoperation auf Datenströmen

Die Definition der Verbundoperation über Datenströmen erfolgt zunächst bezüglich der für die Semantikdefinitionen ausgelegten logischen Datenströme. Danach wird daraus die Verbundoperation für physische Datenströme abgeleitet.

8.4.1. Verbundoperation auf logischen Datenströmen

In der logischen Sicht auf einen Datenstrom muss für jeden Zeitpunkt t und jedes Element e der Ausgabedomäne festgelegt werden, wie oft dieses zu dem Zeitpunkt gültig ist. Dazu werden jeweils alle Paare von Eingabeelementen gebildet, die das Verbundprädikat

erfüllen, bei Anwendung der Ergebnisfunktion dieses Element e ergeben und die zu jenem Zeitpunkt t in den Eingabeströmen gültig sind. Durch Aufsummieren wird deren Anzahl als Ergebnis ermittelt. Analog zum Verbund über Multimengen werden dabei die Vielfachheiten der Elemente zur Bildung aller möglichen Kombinationen multipliziert.

Definition 8.5 (Verbundoperation auf logischen Datenströmen)**.** Seien $\mathcal{T}_1, \mathcal{T}_2$ und $\mathcal{T}$ Typen, P ein Prädikat auf $\Omega_{\mathcal{T}_1} \times \Omega_{\mathcal{T}_2}$ und $r : \Omega_{\mathcal{T}_1} \times \Omega_{\mathcal{T}_2} \to \Omega_{\mathcal{T}}$ eine Ergebnisfunktion. Der allgemeine Verbund über logischen Datenströmen $\bowtie^l$: $\mathbb{S}^l_{\mathcal{T}_1} \times \mathbb{S}^l_{\mathcal{T}_2} \to \mathbb{S}^l_{\mathcal{T}}$ ist definiert durch

$$\bowtie^l_{P,r}(S_1, S_2) := \{(e, t, n) \mid n = \sum_{(e_1,t,n_1)\in S_1,(e_2,t,n_2)\in S_2 : P(e_1,e_2)\wedge e=r(e_1,e_2)} n_1 \cdot n_2\}$$

Ziel der Definition der Verbundoperation auf logischen Datenströmen ist, dass sie für jeden Zeitpunkt semantisch der Verbundoperation auf Multimengen entspricht. Der folgende Satz zeigt, dass Definition 8.5 dies erreicht.

Satz 8.1. *Die Verbundoperation auf logischen Datenströmen ist Schnappschuss-reduzierbar auf die Verbundoperation auf Multimengen.*

Beweis. Seien $t \in \mathbb{T}$ und $S^l_1 \in \mathbb{S}^l_{\mathcal{T}_1}, S^l_2 \in \mathbb{S}^l_{\mathcal{T}_2}$:

$$\tau(\bowtie^l_{P,r}(S^l_1, S^l_2), t) =$$

$$\tau(\{(e, t, n) \mid n = \sum_{(e_1,t,n_1)\in S^l_1,(e_2,t,n_2)\in S^l_2 : P(e_1,e_2)\wedge e=r(e_1,e_2)} n_1 \cdot n_2 \wedge n > 0\}, t) =$$

$$(f : \Omega_{\mathcal{T}} \to \mathbb{N}, f(e) = \xi(\{(e, t, n) \mid n = \sum_{(e_1,t,n_1)\in S^l_1,(e_2,t,n_2)\in S^l_2 : P(e_1,e_2)\wedge e=r(e_1,e_2)} n_1 \cdot n_2 \wedge n > 0\}, e, t)) =$$

$$(f : \Omega_{\mathcal{T}} \to \mathbb{N}, f(e) = \sum_{(e_1,t,n_1)\in S^l_1,(e_2,t,n_2)\in S^l_2 : P(e_1,e_2)\wedge e=r(e_1,e_2)} n_1 \cdot n_2) =$$

$$(f : \Omega_{\mathcal{T}} \to \mathbb{N}, f(e) = \sum_{e_1\in\Omega_{\mathcal{T}_1},e_2\in\Omega_{\mathcal{T}_2},(e_1,t,n_1)\in S^l_1,(e_2,t,n_2)\in S^l_2,P(e_1,e_2),e=r(e_1,e_2)} n_1 \cdot n_2) =$$

$$(f : \Omega_{\mathcal{T}} \to \mathbb{N}, f(e) = \sum_{e_1\in\Omega_{\mathcal{T}_1},e_2\in\Omega_{\mathcal{T}_2} : P(e_1,e_2)\wedge e=r(e_1,e_2)} \xi(S^l_1, e_1, t) \cdot \xi(S^l_2, e_2, t))$$

$$\bowtie_{P,r}((f_1 : \Omega_{\mathcal{T}} \to \mathbb{N}, f_1(e_1) = \xi(S^l_1, e_1, t)), (f_2 : \Omega_{\mathcal{T}} \to \mathbb{N}, f_2(e_2) = \xi(S^l_2, e_2, t))) =$$

$$\bowtie_{P,r}(\tau(S^l_1, t), \tau(S^l_2, t))$$

□

Für jeden Zeitpunkt wird also die Verbundoperation auf Multimengen für den jeweiligen Schnappschuss der Stromelemente zu diesem Zeitpunkt berechnet. Beschränkt man sich auf einen einzelnen Zeitpunkt, so kann man mit Hilfe der Verbundoperation auf logischen Datenströmen die über Multimengen berechnen, wie der folgende Satz zeigt.

Lemma 8.2. *Die Verbundoperation auf Multimengen kann auf Basis der Verbundoperation auf logischen Datenströmen berechnet werden.*

Beweis. Seinen A, B und C Mengen und $R : A \to \mathbb{N}$ und $S : B \to \mathbb{N}$ Multimengen. Sei P ein Prädikat über $A \times B$ und $r : A \times B \to C$ eine Ergebnisfunktion.

Seien $\mathcal{T}_1, \mathcal{T}_2$ und $\mathcal{T}$ Typen mit $\Omega_{\mathcal{T}_1} = A$, $\Omega_{\mathcal{T}_2} = B$ und $\Omega_{\mathcal{T}} = C$ sowie $t \in \mathbb{T}$. Sei $S_1 := \{(e, \hat{t}, n) \in \Omega_{\mathcal{T}_1} \times \mathbb{T} \times \mathbb{N}^+ \mid n = R(e) \wedge \hat{t} = t\}$ und $S_2 := \{(e, \hat{t}, n) \in \Omega_{\mathcal{T}_2} \times \mathbb{T} \times \mathbb{N}^+ \mid n = S(e) \wedge \hat{t} = t\}$:

$$
\begin{aligned}
&\bowtie_{P,r} (R, S) = \\
&(f : C \to \mathbb{N}, f(e) = \sum_{a \in A, b \in B : P(a,b) \wedge e = r(a,b)} R(a) \cdot S(b)) = \\
&(f : \Omega_{\mathcal{T}} \to \mathbb{N}, f(e) = \sum_{e_1 \in \Omega_{\mathcal{T}_1}, e_2 \in \Omega_{\mathcal{T}_2}, n_1 \in \mathbb{N}, n_2 \in \mathbb{N} : n_1 = R(e_1) \wedge n_2 = S(e_2) \wedge P(e_1,e_2) \wedge e = r(e_1,e_2)} n_1 \cdot n_2) = \\
&(f : \Omega_{\mathcal{T}} \to \mathbb{N}, f(e) = \sum_{(e_1,t,n_1) \in S_1, (e_2,t,n_2) \in S_2 : P(e_1,e_2) \wedge e = r(e_1,e_2)} n_1 \cdot n_2) = \\
&(f : \Omega_{\mathcal{T}} \to \mathbb{N}, f(e) = \xi(\{(e, \bar{t}, n) \mid n = \sum_{(e_1,\bar{t},n_1) \in S_1, (e_2,\bar{t},n_2) \in S_2 : P(e_1,e_2) \wedge e = r(e_1,e_2)} n_1 \cdot n_2\}, e, t))) = \\
&\tau(\{(e, \bar{t}, n) \mid n = \sum_{(e_1,\bar{t},n_1) \in S_1, (e_2,\bar{t},n_2) \in S_2 : P(e_1,e_2) \wedge e = r(e_1,e_2)} n_1 \cdot n_2\}, t) = \\
&\tau(\bowtie^{l}_{P,r} (S_1, S_2), t)
\end{aligned}
$$

□

Somit kann insbesondere auch ein relationaler Verbund berechnet werden.

Korollar 8.1. *Die Verbundoperation der erweiterten relationalen Algebra kann auf Basis der Verbundoperation auf logischen Datenströmen berechnet werden.*

Beweis. Die Behauptung folgt direkt aus Lemma 8.2 und Lemma 8.1. □

Die Konstruktionen in den Beweisen zu Lemma 8.1 und Satz 8.1 zeigen zudem, dass der Verbund über Datenströmen für jeden Schnappschuss einen erweiterten relationalen Verbund berechnet, falls die Ein- und Ausgabetypen der Datenströme dem entsprechenden relationalen Schema genügen. Dadurch wird die Semantik des Datenstromverbundes für einen breiten Benutzerkreis leicht zugänglich, da der Verbund der erweiterten relationalen Algebra aus relationalen DBMS bekannt ist und für viele Softwareentwickler zu den Standardwerkzeugen zählt. Dies entspricht der in Abschnitt 2.4.1 dargelegten Motivation der Datenstromsemantik über die Schnappschuss-Reduzierung.

8.4.2. Verbundoperation auf physischen Datenströmen

Passend zur Semantik der Verbundoperation über logischen Datenströmen muss nun für physische Datenströme eine geeignete Implementierung angegeben werden. Da die Abbildung von physischen auf logische Datenströme nicht injektiv ist, gibt es dabei mehrere Möglichkeiten. Die folgende Definition ist derart ausgelegt, dass sie gut in konkrete Algorithmen umgesetzt werden kann.

Definition 8.6 (Verbundoperation auf physischen Datenströmen)**.** Seien $\mathcal{T}_1, \mathcal{T}_2$ und $\mathcal{T}$ Typen, P ein Prädikat auf $\Omega_{\mathcal{T}_1} \times \Omega_{\mathcal{T}_2}$ und $r : \Omega_{\mathcal{T}_1} \times \Omega_{\mathcal{T}_2} \to \Omega_{\mathcal{T}}$ eine Ergebnisfunktion. Der *allgemeine Verbund über physischen Datenströmen* $\bowtie^p_{P,r} \colon \mathbb{S}^p_{\mathcal{T}_1} \times \mathbb{S}^p_{\mathcal{T}_2} \to \mathbb{S}^p_{\mathcal{T}}$ ist definiert durch

$$\bowtie^p_{P,r}(S_1, S_2) := \omega(f : (\Omega_{\mathcal{T}} \times I_{\mathbb{T}}) \to \mathbb{N}, f((e,i)) = \sum_{\substack{e_1 \in \Omega_{\mathcal{T}_1}, e_2 \in \Omega_{\mathcal{T}_2}, i_1 \in I_{\mathbb{T}}, i_2 \in I_{\mathbb{T}}: \\ P(e_1,e_2) \wedge e = r(e_1,e_2) \wedge i = i_1 \cap i_2}} \bar{\omega}(S_1)(e_1, i_1) \cdot \bar{\omega}(S_2)(e_2, i_2))$$

Das folgende Lemma zeigt auf, dass diese Definition korrekt ist, also semantisch dem Verbund auf logischen Datenströmen entspricht.

Lemma 8.3. *Die Verbundoperation auf physischen Datenströmen $\bowtie_p$ implementiert die Verbundoperation auf logischen Datenströmen $\bowtie_l$.*

Beweis. Seien $\mathcal{T}_1, \mathcal{T}_2$ und $\mathcal{T}$ Typen, P ein Prädikat auf $\Omega_{\mathcal{T}_1} \times \Omega_{\mathcal{T}_2}$, $r : \Omega_{\mathcal{T}_1} \times \Omega_{\mathcal{T}_2} \to \Omega_{\mathcal{T}}$ eine Ergebnisfunktion und $S^p_1 \in \mathbb{S}^p_{\mathcal{T}_1}, S^p_2 \in \mathbb{S}^p_{\mathcal{T}_2}$:

$$\varphi^{p \mapsto l}(\bowtie^p_{P,r}(S^p_1, S^p_2)) =$$

$$\varphi^{p \mapsto l}(\omega(f : (\Omega_{\mathcal{T}} \times I_{\mathbb{T}}) \to \mathbb{N}, f((e,i)) = \sum_{\substack{e_1 \in \Omega_{\mathcal{T}_1}, e_2 \in \Omega_{\mathcal{T}_2}, i_1 \in I_{\mathbb{T}}, i_2 \in I_{\mathbb{T}}: \\ P(e_1,e_2) \wedge e = r(e_1,e_2) \wedge i = i_1 \cap i_2}} \bar{\omega}(S^p_1)(e_1, i_1) \cdot \bar{\omega}(S^p_2)(e_2, i_2))) =$$

$$\Bigg\{(e,t,n) \in \Omega_{\mathcal{T}} \times T \times \mathbb{N}^+ \;\Bigg|\; n = \Big|\Big\{j \to (e, [t_{s_j}, t_{e_j})) \in$$

$$\omega(f : (\Omega_{\mathcal{T}} \times I_{\mathbb{T}}) \to \mathbb{N}, f((\hat{e},i)) = \sum_{\substack{e_1 \in \Omega_{\mathcal{T}_1}, e_2 \in \Omega_{\mathcal{T}_2}, i_1 \in I_{\mathbb{T}}, i_2 \in I_{\mathbb{T}}: \\ P(e_1,e_2) \wedge \hat{e} = r(e_1,e_2) \wedge i = i_1 \cap i_2}} \bar{\omega}(S^p_1)(e_1, i_1) \cdot \bar{\omega}(S^p_2)(e_2, i_2)) \Big|$$

$$t \in [t_{s_j}, t_{e_j})\Big\} \Big| \Bigg\} =$$

$$\Big\{(e,t,n) \in \Omega_{\mathcal{T}} \times T \times \mathbb{N}^+ \;\Big|\; n = \sum_{\substack{e_1 \in \Omega_{\mathcal{T}_1}, e_2 \in \Omega_{\mathcal{T}_2}, i_1 \in I_{\mathbb{T}}, i_2 \in I_{\mathbb{T}}: \\ P(e_1,e_2) \wedge e = r(e_1,e_2) \wedge i = i_1 \cap i_2 \wedge t \in i}} \bar{\omega}(S^p_1)(e_1, i_1) \cdot \bar{\omega}(S^p_2)(e_2, i_2)\Big\} =$$

$$\Big\{(e,t,n) \in \Omega_{\mathcal{T}} \times T \times \mathbb{N}^+ \;\Big|\; n = \sum_{\substack{e_1 \in \Omega_{\mathcal{T}_1}, e_2 \in \Omega_{\mathcal{T}_2}, n_1 \in \mathbb{N}^+, n_2 \in \mathbb{N}^+: \\ n_1 = \sum_{i_1 \in I_{\mathbb{T}}: t \in i_1} \bar{\omega}(S^p_1)(e,i_1) \wedge \\ n_2 = \sum_{i_2 \in I_{\mathbb{T}}: t \in i_2} \bar{\omega}(S^p_2)(e,i_2) \wedge \\ P(e_1,e_2) \wedge e = r(e_1,e_2)}} n_1 \cdot n_2\Big\} =$$

$$\Big\{(e,t,n) \in \Omega_{\mathcal{T}} \times T \times \mathbb{N}^+ \;\Big|\; n = \sum_{\substack{(e_1,t,n_1) \in \{(e,t,n) \in \Omega_{\mathcal{T}} \times T \times \mathbb{N}^+ \mid n = \sum_{i_1 \in I_{\mathbb{T}}: t \in i_1} \bar{\omega}(S^p_1)(e,i_1)\}, \\ (e_2,t,n_2) \in \{(e,t,n) \in \Omega_{\mathcal{T}} \times T \times \mathbb{N}^+ \mid n = \sum_{i_2 \in I_{\mathbb{T}}: t \in i_2} \bar{\omega}(S^p_2)(e,i_2)\}: \\ P(e_1,e_2) \wedge e = r(e_1,e_2)}} n_1 \cdot n_2\Big\} =$$

$$\Big\{(e,t,n) \in \Omega_{\mathcal{T}} \times T \times \mathbb{N}^+ \;\Big|\; n = \sum_{\substack{(e_1,t,n_1) \in \{(e,t,n) \in \Omega_{\mathcal{T}} \times T \times \mathbb{N}^+ \mid n = |\{i \to (e,i) \in S^p_1 \mid t \in i\}|\}, \\ (e_2,t,n_2) \in \{(e,t,n) \in \Omega_{\mathcal{T}} \times T \times \mathbb{N}^+ \mid n = |\{i \to (e,i) \in S^p_2 \mid t \in i)\}|\}: \\ P(e_1,e_2) \wedge e = r(e_1,e_2)}} n_1 \cdot n_2\Big\} =$$

$$\left\{(e,t,n) \in \Omega_{\mathcal{T}} \times T \times \mathbb{N}^+ \;\middle|\; n = \sum_{(e_1,t,n_1)\in\varphi^{p\mapsto l}(S_1^p),(e_2,t,n_2)\in\varphi^{p\mapsto l}(S_2^p):P(e_1,e_2)\wedge e=r(e_1,e_2)} n_1 \cdot n_2\right\} =$$

$$\bowtie^{l}_{P,r}(\varphi^{p\mapsto l}(S_1^p), \varphi^{p\mapsto l}(S_2^p))$$

□

Anschauung

Für eine anschauliche Interpretation der Verbundoperation auf physischen Datenströmen bietet es sich an, die Frage zu betrachten, wann im Ausgabedatenstrom ein Stromelement (e,i), also der Wert e mit Gültigkeitsintervall i erzeugt wird. Dazu muss es Werte e_1 und e_2 sowie Gültigkeitsintervalle i_1 und i_2 geben, derart dass das Verbundprädikat für (e_1,e_2) erfüllt ist, die Ergebnisfunktion angewendet auf (e_1,e_2) das Element e ergibt und der Schnitt der Intervalle $i_1 \cap i_2$ genau das Intervall i ergibt. Anderenfalls wäre die Summe in Definition 8.6 leer. Damit nicht einer der Faktoren des Produktes, über das summiert wird, null wird, muss zudem (e_1,i_1) in S_1 sowie (e_2,i_2) in S_2 vorkommen.

Um ein Ergebnis zu produzieren, müssen als notwendige Bedingung also zwei Eingabestromelemente (e_1,i_1) in S_1 und (e_2,i_2) in S_2 existieren, deren Elemente das Joinprädikat erfüllen und deren Gültigkeitsintervalle sich schneiden. Durch Anwendung der Ergebnisfunktion auf die Elemente und durch Schneiden der Gültigkeitsintervalle erhält man das Ergebnisstromelement $(r(e_1,e_2), i_1 \cap i_2)$.

Wie oft dieses im Ergebnisstrom vorkommt, hängt davon ab, wie oft die jeweiligen Eingabestromelemente in den Eingabeströmen auftreten. Die Summe läuft hierbei genau

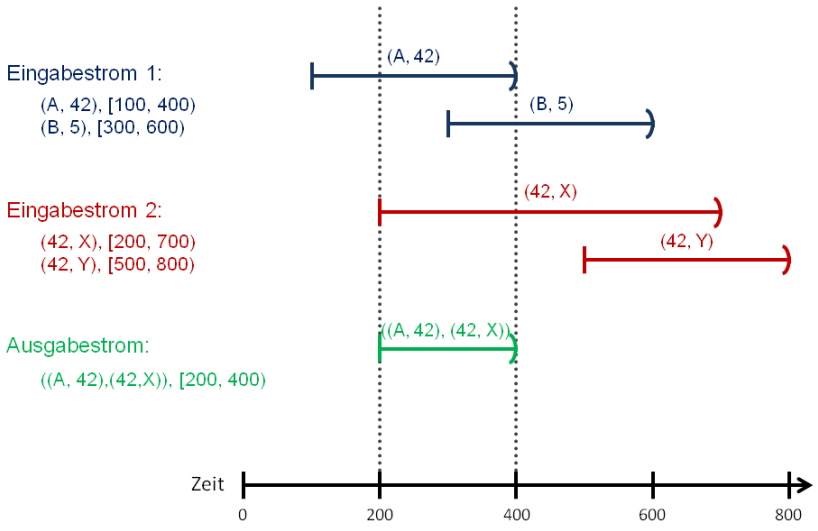


Abbildung 8.1.: Beispiel zum Verbund physischer Datenströme

über alle möglichen Kombinationen (e_1, i_1) und (e_2, i_2), wobei jede dieser Kombinationen mit dem Produkt aus den Vielfachheiten der Eingabestromelemente eingeht. Dies entspricht genau der Zahl aller möglichen Paare aus den entsprechenden einzelnen Stromelementen. Die oben beschriebene notwendige Bedingung für die Erzeugung eines Ergebnisstromelementes ist also auch hinreichend.

Abbildung 8.1 zeigt beispielhaft den physischen Verbund zweier Datenströme mit dem Verbundprädikat $P((t_1, n_1), (n_2, t_2)) := (n_1 = n_2)$ und Ergebnisfunktion $r((t_1, n_1), (n_2, t_2)) := ((t_1, n_1), (n_2, t_2))$.

Das Stromelement mit Wert $(B, 5)$ des ersten Datenstromes findet auf Grund der Zahl 5, die ansonsten nicht vorkommt, keinen Verbundpartner. Das Stromelement mit Wert $(A, 42)$ passt hingegen zu den Werten der beiden Stromelemente des zweiten Eingabestromes. Allerdings sind die Gültigkeitsintervalle $[100, 400)$ und $[500, 800)$ disjunkt. Somit entsteht nur ein Ergebnis, dessen Gültigkeitsintervall $[200, 400)$ sich als Schnitt der beteiligten Eingabegültigkeitsintervalle $[100, 400)$ und $[200, 700)$ ergibt.

Zusammenhang mit Verbund über Multimengen

Betrachtet man nur die in den physischen Datenströmen enthaltenen Elementmengen, so kann man den Verbund über physischen Datenströmen auch wieder mittels des Verbundes über Multimengen ausdrücken. Durch Anordnung der Ergebniselemente zu einem physischen Datenstrom mit Hilfe der Funktion $\bar{\omega}$ erhält man dann wieder das Ergebnis der Verbundoperation über physischen Datenströmen.

Lemma 8.4. *Die Verbundoperation auf physischen Datenströmen $\bowtie_p$ kann mittels der Verbundoperation auf Multimengen $\bowtie$ berechnet werden.*

Beweis. Seien $\mathcal{T}_1, \mathcal{T}_2$ und $\mathcal{T}$ Typen, P ein Prädikat auf $\Omega_{\mathcal{T}_1} \times \Omega_{\mathcal{T}_2}$, $r : \Omega_{\mathcal{T}_1} \times \Omega_{\mathcal{T}_2} \to \Omega_{\mathcal{T}}$ eine Ergebnisfunktion sowie S_1^p und S_2^p physische Datenströme vom Typ $\mathcal{T}_1$ beziehungsweise $\mathcal{T}_2$.

Dann sind $A := \Omega_{\mathcal{T}_1} \times I_{\mathbb{T}}$, $B := \Omega_{\mathcal{T}_2} \times I_{\mathbb{T}}$ und $C := \Omega_{\mathcal{T}} \times I_{\mathbb{T}}$ Mengen und $R : A \to \mathbb{N}$ mit $R((e_1, i_1)) := \bar{\omega}(S_1)(e_1, i_1)$ und $S : B \to \mathbb{N}$ mit $S((e_2, i_2)) := \bar{\omega}(S_2)(e_2, i_2)$ Multimengen. Weiterhin ist P' mit $P'((e_1, i_1), (e_2, i_2)) := P(e_1, e_2) \wedge i_1 \cap i_2 \neq \emptyset$ ein Prädikat über $A \times B$ und $r' : A \times B \to C$ mit $r'((e_1, i_1), (e_2, i_2)) := (r(e_1, e_2), i_1 \cap i_2)$ eine Ergebnisfunktion. Damit gilt:

$$
\begin{aligned}
&\bowtie_{P',r'} (R, S) = \\
&\Big(f : C \to \mathbb{N}, f(c) = \sum_{a \in A, b \in B : P'(a,b) \wedge c = r'(a,b)} R(a) \cdot S(b)\Big) = \\
&\Big(f : C \to \mathbb{N}, f(c) = \sum_{\substack{(e_1,i_1) \in \Omega_{\mathcal{T}_1} \times I_{\mathbb{T}}, (e_2,i_2) \in \Omega_{\mathcal{T}_1} \times I_{\mathbb{T}} : \\ P'((e_1,i_1),(e_2,i_2)) \wedge c = r'((e_1,i_1),(e_2,i_2))}} R((e_1, i_1)) \cdot S((e_2, i_2))\Big) = \\
&\Big(f : (\Omega_{\mathcal{T}} \times I_{\mathbb{T}}) \to \mathbb{N}, f((e, i)) = \sum_{\substack{(e_1,i_1) \in \Omega_{\mathcal{T}_1} \times I_{\mathbb{T}}, (e_2,i_2) \in \Omega_{\mathcal{T}_1} \times I_{\mathbb{T}} : \\ P'((e_1,i_1),(e_2,i_2)) \wedge (e,i) = r'((e_1,i_1),(e_2,i_2))}} R((e_1, i_1)) \cdot S((e_2, i_2))\Big) =
\end{aligned}
$$

$$\Big(f : (\Omega_{\mathcal{T}} \times I_{\mathbb{T}}) \to \mathbb{N}, f((e,i)) = \sum_{\substack{e_1 \in \Omega_{\mathcal{T}_1}, e_2 \in \Omega_{\mathcal{T}_2}, i_1 \in I_{\mathbb{T}}, i_2 \in I_{\mathbb{T}}: \\ P(e_1,e_2) \wedge e = r(e_1,e_2) \wedge i = i_1 \cap i_2}} \bar{\omega}(S_1^p)(e_1,i_1) \cdot \bar{\omega}(S_2^p)(e_2,i_2)\Big) =$$

$$\bar{\omega}\Bigg(\omega\Big(f : (\Omega_{\mathcal{T}} \times I_{\mathbb{T}}) \to \mathbb{N}, f((e,i)) = \sum_{\substack{e_1 \in \Omega_{\mathcal{T}_1}, e_2 \in \Omega_{\mathcal{T}_2}, i_1 \in I_{\mathbb{T}}, i_2 \in I_{\mathbb{T}}: \\ P(e_1,e_2) \wedge e = r(e_1,e_2) \wedge i = i_1 \cap i_2}} \bar{\omega}(S_1^p)(e_1,i_1) \cdot \bar{\omega}(S_2^p)(e_2,i_2)\Big)\Bigg) =$$

$$\bar{\omega}(\bowtie^p_{P,r}(S_1^p, S_2^p))$$

□

Bezüglich der Verbundoperation auf Multimengen werden die Gültigkeitsintervalle der Elemente der physischen Datenströme dabei zu einem Teil der Eingabeelemente, und das Verbundprädikat sowie die Ergebnisfunktion werden dahingehend erweitert, dass das Gültigkeitsintervall eines Ergebnisses dem Schnitt der Gültigkeitsintervalle der dabei verbundenen Elemente entsprechen muss. Diese Erkenntnis spielt für die im folgenden Kapitel behandelte Implementierung der Verbundoperation eine wesentliche Rolle.

8.5. Äußerer Verbund

Neben dem allgemeinen Verbund spielt in DBMS auch der äußere Verbund eine wichtige Rolle. Auch dieser kann auf DSMS übertragen werden, wie dieser Abschnitt aufzeigt. Ähnliche Überlegungen gibt es auch für temporale Datenbanken.[SG89, GJSS05]

8.5.1. Motivation

Verbünde bringen Elemente aus zwei (Multi-)Mengen oder Datenströmen zusammen. Über die Ergebnisfunktion gehen die beteiligten Elemente dann in das Ergebnis ein, falls sich ein Element der anderen Eingabe findet, mit dem es joined. Alle anderen Elemente sind im Ergebnis des Verbundes nicht vertreten. Dieser Verlust an in der Eingabe vorhandener Information ist in manchen Anwendungen nicht gewünscht.

8.5.2. Definition

Da der Ausgabestrom einen festen Typ haben muss, muss eine Möglichkeit gefunden werden, Eingabeelemente ohne Partner mit Hilfe der Ergebnisfunktion auf ein Element des Ausgabetyps abzubilden. Dies ist elegant möglich, wenn man als Ergebnisfunktion die Paarbildung verwendet. Um diese mit nur einem Eingabeparameter anwenden zu können, ersetzt man dabei den fehlenden durch ein spezielles Symbol $\perp$, dass nicht Teil der Wertebereiche ist.

Da nicht immer zwangsweise die Informationen beider Eingaben komplett erhalten bleiben sollen, gibt es noch zwei zusätzliche Varianten des äußeren Verbundes, bei denen jeweils nur eine Seite komplett erhalten bleibt. Alle drei Arten des äußeren Verbundes werden in Definition 8.7 bezüglich logischer Datenströme definiert.

Definition 8.7 (Äußere Verbundoperationen auf logischen Datenströmen)**.** Seien $\mathcal{T}_1, \mathcal{T}_2$ und $\mathcal{T}$ Typen mit $\bot \notin \Omega_{\mathcal{T}_1}$, $\bot \notin \Omega_{\mathcal{T}_2}$ sowie $\Omega_{\mathcal{T}} = (\Omega_{\mathcal{T}_1} \cup \{\bot\}) \times (\Omega_{\mathcal{T}_2} \cup \{\bot\})$, P ein Prädikat auf $\Omega_{\mathcal{T}_1} \times \Omega_{\mathcal{T}_2}$ und $r : \Omega_{\mathcal{T}_1} \times \Omega_{\mathcal{T}_2} \to \Omega_{\mathcal{T}}$ eine Ergebnisfunktion mit $r(e_1, e_2) = (e_1, e_2)$. Der äußere Verbund über logischen Datenströmen $=\!\!\bowtie\!\!=^l\colon \mathbb{S}^l_{\mathcal{T}_1} \times \mathbb{S}^l_{\mathcal{T}_2} \to \mathbb{S}^l_{\mathcal{T}}$ ist definiert durch

$$
\begin{aligned}
=\!\!\bowtie\!\!=^l_{P,r}(S_1, S_2) := {} & \bowtie^l_{P,r}(S_1, S_2) \cup \\
& \{((e_1, \bot), t, n) \mid (e_1, t, n) \in S_1 \wedge \nexists (e_2, t, n_2) \in S_2 : P(e_1, e_2)\} \cup \\
& \{((\bot, e_2), t, n) \mid (e_2, t, n) \in S_2 \wedge \nexists (e_1, t, n_1) \in S_1 : P(e_1, e_2)\}
\end{aligned}
$$

Der linke äußere Verbund über logischen Datenströmen $=\!\!\bowtie^l\colon \mathbb{S}^l_{\mathcal{T}_1} \times \mathbb{S}^l_{\mathcal{T}_2} \to \mathbb{S}^l_{\mathcal{T}}$ ist definiert durch

$$
\begin{aligned}
=\!\!\bowtie^l_{P,r}(S_1, S_2) := {} & \bowtie^l_{P,r}(S_1, S_2) \cup \\
& \{((e_1, \bot), t, n) \mid (e_1, t, n) \in S_1 \wedge \nexists (e_2, t, n_2) \in S_2 : P(e_1, e_2)\}
\end{aligned}
$$

Der rechte äußere Verbund über logischen Datenströmen $\bowtie\!\!=^l\colon \mathbb{S}^l_{\mathcal{T}_1} \times \mathbb{S}^l_{\mathcal{T}_2} \to \mathbb{S}^l_{\mathcal{T}}$ ist definiert durch

$$
\begin{aligned}
\bowtie\!\!=^l_{P,r}(S_1, S_2) := {} & \bowtie^l_{P,r}(S_1, S_2) \cup \\
& \{((\bot, e_2), t, n) \mid (e_2, t, n) \in S_2 \wedge \nexists (e_1, t, n_1) \in S_1 : P(e_1, e_2)\}
\end{aligned}
$$

Diese äußeren Verbünde sind im Gegensatz zu den Halb- und Anti-Verbünden kein Spezialfall des allgemeinen Verbundes auf Datenströmen. Dies ist leicht daran ersichtlich, dass in den äußeren Verbünden Ausgabeelemente zu Zeitpunkten gültig sein können, zu denen nur einer der Eingabeströme über gültige Elemente verfügt. Dies ist beim allgemeinen Verbund nicht der Fall.

Beim Übergang zu physischen Datenströmen bleiben jeweils die aus dem Verbund $\bowtie^l_{P,r}(S_1, S_2)$ resultierenden Datenstromelemente erhalten. Es kommen jedoch solche hinzu, die neben einem Wert e als andere Komponente des Paares $\bot$ enthalten. Deren Gültigkeitsintervalle müssen jeweils diejenigen Zeitpunkte überdecken, zu denen e an keinem Verbund teilgenommen hat. Die Vielfachheit bleibt dabei erhalten, so dass für jedes Eingabestromelement, in dem e enthalten ist, für die genannten Zeiträume auch ein Ausgabestromelement generiert werden muss.

9. Implementierung

Da die Entwicklung von DSMS sich an der von DBMS orientiert und dementsprechend die Definition der Semantik der Verbundoperation auf Basis derer für Multimengen beziehungsweise der der erweiterten relationalen Algebra erfolgte, bietet es sich an, auch bei der Entwicklung von Algorithmen, die den Verbund durchführen, die vielfältigen in DMBS bewährten Techniken einfließen zu lassen.

In diesem Kapitel werden daher zunächst in 9.1 klassische Verbundtechniken für Multimengen diskutiert. In 9.2 wird dann mit der SweepArea eine für die sortierbasierte Verbundberechnung über Multimengen bewährte Datenstruktur vorgestellt. Aufbauend darauf werden in 9.3 konkrete Verbundalgorithmen für Datenströme vorgestellt. In 9.4 wird dann noch diskutiert, wie sich auch das Konzept des äußeren Verbundes auf die Datenstromverarbeitung übertragen lässt.

9.1. Verbundtechniken für Multimengen

Dieser Abschnitt gibt einen kurzen Überblick über die wichtigsten Verbundtechniken für Multimengen und beschreibt die wesentlichen Ideen, die die Entwicklung von weitergehenden Verbundoperationen beeinflusst haben. Eine breitere Übersicht über Verbundoperationen für Datenbanken gibt 17.2.

9.1.1. Problemanalyse

Die Verbundoperation sucht alle Paare gebildet aus einem Element der einen und der anderen Eingabemenge, die das Verbundprädikat erfüllen, und wendet dann die Ergebnisfunktion darauf an. Während die Anwendung der Ergebnisfunktion kaum Optimierungspotential bietet, da diese in der Praxis zumeist eher einfacher Natur ist, kommt dem Verbundprädikat die entscheidende Rolle beim Design der Verbundalgorithmen zu. Für ein allgemeines Verbundprädikat bleibt nichts anderes übrig, als alle möglichen Paare zu bilden und gegen das Prädikat zu testen. Bei n Einzelelementen in der linken und m in der rechten Eingabemultimenge werden dazu $O(n \cdot m)$ Paare gebildet und getestet. Ziel bei der Entwicklung von Verbundalgorithmen ist daher immer, unter diese obere Schranke für die Komplexität zu kommen. Liegt der relative Anteil der Paare, die das Verbundprädikat erfüllen, die so genannte *Selektivität* des Verbundes, jedoch bei 1, so kann die Komplexität nicht unter diese Schranke gedrückt werden, da $n \cdot m$ Ergebnisse produziert werden müssen. Bei der Angabe der Komplexität eines Verbundalgorithmus wird daher auch die Zahl der Ergebnisse r berücksichtigt.

Neben dem Verbundprädikat können auch spezielle Eigenschaften der Eingabemengen beziehungsweise der sie liefernden Operatoren Einfluss auf Design und Komplexität von

Verbundalgorithmen haben. Insbesondere die Fälle, dass eine oder beide Eingaben einer Ordnung folgend oder über Zugriffsstrukturen gelesen werden können, spielen hierbei eine wichtige Rolle.

9.1.2. Klassische Verbundtechniken

Die klassischen Verbundalgorithmen wurden für DBMS entwickelt und gehen davon aus, dass die Eingaben über eine ONC-Schnittstelle oder gegebenenfalls über zusätzliche Zugriffsstrukturen gelesen werden können.

Nested-Loops-Join

Der *Nested-Loops-Join* bildet mit Hilfe der namensgebenden verschachtelten Schleifen alle möglichen Paare und testet diese mit dem Verbundprädikat. Die äußere Eingabe (mit n Einzelelementen) wird dabei einmal, die innere mit m Einzelelementen n-mal gelesen. Der Nachteil des Nested-Loops-Joins ist die mit $O(n \cdot m)$ schlechte Komplexität, sein entscheidender Vorteil ist die Tatsache, dass er unabhängig von Verbundprädikat und Zusatzeigenschaften der Eingaben immer anwendbar ist. Daher wird er immer dann eingesetzt, wenn die Voraussetzungen für den Einsatz eines effizienteren Verfahrens nicht gegeben sind. Zudem lohnt sich sein Einsatz bei hohen Selektivitäten, da in diesem Fall ohnehin fast alle Paare erzeugt werden müssen.

Index-Join

Beim *Index-Join* wird die innere Schleife des Nested-Loops-Joins, die zu einem Element der äußeren Schleife alle Elemente der inneren Eingabe betrachtet, durch einen Zugriff auf eine Indexstruktur ersetzt, die die innere Eingabe indiziert. Die Existenz eines solchen Index ist dabei keine Voraussetzung für den Einsatz des Verfahrens, denn dieser kann bei Bedarf als erster Schritt des Verfahrens aufgebaut werden. Wichtig ist vielmehr, dass der Index es bezüglich des Verbundprädikates ermöglicht, zu einem gegebenen Element e eine echte Teilmenge seiner Elemente zu liefern und zu garantieren, dass alle anderen dieses nicht mit e erfüllen. Würde immer der gesamte Inhalt geliefert, erhielte man eine Variante des Nested-Loops-Joins.

Den wichtigsten Spezialfall stellt dabei der sogenannten Streu-Verbund (Hash-Join) dar, bei dem eine Hash-Tabelle den Index bildet. Diese sind zwar extrem effizient, unterstützen allerdings auch nur ein sehr begrenztes Spektrum von Anfragen, nämlich im Allgemeinen nur die Punktanfrage mit Hilfe eines Schlüssels. Damit lassen sich jedoch zwei extrem wichtige Typen von relationalen Verbundprädikaten unterstützen, nämlich die des Gleichverbunds (Equi-Joins) und des natürlichen Verbundes (Natural-Joins). Dazu wird ein Index benötigt, der die Elemente über den Attributen zugreifbar macht, auf denen Gleichheit gefordert ist. Mit den entsprechenden Werten aus einem Element der äußeren Eingabe kann der Index dann angefragt werden.

Sort-Merge-Join

Die Anwendung des *Sort-Merge-Join*s[BE77] setzt die Existenz einer geeigneten globalen Ordnung auf den Elementen beider Eingaben voraus. Diese werden dann zunächst separat bezüglich dieser Ordnung sortiert (Sort), soweit sie nicht bereits so zugreifbar sind. Dann werden alle Elemente in der Reihenfolge der globalen Ordnung verarbeitet, indem als nächstes zu verarbeitendes Element jeweils das Minimum der Minima der noch nicht verarbeiteten Elemente der Eingaben ausgewählt wird (Merge).

Für beide Eingaben wird jeweils eine Teilmenge der Elemente verwaltet, die aus den jeweils zuletzt verarbeiteten Elementen besteht. Bei der Verarbeitung eines Elementes wird dieses dann gegen die Teilmenge der anderen Eingabe mit dem Verbundprädikat getestet. Der Sort-Merge-Join arbeitet dann effizient, wenn es gelingt, die auf beiden Seiten verwalteten Teilmengen durch frühzeitiges Entfernen von Elementen möglichst klein zu halten und trotzdem zu garantieren, dass ein einmal entferntes Element mit keinem der später noch eintreffenden Elemente der Gegenseite das Verbundprädikat erfüllen kann. Als Kriterium für das Entfernen und diese Garantie dient dabei das Verbundprädikat in Verbindung mit der Ordnung. Als Struktur für das Verwalten der Teilmengen können dabei wie in 9.2.3 erläutert die in 9.2 vorgestellten SweepAreas zum Einsatz kommen.

9.1.3. Verbundtechniken mit frühen Ergebnissen

Beim Design klassischer Verbundalgorithmen war das Optimierungskriterium immer die Minimierung der Gesamtlaufzeit. Dabei haben viele der entwickelten Algorithmen die Eigenschaft, zunächst umfangreiche Vorarbeiten zu leisten und am Ende in schneller Folge alle Ergebnisse zu liefern. Für die Entwicklung von Verbundoperationen mit frühen Ergebnissen gab es daher mehrere grundlegende Motivationen.

Zum einen sollten Benutzern von lokalen, aufwendigen Verbünden mit langer Laufzeit schon frühzeitig erste Ergebnisse und Hochrechnungen über Eigenschaften des Gesamtergebnisses bereitgestellt werden. Die Nutzer haben dadurch insbesondere die Option, die Berechnung des Verbundes vorzeitig abzubrechen, wenn ihnen die bereits erhaltenen Ergebnisse oder die Hochrechnungen bereits genügen.

Zum anderen gewannen verteilte Umgebungen an Bedeutung, in denen einerseits die Eingaben nicht beliebig schnell gelesen werden können, was eine bessere Verzahnung des Lesens der Eingaben mit der Ergebnisproduktion sinnvoll macht, und in denen es andererseits sinnvoll ist, Ergebnisse bereits frühzeitig an andere Rechner weiterzuleiten, damit diese bereits damit weiterarbeiten können. Im Unterschied zur Datenstromverarbeitung wurde bei der Entwicklung der folgenden Algorithmen aber noch prinzipiell von endlichen Eingabemengen ausgegangen. Die grundlegenden Verbundtechniken mit frühen Ergebnissen werden an dieser Stelle vorgestellt; eine umfassendere Übersicht gibt Abschnitt 17.2.2.

Ripple-Join

Die Idee aller Varianten des *Ripple-Join*s[HH99] ist es, die beiden Phasen des Index-Joins, nämlich Aufbau mit allen Elementen der einen und danach Anfrage mit allen Elementen

der anderen Eingabe, miteinander zu verschmelzen. Dazu wird der Index iterativ jeweils mit einem Teil der Eingabe erweitert und dann mit einem Teil der anderen angefragt. Dadurch würden jedoch diejenigen Ergebnisse verloren gehen, die ein anfragendes Element mit einem erst nach der Anfrage in den Index eingefügten Element bilden würde. Deshalb bauen Ripple-Joins aus den anfragenden Elementen ebenfalls einen Index auf, den sie mit denjenigen Elementen anfragen, die in den anderen Index eingefügt werden. Insgesamt entsteht ein symmetrisches Verfahren, bei dem Elemente bei ihrer Verarbeitung in den zur eigenen Eingabe gehörigen Index eingefügt werden und den Index der anderen Eingabe befragen.

Beim Ripple-Join obliegt die Kontrolle darüber, in welcher Reihenfolge die Eingabeelemente verarbeitet werden, noch dem Algorithmus. Bei abwechselnd einem Element je Eingabe spricht man von einem *Square-Ripple-Join*, bei wahlfrei je einem Element von einem *Rectangular-Ripple-Join*, bei abwechselnd blockweiser Verarbeitung von einem *Block-Ripple-Join* und bei wahlfreier blockweiser Verarbeitung vom *Rectangular Block-Ripple-Join*. Der übliche Fall, dass als Indices Hashtabellen Verwendung finden, ist auch unter dem Namen *Hash-Ripple-Join*[HH99] bekannt.

Um die Idee des Ripple-Joins für beliebige Verbundprädikate anwendbar zu machen, kann man als Indexstruktur eine Liste verwenden, die bei jeder Anfrage komplett durchlaufen wird. Man erhält eine symmetrische Variante des Nested-Loops-Joins, die frühe Ergebnisse liefert.

X- und M-Join

Der *X-Join*[UF00] setzt die Idee des Hash-Ripple-Join für den Fall fort, dass große Eingabemengen gejoint werden sollen, die über eine langsame Verbindung geliefert werden. Belegen die Hashtabellen den gesamten Hauptspeicher, so werden Partitionen auf den Externspeicher ausgelagert. Die durch das langsame Eintreffen der Daten verfügbaren Ressourcen werden zwischenzeitlich genutzt, um ausgelagerte Teilpartitionen vom Externspeicher wieder einzulesen und miteinander und mit dem Hauptspeicherteil der Partitionen zu verbinden. Nach dem vollständigen Eintreffen der Eingabemengen werden schließlich die noch fehlenden Ergebnisse durch sukzessives Laden und Joinen der Partitionen berechnet. Die Vermeidung von Duplikaten erfolgt mit Hilfe von Zeitstempeln, mit denen die Zeitpunkte von Auslagern und Wiedereinlesen protokolliert werden, wodurch nachvollziehbar ist, welche Eingabeelemente bereits vorher gejoint wurden. Der *M-Join*[VNB03] setzt die Idee des binären X-Joins für mehrdimensionale Joins um.

Progressive-Merge-Join

Der *Progressive-Merge-Join*[DSTW02, DSTW03] (*PMJ*) überträgt die Idee der Produktion früher Ergebnisse auf den Sort-Merge-Join, indem er die Sortierphase mit der Merge-Phase verwebt. Zu diesem Zweck wird der Sortiervorgang mit Hilfe des (externen) Merge-Sort vorgenommen, wobei die Eingaben nicht nacheinander, sondern wechselseitig teilweise sortiert werden. Über den dabei entstehenden sortierten Teilfolgen wird dann bereits der Verbund berechnet. Auf Grund seiner Bedeutung für diese Arbeit wird der PMJ in 11 noch ausführlicher diskutiert.

9.2. SweepAreas

Bevor die konkreten Algorithmen zur Berechnung des Verbundes über physischen Datenströmen vorgestellt werden, wird in diesem Abschnitt mit der so genannten *SweepArea* noch eine Datenstruktur vorgestellt, die in PIPES in verschiedenen Varianten sowohl für die Berechnung von Verbünden, als auch für verschiedenste andere Operator-Implementierungen eine zentrale Rolle spielt.[Krä07] Die prinzipielle Idee für diese Datenstruktur geht auf einen Algorithmus zum Schneiden geometrischer Figuren[NP82] zurück, der mit einer so genannten *SweepLine* arbeitet. Bereits in [DSTW02] wurde die Idee auf die Verarbeitung von Verbünden angewendet.

9.2.1. Motivation

Die Aufgaben einer SweepArea kann man sich gut an der Berechnung eines *räumlichen Verbund*es (*Spatial-Join*), bei dem zwei Mengen von achsenparallelen Rechtecken bezüglich des Verbundprädikates nichtleerer Schnitt miteinander verbunden werden, verdeutlichen, der mit Hilfe des Sort-Merge-Joins berechnet wird.

Projiziert man das Problem zunächst auf eine räumliche Achse, so erhält man das Problem des Intervall-Verbundes. Als Sortierkriterium kann man dabei die Intervallanfänge benutzen. Erreicht die global sortierte Verarbeitung nun ein Intervall i, so hat sie vorher schon alle Intervalle der anderen Eingabemenge erreicht, die vor diesem beginnen. Alle diejenigen, die nicht vor i auch wieder enden, erfüllen mit i das Verbundprädikat. Alle anderen jedoch erfüllen es nicht und werden auch mit keinem weiteren Element verbunden werden, da diese ja auf Grund der globalen Ordnung noch später beginnen. Sie können also aus der verwalteten Teilmenge entfernt werden, wodurch die Anwendung des Sort-Merge-Joins ihre Effizienz bezieht. Die dabei nicht entfernten Elemente erfüllen jetzt mit i alle das Verbundprädikat.

Soll eine SweepArea die Verwaltung der Teilmenge erledigen, so muss sie also drei Eigenschaften haben: Man muss Elemente einfügen können, man muss möglichst effizient alle Elemente entfernen können, die fortan nicht mehr benötigt werden, und man muss alle verbliebenen Elemente aufzählen können.

Betrachtet man nun wieder das Problem des räumlichen Verbundes, so kann man die Rechtecke völlig analog bezüglich des Intervalls auf einer der Achsen verarbeiten. Schneiden sich diese Intervalle zweier Rechtecke nicht, so können sich auch die Rechtecke nicht schneiden. Das Prinzip des Entfernens aus der SweepArea bleibt also gleich. Jedoch geht die Eigenschaft verloren, dass alle dabei verbleibenden Elemente mit dem gerade zu verarbeitenden das Verbundprädikat erfüllen, denn dazu müssen sich die Rechtecke auch bezüglich der anderen Achse schneiden. Zu einem gegebenen Element, in diesem Falle einem Rechteck, sollte die SweepArea also auch noch effizient diejenigen Elemente liefern können, die damit das Verbundprädikat erfüllen.

9.2.2. Spezifikation

Da der abstrakte Datentyp SweepArea für die Verfahren in dieser Arbeit eine zentrale Rolle spielt, wird er in Datenstruktur 1 noch näher spezifiziert. Aus Gründen der

Übersichtlichkeit bleiben die Typen der Elemente dabei unberücksichtigt.

Datenstruktur 1: SweepArea

```
1  Prädikat P_query;
2  Prädikat P_remove;
3  SweepArea(Prädikat p_query, Prädikat p_remove) {
4     Legt eine neue SweepArea mit den übergebenen Prädikaten an;
5  }
6  Prozedur insert(Element e) {
7     Fügt e in die SweepArea ein;
8  }
9  Iterator iterator() {
10    Liefert einen Iterator über alle Elemente in der SweepArea;
11 }
12 Iterator query(Element s, Index j) {
13    Liefert einen Iterator über alle Elemente t in der SweepArea, für die P_query(t, s) gilt;
14 }
15 Prozedur remove(Element s, Index j) {
16    Löscht alle Elemente t aus der SweepArea, für die P_remove(t, s) gilt;
17 }
18 int size() {
19    Liefert die Zahl der Elemente in der SweepArea;
20 }
```

9.2.3. Anwendung

An dieser Stelle wird nun zunächst erläutert, wie das Konzept der SweepArea für die Verbundberechnung über Multimengen eingesetzt werden kann. Dort wird es insbesondere in sortierbasierten Verfahren eingesetzt.

Sort-Merge-Join

Bezeichnung	Daten	$P(u, v)$
Kreuzprodukt	beliebig	wahr
Theta-Verbund	beliebig	θ
Gleichverbund	Attribut A	$u.A = v.A$
Natürlicher Verbund	Attibute $A_1, \ldots, A_n$	$\forall i : u.A_i = v.A_i$
Band-Verbund	Attribut A	$\lvert u.A - s.A \rvert < \epsilon$
Intervall-Verbund	Intervall $i = [min, max]$	$u.i \cap v.i \neq \emptyset$
Temporaler Verbund	Intervall $i = [min, max)$	$u.i \cap v.i \neq \emptyset$
Räumlicher Verbund	Rechteck $R = ([x_{min}, x_{max}], [y_{min}, y_{max}])$	$u.R \cap v.R \neq \emptyset$

Tabelle 9.1.: Wichtige Spezialfälle der Verbundoperation

Tabelle 9.1 listet diverse wichtige Spezialfälle der Verbundoperation über Multimengen

auf. Darin sind leicht vereinfachend die Anforderungen an die Eingabemengen angegeben; dabei sind die dort aufgeführten Fälle und insbesondere die Verbundprädikate P alle symmetrisch.

Möchte man zu deren Berechnung nun den Sort-Merge-Join unter Verwendung von SweepAreas als Datenstrukturen für die in Verarbeitung befindlichen Teilmengen bezüglich eines Verbundprädikates P_{query} anwenden, so müssen lediglich drei Parameter festgelegt werden: Die globale Ordnung, nach der die Eingaben sortiert sein beziehungsweise werden sollen, sowie die Anfrage- und Entfernungsprädikate für die SweepAreas.

Algorithmus 3 zeigt den Sort-Merge-Join für symmetrische Prädikate unter Verwendung von SweepAreas. Nach der anfänglichen Sortierung wird in der Schleife jeweils das als nächstes zu verarbeitende minimale Element e ausgewählt. Die drei wesentlichen Verarbeitungsschritte Einfügen, Entfernen und Anfragen sind in Algorithmus 2 zusammengefasst.

Algorithmus 2: VerarbeiteElement

Input : SweepArea S_1, SweepArea S_2, Element e, Index j
Output : Elementemenge E

```
1 k ← 3 − j;
2 S_j.insert(e);
3 S_k.remove(e, j);
4 E ← S_k.query(e, j);
```

Algorithmus 3: Sort-Merge-Join

Input : Elementemenge R_1, Elementemenge R_2, Ordnung ω, Prädikat P_{query_1}, Prädikat P_{remove_1}, Prädikat P_{query_2}, Prädikat P_{remove_2}, Ergebnisfunktion r
Output : Elementemenge E

```
1  R̂_1 ← Sortiere_ω R_1;
2  R̂_2 ← Sortiere_ω R_2;
3  SweepArea S_1 ← SweepArea(P_query1, P_remove1);
4  SweepArea S_2 ← SweepArea(P_query2, P_remove2);
5  while R̂ ≠ ∅ ∨ Ŝ ≠ ∅ do
6      if min(R̂_1) < min(R̂_2) then
7          i ← 1;
8      else i ← 2;
9      e ← min(R̂_i);
10     R̂_i ← R̂_i \ e;
11     Z ← VerarbeiteElement(S_1, S_2, e, i);
12     foreach z ∈ Z do
13         if i < 2 then
14             E ← E ∪ r(e, z);
15         else E ← E ∪ r(z, e);
```

Die globale Ordnung muss dabei derart gewählt werden, dass Elemente, die miteinander das Verbundprädikat erfüllen, nicht zu weit voneinander entfernt stehen können. Dadurch ist es dann möglich, die Entfernungsprädikate P_{remove} so zu wählen, dass die SweepAreas nicht zu sehr anwachsen und insbesondere immer in den Hauptspeicher passen. Dabei dürfen jedoch keine Elemente verworfen werden, die noch am Verbund bezüglich des Prädikates P_{join} teilnehmen können; es muss also gelten:

$$u < v \wedge P_{remove_1}(u, v) \Rightarrow \forall w > v : \neg P_{join}(u, w) \tag{9.1}$$

$$u < v \wedge P_{remove_2}(u, v) \Rightarrow \forall w > v : \neg P_{join}(w, u) \tag{9.2}$$

Als Anfrageprädikat kann im Allgemeinen unmittelbar das Verbundprädikat P_{join} gewählt werden, da dieses ja gerade angibt, welche Paare sich für das Ergebnis qualifizieren. Für die zweite SweepArea müssen in asymmetrischen Fällen lediglich die Eingabeparameter des Prädikates vertauscht werden. Allerdings kann unter Beachtung der Tatsache, dass durch Anwendung von remove schon Elemente entfernt wurden, die sich nicht mehr für den Verbund qualifizieren können, das Anfrageprädikat noch optimiert werden. Beim Anwendung von $P_{query_i}(u, v)$ in der Anfrage ist nämlich bereits $\neg P_{remove_i}(u, v)$ garantiert. Zudem liefert die global geordnete Verarbeitung $u \leq v$, da u in die SweepArea eingefügt werden musste, bevor es von v angefragt werden kann. Bezüglich P_{query} ist für die Korrektheit des Algorithmus daher hinreichend, wenn gilt:

$$u < v \wedge \neg P_{remove_1}(u, v) \wedge P_{query_1}(u, v) \Rightarrow P_{join}(u, v) \tag{9.3}$$

$$u < v \wedge \neg P_{remove_2}(u, v) \wedge P_{query_2}(u, v) \Rightarrow P_{join}(v, u) \tag{9.4}$$

In [DSTW02] findet sich eine Tabelle mit den passenden Ordnungen und Prädikaten für wichtige Verbünde. Tabelle 9.2 erweitert diese und zeigt für alle Verbünde aus Tabelle 9.1 auf, wie das Anfrageprädikat der SweepArea vom Verbundprädikat abweichend gewählt werden kann.

Bezeichnung	Ordnung	$P_{query}(u, v)$	$P_{remove}(u, v)$
Kreuzprodukt	beliebig	wahr	falsch
Theta-Verbund	beliebig	θ	falsch
Gleichverbund	A	wahr	$u.A \neq v.A$
Natürlicher Verbund	$A_1, \ldots, A_n$	wahr	$\exists i : u.A_1 \neq v.A_1$
Band-Verbund	A	wahr	$\lvert u.A - s.A \rvert \geq \epsilon$
Intervall-Verbund	$i.min$	wahr	$u.max < v.min$
Temporaler Verbund	$i.min$	wahr	$u.max \leq v.min$
Räumlicher Verbund	x_{min}	$u.[y_{min}, y_{max}] \cap v.[y_{min}, y_{max}] \neq \emptyset$	$u.x_{max} < v.x_{min}$

Tabelle 9.2.: Parameter des Sort-Merge-Joins für wichtige Spezialfälle der Verbundoperation

Progressive-Merge-Join

Der Einsatz der SweepAreas im Progressive-Merge-Join erfolgt im Wesentlichen analog zu der im Sort-Merge-Join und wird in 11 näher erläutert.

Ripple-Join

Auch der Ripple-Join verwaltet für beide Eingaben jeweils eine Datenstruktur, in die Elemente in der Reihenfolge ihrer Verarbeitung eingefügt werden. Dazu kann ebenfalls eine SweepArea eingesetzt werden, wobei deren Funktionalität zum Löschen von Elementen ungenutzt bleibt.

9.2.4. Implementierung

Eine optimale Implementierung einer SweepArea sollte eine integrierte Datenstruktur sein, die Einfügen, Anfragen und Löschen effizient unterstützt. Angesichts der Vielzahl an möglichen Kombinationen an Anfrage- und Löschprädikaten bietet es sich aber an, nach Möglichkeiten zu suchen, beide Probleme getrennt zu behandeln und aus den Lösungen die benötigten SweepAreas zu generieren. Dazu werden zwei Datenstrukturen gewählt, von denen die eine Anfragen bezüglich des Lösch-, und die andere bezüglich des Anfrageprädikates effizient unterstützt.

Datenstruktur 4: Anfragestruktur

```
1  Prädikat P_query;
2  Anfragestruktur(Prädikat p_query) {
3     Legt eine neue Anfragestruktur mit dem übergebenen Anfrageprädikat an;
4  }
5  Prozedur insert(Element e) {
6     Fügt e in die Anfragestruktur ein;
7  }
8  Iterator iterator() {
9     Liefert einen Iterator über alle Elemente in der Anfragestruktur;
10 }
11 Iterator query(Element s, Index j) {
12    Liefert einen Iterator über alle Elemente t in der Anfragestruktur, für die P_query(t, s) gilt;
13 }
14 Prozedur remove(Elementemenge S) {
15    Löscht alle Elemente in S aus der Anfragestruktur;
16 }
17 int size() {
18    Liefert die Zahl der Elemente in der Anfragestruktur;
19 }
```

Datenstruktur 4 zeigt, wie eine Anfragestruktur beschaffen sein muss. Sie hängt nur von dem Anfrageprädikat P_{query} ab und unterstützt in query Anfragen bezüglich dieses Prädikates. Zusätzlich muss sie in der Lage sein, eine vorgegebene Menge von Elementen zu löschen.

Datenstruktur 5: Löschstruktur

```
1  Prädikat P_remove;
2  Löschstruktur(Prädikat p_remove) {
3     Legt eine neue Löschstruktur mit dem übergebenen Prädikat an;
4  }
5  Prozedur insert(Element e) {
6     Fügt e in die Löschstruktur ein;
7  }
8  Elementemenge remove(Element s, Index j) {
9     Löscht alle Elemente t aus der Löschstruktur, für die P_remove(t, s) gilt, und gibt diese als
      Menge zurück;
10 }
```

Datenstruktur 5 zeigt die Anforderungen an eine Löschstruktur. Sie hängt nur vom Entfernungsprädikat P_{remove} ab und unterstützt diesbezüglich effizientes Löschen, wobei sie die jeweils gelöschte Elementemenge zurückliefert.

Datenstruktur 6 zeigt, wie eine Einfügestruktur und eine Löschstruktur zu einer zweistufigen SweepArea kombiniert werden. Einzufügende Elemente werden dabei in beide Strukturen eingefügt. Da keine Kopien der Elemente, sondern Referenzen Verwendung finden, entsteht dabei für das Speichern in der zweiten Struktur kein Mehrbedarf an Speicher, der über den der reinen Struktur hinausginge. Anfragen an die Daten oder deren Zahl werden jeweils direkt von der Anfragestruktur bearbeitet. Die Struktur L wird nur beim Löschen benutzt. Die in ihr gelöschten Elemente werden dann auch in der Anfragestruktur gelöscht. Die Prädikate kommen dabei in Datenstruktur 6 nicht explizit vor; sie gehen implizit über die verwendeten Datenstrukturen ein, die gerade die Eigenschaft haben, diese Prädikate als Parameter zu haben.

Die verschiedenen Implementierungen der abstrakten Datentypen für Anfrage- und Löschstrukturen sollten dabei möglichst gut auf die jeweiligen Prädikate zugeschnitten sein, damit die SweepArea insgesamt effizient arbeitet. Zunächst problematisch erscheint dabei die Anforderung an die Anfragestruktur, vorgegebene Elemente effizient zu löschen. Diese relativiert sich jedoch sehr stark, da die meisten Verbundprädikate, aus denen die Anfrageprädikate abgeleitet werden, darauf abzielen, ähnliche Elemente miteinander zu verbinden. Da ein Element sich selber natürlich ähnlich ist, unterstützen Anfragestrukturen in den meisten Fällen problemlos das effiziente Löschen vorgegebener Elemente. In 9.3.5 wird dennoch auch die Möglichkeit diskutiert, aus Effizienzgründen statt eines zweistufigen Aufbaus die Datenstruktur SweepArea direkt zu implementieren.

9.3. Verbundalgorithmen für Datenströme

Auf Basis der im vorstehenden Abschnitt vorgestellten Algorithmen und Datenstrukturen kann nun der Algorithmus für die Verbundoperation über physischen Datenströmen motiviert und vorgestellt werden.

Datenstruktur 6: Zweistufige SweepArea

```
1  Anfragestruktur A;
2  Löschstruktur L;
3  SweepArea(Anfragestruktur Ā, Löschstruktur L̄) {
4     A ← Ā;
5     L ← L̄;
6  }
7  Prozedur insert(Element e) {
8     A.insert(e);
9     L.insert(e);
10 }
11 Iterator iterator() {
12    return A.iterator();
13 }
14 Iterator query(Element s, Index j) {
15    return A.query(s, j);
16 }
17 Prozedur remove(Element s, Index j) {
18    A.remove(L.remove(s, j));
19 }
20 int size() {
21    return A.size();
22 }
```

9.3.1. Motivation

Der Rectangular-Ripple-Join verarbeitet zwei Eingabemengen und kann dabei die Eingabeelemente prinzipiell in beliebiger Reihenfolge lesen. Als Reihenfolge kann man bei der Übertragung auf Eingabedatenströme dabei also auch diejenige wählen, in der die Eingabeelemente eintreffen. Dabei tritt jedoch das Problem auf, dass die Elemente bis zur vollständigen Berechnung des Verbundes gehalten werden, was nicht mit potentiell unendlichen Datenströmen vereinbar ist. Um den Algorithmus also anwenden zu können, müssen mit Hilfe der Gültigkeitsintervalle Elemente wieder aus dem Status entfernt werden. Dabei kommt das Konzept ins Spiel, beim Ripple-Join SweepAreas als Statusstrukturen einzusetzen. Deren Entfernen-Funktionalität kann mit einem geeigneten Entfernungsprädikat dafür sorgen, dass die SweepAreas nicht beliebig anwachsen. Dabei muss jedoch garantiert werden, dass keine Ergebnisse verloren gehen.

Einen alternativen Zugang bietet die Tatsache, dass physische Datenströme sortiert nach Startzeitstempeln eintreffen. Daher könnte man den Merge-Schritt des Sort-Merge-Joins verwenden, wobei man den temporalen Join analog zu einem möglichen Vorgehen für temporale Datenbanken[GJSS05] als Intervall-Verbund, also eindimensionalen räumlichen Verbund, ansieht. Dabei muss jedoch die Verarbeitung nach dem Datenstromparadigma von den eintreffenden Elementen gesteuert werden. Es ergibt sich jedoch das Problem, dass die Eingabeelemente in den Datenströmen zwar jeweils lokal geordnet bezüglich der Ordnung auf Startzeitstempeln eintreffen, aber nicht global sortiert. Der Sort-Merge-Join geht jedoch von einer der globalen Ordnung folgenden Verarbeitung aus.

9.3.2. Erster Algorithmus

Ein erster Algorithmus ergibt sich also, wenn man dafür sorgt, dass die lokal sortiert eintreffenden Ströme synchronisiert werden, und dann die Elemente global sortiert mit dem Merge-Schritt des Sort-Merge-Joins verarbeitet. Die Sortierung auf den Startzeitstempeln ist als Parameter des Verfahrens festgelegt; es müssen also noch die Anfrage- und Entfernungsprädikate für die SweepAreas festgelegt werden. Das Entfernungsprädikat wird dabei analog zum Intervall-Verbund aus Tabelle 9.2 so gewählt, dass ein Stromelement $(e, [t_s, t_e))$ entfernt werden kann, wenn aus der anderen Eingabe ein Element $(\hat{e}, [\hat{t}_s, \hat{t}_e))$ mit $t_e \leq \hat{t}_s$ eintrifft. Wie in 8.4.2 erläutert müssen sich nämlich die Gültigkeitsintervalle zweier Stromelemente schneiden, damit diese joinen können. Analog gilt dies natürlich für Intervalle beim Intervall-Verbund. Da nach dem Element mit Startzeitstempel $\hat{t}_s$ keines mehr verarbeitet wird, dessen Startzeitstempel davor liegt, kann auch keines mehr davon die Schnittbedingung erfüllen. Somit ist die in 9.1 formulierte Bedingung erfüllt, wenn man die Löschprädikate P_{remove} so wählt:

$$\forall i \in \{1, 2\} : P_{remove_i}((e, [t_s, t_e)), (\hat{e}, [\hat{t}_s, \hat{t}_e))) := t_e \leq \hat{t}_s \tag{9.5}$$

Tabelle 9.2 weist für den Intervall-Verbund als Anfrageprädikat für die SweepAreas *wahr* aus. Im Falle des Strom-Verbundes kommt neben der Intervallschnittbedingung jedoch noch das eigentliche Verbundprädikat P hinzu, das zusätzlich erfüllt sein muss:

$$P_{join}((e, [t_s, t_e)), (\hat{e}, [\hat{t}_s, \hat{t}_e))) = P(e, \hat{e}) \wedge [t_s, t_e) \cap [\hat{t}_s, \hat{t}_e) \neq \emptyset$$

Die in 9.3 formulierte Bedingung an die Anfrageprädikate P_{query} wird dabei dann gerade durch die Wahl des Verbundprädikates beziehungsweise des Verbundprädikates mit umgekehrten Parametern als Anfrageprädikate für die SweepAreas erfüllt:

$$P_{query_1}((e, [t_s, t_e)), (\hat{e}, [\hat{t}_s, \hat{t}_e))) := P(e, \hat{e}) \tag{9.6}$$

$$P_{query_2}((e, [t_s, t_e)), (\hat{e}, [\hat{t}_s, \hat{t}_e))) := P(\hat{e}, e) \tag{9.7}$$

Algorithmus 7 berechnet nun den Verbund zweier physischer Datenströme. Zunächst werden die Prädikate für die SweepAreas und die Ergebnisfunktion mit Intervallschnitt definiert. Die Foreach-Schleife stellt den aktiven Teil des Algorithmus dar, da hier das Eintreffen der Elemente aus den Datenströmen die Verarbeitungsreihenfolge vorgibt. Eintreffende Elemente gelangen nun zunächst in eine Warteschlage, in der sie so lange verbleiben, bis in der anderen Warteschlange keine Elemente mit kleinerem Startzeitstempel mehr warten oder erwartet werden. Erst dann werden sie wie im Sort-Merge-Join mit Hilfe von *VerarbeiteElement* verarbeitet. Auf die erhaltenen Anfrageergebnisse wird die Ergebnisfunktion angewendet, und die Ergebnisse werden dem Ausgabestrom S_r zugeführt.

Algorithmus 7: Erster Verbundalgorithmus für Datenströme

Input : physischer Eingabestrom S_1, physischer Eingabestrom S_2, Verbundprädikat P, Ergebnisfunktion r
Output : physischer Ausgabestrom S_r

```
1  P_query1((e,[t_s,t_e)),(ê,[t̂_s,t̂_e))) := P(e,ê) ;
2  P_query2((e,[t_s,t_e)),(ê,[t̂_s,t̂_e))) := P(ê,e) ;
3  P_remove((e,[t_s,t_e)),(ê,[t̂_s,t̂_e))) := t_e ≤ t̂_s;
4  r'((e_1,i_1),(e_2,i_2)) := (r(e_1,e_2), i_1 ∩ i_2);
5  SweepArea A_1 ← SweepArea(P_query1, P_remove);
6  SweepArea A_2 ← SweepArea(P_query2, P_remove);
7  FiFoQueue Q_1 ← ∅; FiFoQueue Q_2 ← ∅;
8  foreach s := (e,[t_s,t_e)) ↩ S_j do
9      Q_j.enQueue(s);
10     while ¬Q_1.isEmpty() ∧ ¬Q_2.isEmpty() do
11         (e_1,[t_s1,t_e1)) ← Q_1.peek();
12         (e_2,[t_s2,t_e2)) ← Q_2.peek();
13         if t_s1 ≤ t_s2 then
14             k ← 1 ;
15         else k ← 2 ;
16         x ← Q_k.deQueue();
17         Z ← VerarbeiteElement(A_1, A_2, x, k) ;
18         foreach z ∈ Z do
19             if k < 2 then
20                 r'(x,z) ↪ S_r;
21             else r'(z,x) ↪ S_r;
22 if ¬Q_1.isEmpty() then
23     k ← 1;
24 else k ← 2 ;
25 while ¬Q_k.isEmpty() do
26     x ← Q_k.deQueue();
27     Z ← VerarbeiteElement(A_1, A_2, x, k) ;
28     foreach z ∈ Z do
29         if k < 2 then
30             r'(x,z) ↪ S_r;
31         else r'(z,x) ↪ S_r;
```

Korrektheit

Lemma 9.1. *Algorithmus 7 berechnet die Verbundoperation auf physischen Datenströmen* $\bowtie_p$.

Beweis. Zu zeigen ist, dass der Algorithmus den korrekten Ausgabestrom erzeugt. Dazu genügt es zu zeigen, dass der Ausgabestrom die richtige Menge an Ausgabeelementen enthält und dass er der Ordnungsbedingung auf den Startzeitstempeln genügt.

Eintreffende Stromelemente werden zunächst in Warteschlangen abgelegt. Sie werden erst durch den im Algorithmus verwendeten Merge-Schritt des Sort-Merge-Joins verarbeitet, wenn für die jeweils andere Eingabe nur noch Stromelemente mit mindestens gleich hohem Startzeitstempel eintreffen können. Der Merge-Schritt verarbeitet die Elemente also in der Reihenfolge der globalen Ordnung auf Startzeitstempeln.

Die Wahl des Verbundprädikates P_{join} sowie der Ergebnisfunktion r' erfolgt analog zur Konstruktion im Beweis zu Lemma 8.4. Die Korrektheit der Ergebnismenge folgt daher aus der Korrektheit der Ergebnismenge des Merge-Schrittes des Sort-Merge-Joins. Die SweepAreas werden dabei entsprechend den Bedingungen 9.5 und 9.6 gewählt.

Das Gültigkeitsintervall eines Stromelementes des Ergebnisstromes wird von der Ergebnisfunktion r' als Schnitt der Gültigkeitsintervalle der beteiligten Eingabestromelemente gesetzt. Da die Stromelemente global sortiert verarbeitet werden, ist der Startzeitstempel des Schnittes immer gleich dem Startzeitstempel des später verarbeiteten Stromelementes. Daher werden die Ausgabeelemente sortiert nach Startzeitstempeln in den Ausgabestrom eingefügt. □

Diskussion

Ein essentielles Konzept der Datenstromverarbeitung ist die datengetriebene Verarbeitung. Daten sollen direkt bei deren Auftreten im System so weit wie möglich weiterverarbeitet werden. Auch wenn dieses Konzept, wie auch in dieser Arbeit noch aufgezeigt wird, nicht immer strikt verwirklicht werden kann, sollte doch nicht unnötig davon abgewichen werden. Der vorstehende erste Algorithmus zur Berechnung des Verbundes verarbeitet jedoch nicht jedes Element unmittelbar, sondern fügt es nur in eine Warteschlange ein. Danach werden dann eventuell Elemente verarbeitet. Je nachdem, wie die Eingabeströme eintreffen, kann es auf diese Art vorkommen, dass viele Elemente eintreffen und jeweils nur in eine Warteschlange gelangen, bevor dann beim Eintreffen eines weiteren Stromelements viele verarbeitet werden müssen. Diese schubweise Verarbeitung ist nicht im Sinne einer gleichmäßigen Auslastung des Systems. Gerade die Verbundoperation, die zu den wichtigsten und zugleich teuersten Operationen in der Stromverarbeitung zählt, sollte jedoch möglichst kontinuierlich arbeiten.

9.3.3. Zweiter Algorithmus

Der Lösungsansatz zur Verwendung des Merge-Schrittes des Sort-Merge-Joins für den ersten Algorithmus war es, aus der vorhandenen lokalen Ordnung die globale zu erzeugen. Für den zweiten Algorithmus ist der Ansatz nun, zu untersuchen, wo die globale Ordnung in diesen Merge-Schritt eingeht und ob dieser derart modifiziert werden kann, dass eine lokal sortierte Verarbeitung ausreicht. Die Verwandtschaft zu den Ripple-Joins zeigt auf,

dass prinzipiell jede Reihenfolge funktionieren kann. Für die Effektivität ist jedoch wichtig, die SweepAreas klein zu halten, um deren Speicherverbrauch und die Anfragekosten so klein wie möglich zu halten.

Die globale Verarbeitungsreihenfolge geht in die Bedingungen an die Entfernungs- (9.1) und Anfrageprädikate (9.3) für die SweepAreas jeweils mit $u < v$ ein. Demnach ergeben sich für nur lokale Verarbeitungsordnung folgende Bedingungen:

$$P_{remove_1}(u, v) \Rightarrow \forall w > v : \neg P_{join}(u, w) \tag{9.8}$$
$$P_{remove_2}(u, v) \Rightarrow \forall w > v : \neg P_{join}(w, u) \tag{9.9}$$

$$\neg P_{remove_1}(u, v) \wedge P_{query_1}(u, v) \Rightarrow P_{join}(u, v) \tag{9.10}$$
$$\neg P_{remove_2}(u, v) \wedge P_{query_2}(u, v) \Rightarrow P_{join}(v, u) \tag{9.11}$$

Die Entfernungsprädikate aus 9.5 erfüllen bereits diese stärkere Bedingung, da für Elemente $(e, [t_s, t_e))$ und $(\hat{e}, [\hat{t}_s, \hat{t}_e))$ aus $t_e \leq \hat{t}_s$ auch unmittelbar $t_s < \hat{t}_s$, also $(e, [t_s, t_e)) \leq (\hat{e}, [\hat{t}_s, \hat{t}_e))$ folgt.

Die Anfrageprädikate 9.6 hingegen nutzen die Bedingung $u < v$ aus, um den Schnitt der Gültigkeitsintervalle zu garantieren. Von den beiden prinzipiellen Möglichkeiten dafür, dass sich zwei nichtleere Intervalle nicht schneiden, wird jetzt jedoch nur noch eine ausgeschlossen. Das anfragende Intervall kann nicht komplett hinter einem angefragten in einer SweepArea liegen, da dieses dann zuvor durch Gültigkeit des Entfernenprädikats gelöscht worden wäre, weil das Löschen in Algorithmus 2 unter anderem zu diesem Zweck vor dem Anfragen erfolgt. Das anfragende Intervall kann jedoch nun komplett vor dem angefragten liegen, da es in diesem Fall nicht mehr vor diesem verarbeitet werden muss. Daher wird das Anfrageprädikat derart ergänzt, dass dieser Fall behandelt wird.

$$P_{query_1}((e, [t_s, t_e)), (\hat{e}, [\hat{t}_s, \hat{t}_e))) := P(e, \hat{e}) \wedge \hat{t}_e > t_s \tag{9.12}$$
$$P_{query_2}((e, [t_s, t_e)), (\hat{e}, [\hat{t}_s, \hat{t}_e))) := P(\hat{e}, e) \wedge \hat{t}_e > t_s \tag{9.13}$$

Natürlich ist es auch möglich, als Zusatzbedingung die beteiligten Intervalle auf nichtleeren Schnitt zu testen, was dann der Überprüfung der kompletten Verbundbedingung für physische Datenstromelemente bezüglich P entspricht. Damit schließt sich der Kreis zur in der Motivation in 9.3.1 formulierten Idee, einen Ripple-Join zu verwenden, wobei man die Verarbeitungsreihenfolge durch das Eintreffen der Elemente vorgeben lässt. Die dort erläuterte Zusatzanforderung, Stromelemente wieder aus den Statusstrukturen zu entfernen, wenn diese in der Folge auf jeden Fall nicht mehr benötigt werden, wird gerade durch das Entfernen mittels der in 9.5 angegebenen Entfernungsprädikate erfüllt.

Allerdings ergibt sich eine neue Problematik, die beim ersten Algorithmus noch nicht auftrat. Durch die nicht mehr global sortierte Verarbeitung der Elemente überträgt sich keine Ordnung mehr auf die Startzeitstempel der produzierten Ergebnisse. Dies gilt insbesondere, da nun der Startzeitstempel eines anfragenden Stromelementes nicht mehr

immer auch der Startzeitstempel der dabei produzierten Ergebnisse wird. Daher dürfen die Ergebnisse nicht bei ihrer Produktion unmittelbar in den Ausgabestrom geschrieben werden, sondern müssen vielmehr zuvor noch geordnet werden.

Da eine Sortierung einer potentiell unendlich großen Ergebnismenge ohnehin ausscheidet und die Ergebnisse frühestmöglich in den Ausgabedatenstrom gelangen sollen, müssen fortlaufend sortierte Teilfolgen weitergeleitet werden, für deren Elemente feststeht, dass keine kleineren mehr auftreten können. Der Startzeitstempel eines anfragenden Elementes überträgt sich zwar nicht mehr immer auf dabei entstehende Ergebnisse, stellt aber eine untere Schranke für Startzeitstempel von Ergebnissen dar, die durch Anfragen von seiner Eingabe entstehen. Dies gilt, da die Elemente der Eingabeströme lokal nach Startzeitstempeln sortiert anfragen und der Anfang des Schnittes der Gültigkeitsintervalle nach vorn durch das Maximum der beteiligten Startzeitstempel gegeben ist. Bildet man das Minimum über die beiden lokalen unteren Schranken, also über die Startzeitstempel der von den beiden Eingaben jeweils zuletzt verarbeiteten Stromelemente, so erhält man eine globale untere Schranke für die Startzeitstempel zukünftig zu produzierender Ergebnisse. Alle Ergebnisse mit kleinerem Startzeitstempel können also in den Ausgabestrom geleitet werden.

Die so berechnete Schranke ist nicht immer optimal. Unter Berücksichtigung der minimalen Startzeitstempel der Elemente in den beiden SweepAreas kann in manchen Fällen eine bessere bestimmt werden. Die Möglichkeit, dadurch gelegentlich Ergebnisse etwas früher weiterleiten zu können, rechtfertigt jedoch im Allgemeinen nicht den dafür benötigten Mehraufwand. Denn die Speicherung der Elemente bezüglich ihrer Zeitstempel erfolgt wie in 9.3.5 erläutert normalerweise anhand der End- und nicht der Startzeitstempel.

Algorithmus 8 setzt vorstehendes Konzept um. Jedes eintreffende Stromelement wird unmittelbar mit Hilfe von *VerarbeiteElement* verarbeitet. Die daraus mit r' erzeugten Ergebnisse werden jedoch nicht direkt in den Ausgabestrom eingefügt, sondern zunächst in eine Prioritätswarteschlange Q eingefügt, die ihre Elemente aufsteigend sortiert nach Startzeitstempeln liefert. Nach der Verarbeitung eines Elements werden dann alle Ergebnisse aus der Warteschlange in den Ausgabedatenstrom eingefügt, die vor der zuvor motivierten unteren Schranke für zukünftig zu produzierende Ergebnisse liegen. Nachdem alle Eingabeelemente verarbeitet wurden, werden noch die in der Warteschlange verbliebenen Ergebnisse geliefert.

Korrektheit

Lemma 9.2. *Algorithmus 8 berechnet die Verbundoperation auf physischen Datenströmen* $\bowtie_p$.

Beweis. Zu zeigen ist, dass der Algorithmus den korrekten Ausgabestrom erzeugt. Dazu genügt es zu zeigen, dass der Ausgabestrom die richtige Menge an Ausgabeelementen enthält und dass er der Ordnungsbedingung auf den Startzeitstempeln genügt.

Der Beweis der richtigen Ausgabemenge verläuft analog zum Korrektheitsbeweis des Sort-Merge-Joins. Ohne das Entfernen aus den SweepAreas und die Anfragebedingung an die Zeitintervalle würden alle möglichen Paare von Elementen gebildet und mit dem Verbundprädikat getestet, da jeweils das später eintreffende Element das andere anfragt.

Algorithmus 8: Zweiter Verbundalgorithmus für Datenströme

Input : physischer Eingabestrom S_1, physischer Eingabestrom S_2, Verbundprädikat P, Ergebnisfunktion r
Output : physischer Ausgabestrom S_r

```
1  P_query1((e,[t_s,t_e)),(ê,[t̂_s,t̂_e))) := P(e,ê) ∧ t̂_e > t_s ;
2  P_query2((e,[t_s,t_e)),(ê,[t̂_s,t̂_e))) := P(ê,e) ∧ t̂_e > t_s ;
3  P_remove((e,[t_s,t_e)),(ê,[t̂_s,t̂_e))) := t_e ≤ t̂_s;
4  r'((e_1,i_1),(e_2,i_2)) := (r(e_1,e_2), i_1 ∩ i_2);
5  SweepArea A_1 ← SweepArea(P_query1, P_remove);
6  SweepArea A_2 ← SweepArea(P_query2, P_remove);
7  PriorityQueue_ts Q ← ∅;
8  m_1 ← −∞; m_2 ← −∞;
9  foreach s := (e,[t_s,t_e)) ↩ S_j do
10     Z ← VerarbeiteElement(A_1, A_2, (e,[t_s,t_e)), j) ;
11     foreach z ∈ Z do
12         if j < 2 then
13             Q.enQueue(r'((e,[t_s,t_e),z)));
14         else Q.enQueue(r'(z,(e,[t_s,t_e))));
15     m_j ← t_s;
16     while ¬Q.isEmpty() do
17         (e',[t'_s,t'_e)) ← Q.peek();
18         if t'_s ≤ min(m_1,m_2) then
19             Q.deQueue() ↪ S_r ;
20         else break ;
21 while ¬Q.isEmpty() do
22     Q.deQueue() ↪ S_r ;
```

Die Konstruktion am Anfang des Abschnittes zeigt, dass auch unter Anwendung des Entfernens und der zusätzlichen Anfragebedingung keine Ergebnisse verloren gehen.

Die Ausgabe der Elemente geordnet nach Startzeitstempeln wird durch die Ausgabewarteschlange Q erreicht. Ein Element wird erst daraus entfernt, wenn kein in der Ordnung vor ihm stehendes mehr eingefügt werden kann. Daher bewirkt das Entfernen des jeweiligen Minimums die Ausgabe in einer korrekten Reihenfolge. □

Diskussion

Mit diesem Algorithmus werden die Eingabestromelemente jetzt unmittelbar verarbeitet und Ergebnisse produziert, eine datengetriebene Verarbeitung findet also statt. Allerdings können die Ergebnisse nicht immer direkt weitergegeben werden, da die Ordnungsbedingung für den Ausgabestrom eingehalten werden muss. Je nach Eingabeverhalten können so schubweise Ergebnisse weitergeleitet werden, was bei stromabwärts gelegenen Operatoren zu Problemen führen kann.

Das generelle Problem ist, dass der Verbund zwei voneinander unabhängig eintreffende lokal nach Startzeitstempeln sortierte Ströme zu einem derart sortierten Ausgabestrom verbinden muss. Da Elemente überschneidende Gültigkeitsintervalle besitzen müssen, um überhaupt joinen zu können, müssen Elemente mit ähnlichen Startzeitstempeln zusammen gebracht werden. Eilt nun einer der Eingabeströme dem anderen zeitlich voraus, so müssen dessen Stromelemente aufbewahrt werden, bis die potentiellen Partner im anderen Strom eintreffen. Dieses Problem löst der erste Algorithmus, indem er die Aufbewahrung vor die Verarbeitung setzt, während der zweite möglichst früh verarbeitet, um die Synchronisation danach vorzunehmen.

9.3.4. Hybrider Algorithmus

Sind beide Eingabeströme annähernd synchron, so verhalten sich beide vorgestellten Algorithmen fast gleich. Die Frage, welchen man verwenden sollte, stellt sich also im Falle zeitlich gegeneinander verschobener Datenströme. Dabei müssen beide Algorithmen den früher eintreffenden Strom teilweise zwischenspeichern, der erste in einer Warteschlange, der zweite in einer SweepArea. Der erste Algorithmus hat den Nachteil, dass er für die Verbundberechnung dann beide SweepAreas wie bei unverschobenen Strömen benutzt. Der zweite hingegen kann in der zum später eintreffenden Eingabestrom gehörigen SweepArea Elemente frühzeitig entfernen und spart damit Speicher. Dafür sind die Anfragen an die SweepAreas aufwendiger, da diese auch Elemente enthalten, die bezüglich ihres Gültigkeitsintervalls mit anfragenden Elementen nicht verbunden werden können. Zudem muss der zweite Algorithmus die Ergebnisse zwischenspeichern, was im Falle eines expansiven Verbundes, also im Falle einer hohen Selektivität, erheblichen Speicherbedarf erzeugt.

Neben dem Grad der Verschiebung der Ströme zueinander kann sich während der langen Laufzeit eines Stromverbundes insbesondere dessen Selektivität ändern. Somit kann sich auch die Wahl des geeigneteren Algorithmus immer wieder ändern. Aus diesem Grund bietet es sich an, einen hybriden Algorithmus bereitzustellen, der die Ideen der beiden Verfahren in sich vereinigt. Dabei macht es keinen Sinn, Puffer für Ein- und

Ausgabeströme dauerhaft gleichzeitig zu benutzen, da der Ausgabepuffer bei synchroner Verarbeitung mittels Pufferung an den Eingaben gar nicht benötigt wird. Vielmehr kann man aber zur Laufzeit zwischen den beiden Verfahren wechseln.

Der Wechsel kann jedoch nicht erfolgen, ohne dabei den aktuellen Zustand des Verbundes zu berücksichtigen, und benötigt teilweise sogar eine Übergangszeit. Der Wechsel kann daher nicht per Befehl direkt ausgeführt werden, sondern muss angefordert werden und wird dann ausgeführt. Dabei können dann übergangsweise Ein- und Ausgangspuffer parallel verwendet werden. Es soll aber möglich sein, jederzeit einen Wechsel anzufordern, unabhängig davon, ob bereits einer im Gange ist.

Als Parameter für die SweepAreas werden die Prädikate aus dem zweiten Algorithmus verwendet, weil damit beide Algorithmen funktionieren, da der Unterschied nur darin liegt, dass der zweite Algorithmus ein verschärftes Anfrageprädikat nutzt, das aber keine korrekten Verbundergebnisse abweist.

Wechsel

Der Wechsel hin zu einem Ausgabepuffer kann jederzeit unmittelbar erfolgen. Dazu wird zunächst die Ausgabewarteschlange, soweit noch nicht vorhanden, installiert. Dann werden unmittelbar alle Elemente in den Eingabeschlangen verarbeitet. Global muss dabei keine Verarbeitungsordnung beachtet werden, denn die Synchronisation übernimmt nun die neu installierte Ausgabewarteschlange. Eine der Eingabewarteschlangen sollte ohnehin leer sein.

Umgekehrt aber kann ein Wechsel von der Verwendung eines Ausgabepuffers hin zu Eingabepuffern nicht unmittelbar erfolgen. Die Eingabepuffer können zwar unmittelbar installiert werden, der Ausgabepuffer aber nicht sofort entfernt werden. Dies liegt zum einen darin begründet, dass der Ausgabepuffer bereits Ergebnisse enthalten kann, zwischen die sich im Ausgabestrom noch andere Ergebnisse einreihen müssen, die aus Eingabeelementen gejoint werden, die noch gar nicht im Verbund eingetroffen sind. Zum anderen werden durch den Zustand der SweepAreas wie im zweiten Algorithmus die Ergebnisse nicht unmittelbar sortiert generiert.

Es kann aber ein logischer Zeitpunkt gefunden werden, zu dem die Ausgabewarteschlange geleert und entfernt werden kann. Als solcher eignet sich der maximale Startzeitstempel t_m, mit dem vor der Installation der Eingabepuffer bereits ein Eingabeelement verarbeitet wurde. Zum einen ist das der maximale Startzeitstempel, mit dem bis zu diesem Zeitpunkt Ergebnisse produziert werden konnten. Zum anderen durchlaufen alle Elemente mit einem größeren Startzeitstempel die Eingabepuffer und werden daher nach der globalen Ordnung verarbeitet, was zu geordneten Ergebnissen führt; und für Ergebnissen mit früheren Startzeitstempeln müssen beide joinenden Eingabeelemente frühere Startzeitstempel haben. Erreicht das kleinste Ergebnis im Ausgabepuffer also diesen Zeitpunkt t_m, so kann er geleert und entfernt werden.

Da ein Wechsel jederzeit angefordert werden können soll, bleibt zu klären, was dann während eines laufenden Wechsels von Aus- auf Eingabepuffer passiert. Wird nochmals ein Wechsel hin zu Eingabepuffern angefordert, muss natürlich nichts geändert werden. Wird ein Wechsel hin zu einem Ausgabepuffer angefordert, so kann dies wieder unmittelbar geschehen. Der Ausgabepuffer existiert in diesem Falle noch, es müssen also

nur die Eingabepuffer verarbeitet und entfernt werden.

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="3">aktuelle Position</td></tr>
<tr><td>nur an
Ausgabe</td><td>an Eingabe
und Ausgabe</td><td>nur an
Eingabe</td></tr>
<tr><td rowspan="2">gewünschte
Position</td><td>Ausgabe</td><td></td><td colspan="2">- Installation Ausgabepuffer
- Verarbeitung Eingabepuffer
- Entfernung Eingabepuffer
- neuer Status: nur Ausgabe</td></tr>
<tr><td>Eingabe</td><td>- Bestimmung t_m
- Installation
Eingabepuffer
- neuer Status:
Ein- und Ausgabe</td><td></td><td></td></tr>
</table>

Tabelle 9.3.: Wechsel der Pufferplatzierung beim hybriden Algorithmus

Tabelle 9.3 zeigt zusammenfassend die Übergänge zwischen den verschiedenen Zuständen bei Anforderungen eines Wechsels. Angegeben sind jeweils die unmittelbar durchzuführenden Schritte. Zusätzlich erfolgt wie vorstehend beschrieben durch Ausgeben und Entfernen des Ausgabepuffers der Wechsel zum Status *nur an Eingabe*, sobald im Ausgabepuffer t_m erreicht wird.

9.3.5. Wahl der SweepAreas

Die bisherige Diskussion der Algorithmen bezog sich auf deren grundlegende Struktur und Korrektheit, die wiederum wesentlich von der Einhaltung der Spezifikation durch die verwendeten SweepAreas abhängt. Daher wird nun diskutiert, welche SweepAreas dazu geeignet sind und zugleich eine effiziente Durchführung des Verbundes gewährleisten.

Für den ersten Algorithmus sind die Anfrageprädikate 9.6 alleine von den Werten der Stromelemente abhängig. Für die beiden anderen Algorithmen kommt bei den Anfrageprädikaten 9.12 zwar eine Bedingung an das Gültigkeitsintervall hinzu, die aber in den meisten praktischen Fällen für einen Großteil der Elemente in den SweepAreas erfüllt ist. Im Gegensatz dazu sind die Entfernungsprädikate 9.5 bei allen Algorithmen allein von den Gültigkeitsintervallen abhängig.

Daher bietet sich die Verwendung der in 9.2.4 motivierten zweistufigen Struktur einer SweepArea an. Die Anfragestruktur ist dabei auf die Elemente zu beziehen, die Löschstruktur auf die zugehörigen Gültigkeitsintervalle.

Zeitbezogene Löschstrukturen

Die Entfernungsprädikate 9.5 der Algorithmen, die durch die Löschstruktur effizient unterstützt werden sollen, entfernen ein Element, wenn dessen Endzeitstempel kleiner oder gleich dem Startzeitstempel des Elementes ist, das das Löschen anstößt. Betrachtet man die Abfolge der Löschoperationen, so werden diese ja alle von Elementen desselben

Stromes ausgelöst, die wiederum nach Startzeitstempeln geordnet verarbeitet werden. Die Löschstruktur muss also für eine monoton steigende Folge von Zeitpunkten jeweils die Elemente entfernen, deren Endzeitstempel nicht größer als der jeweilige Zeitpunkt sind.

Um dieses Löschmuster zu unterstützen, kann man die Elemente sortiert nach Endzeitstempeln löschen, und zwar jeweils alle bis zu dem auslösenden Startzeitstempel. Zwischen den Aufrufen von *remove* können aber auch noch neue Elemente eingefügt werden. Gesucht wird also eine dynamische Struktur, die effizientes Einfügen und effizientes Löschen des Elementes mit kleinstem Endzeitstempel unterstützt. Dazu bietet sich ein Min-Heap bezüglich der Endzeitstempel an.

Eine auf einem solchen Heap basierende Löschstruktur eignet sich unabhängig vom Verbundprädikat P für alle vorgestellten Algorithmen für den Verbund von Datenströmen. In 10.2.1 werden noch Varianten diskutiert, die mit anderen Strukturen arbeiten. Da die Operationen auf dem Heap mit logarithmischem Aufwand bezogen auf die Zahl der enthaltenen Elemente erfolgen, wird der in [GÖ05] geäußerten Befürchtung vorgebeugt, eine für das Löschen nach Endzeitstempeln geeignete Struktur erfordere beim Einfügen oder Löschen linearen Aufwand.

Wertbezogene Anfragestrukturen

Die Wahl der Anfragestrukturen hängt nun ganz wesentlich vom Anfrageprädikat P ab. Für die wichtigsten Typen von Verbundprädikaten werden daher an dieser Stelle passende Anfragestrukturen diskutiert.

Allgemeines Prädikat Im allgemeinsten Fall, also wenn über das Prädikat P nichts weiter bekannt ist, bleibt bei einer Anfrage mit einem Stromelement des anderen Stromes nichts anderes übrig, als alle Elemente in der SweepArea damit bezüglich ihres Wertes auf Erfülltsein des Verbundprädikates P zu testen. Im Falle des zweiten und des hybriden Algorithmus muss zusätzlich noch die Bedingung, die den nichtleeren Schnitt der Gültigkeitsintervalle abfragt, getestet werden.

Als Struktur, die Einfügen und das Traversieren aller enthaltener Elemente erlaubt, würde sich schon eine einfache Liste eignen. Das daraus resultierende Verfahren entspricht einer Stromvariante des in 9.1.2 diskutierten Nested-Loops-Joins, bei der für jedes Element des einen Stromes in einer Schleife alle Elemente des anderen Stromes durchlaufen und getestet werden. Auf Basis der Zeitintervalle wird jedoch mit Hilfe der zeitlichen Ordnung und der Löschstrukturen die Zahl der zu testenden Kombinationen wesentlich eingeschränkt.

Der Nachteil an der Wahl einer Liste ist es, dass diese nicht effizient das Löschen der von der Löschstruktur ermittelten Elemente erlaubt. Falls also für die Stromelemente eine geeignete Hashfunktion angegeben werden kann, bietet es sich an, statt einer Liste eine Hashtabelle einzusetzen. Diese ermöglicht ebenfalls effizientes Einfügen und Abfragen aller Elemente, aber auch effizientes Löschen von Elementen. Eine Liste sollte also nur im allgemeinsten Fall zum Einsatz kommen.

Kreuzprodukt Die gleichen Überlegungen gelten auch für den Spezialfall des Kreuzproduktes, dass ebenfalls keine Einschränkung der zu traversierenden Elemente anhand des Verbundprädikates, dass dann *wahr* entspricht, erlaubt.

Gleichverbund Das übliche Verfahren zur Berechnung eines Gleich-Verbundes in DBMS ist der Hash-Join (siehe 9.1.2). Die Elemente mit gleichem Wert auf dem Attribut, für das dies verlangt ist, bilden dabei Äquivalenzklassen, die durch eine geeignete Hashfunktion auf die Hashbuckets verteilt werden. Dadurch können passende Elemente effizient aufgefunden werden.

Dementsprechend kann als Anfragestruktur eine Hashtabelle Anwendung finden, wie auch schon zuvor für das allgemeine Prädikat. Der wesentliche Unterschied liegt nun darin, dass bei Anfragen nicht mehr eine Schleife über alle Elemente in der Hashtabelle läuft, sondern nur noch über das zum anfragenden Element passende Hashbucket. Damit reduziert sich der Anfrageaufwand erheblich.

Analog kann vorgegangen werden, wenn Gleichheit auf mehreren Attributen gefordert ist. Optimalerweise findet eine Hashfunktion Anwendung, die alle Werte auf den Attributen berücksichtigt. Das Verfahren funktioniert jedoch auch, wenn nur eine Teilmenge davon in die Hashfunktion eingeht.

Natürlicher Verbund Der natürliche Verbund ist ein Spezialfall des Gleichverbundes auf mehreren Attributen und wird daher entsprechend behandelt.

Äquivalenzklassenverbünde Die Effizienz des Einsatzes der Hashtabellen rührt aus der Eigenschaft her, die Elemente bezüglich des Joinprädikates in Äquivalenzklassen einteilen zu können. Neben Gleichverbünden gibt es noch einige nicht exotische Verbundprädikate, bei denen das möglich ist, beispielsweise solche der Form $P((x_1, \ldots, x_n), (y_1, \ldots, y_m)) = (f((x_{i_1}, \ldots, x_{i_k})) = (y_{j_1}, \ldots, y_{j_l}))$ für eine Funktion f. Bei diesen kann auf die Elemente der einen Eingabe vor Anwendung der Hashfunktion die Funktion f angewendet werden.

Band-Verbund Beim Band-Verbund joinen zwei Elemente, deren Komponenten sich nur um einen Maximalwert ϵ unterscheiden, das Verbundprädikat hat also etwa die Form $P((x_1, \ldots, x_n), (y_1, \ldots, y_m)) = (|x_k - y_l| \leq \epsilon)$. Zu einem Element $(x_1, \ldots, x_n)$ müssen also Elemente $(y_1, \ldots, y_m)$ in der anderen SweepArea gefunden werden, für die $x_k - \epsilon \leq y_l \leq x_k + \epsilon$ gilt. Dies entspricht einer Bereichsanfrage auf einer Datenstruktur, welche die Elemente der zweiten Eingabe nach den Werten y_l geordnet verwaltet. Neben den Bereichsanfragen müssen wiederum Einfügen und Löschen einzelner Elemente effizient unterstützt werden. Als Hauptspeicherstruktur eignen sich hierfür Suchbäume wie der AVL-Baum[AVL62] oder der Rot-Schwarz-Baum[Bay72].

Intervall-Verbund Beim Intervall-Verbund beinhalten die Elemente der beteiligten Ströme Wert-Intervalle, die sich als Verbundbedingung schneiden müssen, das Verbundprädikat ist also von der Form $P((\ldots, [s_1, e_1], \ldots), (\ldots, [s_2, e_2], \ldots)) = ([s_1, e_1] \cap [s_2, e_2] \neq \emptyset)$.

In diesem Falle müssen die Elemente bezüglich Intervallen in den SweepAreas verwaltet werden, an die dann Bereichsanfragen in Form der Intervalle im anfragenden Element gestellt werden. Als Hauptspeicherstruktur eignet sich hierbei ein dynamischer Intervallbaum[Ede80].

Neben der Schnittbedingung an die Werteintervalle ist dafür, dass zwei physische Stromelemente verbunden werden, immer noch der Schnitt derer Gültigkeitsintervalle notwendig. Insgesamt werden also zwei Stromelemente verbunden, wenn sich deren Intervalle auf zwei unabhängigen Achsen jeweils schneiden. Dies entspricht einem räumlichen Verbund, bei dem sich Rechtecke schneiden müssen. Zur Berechnung eines räumlichen Verbundes über Multimengen eignet sich besonders ein Sort-Merge-Join unter Verwendung einer Sortierung bezüglich der Intervallbeginne der einen Achse und von Intervallbäumen[Ede80] über der anderen Achse in den Statusstrukturen.[APR+98] Somit bestehen bei der Verwendung des ersten Algorithmus zur Berechnung eines Intervall-Verbundes starke Parallelen zum räumlichen Verbund über Multimengen.

Räumlicher Verbund Beim Räumlichen Verbund kommt nun im Vergleich zum Intervall-Verbund eine weitere Achse hinzu, nämlich eine zweite Werte-Achse. Da die Verarbeitung nach wie vor entlang der Zeitachse erfolgt, muss die Anfragestruktur beide Werteachsen berücksichtigen. Hierzu bieten sich Hauptspeicherstrukturen an, die zum Indexieren von Rechtecken geschaffen wurden. Einige gängige Strukturen sind dabei aber nicht unmodifiziert verwendbar, da sie beim Einfügen ihre Struktur verfeinern, diese beim Löschen aber nicht wieder bereinigen können. Da aus den SweepAreas aber auch fortwährend Elemente wieder entfernt werden, darf die Struktur aber auch in diesem Falle nicht entarten. Eine Übersicht der Strukturen findet sich zum Beispiel in [GG98].

Beim räumlichen Verbund komplexerer geometrischer Objekte wie Polygonen werden diese zumeist konservativ durch achsenparallele Rechtecke approximiert, die dann wie zuvor beschrieben behandelt werden. Das eigentliche Verbundprädikat wird dann in einem Verfeinerungsschritt getestet.

Ähnlichkeits-Verbund Beim Ähnlichkeits-Verbund (*Similarity-Join*) gehen punktförmige Elemente den Verbund ein, wenn sich diese nicht zu sehr unterscheiden, wobei der Unterscheidungsgrad beispielsweise mit Hilfe einer Metrik bestimmt wird. In den Anfragestrukturen sind also punktförmige Objekte zu speichern, die Anfragen sind aber Bereichsanfragen, da alle Objekte in deren Umfeld gefunden werden müssen. Passt die Bereichsanfrage nicht exakt auf das Ähnlichkeitsprädikat, so wird wieder eine konservative Annäherung angefragt, und die Ergebnisse werden dann gegen das eigentliche Prädikat getestet.

Analog zu den Überlegungen zum Räumlichen Verbund finden hier Hauptspeicherstrukturen Verwendung, die punktförmige Objekte speichern und Bereichsanfragen unterstützen. Wiederum ist es wichtig, dass die Strukturen nicht durch fortlaufendes Einfügen und Löschen entarten. Eine Übersicht der Strukturen findet sich auch hierfür in [GG98].

Asymmetrische Anfragestrukturen

Die Verbundprädikate, für die bisher die Wahl der Anfragestrukturen diskutiert wurde, waren strukturell symmetrisch. Zwar konnten sich beispielsweise bei einem natürlichen Verbund die Attribute auf beiden Seiten unterscheiden, die für das Verbundprädikat entscheidenden Elemente waren jedoch jeweils identisch. Auch der allgemeine Verbund ist in dem Sinne symmetrisch, dass auf beiden Seiten keine Eigenschaften des Verbundprädikates ausgenutzt werden können. Dementsprechend wurden die SweepArea-Strukturen für beide Eingaben gleich gewählt. Es gibt jedoch auch Verbünde, bei denen eine asymmetrische Wahl günstig ist.[KNV03]

Asymmetrische Verbundprädikate Auch wenn sie nicht unbedingt zu den Standardanfragen gehören, gibt es doch verschiedenste Arten sinnvoller asymmetrischer Verbundprädikate. Beispielsweise kann man einen Strom von Punkten und einen von Rechtecken daraufhin joinen, dass ein Punkt in einem Rechteck enthalten sein muss. In diesem Falle müssen in der ersten SweepArea Punkte gespeichert werden. Angefragt werden jeweils Punkte, die in einem eintreffenden Rechteck enthalten sind, was einer Bereichsanfrage entspricht. Für diese SweepArea eignet sich also eine Anfragestruktur analog zum Ähnlichkeits-Verbund. Umgekehrt müssen in der anderen SweepArea Rechtecke gespeichert werden, und es werden Punktanfragen gestellt. Dafür eignen sich wiederum Hauptspeicherstrukturen zum Abspeichern von Rechtecken, analog zum räumlichen Verbund. Somit werden für die beiden SweepAreas in diesem Falle also zwei verschiedene Anfragestrukturen verwendet, um den Verbund effizient zu unterstützen.

Asymmetrische Verbundcharakteristika Auch im Falle strukturell symmetrischer Verbundprädikate kann es aber sinnvoll sein, die Anfragestrukturen in den SweepAreas asymmetrisch zu wählen. Dafür müssen die beteiligten Datenströme spezielle Charakteristika aufweisen. Führen beispielsweise die Ströme dazu, dass die eine SweepArea im Gegensatz zur anderen immer nur wenige Elemente enthält, so kann es sinnvoller sein, für sie eine einfache Liste zu verwenden, da eine komplexe Indexstruktur unnötigen Aufwand verursachen würde.

Integrierte SweepAreas

Obwohl das zweistufige Design viele Vorteile bringt und die Implementierung erheblich vereinfacht, bringt es natürlich auch Nachteile mit sich. Neben dem üblichen Overhead durch die erhöhte Abstraktion fällt dabei insbesondere ins Gewicht, dass ein in der Löschstruktur zur Löschung ausgewähltes Element in der Anfragestruktur nochmals aufgefunden werden muss, um es auch daraus löschen zu können. Daher kann es sich lohnen, für besonders häufig benötigte Kombinationen aus Anfrage- und Löschstruktur stattdessen eine direkte integrierte Implementierung des Datentyps SweepArea bereitzustellen.

Hash-Anfragestruktur Die mit Abstand wichtigste Kombination bildet dabei diejenige aus einer Heap-basierten Löschstruktur und ein hash–basierten Anfragestruktur. Das

Löschen mit Hilfe eines Heaps eignet sich für variable Längen der Gültigkeitsintervalle, und durch wertbasiertes Hashing werden insbesondere die in der Praxis besonders häufig verwendeten Gleichverbünde, darunter der Spezialfall Natürlicher Verbund, unterstützt. Um eine integrierte Struktur zu erhalten, kann man hier als Hashbuckets doppelt verkettete Listen wählen. Diese haben den Vorteil, dass man Elemente, auf die man eine Referenz hat, mit konstantem Aufwand daraus löschen kann. In den Heap fügt man nun nicht die ursprünglichen Elemente, sondern die Elemente der doppelt verketten Listen ein, wobei man die zeitliche Ordnung nach Endzeitstempeln der Gültigkeitsintervalle entsprechend überträgt. In der so konstruierten Struktur werden Elemente beim Einfügen zunächst in das Hashbucket eingefügt, und das dabei erstellte Listenelement dann direkt in den Heap. Anfragen laufen wie gehabt komplett über die Hashtabelle ab. Beim Löschen wird die Spitze des Heaps entfernt, die dann direkt auch aus ihrer Liste in der Hashtabelle entfernt werden kann. Danach muss nur wie gehabt die Heapbedingung wiederhergestellt werden. Wie gewünscht entfällt also die zusätzliche Suche in der Hashtabelle beim Löschen.

Auf ähnliche Weise kann man auch eine FiFo-Queue als Löschstruktur mit einer hash–basierten Anfragestruktur verweben, indem man wiederum Elemente von als Hashbuckets verwendeten doppelt verketteten Listen zusätzlich in einer als FiFo-Queue verwendeten einfach verketteten Liste aufreiht.

Während diese integrierten Strukturen bezüglich der Verarbeitung Vorteile bieten, soll aber auch nicht unerwähnt bleiben, dass sie im Vergleich zu einfach gehaltenen zweistufigen Strukturen mehr Speicher benötigen können, beispielsweise durch die doppelte statt einfache Verkettung von als Hashbuckets verwendeten Listen.

Anfragestruktur für allgemeines Prädikat Im Falle einer Anfragestruktur für ein allgemeines Prädikat, die ohnehin bei einer Anfrage für jedes enthaltene Element das Verbundprädikat überprüfen muss, kann man auf diese Anfragestruktur ganz verzichten, wenn eine Löschstruktur verwendet wird, die es erlaubt, über alle enthaltenen Elemente zu iterieren. Dies sollte fast immer möglich sein. Die Anfragen iterieren dann einfach über die Löschstruktur und prüfen für die gelieferten Elemente jeweils das Verbundprädikat mit dem anfragenden Element.

Dieses Vorgehen spart nicht nur den zusätzlich für die Speicherstruktur benötigten Speicherplatz ein, sondern insbesondere auch den mit ihr sonst verbundenen Aufwand für Einfügen und Löschen. Wie zuvor diskutiert verhält sich insbesondere der Löschaufwand im Falle der Verwendung einer Liste als Anfragestruktur linear zu deren Größe und sollte daher vermieden werden.

9.4. Äußerer Verbund

Da der äußere Verbund wie in 8.5 erläutert kein Spezialfall des allgemeinen Verbundes ist, werden für ihn eigene Algorithmen benötigt. Da der zugehörige Verbund aber jeweils ein Teilstrom der Ausgabe des äußeren Verbundes ist, bietet es sich an, die Algorithmen für den allgemeinen Verbund so zu modifizieren, dass sie die zusätzlich benötigten Ausgabeelemente produzieren.

Die nachfolgenden Überlegungen beziehen sich dabei auf den äußeren Verbund und können problemlos auch auf den linken und rechten äußeren Verbund übertragen werden.

9.4.1. Motivation

Die Idee beim äußeren Verbund ist es, keine Informationen aus den Eingabeströmen zu verlieren. Dazu könnte man zunächst aus allen Eingabeelementen durch Paarbildung mit $\perp$ Ausgabeelemente bilden, und diese dem normalen Verbundergebnis hinzufügen. Die so gebildeten Ausgabeelemente wären aber auch zu Zeitpunkten gültig, zu denen die darin eingegangenen Eingabeelemente an einem Verbund beteiligt waren. Dies darf aber gerade nicht sein. Daher müssen die Zeitpunkte ermittelt werden, zu denen ein Eingabestromelement an mindestens einem Verbund beteiligt war, und nur für die verbliebenen Zeitpunkte dessen Gültigkeitsintervalls darf es mit $\perp$ gepaart in die Ausgabe weitergeleitet werden.

In den folgenden Abschnitten werden zwei Verfahren vorgestellt, die die Berechnung des äußeren Verbundes erlauben.

9.4.2. Modifikation der Verbundalgorithmen für Datenströme

Erste Verfahren ergeben sich durch Modifikation der in 9.3 vorgestellten Algorithmen zur Berechnung des allgemeinen Verbundes. Diese werden zunächst in einer prinzipiellen Form motiviert. Danach werden Optimierungen vorgestellt, die sowohl Speicherbedarf als auch Berechnungsaufwand stark reduzieren.

Grundlegende Modifikationen

Die folgenden Änderungen an den allgemeinen Verbundalgorithmen ergeben zunächst äußere Verbundalgorithmen:

- Zu jedem Stromelement in den SweepAreas wird zusätzlich eine Liste von paarweise disjunkten Zeitintervallen gespeichert, die zusammen die Zeitpunkte abdecken, für die das Element noch an keinem Verbund teilgenommen hat. Beim Einfügen in die SweepArea wird diese Liste demnach mit einem einzigen Intervall initialisiert, nämlich dem Gültigkeitsintervall des Elementes.

- Erfüllt ein Element mit einem anderen Stromelement (e_2, i_2) das Verbundprädikat, so wird die zu ihm gespeicherte Liste aktualisiert. Alle Intervalle, die in i_2 enthalten sind, werden entfernt. Alle anderen Intervalle werden um ihren Schnitt mit i_2 gekürzt und dabei gegebenenfalls in zwei Intervalle aufgeteilt. Die Menge deckt danach also den alten Bereich ohne i_2 ab. Diese Aktualisierung erfolgt nicht nur bei Anfragen an die SweepArea, sondern auch bei der Anfrage der gegenüberliegenden SweepArea mit dem Element.

- Wird ein Element durch das Entfernungsprädikat aus einer SweepArea entfernt, so wird es an keinem weiteren Verbund mehr teilnehmen. Die Liste der dazu gespeicherten Intervalle gibt also nun die Zeitintervalle an, an denen es insgesamt

nicht am Verbund teilnimmt. Daher wird nun für jedes solche Intervall aus dem Element und $\perp$ ein Paar gebildet und mit dem Intervall als Gültigkeit in die Ergebniswarteschlange eingefügt.

- Die Verarbeitung der je nach Algorithmus gegebenenfalls hinzuzufügenden Ausgabewarteschlange wird dahingehend modifiziert, dass die so zusätzlich entstehenden Ergebnisse korrekt in den Ausgabestrom eingereiht werden. Es darf nun kein Ergebnis mehr weitergeleitet werden, so lange sich in einer der zusätzlich bei den Elementen gespeicherten Intervalllisten noch ein Intervall mit kleinerem Startzeitstempel existiert. Da der minimale Startzeitstempel in allen diesen Mengen nur aufwendig bestimmt werden kann, kann er konservativ durch das Minimum der Startzeitstempel der Elemente in den SweepAreas approximiert werden.

Dieses Verfahren ermittelt somit für jedes Eingabeelement die Zeiträume, für die es an keinem Verbund teilnimmt, und produziert dafür die zusätzlichen äußeren Ergebnisse.

Optimierung

Die Änderungen an der Liste von Intervallen zu einem Stromelement durch die Anfrage der gegenüberliegenden SweepArea erfolgt nur einmal, und zwar direkt bei dessen Verarbeitung. Da keine Aussagen darüber getroffen werden können, in welcher Reihenfolge dort Elemente zugegriffen werden, können Änderungen an der Liste hier prinzipiell überall auftreten. Es könnte also aus Effektivitätsgründen Sinn machen, hier zunächst eine komplexere Struktur, wie zum Beispiel einen Intervallbaum, zu verwenden. Allerdings ist der Fall eher unüblich, dass ein Stromelement mit langer Gültigkeit einen Verbund mit sehr vielen Elementen kurzer Gültigkeit eingeht, was nötig wäre, um die Struktur merklich anwachsen zu lassen. Daher genügt üblicherweise die Verwendung einer einfachen Liste.

Alle weiteren Änderungen an der Liste werden dann durch Anfragen aus der anderen Eingabe ausgelöst. Da diese Eingabe sortiert nach Startzeitstempeln eintrifft, können gar keine Elemente mehr dazukommen, die vordere Einträge in der Liste modifizieren. Alle noch eintreffenden Elemente müssen nämlich mindestens den Startzeitstempel haben, der dem Endzeitstempel des vorletzten Elementes in der Liste entspricht. An diesem Zeitpunkt muss nämlich das Gültigkeitsintervall eines vorher in die gegenüberliegende SweepArea eingefügten Elementes begonnen haben.

Auf Grund dieser Betrachtung genügt es nun, nach dem Anfragen der gegenüberliegenden SweepArea lediglich das Teilintervall mit dem höchsten Startzeitstempel, das nicht an einem Verbund teilgenommen hat, zu speichern. Alle anderen können sofort zusammen mit $\perp$ als äußeres Ergebnis identifiziert und in die Ausgabewarteschlange eingefügt werden. Im Falle der Modifikation des ersten Verbundalgorithmus kann in diesem Schritt ohnehin nur ein Intervall entstehen, da alle Elemente, mit denen ein Verbund eingegangen werden kann, höchstens denselben Startzeitstempel haben wie das gerade eingetroffene.

Durch die nach Startzeitstempeln geordnete Verarbeitung der Elemente kann auch bei Anfragen an die SweepArea immer nur das zeitlich letzte Intervall in der Liste von

Änderungen betroffen sein. Daher ist es während des gesamten Verfahrens hinreichend, statt der Liste nur deren letzten Eintrag zu dem Element der SweepArea zu speichern und mit anderen Einträgen jeweils unmittelbar Ergebnisse zu produzieren und in die Ausgabewarteschlange einzufügen. Überlappt das Gültigkeitsintervall eines neuen Verbundpartners also das gespeicherte Intervall vom Beginn an, so wird es entsprechend verkürzt. Schneidet es erst hinter dem Startzeitstempel, so wird für den nicht schneidenden vorderen Teil ein äußeres Ergebnis produziert. Verbleibt hinter dem Schnitt noch ein Rest, so wird dieser wieder als noch nicht am Verbund beteiligtes Gültigkeitsintervall gespeichert.

Insgesamt muss also zu jedem Element nur das Teilintervall mit dem höchsten Startzeitstempel, das noch an keinem Verbund teilgenommen hat, gespeichert werden. Entspricht dessen Ende nicht dem Endzeitstempel des Elementes, so kann es ebenfalls unmittelbar für einen äußeren Verbund herangezogen werden, da schon ein Verbund mit späterem Startzeitstempel stattgefunden hat. Somit genügt es sogar, statt dem einen Intervall pro Element in den SweepAreas nur jeweils dessen Startzeitstempel zu speichern.

Durch die vorstehenden Optimierungen vereinfacht sich auch die Ermittlung des Zeitpunktes, bis zu dem Ergebnisse in den Ausgabestrom ausgegeben werden dürfen. Dieser wird nun zusätzlich zu den ursprünglichen Limitierungen des zu Grunde liegenden Algorithmus durch das Minimum der zusätzlich an den Elementen der SweepAreas gespeicherten Zeitstempel begrenzt. Diese stellen nämlich die Zeitpunkte dar, an denen noch zu produzierende äußere Ergebnisse beginnen können. Die Optimierungen vermindern also nicht nur den Aufwand und Speicherverbrauch der äußeren Verbundalgorithmen, sondern erlauben auch eine frühere Ausgabe der Ergebnisse.

9.4.3. Subtraktionsmethode

Der Nachteil der Modifikation des Algorithmus ist es, dass nicht nur der Kernalgorithmus, sondern auch die jeweiligen SweepAreas modifiziert werden müssen, um die Stromelemente nicht in zusätzlichen Datenstrukturen redundant speichern zu müssen. Zudem ist die Modifikation nicht allgemein auf andere Verbundalgorithmen übertragbar und müsste daher für Implementierungen anderer Verbundalgorithmen jeweils neu entworfen und programmiert werden. Daher wird an dieser Stelle eine alternative Verfahrensweise vorgestellt, durch die aus einem fast beliebigen Verbundalgorithmus ein Äußerer Verbund konstruiert werden kann.

Bei dem Verfahren entsteht kein direkter Äußerer Verbundoperator, sondern der benutzte allgemeine Verbundoperator wird mit anderen bestehenden Operatoren und zusätzlichen Komponenten kombiniert. Dabei entsteht ein Operatorgraph mit Eingängen und einem Ausgang. Solche Graphen können ihrerseits als virtuelle Operatoren angesehen werden, in diesem Fall als virtueller äußerer Verbundoperator.

Die Idee dabei entspricht dem in 9.4.1 erläuterten Konzept, dass der äußere Verbund drei Teilströme vereinigt. Zum einen den normalen allgemeinen Verbund, zum anderen die Teile der beiden Eingabeströme, die nicht am Verbund teilnehmen, jeweils gepaart mit $\perp$. Für die Vereinigung gibt es den Vereinigungsoperator, für die Paarbildung mit $\perp$ kann jeweils ein Mapping-Operator verwendet werden und von den Eingabeströmen kann man mit Hilfe des Differenzoperators die Teile abziehen, die am Verbund teilnehmen.

Dazu wird allerdings für jede Eingabe ein Strom der am Verbund teilnehmenden Elemente benötigt, bei dem die Gültigkeitsintervalle der Elemente angeben, für welches Intervall das jeweilige Element am Verbund teilgenommen hat. Diese Ströme kann man aber gewinnen, indem man den Ausgabestrom des Verbundes wieder in seine Komponenten aufteilt. Dazu wird dieser mit Hilfe des Mappingoperators auf die jeweils benötigte Komponente projiziert. Aus einem Ergebnis $((e_1, e_2), i)$ des Verbundes wird also ein Element (e_1, i) erzeugt, dass dann von der linken Eingabe abgezogen wird, und ein Element (e_2, i), das von der rechten Eingabe abgezogen wird. Dies ist immer möglich, da die Ergebnisfunktion des hier zur Berechnung des äußeren Verbundes verwendeten Verbundoperators die Paarbildung ist. Abbildung 9.1 zeigt den Operatorgraphen, der insgesamt als virtueller Operator den äußeren Verbund berechnet.

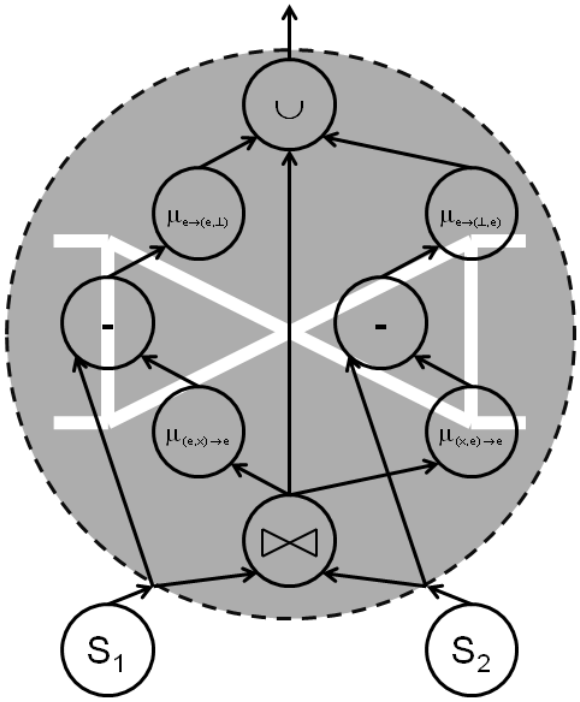


Abbildung 9.1.: Virtueller Äußerer Verbund

Durch Weglassen des jeweils anderen Pfades in dem virtuellen Operator können natürlich auch virtuelle Operatoren für den linken und rechten äußeren Verbund erzeugt werden.

9.4.4. Erweiterung

Die bisherigen Betrachtungen zum äußeren Verbund gingen davon aus, dass der dabei um äußere Elemente erweiterte Verbund als Ergebnisfunktion die Paarbildung verwendet. Es stellt sich nun die Frage, ob dieser auch für andere Ergebnisfunktionen sinnvoll definiert und implementiert werden kann.

Die Definition kann einfach um eine Ergebnisfunktion auf dem Kreuzprodukt aus den Eingabetypen, jeweils erweitert um $\perp$, ergänzt werden, die dann durch Nachschalten eines Mappings mit dieser Ergebnisfunktion implementiert werden kann. Dies funktioniert sowohl für die in 9.4.2 beschriebene Modifikation des Verbundalgorithmus als auch für die in 9.4.3 beschrieben Subtraktionsmethode.

Es bleibt die Frage, ob auch eine vorgegebene Implementierung eines Verbundes zu einer äußeren Verbundimplementierung erweitert werden kann. Den Ausgabetyp des implementierten Verbundes, also den Wertebereich von dessen Ergebnisfunktion, kann man um die mit $\perp$ und den Eingabetypen bildbaren Paare zu einem Ausgabetypen für den äußeren Verbund vereinigen. Damit ist der äußere Verbund dann als Vereinigung aus dem Verbundergebnis und den äußeren Bestandteilen wieder definiert.

Die Subtraktionsmethode erfordert jedoch nun, dass die ursprüngliche Ergebnisfunktion des Verbundes bijektiv sein muss, da die an einem Verbundergebnis beteiligten Elemente rekonstruierbar sein müssen, um sie den Differenzen zuzuführen. Ist dies nicht der Fall, so muss anderweitig ermittelt werden, welche Eingabeelemente am Verbund teilgenommen haben. Dazu ist es beispielsweise hinreichend, die Ergebnisfunktion derart dekorieren zu können, dass sie ihre Parameter zusätzlich nach außen liefert. Häufiger anwendbar dürfte jedoch die Möglichkeit sein, das Verbundprädikat so zu dekorieren, dass es seine Parameter dann nach außen gibt, wenn es zu wahr ausgewertet wird. In beiden Fällen werden die Elemente jetzt nicht zeitlich sortiert geliefert, so dass sie vor der Verwendung in der Differenz noch analog zur Ausgabe des zweiten Verbundalgorithmus mit Hilfe von Prioritätswarteschlangen geordnet werden müssen. Die Schranken, bis wohin die Elemente jeweils ausgegeben werden dürfen, können dabei aus dem Ergebnisstrom des Verbundes gewonnen werden, da dessen Startzeitstempel untere Schranken für die noch zu produzierenden Verbundergebnisse bilden.

10. Optimierung

Die Verbundoperation spielt analog zu DBMS in Datenstrommanagementsystemen eine zentrale Rolle, insbesondere in solchen, die sich am relationalen Datenmodell orientieren. Daher ist es besonders wichtig, den Verbund möglichst optimal auszuführen. Oftmals werden Bemühungen zur Optimierung des Verbundes dabei dann isoliert für den Verbund betrachtet. Für ein DSMS ist aber vielmehr ein Angebot an effizienten Verbundimplementierungen, die sich nahtlos in das System integrieren und mit diesem interagieren, von zentraler Bedeutung.

Beispielsweise kann man die gleiche Menge an Verbundergebniselementen effizienter berechnen, wenn man die Bedingung fallen lässt, dass diese nach Startzeitstempeln geordnet produziert werden müssen. Dazu muss man lediglich im zweiten Algorithmus für den allgemeinen Verbund 8 die Ausgabewarteschlange entfernen, was offensichtlich Ressourcen spart. Allerdings ist die dabei produzierte Ausgabe nicht innerhalb eines DSMS einsetzbar, das nach Startzeitstempeln sortierte Datenströme verarbeitet. Diese Art der Optimierung würde also den Aufwand oftmals nur verlagern, aber nicht wirklich reduzieren.

Für die Optimierung ist es daher wichtig zu analysieren, mit welchen Zielsetzungen diese in einem DSMS erfolgen sollte, um daraus dann ableiten zu können, welche Eigenschaften die Verbundimplementierungen haben sollten und welche Informationen über die Implementierungen benötigt werden, um diese sinnvoll im Gesamtkomplex DSMS verwenden zu können.

In diesem Kapitel wird in 10.1 zunächst die kostenbasierte Optimierung bezogen auf das gesamte DSMS diskutiert, wobei die Rolle des Verbundes dabei genauer behandelt wird. Danach werden in 10.2 interne Optimierungen der vorgestellten Verbundalgorithmen vorgestellt und der Einfluss des Verbundes auf DSMS-Optimierungstechniken beleuchtet. Mit der mehrdimensionalen Verbund wird danach in 10.3 ein Operator vorgestellt, der es erlaubt, mehrere Verbünde zusammenzufassen und dadurch effizienter auszuführen.

10.1. Kostenbasierte Optimierung

In diesem Abschnitt wird der Prozess der kostenbasierten DSMS-Optimierung erläutert und dann die Rolle der Verbundoperation darin beleuchtet. Dazu werden zunächst die Optimierungsziele diskutiert. Danach wird ein Kostenmodell vorgestellt, mit dessen Hilfe verschiedene Alternativen zur Ausführung einer Anfrage verglichen werden können. Dann wird erläutert, wie sich die Verbundoperation in das Kostenmodell einfügt. Abschließend wird aufgezeigt, wie das Kostenmodell zur Optimierung eingesetzt werden kann.

Das Kostenmodell, welches hier beschrieben wird, und Methoden zur kostenbasier-

ten Optimierung wurden bereits in [CKSV08] vorgestellt. Diese Arbeiten werden hier präzisiert und insbesondere in Hinblick auf Verbundoperationen erweitert. Dies beinhaltet insbesondere eine detaillierte Diskussion der Kosten der bedeutendsten SweepAreas. Am Ende dieses Kapitels wird in 10.3.5 zudem noch die Anwendung des Modells auf den mehrdimensionalen Verbundoperator diskutiert.

10.1.1. Zielsetzung

Die Aufgabe eines Optimierers in DBMS ist primär die Optimierung einzelner Anfragen, die an das System gestellt werden und die dann nach der Optimierung möglichst optimal ausgeführt werden sollen. Ziel ist dabei insbesondere eine möglichst kurze Laufzeit der einzelnen Anfrage. Allerdings gibt es auch in DBMS eine Vielzahl von zusätzlichen Anforderungen, die erfüllt sein müssen, um ein gut funktionierendes DBMS bereitzustellen. Beispielsweise sollte es mehrere Transaktionen parallel oder zumindest nebenläufig verarbeiten können, was unter anderem erfordert, dass eine einzelne Anfrage nicht derart optimiert wird, dass sie während ihrer Laufzeit sämtliche Systemressourcen wie Prozessor, Hauptspeicher oder Externspeicherbandbreite blockiert.

In einem DBMS können auch außerhalb der Laufzeit einer Anfrage Optimierungsmaßnahmen stattfinden. So können Indices angelegt werden, die dann gegebenenfalls das Anfrageverhalten des Systems verbessern, aber das Änderungsverhalten möglicherweise negativ beeinflussen. Oftmals wird der Einsatz solcher Techniken noch einem Systemadministrator überlassen und nicht automatisch gesteuert. Dies ist auch nur bedingt möglich, da eine optimale Wahl Informationen über das zukünftige Anfrageverhalten benötigen würde.

Möchte man nun zur Optimierung einer DSMS-Anfrage auf die Erfahrungen aus DBMS aufbauen, so fällt das dort wesentliche Kriterium der Gesamtlaufzeit einer Anfrage komplett weg. Eine DSMS-Anfrage läuft von dem Zeitpunkt an, zu dem sie an das System gestellt wird, bis sie wieder entfernt wird oder die zugehörigen Datenquellen versiegen und alle danach noch ausstehenden Verarbeitungsschritte erfolgt sind; ihre Gesamtlaufzeit ist also fast völlig unabhängig von der Art ihrer Ausführung, falls diese kontinuierlich und nicht unter ständiger Überlastung des Systems erfolgt.

Während der Laufzeit einer Anfrage verarbeitet diese kontinuierlich Eingabeelemente und produziert dabei in der Regel kontinuierlich Ergebnisse. Man könnte also die Zeit vom Eintreffen von Eingabeelementen bis zur Produktion aller daraus resultierender Ergebnisse als Kriterium in Erwägung ziehen. Da beispielsweise Anfragen, deren Ergebnisse mit Alarmsituationen der realen Welt korrelieren, oftmals lange Zeit keine Ergebnisse liefern, müsste man das Kriterium dahingehend modifizieren, dass auch die Feststellung der Tatsache, dass ein Eingabeelement keine Ergebnisse bewirkt, eine Zeitspanne benötigt. Insgesamt ergäbe sich als Optimierungsziel also eine möglichst kurze Zeit zur Erledigung aller aus einem Eingabeelement resultierender Berechnungen. Allerdings kann dieses Element auch noch die Ergebnisse, die erst aus nachfolgenden Elementen berechnet werden können, noch beeinflussen, insbesondere auch beim Verbund. Es spielt also auch eine Rolle, welche Schritte bei der Verarbeitung eines Elementes unternommen werden, um die Verarbeitung weiterer Elemente zu unterstützen. Beispielsweise begünstigt das Einfügen von Elementen in Indexstrukturen in SweepAreas das spätere Auffinden dieser

Elemente bei der Suche nach Verbundergebnissen. Als Ziel sollte also angestrebt werden, nicht die Berechnungszeit, die aus einem einzelnen Element heraus entsteht, sondern die, die im Mittel aus allen entsteht, zu minimieren. Mit anderen Worten sollte die aus einer Anfrage resultierende Belastung der CPU möglichst gering gehalten werden.

Da DSMS in der Regel im Wesentlichen Hauptspeichersysteme sind und die zusätzliche Benutzung des erheblich langsameren Externspeichers nur begrenzt möglich ist, ist der Hauptspeicher die zweite wesentliche begrenzte Ressource. Dementsprechend sollten Anfragen möglichst wenig Hauptspeicher und möglichst wenig Externspeicherbandbreite benötigen.

Ein essentielles Kriterium dabei ist insbesondere, dass die Auslastung der CPU durch die Anfragen nicht lange über 100% liegen darf, da dies bedeuten würde, dass schneller Eingabeelemente eintreffen, als die dadurch nötigen Berechnungen vorgenommen werden können. Die Eingabeelemente müssten zwischengespeichert werden, was wiederum Speicher-Ressourcen benötigen und langfristig zum sicheren Kollaps des Systems führen würde.

Für die Wahl zwischen verschiedenen Möglichkeiten zur Ausführung einer Anfrage ist es also wichtig, den jeweils resultierenden Verbrauch der Ressourcen CPU-Zeit und Hauptspeicher abzuschätzen. Dieser hängt natürlich wesentlich von den Eigenschaften der Eingabeströme der Anfrage ab. Aufgrund der Vielzahl an möglichen Anfragen sollte dabei die Berechnung der Kosten ebenso wie die Berechnung der Anfrage auf die einzelnen dabei verwendeten Operatoren herunter gebrochen werden.

Allerdings können dabei weder die Eingabeströme noch die daraus resultierenden Berechnungen exakt vorausgesagt oder berechnet werden, da dies den gleichen Aufwand wie die Berechnung der Anfrage selber verursachen könnte. Es müssen vielmehr Abschätzungen mit Hilfe eines vereinfachenden Kostenmodells vorgenommen werden. Dieses muss die charakteristischen Eigenschaften von physischen Datenströmen und Anfragen modellieren und für die einzelnen Operatoren abschätzen können, welche Ressourcen diese für derart gegebene Eingabeströme benötigen. Da die von den Operatoren produzierten Ergebnisströme wiederum in weiteren Operatoren als Eingabeströme dienen, muss zudem für jeden Operator die Charakteristik seiner Ausgabeströme abgeschätzt werden können.

10.1.2. Stromcharakteristik

Die Charakteristik eines Datenstromes soll dabei mit Hilfe einiger weniger Parameter ausgedrückt werden.

Parameter d

Die wohl wichtigste Information zur Abschätzung der Kosten von Anfragen in DBMS ist die Größe der beteiligten Eingaben. Dieses Kriterium kann nicht unmittelbar auf Datenströme übertragen werden, da diese potentiell unbeschränkt sind. Allerdings liegen diese auch nicht komplett zur sofortigen Verarbeitung vor, sondern treffen im Laufe der Zeit ein. Wesentlich für den Verarbeitungsaufwand ist demnach die zeitliche Abfolge, mit der Elemente eintreffen. Um das Kostenmodell hinreichend einfach zu halten, wird diese

durch einen einfachen Wert modelliert, und zwar durch den mittleren zeitlichen Abstand zwischen dem Eintreffen zweier aufeinanderfolgender Elemente des Datenstromes. Dieser Abstandsparameter wird mit d (*distance*) bezeichnet.

Logische vs. physische Zeit

Für die reale Belastung der CPU ist natürlich die physische Zeit zwischen dem Eintreffen der Elemente entscheidend, nicht die logische. Die Zeitstempel in den Stromelementen, deren Differenz sich leicht bestimmen lässt, sind jedoch logische Zeitstempel, und auf diesen Zeitstempeln basiert auch die Verarbeitungslogik, beispielsweise das für den Speicherverbrauch des Verbundes besonders wichtige Entfernen von Elementen aus den Statusstrukturen. Daher wurde für das Kostenmodell die Entscheidung getroffen, sämtliche Stromcharakteristika auf die logischen Zeitstempel zu beziehen. In fast allen üblichen Anwendungen sollten die logische und die physische Zeit ohnehin im Wesentlichen synchron oder zeitversetzt laufen. In Simulationen kann zudem eine beschleunigte oder verlangsamte Ausführung auftreten. In diesen Fällen skaliert jedoch dann die Prozessorbelastung entsprechend linear, während die mittlere Speichernutzung unverändert ist. Das Kostenmodell ist also auch in diesen Fällen anwendbar.

Der Parameter d gibt somit den mittleren Abstand der logischen Startzeitstempel innerhalb eines physischen Datenstromes an.

Parameter l

Nachdem ein Stromelement im System eingetroffen ist, muss es dort in vielen Fällen gespeichert werden, da es gemeinsam mit nachfolgend im gleichen oder in anderen Strömen eintreffenden Elementen zu Ergebnissen beitragen kann. Durch die Gültigkeitsintervalle können bei allen auf Basis der Schnappschuss-Äquivalenz definierten Berechnungen aber lediglich Stromelemente gemeinsam zu einem logischen Ergebnis für einen Zeitpunkt beitragen, wenn sich ihre Gültigkeitsintervalle schneiden. Haben alle Eingabeströme also bereits einen Zeitpunkt t erreicht und wurden alle damit verbundenen Berechnungen angestellt, so können Stromelemente mit Endzeitstempel kleiner als t keine weitere Rolle mehr spielen und somit aus den Statusstrukturen der Operatoren entfernt werden.

Nach dem Eintreffen eines Stromelementes bestimmt die Länge seines Gültigkeitsintervalls also im Wesentlichen die Zeit, die es das System beeinflusst, wobei dieses Speicher und CPU-Ressourcen auf das Element verwenden muss. Als zweiter Parameter für die Stromcharakteristik wird daher die mittlere Länge der Gültigkeitsintervalle des Stroms gewählt. Dieser Abstandsparameter wird mit l (*length*) bezeichnet.

Synchronität

Für die physische Dauer, die ein Eingabestromelement die Ergebnisproduktion in einem DSMS beeinflusst, spielt neben der Länge seines Gültigkeitsintervalls auch eine Rolle, wann andere Elemente eintreffen, deren Gültigkeitsintervalle dieses schneiden. Dies muss nicht zwangsweise zeitnah geschehen, da die Datenströme lokal nach Startzeitstempeln sortiert verarbeitet werden, und nicht global sortiert. Daher kann ein logischer Zeitpunkt

in einem Eingabestrom wesentlich später erreicht werden als in einem anderen. Während das Design eines DSMS solche Verschiebungen tolerieren sollte, wirkt sich ein dauerhaftes Auftreten erheblicher Verschiebungen natürlich negativ auf dessen Antwortverhalten aus. Daher sollten diese in der Praxis eher eine temporäre Ausnahme und nicht die Regel darstellen. Da das Verhalten in Ausnahmesituationen ohnehin schwer prognostiziert werden kann, wird das System für das Regelverhalten optimiert. Daher wurde bei der Entwicklung des Kostenmodells die vereinfachende Annahme getroffen, dass die Eingabeströme nicht gegeneinander verschoben sind.

Parameter g

Betrachtet man einen logischen Zeitpunkt t, an dem in den Eingabedatenströmen weder ein Gültigkeitsintervall beginnt, noch eines endet, so stellt man fest, dass für diesen Zeitpunkt das Ergebnis mit dem für $t-1$ übereinstimmen muss, da bei Umwandlung in logische Eingabeströme die Teilmengen mit den Zeitstempeln $t-1$ und t übereinstimmen. Es gibt also keinen Grund, warum in den physischen Ergebnisströmen ein Gültigkeitsintervall zum Zeitpunkt t enden oder beginnen sollte, wenngleich dies in einem korrekten Ergebnis möglich wäre. Dadurch wird die Menge der praktisch möglichen Ergebnisströme eingeschränkt, was den Berechnungs- und Speicheraufwand reduzieren kann.

Daher bietet sich ein Parameter an, der die Vielfalt der im Strom vorkommenden Zeitstempel modelliert, also die Teilmenge der Zeitdomäne, die im Strom vorkommen kann. Da Einschränkungen in der Praxis zumeist auf die Verwendung bestimmter Zeiteinheiten beruhen, die Vielfache der kleinsten Zeiteinheit sind, werden für das Kostenmodell als mögliche Teilmengen vereinfachend nur Unterringe von $\mathbb{Z}$ betrachtet. Als dritter Parameter für die Stromcharakteristik wird dann das größte g gewählt, für das alle Zeitstempel im Strom in $g\mathbb{Z}$ liegen. Dieser Granularitätsparameter wird entsprechend mit g (*granularity*) bezeichnet.

Auswahl

Die gewählten Parameter beziehen sich lediglich auf die Zeitintervalle der Stromelemente und lassen die eigentlichen Elemente unberücksichtigt. Ein wichtiger Grund dafür ist, dass Werteverteilungen in den Strömen vor deren Eintreffen in den seltensten Fällen vorhergesagt werden können, wie dies bei DBMS über Statistiken oftmals möglich ist. Im Gegensatz zu den Werteverteilungen können Informationen über die Gültigkeitsintervalle allerdings häufig schon aus statischen Metadaten (siehe 3.3.1) gewonnen werden. Eine Erweiterung des Kostenmodells um wertbasierte Stromcharakteristiken wäre jedoch ein interessanter Ansatzpunkt für weitere Forschungen. In der Praxis hat sich jedoch das zunächst auf die drei vorgestellten Parameter beschränkte Modell als sehr nützlich erwiesen, da die Gültigkeitsintervalle das Systemverhalten am meisten beeinflussen. In der Charakteristik unberücksichtigt bleibt auch der mittlere Speicherverbrauch eines Elementes in einem Datenstrom. Dieser kann oftmals exakt oder vereinfachend als statisches Metadatum verwaltet werden, das in Abhängigkeit vom Schema für den Operatorgraphen hochgerechnet werden kann. Bei Verwendung von Typen mit Werten

mit sehr verschiedenem Speicherverbrauch kann er aber auch zur Laufzeit als dynamisches Metadatum überwacht werden und aktualisiert in neue Kostenrechnungen einfließen.

Zusammenfassung

Die Charakteristik eines physischen Datenstromes ist also ein Tripel (d, l, g), bestehend aus Abstandsparameter, Längenparameter und Granularitätsparameter. Alle Parameter charakterisieren lediglich die Gültigkeitsintervalle im Strom, nicht die Daten der Elemente.

10.1.3. Kostenmodell

Mit Hilfe der Charakteristik wird nun ein Kostenmodell aufgebaut, mit dem der Ressourcenverbrauch eines Anfragegraphen innerhalb eines DSMS abgeschätzt werden kann. Nach einigen grundsätzlichen Betrachtungen werden die Kosten von Operatoren und, darauf aufbauend, von Anfragegraphen betrachtet. Abschließend werden Anwendungsmöglichkeiten des Kostenmodells diskutiert.

Äquivalenz und Charakteristik

Die Charakteristik ist im Kostenmodell eine Eigenschaft eines physischen Datenstromes. Zwei äquivalente physische Datenströme, also solche mit identischer logischer Darstellung, müssen dabei nicht die gleiche Charakteristik haben. Betrachten man den Datenstrom $S_1 = (a)$ mit $a_i := (x, [2i - 2, 2i))$, so ist dieser äquivalent zum Strom $S_2 = (b)$ mit $b_i := (x, [i - 1, i))$, da er durch Aufteilung aller Elemente in der Mitte ihres Gültigkeitsintervalls entsteht. S_1 hat dabei die Charakteristik $(2, 2, 2)$, während S_2 die Charakteristik $(1, 1, 1)$ hat. Erhöht oder verringert sich bei Umwandlung in einen äquivalenten Strom der Parameter d, so muss dies analog für l gelten, da die Gültigkeitsintervalle die gleichen Zeitpunkte überdecken müssen. Wenn also die Intervalle in größeren Abständen beginnen, müssen diese auch länger sein.

Da die Semantik von Operatoren oft nur über die logischen Datenströme definiert wird, wird durch die Eingabeströme und die Semantik eines Operators nicht die Charakteristik von dessen Ausgabestrom determiniert. Der produzierte physische Ausgabestrom und dessen Charakteristik sind vielmehr implementierungsabhängig. Physisch äquivalente physische Datenströme haben jedoch generell die gleiche Charakteristik, da sie bis auf Umordnung dieselben Stromelemente enthalten.

Operatoren

Formeln Um den Ressourcenverbrauch eines Operatorgraphen abschätzen zu können, muss dieses Problem zunächst für dessen kleinste Einheiten, die Operatoren, gelöst werden. Für jede Operatorimplementierung müssen also Kostenformeln gefunden werden, mit denen für gegebene Charakteristiken der Eingaben der Ressourcenverbrauch geschätzt werden kann. Neben den Charakteristiken spielen jedoch weitere Faktoren eine Rolle, nämlich die Elemente in den Eingabeströmen, deren Werteverteilung wie in 10.1.2 erläutert nicht Teil der Charakteristik ist. So hängt der Berechnungsaufwand des Verbundes nicht

unwesentlich von dessen Selektivität ab, die wiederum von den Werten der Elemente in den Eingabeströmen abhängt.

Da der Ausgabestrom eines Operators im Operatorgraphen wiederum die Eingabe weiterer Operatoren sein kann, für die ebenfalls Kostenabschätzungen getroffen werden sollen, müssen zudem Formeln für die drei Komponenten der Charakteristik des Ausgabestroms einer Operatorimplementierung gefunden werden. Analog zu den Kostenformeln beeinflussen neben den Eingabecharakteristiken weitere Faktoren die Formeln.

Heuristik Da die jeweiligen Formeln auf vereinfachende Charakteristiken und Eigenschaften der Ströme angewendet werden, können sie natürlich nicht in allen Fällen korrekte Ergebnisse liefern. Beispielsweise haben für zwei Folgen (x) und (y) die Ströme $S_1 = (a)$ mit $a_i := (x_i, [2i, 2i + 1))$ und $S_2 = (b)$ mit $b_i := (y_i, [2i + 1, 2i + 2))$ beide die Charakteristik $(2, 1, 1)$. Das Kreuzprodukt zweier Ströme mit diesen Charakteristika hat im Allgemeinen wieder die Charakteristik $(2, 1, 1)$. Das Kreuzprodukt $S_1 \times S_2$ der konkreten Ströme ist jedoch ein leerer Strom, und zwar sogar unabhängig von Werten der Elemente, also den Folgen (x) und (y), allein auf Grund der besonderen Eigenschaften der Gültigkeitsintervalle. Wie dieses Beispiel veranschaulicht, ist es nicht möglich, mit allgemeinen, beweisbar korrekten Formeln aus den Charakteristiken, die lediglich grobe statistische Eigenschaften der Ströme erfassen, die exakte Charakteristik der Ausgabeströme zu berechnen. Es können lediglich Heuristiken angegeben werden, die auf in der Praxis häufig gegebene Annahmen passen und die sich experimentell bewähren.

Stabiler Zustand Zudem ergibt sich der Effekt, dass die Datenströme zwar potentiell unbeschränkt sind, aber einen Anfang haben, beziehungsweise dass eine Anfrage erst zu einem bestimmten Zeitpunkt auf die Datenströme aufgesetzt wird. Nach dem Start der Anfrage steigt deren Ressourcenverbrauch dann in den meisten Fällen zunächst an, bis er einen mehr oder weniger stabilen Zustand erreicht. Der Grund dafür ist, dass sich die Statusstrukturen des Operators zunächst füllen, bis irgendwann auch Elemente wieder entfernt werden können, die auf Grund ihres Gültigkeitsintervalls für die weitere Verarbeitung der Anfrage keine Rolle mehr spielen.

Im Falle des Verbundes werden beispielsweise Stromelemente der linken Eingabe zunächst nur eingefügt, bis die Startzeitstempel des rechten Eingabestroms zum ersten Mal einen Endzeitstempel des linken Stromes erreichen, was zum Löschen aus der SweepArea führt. Von da an werden durch den zeitlichen Fortschritt jeweils etwa gleich viele Elemente entfernt und hinzugefügt, der Speicherverbrauch und ebenso die Anfragekosten der linken SweepArea bleiben also dann konstant.

Der Zustand ansteigenden Ressourcenverbrauches tritt nur am Anfang der Laufzeit einer Anfrage auf. Interessant für die Beurteilung der Anfrage mit Hinblick auf ihren Einfluss auf das System ist aber vielmehr ihr Verhalten im späteren, lange anhaltenden stabilen Zustand. Daher werden die heuristischen Kosten- und Charakteristikformeln auf diesen Zustand hin ausgelegt.

Anfragegraph

Möchte man nun den geschätzten Ressourcenverbrauch eines Operatorgraphen, also insbesondere einer Anfrage, bestimmen, so muss man lediglich beginnend an dessen Eingabeströmen schrittweise für alle Operatoren deren geschätzten Ressourcenbedarf und deren Ausgabecharakteristik berechnen. Die Ausgabecharakteristik wird benötigt, da der Ausgabestrom eines Operators wieder Eingabestrom für weitere Operatoren sein kann. Da der Operatorgraph kreisfrei ist, kann man durch topologisches Sortieren immer eine Reihenfolge finden, so dass für den jeweils nächsten Operator bereits alle Eingabecharakteristiken berechnet wurden.

Wie die neben den Stromcharakteristiken in den Formeln benötigten Parameter gewählt werden, hängt wesentlich von den Anwendungsszenarien des Kostenmodells ab, die nachfolgend diskutiert werden.

Anwendung

Das Kostenmodell kann bei der Optimierung innerhalb eines DSMS für verschiedene Aufgaben verwendet werden, die hier kurz erläutert werden.

Neue Anfrage Kommt eine neue Anfrage in das System, so muss zu deren Beantwortung ein physischer Operatorgraph (Plan) installiert werden. Für die Wahl dieses Graphen gibt es in der Regel eine Vielzahl von Möglichkeiten, unter denen die günstigste Variante ausgewählt werden soll. Welcher Plan dabei als der günstigste angesehen wird, hängt wesentlich davon ab, welcher Ressourcenverbrauch durch diesen Plan während der Ausführung zu erwarten wäre. Mit Hilfe des Kostenmodells können dafür Abschätzungen getroffen werden, ohne den Plan zur Ausführung bringen zu müssen. Dafür werden zum einen Informationen oder Schätzungen der Eingabecharakteristiken benötigt. Zum anderen müssen die zusätzlich an den Operatoren benötigten Parameter geschätzt werden.

Reoptimierung Ist eine Anfrage schon einige Zeit im System und hat bereits einen stabilen Zustand erreicht, so können oftmals mit Hilfe der in 3.3.2 eingeführten dynamischen Metadaten die praktischen Werte für die zuvor geschätzten Parameter des Kostenmodells gemessen werden. Beispielsweise kann jetzt die Selektivität eines Verbundes bestimmt oder sogar über längere Zeit beobachtet werden. Auf Basis dieser nun besser bekannten Informationen kann mit Hilfe des Kostenmodells geprüft werden, ob eventuell ein anderer Anfragegraph doch günstiger wäre. Wie eine Anfrage zur Laufzeit an einen anderen Anfrageplan übergeben werden kann, wird in 10.1.5 erläutert.

Änderungen an Eingabecharakteristika und Parametern Sowohl die Charakteristika der Eingabeströme, als auch verschiedene andere Parameter des Kostenmodells können während der langen Laufzeit einer Anfrage Änderungen unterworfen sein. Diese lassen sich wiederum mit Hilfe dynamischer Metadaten detektieren. Da ein kontinuierliches Beobachten aller Parameter relativ aufwendig ist, sollte man diese nur eingeschränkt verfolgen. Dazu kann man beispielsweise nacheinander einzelne Anfragen untersuchen. Alternativ kann man bedarfsgesteuert vorgehen und nur bei größeren Änderungen am

Ressourcenverbrauch des Systems den Ursachen nachgehen. Diese Vorgehensweise birgt jedoch die Gefahr, in Zeiten hoher Last durch Reoptimierung zusätzliche Last zu erzeugen. Werden wesentliche Änderungen an den Parametern erkannt, so kann analog zum Absatz zuvor wieder zur Laufzeit eine Reoptimierung in Erwägung gezogen werden.

10.1.4. Kostenmodell der Verbundalgorithmen

Für die bisher vorgestellten Algorithmen für den allgemeinen Verbund über Datenströmen werden nun geeignete Kostenformeln und Formeln für die Ausgabecharakteristik motiviert und angegeben. Dabei verarbeite der Verbund jeweils zwei Eingabeströme S_1 und S_2 mit den Charakteristiken (d_1, l_1, g_1) und (d_2, l_2, g_2).

Ausgabecharakteristik der Verbundalgorithmen

Da alle bisher vorgestellten Algorithmen für den allgemeinen Verbund über physischen Datenströmen ihr Ergebnis laut Definition 8.6 berechnen, welche für gegebene physische Eingabeströme das physische Ergebnis bis auf Vertauschung der Stromelemente mit gleichen Startzeitstempeln determiniert, produzieren sie zu gegebener Eingabe alle physisch äquivalente Ausgabeströme, also insbesondere solche mit gleicher Charakteristik. Zudem sind die im Ausgabestrom enthaltenen Stromelemente unabhängig von der Reihenfolge, in der die Eingabeelemente eintreffen. Für die Betrachtungen bezüglich der Ausgabecharakteristik kann hier also angenommen werden, dass die Verarbeitung der globalen zeitlichen Ordnung folgt.

Parameter d Betrachtet man einen Eingabestrom S_i, $i \in \{1,2\}$ $j := 3 - i$, so treffen während einer Zeiteinheit erwartete $\frac{1}{d_i}$ Elemente dort ein, die eine erwartete Länge von l_i haben. Bis zur Entfernung der Elemente werden im Strom S_j im Mittel $\frac{l_i}{d_j}$ Elemente eintreffen, die mit diesen einen Verbund eingehen können, was $\frac{l_i}{d_i d_j}$ möglichen Paaren entspricht. Dabei wurde jedes erwartete Paar genau einmal gezählt, da jedes Ergebnispaar genau für den Strom gezählt wurde, in dem das erste der beiden beteiligten Elemente eintrifft. In Summe aus beiden Eingabeströmen sind pro Zeiteinheit also $\frac{l_1}{d_1 d_2} + \frac{l_2}{d_2 d_1} = \frac{l_1 + l_2}{d_1 d_2}$ Elemente zu erwarten.

Die Anwendung des Verbundprädikates reduziert nun die Zahl der zu erwartenden Paare um die Zahl derer, die es nicht erfüllen. Da die Wahrscheinlichkeit dafür üblicherweise unabhängig von den Zeitcharakteristika ist, wird für die Selektivität ein neuer Parameter σ eingeführt, der die Wahrscheinlichkeit angibt, dass ein Paar des Verbundprädikat erfüllt.[1] Die Zahl der pro Zeiteinheit zu erwartenden Ergebnisse reduziert sich entsprechend linear auf $\sigma \frac{l_1 + l_2}{d_1 d_2}$. Die Anwendung der Ergebnisfunktion produziert aus jedem verbliebenen Paar genau ein Ergebnis. Der Kehrwert $\frac{d_1 d_2}{\sigma(l_1 + l_2)}$ entspricht dann

[1] Bei dieser Definition der Selektivität handelt es sich konsistent zum Kostenmodell um eine Selektivität bezüglich physischer, nicht bezüglich logischer Datenströme. Die Gültigkeit der jeweiligen Resultate spielt dabei keine Rolle, sondern wird durch den Parameter l des Ausgabestromes erfasst.

wieder dem zu erwartenden mittleren Abstand der zu erwartenden Ergebnisse. Für den Abstandsparameter erhält man also

$$d = \frac{d_1 d_2}{\sigma(l_1 + l_2)}$$

Parameter l Die Länge eines Gültigkeitsintervalls in der Ausgabe ergibt sich als die Länge des Schnittes der Längen der beteiligten Eingabeintervalle. Ist m_1 die Länge des kürzeren und m_2 die Länge des längeren Intervalls, so darf für einen Schnitt das kürzere Intervall frühestens im Abstand m_1 vor und spätestens am Ende des längeren anfangen, insgesamt also in einem Bereich der Länge $m_1 + m_2$. Es wird vereinfachend angenommen, dass es in diesem Bereich gleichwahrscheinlich an jeder Stelle anfangen kann. Die Länge des Schnittes wird m_1 betragen, wenn das kürzere Intervall frühestens am Anfang des längeren Intervalls und spätestens m_1 vor Ende des längeren Intervalls anfängt, also mit einer Wahrscheinlichkeit von $p := \frac{m_2 - m_1}{m_1 + m_2}$. In den restlichen Fällen, also mit Wahrscheinlichkeit $1 - p$, wird die Länge des Ergebnisintervalls ein Teil der Länge m_1 sein, im Mittel also $\frac{m_1}{2}$ betragen. Insgesamt ergibt sich eine erwartete Länge von $pm_1 + (1-p)\frac{m_1}{2} = \frac{m_1}{2}(1+p)$. Wegen $1 + p = 1 + \frac{m_2 - m_1}{m_1 + m_2} = \frac{2m_2}{m_1 + m_2}$ ergibt sich eine erwartete Länge von $\frac{m_1}{2}\frac{2m_2}{m_1+m_2} = \frac{m_1 m_2}{m_1 + m_2}$.

Das Ergebnis ist symmetrisch, es spielt also keine Rolle, welches Intervall das längere ist. Es wird angenommen, dass die Selektion durch das Verbundprädikat alle Längen mit gleicher Wahrscheinlichkeit ausfiltert, die erwartete Länge also nicht ändert. Die Ergebnisfunktion hat ohnehin keinen Einfluss auf die Gültigkeitsintervalle. Setzt man nun die erwarteten Längen für die Eingabeintervalle l_1 und l_2 ein, so erhält man für den Längenparameter also

$$l = \frac{l_1 l_2}{l_1 + l_2}$$

Parameter g Alle Zeitstempel in S_1 liegen in $g_1\mathbb{Z}$, alle in S_2 in $g_2\mathbb{Z}$. Da Ergebnisintervalle durch Schnittbildung entstehen, liegen alle Zeitstempel der Ergebnisstromes in $g_1\mathbb{Z} \cup g_2\mathbb{Z} \subseteq ggT(g_1, g_2)\mathbb{Z}$. Die Granularität des Ergebnisstromes kann aber auch ein Vielfaches von $ggT(g_1, g_2)$ sein, was aber nicht garantiert werden kann. Man erhält für den Granularitätsparameter also

$$g = ggT(g_1, g_2)$$

Kostenformeln der Verbundalgorithmen

Wie für alle Kostenformeln wird auch hier angenommen, dass die Datenströme im Wesentlichen der globalen zeitlichen Ordnung folgend verarbeitet werden. Effekte durch zeitliche Verschiebungen werden also ignoriert. Die durch diese entstehenden Nachteile können jedoch bei Verwendung des in 9.3.4 vorgestellten hybriden Verbundalgorithmus minimiert werden. Da sich alle vorgestellten Algorithmen nur bei der Behandlung der zeitlichen Verschiebung unterscheiden, genügt auch hier eine gemeinsame Analyse.

Speicherverbrauch Der Verbund benötigt Speicher für die beiden SweepAreas sowie für die Warteschlangen, die allerdings hier nicht betrachtet werden, da sie bei im Wesentlichen zeitlich geordneter Verarbeitung ohnehin sehr klein bleiben. Ebenfalls ignoriert wird der konstante oder geringe Speicherverbrauch für die Grundstrukturen des Operators, wie beispielsweise Referenzen auf Quellen und Senken.

Betrachtet man einen Eingabestrom S_i, $i \in \{1,2\}$ $j := 3 - i$, so treffen in der SweepArea SA_i zu dieser Eingabe zu einem Zeitpunkt t im Mittel $\frac{1}{d_i}$ Elemente ein. Diese verbleiben dann mindestens l_i Zeiteinheiten in der SweepArea, bis S_j den Zeitpunkt $t + l_i$ erreicht. Dann muss jedoch noch wirklich ein Element in Strom S_j auftreten, was im Mittel $\frac{d_j}{2}$ Zeiteinheiten dauert. Jedes Element in S_1 verbleibt dort also im Mittel $l_i + \frac{d_j}{2}$ Zeiteinheiten. Im Mittel befinden sich somit $\frac{1}{d_i}(l_i + \frac{d_j}{2})$ Elemente in SA_i.

Ein Stromelement in S_1 verbrauche im Mittel v_1 Bytes Speicher, eines in S_2 entsprechend v_2. Die SweepAreas als komplexe Datenstrukturen haben nun einen unterschiedlichen Bedarf an Speicher für die strukturierte Speicherung der Stromelemente. Sei daher $V_{SA} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ eine Funktion, die einer SweepArea SA in Abhängigkeit von erwarteter Zahl an Elementen und eines erwarteten mittleren Speicherverbrauchs für eines dieser Elemente einen Speicherverbrauch zuordnet. Damit kann nun der erwartete Speicherbedarf der Verbundalgorithmen angegeben werden, und zwar mit

$$M = V_{SA_1}(\frac{1}{d_1}(l_1 + \frac{d_2}{2}), v_1) + V_{SA_2}(\frac{1}{d_2}(l_2 + \frac{d_1}{2}), v_2)$$

CPU-Belastung Auch bei der CPU-Belastung werden einfache Grundoperationen ignoriert und nur eine Menge charakteristischer Operationen betrachtet, denen jeweils eine spezifische Kostenkonstante zugewiesen wird. Dies deckt sich mit der Vorgehensweise aus der Analyse anderer Algorithmen; so wird beispielsweise bei der Analyse der Effizienz von Hauptspeicher-Sortierverfahren üblicherweise lediglich die Zahl der benötigten Vergleiche betrachtet.

Analog zum Speicherverbrauch spielen beim Verbund die SweepAreas auch für die CPU-Belastung eine zentrale Rolle. Deren Operationen Einfügen, Entfernen und Anfragen verursachen dabei jeweils einen Aufwand, der sicherlich von der aktuellen Zahl der Elemente in der SweepArea abhängt. Deshalb werden diese wieder für jede SweepArea durch Kostenfunktionen $K_{SA}^{einf}, K_{SA}^{anfr} : \mathbb{R}^+ \to \mathbb{R}^+$ und $K_{SA}^{entf} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ beschrieben, die zu einer gegebenen Größe der SweepArea jeweils den Aufwand für einmaliges Einfügen, Entfernen oder Anfragen angeben. Beim Entfernen kommt als zweiter Parameter noch die Zahl der im Mittel zu entfernenden Elemente hinzu.

Ein in Eingabestrom S_i, $i \in \{1,2\}$ $j := 3 - i$, eintreffendes Element wird zunächst in die eigene SweepArea eingefügt, deren Größe analog zur Betrachtung für den Speicherverbrauch als $\frac{1}{d_i}(l_i + \frac{d_j}{2})$ angenommen werden kann. Dabei fallen also Kosten von $K_{SA_i}^{einf}(\frac{1}{d_i}(l_i + \frac{d_j}{2}))$ an.

Danach werden in der anderen SweepArea nicht mehr benötigte Elemente entfernt. Durch den zeitlichen Fortschritt werden im stabilen Zustand im Mittel die Elemente so häufig eingefügt wie entfernt. Daher entspricht die mittlere Zahl der aus SA_j beim

einmaligen Entfernen gelöschten Elemente dem Verhältnis $\frac{d_i}{d_j}$ der Eingaberaten. Somit entstehen Kosten von $K^{entf}_{SA_j}(\frac{1}{d_j}(l_j + \frac{d_i}{2}), \frac{d_i}{d_j})$.

Durch das Entfernen sind dann bei der Anfrage nur noch diejenigen Elemente in der anderen SweepArea, die bei optimaler Reorganisation dort anfallen, also $\frac{1}{d_j} l_j$. Die Anfragekosten betragen also $K^{anfr}_{SA_j}(\frac{l_j}{d_j})$.

Alle bisher genannten Kosten fallen pro Zeiteinheit in Strom i im Mittel $\frac{1}{d_i}$ mal an. Zudem müssen noch die Kosten berücksichtigt werden, die durch Anwendung der Ergebnisfunktion r auftreten, die bei einmaliger Anwendung C_r betragen. Mit Rückgriff auf die geschätzte Ausgaberate $\frac{1}{d}$ entstehen somit pro Zeiteinheit Kosten von $\frac{C_r}{d}$.

Insgesamt können die CPU-Kosten pro Zeiteinheit für den Verbund also so abgeschätzt werden:

$$\begin{aligned}
&\frac{1}{d_1}(K^{einf}_{SA_1}(\frac{1}{d_1}(l_1 + \frac{d_2}{2})) + K^{entf}_{SA_2}(\frac{1}{d_2}(l_2 + \frac{d_1}{2}), \frac{d_1}{d_2}) + K^{anfr}_{SA_2}(\frac{l_2}{d_2}))+ \\
&\frac{1}{d_2}(K^{einf}_{SA_2}(\frac{1}{d_2}(l_2 + \frac{d_1}{2})) + K^{entf}_{SA_1}(\frac{1}{d_1}(l_1 + \frac{d_2}{2}), \frac{d_2}{d_1}) + K^{anfr}_{SA_1}(\frac{l_1}{d_1}))+ \\
&\frac{C_r}{d}
\end{aligned}$$

Auf den ersten Blick geht dabei das Verbundprädikat gar nicht in die Kostenformeln ein. Allerdings bestimmt dieses ja gerade die Wahl der Strukturen für die SweepAreas sowie deren Anfrage- und Entfernungsprädikate. Daher geht das Verbundprädikat indirekt über die Kostenfunktionen der SweepAreas in die Kostenformeln ein.

Kostenformeln der SweepAreas

Bei Verwendung einer zweistufigen SweepArea verteilen sich die anfallenden Kosten auf die Anfrage- und die Löschstruktur. Beim Einfügen und Löschen fallen in beiden Strukturen Kosten an, während die Anfragekosten allein von der Anfragestruktur abhängen. Bei einer zweistufigen SweepArea $ZSA(A, L)$, die aus einer Anfragestruktur A und einer Löschstruktur L besteht, berechnen sich die Kosten daher mit zusätzlichen Kostenfunktionen $K^{einf}_A, K^{anfr}_A : \mathbb{R}^+ \to \mathbb{R}^+$ und $K^{entf}_A : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$, sowie K^{einf}_L und $K^{entf}_L : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ zu:

$$\begin{aligned}
K^{einf}_{ZSA(A,L)}(s) &= K^{einf}_A(s) + K^{einf}_L(s) \\
K^{entf}_{ZSA(A,L)}(s, r) &= K^{entf}_A(s, r) + K^{entf}_L(s, r) \\
K^{anfr}_{ZSA(A,L)}(s) &= K^{anfr}_A(s)
\end{aligned}$$

Analog verteilt sich der Speicherverbrauch mit zusätzlichen Funktionen $V_A : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ und $V_L : \mathbb{R}^+ \to \mathbb{R}^+$ auf die Strukturen. Dabei muss jedoch berücksichtigt werden, dass in einem objektorientierten System die Objekte nur einmal im Speicher gehalten

werden müssen, während beide Strukturen Referenzen auf die Objekte verwalten. Der Speicherverbrauch durch die Objekte wird daher nur der Speicherstruktur zugeschlagen. Dadurch wird der Speicherverbrauch der Löschstruktur unabhängig von der mittleren Objektgröße.

$$V_{ZSA(A,L)}(a,s) = V_A(a,s) + V_L(a)$$

Löschstrukturen Für die Löschstruktur wird zunächst in den meisten Fällen ein Heap verwendet. Dieser hat logarithmische Einfügekosten $log_2(s)\ C^{einf}_{heap}$. Die Kosten für das Löschen bestehen zunächst aus einer Konstante C^{test}_{heap}, die unabhängig davon anfällt, ob das Element gelöscht wird. Beim Löschen eines Elementes muss das neue minimale Element bestimmt werden. Die Kosten dafür sind logarithmisch in der Größe des Heaps multipliziert mit einer Konstante C^{entf}_{heap}. Für eine Heap-Löschstruktur gilt also

$$\begin{aligned} K^{einf}_{L_{heap}}(s) &= log_2(s)\ C^{einf}_{heap} \\ K^{entf}_{L_{heap}}(s,r) &= (r+1)\ C^{test}_{heap} + r\ log_2(s)\ C^{entf}_{heap} \end{aligned}$$

Da der Heap üblicherweise als Array von Referenzen verwaltet werden kann, fallen Speicherkosten linear zur Zahl der Referenzen an. Benötigt eine Referenz D_{ref} Speicher, so ergibt sich somit

$$V_{L_{heap}}(a) = a\ D_{ref}$$

Anfragestrukturen Für die verschiedenen Verbundprädikate gibt es ein Vielfalt an Anfragestrukturen, von denen die wichtigsten in 9.3.5 vorgestellt werden. Dabei kann eine Datenstruktur auch auf verschiedene Weise genutzt werden, was die Zahl der Varianten weiter erhöht. Die Kostenanalyse wird an dieser Stelle auf die beiden wichtigsten Varianten einer hash-basierten Anfragestruktur beschränkt.

Die verwendete Hashtabelle bestehe dabei aus b Behältern, die jeweils aus einer einfach verketteten Liste von Elementen bestehen. Für ein Array von Referenzen auf die Listen entstehen somit Speicherkosten von bD_{ref}, für jeden Eintrag in einer Liste von $2D_{ref}$ für Referenzen auf Objekt und nächstes Listenelement. Dazu kommt der Speicherbedarf der eigentlichen Elemente, der bei der Speicherstruktur berücksichtigt werden muss.

$$V_{A_{hash}}(a,s) = (b+2a)D_{ref} + as$$

Für die Kostenanalyse werden die elementaren Operationskosten C_{einf} für das Einfügen eines Elementes an den Anfang einer Liste, C_{entf} für das Entfernen aus einer Liste, C_{hash}

für das Anwenden der Hashfunktion, von der angenommen wird, dass sie perfekt streut, C_{verg} für das Testen zweier Elemente auf Gleichheit und C_p für das Testen zweier Elemente auf Erfülltsein des Verbundprädikates verwendet.

Beim Einfügen in die Hashtabelle wird dann einmal die Hashfunktion angewendet und danach in die Liste eingefügt.

$$K^{einf}_{A_{hash}}(s) = C_{hash} + C_{einf}$$

Beim Entfernen wird ebenfalls per Hashfunktion die richtige Liste gesucht, das Element darin linear gesucht und entfernt. Dabei wird hier vereinfachend angenommen, dass das Element im Mittel in der Mitte der Liste zu finden ist.

$$K^{entf}_{A_{hash}}(s, r) = r(C_{hash} + \frac{sC_{verg}}{2b} + C_{entf})$$

Das Anfragen erfolgt analog, wobei immer die gesamte Liste untersucht werden wird. Hierbei besteht noch das Problem, dass Hashfunktion und Verbundprädikat nicht unabhängig sind, da die Hashfunktion joinende Elemente gleich abbilden muss. Bei hoher Selektivität trifft die Annahme also nicht zu, dass die Hashfunktion perfekt streut. Dann entartet die Hashtabelle zu einer Liste. Als Abschätzung der Zahl der zu untersuchenden Elemente wird daher das Maximum aus perfekter Streuung und geschätzter Ergebniszahl verwendet.

$$K^{anfr}_{A_{hash}}(s) = C_{hash} + max(\frac{s}{b}, \sigma s)\ C_p$$

Wird das Hashing lediglich zum effizienten Löschen benutzt, so muss bei Anfragen die gesamte Tabelle durchsucht werden, die Anfragekosten ändern sich also zu

$$K^{anfr}_{A_{hash}}(s) = s\ C_p$$

Allerdings zeigt diese Analyse deutlich, dass diese Variante einer zweistufigen Sweep-Area aus Effizienzbetrachtungen wenig Sinn macht. Wird die Hashfunktion lediglich ausgenutzt, um ein in der Löschstruktur ohnehin aufgefundenes Element nochmals zu löschen, so ist es natürlich günstiger, wie in 9.3.5 diskutiert erst gar keine Anfragestruktur zu verwenden und für Anfragen alle Elemente in der Löschstruktur zu traversieren.

10.1.5. Dynamische Planmigration

Auf Grund der langen Laufzeit von Anfragen in DSMS ist es von besonderer Bedeutung, Anfragepläne nicht nur geeignet optimiert installieren, sondern diese auch zur weiteren Optimierung zur Laufzeit modifizieren zu können. In [KYC+06] wird daher ein allgemeines Verfahren vorgeschlagen, dass es erlaubt, ohne Unterbrechung und ohne Verfälschung der Ausgabeströme Anfragepläne über physischen Datenströmen in äquivalente Alternativen zu überführen. Dieses Verfahren wird in diesem Unterabschnitt zunächst beschrieben. Zudem wird speziell seine Anwendbarkeit auf Verbundoperationen diskutiert und es werden weitere Optimierungen für diesen und andere Anwendungsfälle vorgeschlagen.

Motivation

Obwohl dem Optimierer eines DBMS für die Anwendung des Kostenmodells zumeist erheblich mehr Informationen über die Daten zur Verfügung stehen als in einem DSMS, kann es auf Grund der Abschätzungen im Kostenmodell dennoch vorkommen, dass der Optimierer eine ungünstige Entscheidung trifft. Zur Laufzeit einer Anfrage kann dieses jedoch zumeist erkannt werden, beispielsweise schlicht dadurch, dass die erwartete Laufzeit erheblich überschritten wird, oder durch Überwachung von zuvor geschätzten Selektivitäten.

Eine einfache Art der Reaktion darauf ist es, die Ausführung des gewählten Anfrageplans abzubrechen, anhand der zusätzlich gewonnenen Informationen neu zu optimieren und einen dabei ermittelten Alternativplan auszuführen. Der Nachteil dabei ist jedoch, dass die gesamte bereits investierte Arbeit inklusive der bereits ermittelten Ergebnisse verworfen wird. Daher wurden, speziell für die Verbundoperation[LSM+07], Techniken entwickelt[BBD05, EFP06], um unter Erhaltung bereits produzierter Ergebnisse und teilweise auch Zwischenergebnisse einen Anfrageplan zu seiner Laufzeit zu modifizieren.

Wie bereits in 10.1.3 erläutert steigt in DSMS auf Grund der kontinuierlichen Ausführung von Anfragen der Bedarf an Mechanismen zur Reoptimierung der Anfrageausführung zur Laufzeit, da hier zu der Problematik möglicherweise falscher Kostenabschätzungen bei Wahl des Ausführungsplans noch hinzu kommt, dass sich Parameter wie Selektivitäten und Stromcharakteristika in vielen Fällen während der langen Laufzeit einer Anfrage immer wieder ändern können.

Voraussetzung

Wollte man einen völlig allgemeinen Anfrageplan reoptimieren, so müsste man im Allgemeinen den Status des bereits in Ausführung befindlichen Planes in einen initialen Status des neuen Anfrageplanes umrechnen, da der alte Status Informationen enthalten könnte, die fortan für alle weiteren Ergebnisse benötigt werden und die anderweitig nicht mehr gewonnen werden können, da vom alten Plan konsumierte Stromelemente nicht erneut gelesen werden dürfen. Da eine solche Umrechnung schon für verschiedene Implementierungen eines einzelnen Operators extrem schwierig sein kann und nicht allgemein anzugeben ist, beschränken sich die folgenden Betrachtungen zur Planmigration

auf den wichtigen Fall von Schnappschuss-reduzierbaren Operatorgraphen, der in 2.4.5 beschrieben ist.

Technik

Ein einfaches Verwerfen des alten Anfrageplans und Aufsetzen eines neuen auf die Eingabeströme an der bis dahin noch nicht verarbeiteten Stelle ist nicht sinnvoll, weil dabei Informationen aus den Statusstrukturen des alten Anfrageplans und damit möglicherweise Ergebnisse verloren gehen, also weder vom alten noch vom neuen Anfrageplan produziert werden.

Betrachten man beispielsweise den physischen Datenstrom $S_1 = (a)$ mit $a_i := (i, [i, i+2))$, so ergibt sich als Kreuzprodukt des Stroms mit sich selber ein Strom bestehend aus Elementen der Form $((i, i), [i, i+2))$, $((i, i+1), [i+1, i+2))$ und $((i+1, i), [i+1, i+2))$. Verarbeitet man mit dem alten Anfrageplan alle Elemente bis zu einem Startzeitstempel t, so ist das letzte im alten Plan generierte Ergebnis $((t, t), [t, t+2))$. Im neuen Plan werden alle Elemente ab Startzeitstempel $t+1$ verarbeitet, weshalb das erste dort generierte Ergebnis $((t+1, t+1), [t+1, t+1+2))$ ist. Die Ergebnisse $((t, t+1), [t+1, t+2))$ und $((t+1, t), [t+1, t+2))$ werden hingegen in keinem der Pläne generiert, da an ihrer Produktion gerade physische Stromelemente beteiligt wären, von denen das eine nur dem alten und das andere nur dem neuen Plan zur Verfügung stehen. Führt man die noch benötigten Elemente des neuen Plans als alternativer Ansatz beiden Plänen zu, so erhält man das Ergebnis $((t+1, t+1), [t+1, t+1+2))$ auch im alten Plan, also doppelt, weshalb auch dieses Ergebnis nicht korrekt ist.

Die prinzipielle Idee der zeitlichen Aufteilung der Berechnung zwischen den beiden Anfragepläne führt aber zum Ziel. Die logische Repräsentation S^l des zu berechnenden Ausgabestromes ist eine Menge von Tripeln der Form (e, t, n) und kann demnach für einen Zeitpunkt $\hat{t}$ disjunkt in zwei Teilmengen $S^l_a := \{(e, t, n) | t < \hat{t}\}$ und $S^l_n := \{(e, t, n) | t \geq \hat{t}\}$ aufgeteilt werden. Gelingt es also, mit Hilfe des alten und neuen Anfrageplans zu S^l_a und S^l_n äquivalente physische Ausgabeströme zu produzieren, so ergibt deren Vereinigung das gewünschte Anfrageergebnis.

Um einen zu S^l_a äquivalenten Strom zu generieren, genügt es, einen für alle Zeitpunkte kleiner $\hat{t}$ zu S^l äquivalenten Strom zu produzieren und daraus alle Einträge für Zeitpunkte größer oder gleich t zu entfernen. Für physische Datenströme erreicht man dies, indem man Stromelemente $(e, [t_s, t_e))$ mit $t_e \leq \hat{t}$ erhält, Stromelemente $(e, [t_s, t_e))$ mit $t_s \geq \hat{t}$ entfernt und alle anderen Stromelemente $(e, [t_s, t_e))$ durch Verkürzung des Gültigkeitsintervalls zu $(e, [t_s, \hat{t}))$ modifiziert.

Analog erhält man aus einem für alle Zeitpunkte größer oder gleich $\hat{t}$ zu S^l äquivalenten Strom einen zu S^l_n äquivalenten Strom, indem man Stromelemente $(e, [t_s, t_e))$ mit $t_e \leq \hat{t}$ entfernt, Stromelemente $(e, [t_s, t_e))$ mit $t_s \geq \hat{t}$ erhält und alle anderen Stromelemente $(e, [t_s, t_e))$ durch Verkürzung des Gültigkeitsintervalls zu $(e, [\hat{t}, t_e))$ modifiziert. Daraus ergibt sich die folgende Definition:

Definition 10.1 (Zeitliche Projektion auf physischen Datenströmen)**.** Seien $\mathcal{T}$ und $\mathcal{T}_1, \ldots, \mathcal{T}_n$ Typen und $t \in \mathbb{T}$ ein Zeitpunkt. Die zeitlichen Projektionen über physischen

Datenströmen $\Pi_{<t} : \mathbb{S}^p_{\mathcal{T}} \to \mathbb{S}^p_{\mathcal{T}}$ und $\Pi_{\geq t} : \mathbb{S}^p_{\mathcal{T}} \to \mathbb{S}^p_{\mathcal{T}}$ sind definiert durch

$$\Pi_{<t}(S) := \omega(f : (\Omega_{\mathcal{T}} \times I_{\mathbb{T}}) \to \mathbb{N}, f((e,i)) = \sum_{j \in I_{\mathbb{T}} : i = j \cap [-\infty, t)} \bar{\omega}(S)(e,j))$$

$$\Pi_{\geq t}(S) := \omega(f : (\Omega_{\mathcal{T}} \times I_{\mathbb{T}}) \to \mathbb{N}, f((e,i)) = \sum_{j \in I_{\mathbb{T}} : i = j \cap [t, \infty)} \bar{\omega}(S)(e,j))$$

Sie wird auf mehrere Datenströme entsprechend fortgesetzt. $\Pi_{<t} : \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n} \to \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n}$ und $\Pi_{\geq t} : \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n} \to \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n}$ sind definiert durch

$$\Pi_{<t}((S_1, \ldots, S_n)) := (\Pi_{<t}(S_1), \ldots, \Pi_{<t}(S_n))$$
$$\Pi_{\geq t}((S_1, \ldots, S_n)) := (\Pi_{\geq t}(S_1), \ldots, \Pi_{\geq t}(S_n))$$

Bleibt die Frage, welche Eingaben für die beiden Anfragepläne benötigt werden. Da die Ausgaben nur auf den Zeitpunkten kleiner beziehungsweise größer gleich $\hat{t}$ erhalten bleiben und die Anfragepläne Schnappschuss-reduzierbar sind, spielen auch nur die Werte der Eingabeströme auf den Zeitpunkten kleiner beziehungsweise größer gleich $\hat{t}$ eine Rolle. Hier wird die Eigenschaft der Schnappschuss-Reduzierung ausgenutzt, dass das Ergebnis für einen Zeitpunkt alleine vom Wert der Eingaben für diesen Zeitpunkt abhängen darf. Diese erlaubt es, die auf der Ausgabe bezüglich $\hat{t}$ vorgenommene Trennung auf die Berechnung zu übertragen.

Da dem neuen Anfrageplan bereits konsumierte Elemente der Eingabe nicht mehr zur Verfügung stehen, kann $\hat{t}$ nicht beliebig gewählt werden. Bei Installation der Planmigration muss vielmehr $\hat{t}$ derart bestimmt werden, dass noch kein Element verarbeitet wurde, das zum Zeitpunkt $\hat{t}$ oder später gültig war. Führt man dem neuen Anfrageplan fortan alle Eingabestromelemente zu, so sind insbesondere alle Zeitpunkte ab $\hat{t}$ abgedeckt. Ziel der Migration ist es aber insbesondere, den alten Plan außer Betrieb zu nehmen. Da dieser nur Elemente benötigt, die vor $\hat{t}$ gültig sind, kann man dies tun, sobald alle Eingabeströme mit ihren Startzeitstempeln $\hat{t}$ erreicht haben und vollständig verarbeitet wurden, weil die zeitliche Ordnung der physischen Datenströme garantiert, dass danach keine frühere Zeitpunkte betreffenden Elemente mehr eintreffen.

Optimierung

Das im vorangegangenen Abschnitt vorgestellte Verfahren leistet zwar das Gewünschte, bietet allerdings noch diverse Möglichkeiten zur Optimierung. Hier werden zunächst diejenigen besprochen, die ohne weitere Annahmen auskommen.

Minimierung der Eingaben Für die Ergebnisströme der Anfragepläne wird sichergestellt, dass aus dem alten Plan nur die Ergebnisse für die Zeitpunkte vor $\hat{t}$ und aus dem neuen nur die für Zeitpunkte ab $\hat{t}$ verwendet werden. Wegen der Schnappschuss-Reduzierbarkeit der Anfrage hängen diese auch jeweils nur von den Eingaben bezüglich dieser Zeiträume ab. Somit können die physischen Operatoren $\Pi_{<\hat{t}}$ und $\Pi_{\geq \hat{t}}$, welche auf die Ausgabeströme der Anfragepläne angewendet werden, auch auf die entsprechenden

Eingabeströme angewendet werden. Dadurch reduziert sich die Zahl der Zeitpunkte, für die die Anfragepläne Ergebnisse berechnen müssen und damit in vielen Fällen der Berechnungs- und Speicheraufwand der Pläne.

Split-Operator Erlaubt man in physischen Operatorgraphen auch spezielle Operatoren, die nicht an alle ihre Senken die gleichen Stromelemente liefern, so kann man die zur Minimierung der Eingaben benutzten Paare von Operatoren $\Pi_{<\hat{t}}$ und $\Pi_{\geq\hat{t}}$ jeweils zu einem Operator $\Pi_{\hat{t}}$ zusammenfassen, der Stromelemente der Form $(e, [t_s, t_e))$ mit $t_e \leq \hat{t}$ nur in den alten Plan leitet, solche mit $t_s \geq \hat{t}$ nur in den neuen Plan leitet und alle anderen in je zwei Elemente $(e, [t_s, \hat{t}))$ und $(e, [\hat{t}, t_e))$ umwandelt, wobei das erste in den alten und das zweite in den neuen Anfrageplan weitergeleitet wird. Dadurch werden weniger Operatoren und Vergleiche benötigt, was den Berechnungsaufwand reduziert.

Vereinigung der Ergebnisse Jede n-stellige Anfrage in den Anfrageplänen ist Schnappschuss-reduzierbar und berechnet daher für jeden Zeitpunkt einen mengenbasierten Operator $op(R_1, \ldots, R_n)$. Gilt für alle diese Operatoren $op(\emptyset, \ldots, \emptyset) = \emptyset$, so enthält der logische Ausgabestrom der Anfragepläne keine Elemente für Zeitpunkte, die in keiner der Eingaben vorkommen. Wird in diesem Fall eine Minimierung der Eingaben, etwa durch Verwendung von Split-Operatoren, vorgenommen, so müssen die Operatoren $\Pi_{<\hat{t}}$ und $\Pi_{\geq\hat{t}}$ nicht mehr auf die Ausgabeströme der Pläne angewendet werden, da sie diese gerade nicht verändern.

Zudem benötigt man zur Vereinigung der Ergebnisse aus den beiden Plänen nicht den normalen temporalen Vereinigungsoperator, der aus nach Startzeitstempeln lokal geordneten physischen Strömen einen global geordneten erzeugt. Vielmehr ist bekannt, dass die Startzeitstempel der Stromelemente aus dem alten Plan alle vor denen aus dem neuen Plan liegen müssen. Daher ist es hinreichend, zunächst alle Stromelemente aus dem alten Plan weiterzuleiten und die aus dem neuen zu puffern, um dann nach dem Schließen des alten Planes den Inhalt des Puffers weiterzuleiten und fortan die Stromelemente aus dem neuen Plan direkt weiterzuleiten.

Verbesserung der Ausgabecharakteristik Die Aufteilung der Eingabestromelemente bezüglich des Zeitpunktes $\hat{t}$ führt in vielen Fällen dazu, dass die vereinigten Ausgabeströme der beiden Pläne in einer Umgebung dieses Zeitpunktes anders aussehen als die Ausgabeströme der beiden Pläne, hätte man diese auf die Eingabeströme angewendet. Denn in den meisten Fällen, in denen einer der Pläne ein Ergebnis $(e, [t_s, t_e))$ mit $t_s < \hat{t} < t_e$ erzeugt hätte, werden bei der Planmigration zwei Ergebnisse $(e, [t_s, \hat{t}))$ und $(e, [\hat{t}, t_e))$ erzeugt. Betrachtet man die Charakteristik des dabei entstehenden physischen Ausgabestromes lokal, so sinkt die durchschnittliche Länge l bis zu $\hat{t}$ hin ab, um danach wieder anzusteigen. d bleibt zwar konstant, allerdings mit der Ausnahme des Zeitpunktes $\hat{t}$, zu dem die Gültigkeitsintervalle extrem vieler Ergebnisstromelemente beginnen können. Als Folge ist zu erwarten, dass der Verarbeitungsaufwand in nachfolgenden Operatoren während der Planmigration ansteigt. Obwohl ein solcher Effekt immer vermieden werden sollte, ist er hier besonders problematisch, da zugleich die parallele Verwendung von altem und neuen Plan das System belasten.

Die grundlegende Idee zur Behebung des Problems ist es nun, von den beiden Plänen produzierte Ergebnisse $(e, [t_s, \hat{t}))$ und $(e, [\hat{t}, t_e))$ wieder zu einem Ergebnis $(e, [t_s, t_e))$ zusammenzusetzen. Dazu kann die im vorherigen Abschnitt beschriebene vereinfachte Vereinigung so modifiziert werden, dass sie die aus dem alten Plan erhaltenen Ergebnisse jeweils erst dann weiterleitet, wenn aus dem neuen Plan die passende Ergänzung eintrifft. Um die Ausgabe sortiert nach Startzeitstempeln zu garantieren, müssen dabei aber auch nachfolgende Elemente zurückgehalten werden, bis zu allen vorherigen die passenden Ergänzungen eingetroffen sind. Dies stellt an sich kein Problem dar, da diese in der Regel etwa zeitgleich berechnet werden. Das Problem ist vielmehr, dass im alten Plan Ergebnisse produziert werden können, die keine passende Ergänzung haben, nämlich immer dann, wenn sich in der logischen Repräsentation des Ergebnisses die Gültigkeit eines Elementes e zum Zeitpunkt $\hat{t}$ von der zum Zeitpunkt $\hat{t} - 1$ unterscheidet. Das Fehlen einer passenden Ergänzung kann aber erst erkannt werden, sobald der neue Anfrageplan ein Stromelement mit Startzeitstempel größer als $\hat{t}$ produziert. Somit würde das Warten auf nicht existierende Ergänzungen die Vereinigung vorübergehend blockieren, was in der kontinuierlichen Anfrageverarbeitung vermieden werden sollte.

Am einfachsten kann dieses Problem gelöst werden, wenn $\hat{t}$ so gewählt werden kann, dass es in den Ausgabeströmen des zu migrierenden Anfrageplans nicht als Start- oder Endzeitstempel auftreten kann. Dies ist insbesondere immer dann der Fall, wenn die Granularitäten der Ausgabeströme einen größten gemeinsamen Teiler haben, der größer als eins ist.

Scheidet diese Lösung aus, etwa weil einer der Ausgabeströme eine Granularität von eins hat, so kann man eine approximative Lösung einsetzen. Dazu modifiziert man den Split-Operator dahingehend, dass er Elemente der Form $(e, [t_s, t_e))$ mit $t_s \leq \hat{t} < t_e$ in zwei Elemente $(e, [t_s, \hat{t} + 1))$ und $(e, [\hat{t}, t_e))$ aufteilt, die er in den alten beziehungsweise neuen Plan leitet. Elemente der Form $(e, [t_s, \hat{t}))$ werden unverändert nur in den alten Plan geleitet. Damit berechnet der alte Plan jetzt die Anfrage für alle Zeitpunkte einschließlich $\hat{t}$. Entsteht nun im alten Plan ein Ergebniselement der Form $(\bar{e}, [\bar{t}_s, \hat{t} + 1))$, so muss $\bar{e}$ auch im neuen Plan zum Zeitpunkt $\hat{t}$ gültig sein, es existiert also garantiert eine passende Ergänzung $(\bar{e}, [\hat{t}, \bar{t}_e))$. Diese beiden Ergebnisse werden somit zu $(\bar{e}, [\bar{t}_s, \bar{t}_e))$ verschmolzen, wobei die zur Kenntlichmachung hinzugefügte doppelte Gültigkeit zum Zeitpunkt $\hat{t}$ gerade wieder entfernt wird. Ergebnisse der Form $(\bar{e}, [\bar{t}_s, \hat{t}))$ müssen keine Ergänzung haben und werden unverändert dem Ergebnisstrom hinzugefügt.

Umgekehrt muss aber nicht zu jeder vorhandenen Ergänzung ein Ergebnis der Form $(\bar{e}, [\bar{t}_s, \hat{t} + 1))$ erzeugt werden, da der alte Plan alternativ auch zwei Ergebnisse $(\bar{e}, [\bar{t}_s, \hat{t}))$ und $(\bar{e}, [\hat{t}, \hat{t} + 1))$ erzeugen darf. Das erste dieser Elemente wird dann wie oben beschrieben unverändert weitergeleitet. Elemente der Form $(\bar{e}, [\hat{t}, \hat{t} + 1))$ werden am Ausgang des alten Planes verworfen. Da dieser Spezialfall in der Praxis sehr selten auftritt, findet dieses approximative Verfahren fast alle Möglichkeiten, ein Ergebnis passend zu ergänzen. Die übrigen Ergebnisse werden ebenfalls korrekt verarbeitet und führen nur zu einer geringfügigen Mehrbelastung der Folgeoperatoren. Zudem besteht auch keine Gefahr des zeitweiligen Blockierens mehr, weil alle aus dem alten Plan eintreffenden Elemente direkt verarbeitet werden, da diese nach Startzeitstempeln sortiert eintreffen und das Verfahren jeweils Ergebnisse mit gleichem Startzeitstempel weiterleitet.

Referenzpunktmethode

Wenn noch zusätzliche Annahmen getroffen werden, die in der Praxis häufig erfüllt sind, kann das Verfahren noch weiter optimiert werden.

Motivation Bei dem im vorstehenden Abschnitt beschriebenen Verfahren fällt auf, dass beim Ergänzen das Stromelement aus dem neuen Anfrageplan lediglich benötigt wird, um den Endzeitstempel des verschmolzenen Ergebnisses zu ermitteln. Daher liegt die Idee nahe, auch diesen noch durch den alten Graphen bestimmen zu lassen. Verlängert man die Elemente, die beim Split in den alten Graphen gelangen, dort nicht nur um eine Zeiteinheit hinter $\hat{t}$, sondern belässt ihnen ihr ursprüngliches Gültigkeitsintervall, so werden vom alten Graphen weiterhin die kompletten Ergebnisse berechnet, die auch ohne Planmigration berechnet worden wären. Zu einem Ergebnis $(\bar{e}, [t_s, t_e))$ mit $t_s < \hat{t} < t_e$ aus dem alten Plan müsste dennoch weiterhin ein Ergebnis $(\bar{e}, [\hat{t}, t_e))$ am Ausgang des neuen Plans entfernt werden, obwohl die komplette Information über das Ausgabestromelement bereits aus dem alten Plan verfügbar ist. Damit das Verfahren also einen Vorteil bringt, müssen die Stromelemente, die am Ausgang des neuen Plans entfernt werden, einfacher erkannt werden. Die Idee dazu ist es, dual zur im vorherigen Ansatz beschriebenen Kennzeichnung der Stromelemente, die eine Ergänzung finden, diesmal die Gültigkeitsintervalle der Stromelemente, die beim Split für den neuen Plan erzeugt werden, um eine Zeiteinheit nach vorne zu verlängern. Leider tritt dabei analog auch das Problem auf, dass diese Art der Kennzeichnung nicht zwingend alle Stromelemente erfasst. Dieses muss hier jedoch gelöst werden, da es diesmal nicht um optionale Ergänzungen, sondern um zwingend nötiges Entfernen doppelt produzierter Ergebnisse geht.

Das Problem weitet sich sogar noch auf spätere Zeiteinheiten aus, denn statt eines Ergebniselementes $(\bar{e}, [\hat{t} - 1, t_e))$ darf der neue Plan beispielsweise auch zwei Ergebniselemente $(\bar{e}, [\hat{t} - 1, \check{t}))$ und $(\bar{e}, [\check{t}, t_e))$ erzeugen. Um hierbei korrekt sicherzustellen, dass alle schon im alten Plan erzeugten Ergebnisteile im neuen verworfen werden, müssten beide Elemente erkannt werden, nicht nur das erste.

Voraussetzungen Um diese Probleme lösen zu können, müssen die Voraussetzungen zunächst definiert werden.

Definition 10.2 (Historisch kontextfrei)**.** Seien $\mathcal{T}^e_1, \ldots, \mathcal{T}^e_n$ und $\mathcal{T}^a_1, \ldots, \mathcal{T}^a_n$ Typen.

Ein physischer Operatorgraph $g^p : \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n} \to \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_m}$ ist *historisch kontextfrei*, wenn $\forall (S^p_1, \ldots, S^p_n) \in \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n} : \Pi_{\geq t}(g^p(S^p_1, \ldots, S^p_n)) \equiv^p g^p(\Pi_{\geq t}(S^p_1, \ldots, S^p_n))$.

Ein physischer Operatorgraph ist also historisch kontextfrei, wenn er mit $\Pi_{\geq t}$ kommutiert. Die Eigenschaft drückt aus, dass die physischen Ergebnisse des Operatorgraphen unabhängig davon sein müssen, ob man logisch alle Gültigkeiten bis zu einem Zeitpunkt t vor oder nach dem Plan entfernt.

Mit Hilfe dieser Definition werden nun folgende Anforderungen für das Verfahren gestellt:

- Alter und neuer Anfrageplan müssen physisch äquivalent sein.

- Der neue Anfrageplan muss historisch kontextfrei sein.

Die erste Forderung verschärft die für die allgemeine Planmigration benötigte logische Äquivalenz. Die zweite Bedingung stellt sicher, dass sich daran bezüglich des Zeitraumes ab $\hat{t} - 1$ nichts ändert, wenn man alle Zeiteinheiten vor $\hat{t} - 1$ bei den Berechnungen ignoriert.

Verfahren Unter diesen Voraussetzungen kann das Verfahren zur Planmigration durchgeführt werden, das hier zusammengefasst wird:

- Es wird ein Zeitpunkt $\hat{t}$ bestimmt, so dass keiner der Eingabeströme bereits ein Stromelement geliefert hat, dass zum Zeitpunkt $\hat{t} - 1$ oder später gültig war.
- Der neue Plan wird neben dem alten installiert.
- Alle Eingabeströme werden mit einem modifizierten Split auf die beiden Pläne verteilt. Alle Elemente werden in den alten Plan geleitet. Elemente der Form $(e, [t_s, t_e))$ werden nicht in den neuen Plan geleitet, falls $t_e \leq \hat{t}$ ist und komplett in den neuen Plan geleitet, falls $t_s \geq \hat{t}$. Anderenfalls wird $(e, [\hat{t} - 1, t_e))$ in den neuen Plan geleitet.
- Die Ausgabeströme der beiden Pläne werden paarweise zum Ausgabestrom vereinigt. Dabei werden zunächst alle Stromelemente aus dem alten Plan weitergeleitet, deren Startzeitstempel kleiner als $\hat{t}$ ist. Alle Stromelemente aus dem neuen Plan, deren Startzeitstempel nicht $\hat{t} - 1$ ist, werden gespeichert.
- Sobald der alte Anfragegraph auf einer Ausgabe ein Stromelement mit Startzeitstempel $\hat{t}$ oder größer produziert hat, werden alle in der entsprechenden Vereinigung gepufferte Stromelemente weitergeleitet. Sobald dies auf allen Ausgaben der Fall war, werden der alte Plan und alle Hilfsoperatoren entfernt und die Planmigration ist abgeschlossen.

Korrektheit Zu zeigen ist, dass das Verfahren Ausgabeströme liefert, die logisch äquivalent zu denen sind, die der alte Anfrageplan angewendet auf die kompletten Eingaben produziert hätte. Dazu wird hier gezeigt, dass die produzierten Ausgabeströme sogar physisch äquivalent dazu sind. Dazu wird zunächst folgendes Verfahren betrachtet: Beide Anfragepläne werden auf die komplette Eingabe angewendet und die Ergebnisse vereinigt. Dabei entsteht jedes physische Stromelement aus dem alten Plan offensichtlich zusätzlich genau nochmals im neuen Plan, da die Pläne nach Voraussetzung physisch äquivalent sind. Zur Duplikateliminierung werden daher von den Ergebnisse aus dem alten Plan die mit Startzeitstempel größer oder gleich $\hat{t}$ entfernt, von denen aus dem neuen Plan diejenigen mit Startzeitstempel kleiner als $\hat{t}$. Da jeder Startzeitstempel genau eines der Kriterien erfüllt, wird somit die Gesamtergebnismenge für jeden Ausgabestrom in zwei disjunkten Teilmengen produziert. Da die Startzeitstempel in der einen alle kleiner sind als die in der anderen, liefert die Hintereinanderanordnung der Ströme jeweils das korrekte Ergebnis.

Modifiziert man dies Verfahren nun so, dass auf alle Ausgaben des neuen Planes $\Pi_{\geq \hat{t}-1}$ angewendet wird, so funktioniert das Verfahren offensichtlich immer noch. Dies

liegt daran, dass die Stromelemente des neuen Plans mit Startzeitstempel größer oder gleich $\hat{t}$, die daraus weitergeleitet werden müssen, unverändert produziert werden, und diejenigen mit Startzeitstempel kleiner als $\hat{t}$, die verworfen werden müssen, entweder komplett fehlen oder den Startzeitstempel $\hat{t} - 1$, der kleiner als $\hat{t}$ ist, haben.

Modifiziert man das Verfahren weiter so, dass $\Pi_{\geq \hat{t}-1}$ auf die Ein-, statt die Ausgaben des neuen Plans angewendet wird, so ändert sich nichts, da der neue Anfrageplan nach Voraussetzung historisch kontextfrei ist. Wegen der Anwendung von $\Pi_{\geq \hat{t}-1}$ genügt es nun, dem neuen Plan nicht alle Eingabestromelemente, sondern nur diejenigen, die zum Zeitpunkt $\hat{t} - 1$ oder später gültig sind, zuzuführen. Da $\hat{t}$ im ersten Schritt des Verfahrens so gewählt wird, dass noch kein solches Element verarbeitet wurde, ist gewährleistet, dass alle diese Stromelemente den neuen Plan erreichen.

Sobald der alte Plan in jedem Ausgabestrom ein Stromelement mit Startzeitstempel $\hat{t}$ oder größer produziert hat, das verworfen wird, wird er wegen der zeitlichen Ordnung auf den physischen Strömen nur noch weitere solche Elemente produzieren, die ebenfalls verworfen würden. Daher kann der alte Plan zu diesem Zeitpunkt folgenlos entfernt werden. Nach dem Verwerfen eines Elementes aus dem alten Plan kann jeweils in der jeweiligen Vereinigung die Ausgabe aus dem neuen Plan beginnen, da die aus dem alten abgeschlossen ist.

Das hier schrittweise als korrekt entwickelte Verfahren entspricht demjenigen, dessen Korrektheit zu zeigen war.

Optimierung Die Referenzpunktmethode hat den angestrebten Vorteil, bei der Vereinigung der Ergebnisse aus den beiden Plänen ohne komplexe Operationen auszukommen und dennoch die gleichen physischen Ausgabeströme zu liefern, die auch ohne Planmigration entstanden wären. Sie hat jedoch den Nachteil, dass gegebenenfalls mehr Stromelemente in den alten Plan geleitet werden müssen als bei der zuvor beschriebenen allgemein anwendbaren Planmigration. Unter Hinzunahme einer Bedingung an den alten Anfrageplan kann diese Problematik jedoch auf ein Minimum reduziert werden.

Definition 10.3 (Zeitliche Selektion auf physischen Datenströmen)**.** Seien $\mathcal{T}$ und $\mathcal{T}_1, \ldots, \mathcal{T}_n$ Typen und $t \in \mathbb{T}$ ein Zeitpunkt. Die zeitlichen Selektion über physischen Datenströmen $\Sigma_{<t} : \mathbb{S}^p_{\mathcal{T}} \to \mathbb{S}^p_{\mathcal{T}}$ ist definiert durch

$$\Sigma_{<t}(S) := \omega\left(f : (\Omega_{\mathcal{T}} \times I_{\mathbb{T}}) \to \mathbb{N}, f((e,[t_s,t_e)) = \begin{cases} \bar{\omega}(S)(e,[t_s,t_e))) & t_s < t \\ 0 & \textit{sonst} \end{cases}\right)$$

Sie wird auf mehrere Datenströme entsprechend fortgesetzt. $\Sigma_{<t} : \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n} \to \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n}$ ist definiert durch

$$\Sigma_{<t}((S_1, \ldots, S_n)) := (\Sigma_{<t}(S_1), \ldots, \Sigma_{<t}(S_n))$$

Die zeitliche Selektion liefert also den Anfang des Eingabestromes bis zum Erreichen des Startzeitstempels t, ab dem die Elemente nicht mehr geliefert werden.

Definition 10.4 (Verzögerungsfrei)**.** Seien $\mathcal{T}^e_1, \ldots, \mathcal{T}^e_n$ und $\mathcal{T}^a_1, \ldots, \mathcal{T}^a_n$ Typen.

Ein physischer Operatorgraph $g^p : \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n} \rightarrow \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_m}$ ist *verzögerungsfrei*, wenn $\forall (S^p_1, \ldots, S^p_n) \in \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n} : \Sigma_{<t}(g^p(S^p_1, \ldots, S^p_n)) \equiv^p g^p(\Sigma_{<t}(S^p_1, \ldots, S^p_n))$.

Ein physischer Operatorgraph ist also verzögerungsfrei, wenn er alle Ergebnisse bis zu einem Startzeitstempel $\check{t}$ produziert, wenn alle Eingabeströme bis zum Startzeitstempel $\check{t}$ vorliegen. Physische Operatoren wie typische Implementierungen von Selektion und Projektion haben beispielsweise diese Eigenschaft, der Aggregationsoperator hat sie aber im Allgemeinen nicht.

Bei der Referenzpunktmethode werden nun aus dem alten Anfrageplan genau diejenigen Ergebnisse mit einem Startzeitstempel kleiner als $\hat{t}$ benötigt. Ist der alte Plan nun verzögerungsfrei, so genügt es, ihm statt aller Eingabeelemente nur diejenigen zuzuführen, die bei Anwendung von $\Sigma_{<\hat{t}}$ auf die Eingaben entstehen. Der modifizierte Split kann also so weiter modifiziert werden, dass er dem alten Anfrageplan statt aller nur die Stromelemente mit Startzeitstempel kleiner als $\hat{t}$ zuführt. Da dieser nun nur noch Ergebnisse mit Startzeitstempel kleiner als $\hat{t}$ produziert, kann das Ausfiltern anderer Ergebnisse aus dem alten Plan entfallen. Zudem muss noch das Kriterium geändert werden, das bestimmt, wann der alte Plan entfernt werden kann. Dies ist nun der Fall, wenn diesem alle Elemente bis zum Startzeitstempel $\hat{t} - 1$ zugeführt sind und diese darin vollständig verarbeitet wurden.

Da bei Nichtverarbeitung von Eingabeelementen mit Startzeitstempel kleiner als $\hat{t} - 1$ dem alten Plan Informationen fehlen würden, um die Ergebnisse für den Zeitpunkt $\hat{t} - 1$ korrekt zu berechnen, verarbeitet diese optimierte Variante der Referenzpunktmethode nun im alten Anfrageplan ein Minimum an Eingabeelementen. Das so entstandene Verfahren ist aber nur anwendbar, wenn der alte Anfrageplan verzögerungsfrei und der neue historisch kontextfrei ist.

Anwendbarkeit auf Verbundoperationen

Lemma 10.1. *Der allgemeine Verbund über physischen Datenströmen $\bowtie^p$ ist historisch kontextfrei und verzögerungsfrei.*

Beweis. Die Verbundergebnisse mit Startzeitstempel t_s der Form $(e, [t_s, t_e))$ entstehen aus Eingabestromelementen $(e_1, [t_{s_1}, t_{e_1}))$ und $(e_2, [t_{s_2}, t_{e_2}))$ mit $[t_s, t_e) = [t_{s_1}, t_{e_1}) \cap [t_{s_2}, t_{e_2})$. Demnach gilt $t_s = max(t_{s_1}, t_{s_2})$ und somit $t_s \geq t_{s_1} \wedge t_s \geq t_{s_2}$, aber nicht $t_s > t_{s_1} \wedge t_s > t_{s_2}$. Somit ist der allgemeine Verbund über physischen Datenströmen verzögerungsfrei.

Bei Anwendung von $\Pi_{\geq t}$ wird $(e_1, [t_{s_1}, t_{e_1}))$ zu $(e_1, [max(t, t_{s_1}), t_{e_1}))$, wobei das Stromelement für $max(t, t_{s_1}) \geq t_{e_1}$ nicht existiert. Analog wird $(e_2, [t_{s_2}, t_{e_2}))$ zu $(e_2, [max(t, t_{s_2}), t_{e_2}))$. Soweit daraus noch ein Verbundergebnis gebildet werden kann, hat es die Form $(e, [max(t, t_{s_1}, t_{s_2}), t_e)$ mit $max(t, t_{s_1}, t_{s_2}) = max(t, max(t_{s_1}, t_{s_2})) = max(t, t_s)$. Dies entspricht somit dem Ergebnis der Form $(e, [max(t, t_s), t_e))$, das bei Anwendung von $\Pi_{\geq t}$ auf das Ausgabeelement $(e, [t_s, t_e))$ entsteht. Somit ist der allgemeine Verbund über physischen Datenströmen historisch kontextfrei. □

Somit kann eine konkrete Implementierung des allgemeinen Verbunds über physischen Datenströmen per Referenzpunktmethode zur Laufzeit gegen eine andere ausgetauscht werden. Das folgende Korollar führt jedoch zu wesentlich wichtigeren Anwendungsfällen:

Korollar 10.1. *Physische Operatorgraphen, die aus allgemeinen Verbünden über physischen Datenströmen bestehen, sind historisch kontextfrei und verzögerungsfrei.*

Beweis. Folgt direkt per struktureller Induktion aus Lemma 10.1. □

Somit können unter Verwendung der Referenzpunktmethode zur Laufzeit verschiedene Operatorgraphen gegeneinander ausgetauscht werden, wenn sie physisch äquivalent sind und aus allgemeinen Verbünden über physischen Datenströmen bestehen. Solche Operatorgraphen erhält man aus logisch äquivalenten Operatorgraphen, die nur aus allgemeinen Verbünden über logischen Datenströmen bestehen, indem man die logischen Verbünde entsprechend ersetzt. Da die Semantik der logischen Operatorgraphen über die Schnappschuss-Reduzierung (siehe 2.4.5) auf die der erweiterten relationalen Algebra zurückgeführt werden kann, können mit Hilfe der Optimierungsregeln für DBMS viele Spezialfälle von Strom-Verbünden über relationalen Schemata optimiert werden. Die Referenzpunktmethode erlaubt dann zur Laufzeit den Wechsel zu dem anhand der zur Laufzeit ermittelten aktuellen Parameter günstigsten Ausführungsplan für den Verbundgraphen.

Da sich die Aussagen von Lemma 10.1 und Korollar 10.1 problemlos um die in dieser Arbeit nicht weiter formal behandelten Operatoren Selektion und Projektion erweitern lassen, gilt das Ergebnis analog für so genannte SPJ-Anfragepläne, die aus Joins, Projektionen und Selektionen bestehen. Diese stellen einen wichtigen Spezialfall in der Anfrageplanoptimierung dar.[Cha98]

10.2. Verbundbezogene Optimierung

In diesem Abschnitt wird das Zusammenwirken des Verbundes mit dem umgebenden System auf Optimierungsmöglichkeiten hin beleuchtet. Dabei wird in 10.2.1 zunächst aufgezeigt, wie die in den Verbundimplementierungen verwendeten Löschstrukturen unter bestimmten Umständen vereinfacht werden können. Danach wird in 10.2.2 der Einfluss des Verbundes auf den zeitlichen Fortschritt und in 10.2.3 der auf das Scheduling des Systems untersucht. In 10.2.4 wird abschließend der Spezialfall eines Verbundes eines Datenstromes mit sich selbst diskutiert.

10.2.1. Vereinfachte Löschstrukturen

Wie in 9.3.5 beschrieben werden für die in 9.3 vorgestellten Algorithmen zur Berechnung der Verbundoperation im Allgemeinen zeitbezogene Löschstrukturen auf Basis eines Heaps eingesetzt. Der Heap dient dabei dazu, jeweils das Element mit minimalem Endzeitstempel in der SweepArea zu liefern. Da die Stromelemente in diese sortiert nach Startzeitstempeln eingefügt werden, wird der Heap benötigt, um eine Ordnung auf den Endzeitstempeln herzustellen. In konkreten Anwendungen haben jedoch viele physische Datenströme folgende nützliche Eigenschaft:

Definition 10.5 (Endzeitstempel-Monotonie)**.** Sei $\mathcal{T}$ ein Typ. Ein physischer Datenstrom b mit $b_i = (e_i, [t_{s_i}, t_{e_i})) \in \Omega_{\mathcal{T}} \times I_{\mathbb{T}}$ ist *Endzeitstempel-monoton*, wenn $\forall i \in \mathbb{N} : t_{e_i} \leq t_{e_{i+1}}$

Insbesondere sind alle physischen Datenströme, deren Elemente alle Gültigkeitsintervalle gleicher Länge haben, Endzeitstempel-monoton, da sie nach Startzeitstempeln sortiert eintreffen und das Addieren einer Konstanten auf alle Glieder einer Folge von ganzen Zahlen nichts an deren Ordnung ändert. Solche Ströme entstehen insbesondere bei Anwendung von gleitenden zeitlichen Fenstern auf Ströme, die zuvor nur Elemente der Form $(e, [t, t+1))$ enthielten.

Aus zeitbezogenen Löschstrukturen können die Elemente eines Endzeitstempel-monotonen physischen Datenstromes nun genau in der Reihenfolge wieder entfernt werden, in der sie eingefügt wurden, da diese nach Definition eine aufsteigende Sortierung auf den Endzeitstempeln darstellt. Daher genügt als Datenstruktur innerhalb der zeitbezogenen Löschstruktur in diesem Fall eine FiFo-Queue. Da diese konstante Kosten C_{FiFo}^{einf} für das Einfügen, C_{FiFo}^{entf} für das Entfernen und C_{FiFo}^{test} für das Testen des ersten Elements hat, spart diese Optimierung gegenüber den in 10.1.4 analysierten Kosten der Heap-Löschstruktur CPU-Ressourcen ein:

$$\begin{aligned} K_{L_{FiFo}}^{einf}(s) &= C_{FiFo}^{einf} \\ K_{L_{FiFo}}^{entf}(s, r) &= (r+1)\, C_{FiFo}^{test} + r\, C_{FiFo}^{entf} \end{aligned}$$

Der Einsatz dieser Optimierung kann unabhängig von der anderen für eine der SweepAreas eines Verbundes gewählt werden, falls der zugehörige physische Eingabestrom garantiert Endzeitstempel-monoton ist.

Um sie in einem Operatorgraphen möglichst häufig nutzen zu können, ist es wichtig, möglichst viele physische Datenströme darin als garantiert Endzeitstempel-monoton zu identifizieren. Da die Semantik von Operatoren zumeist auf Basis von logischer oder physischer Äquivalenz definiert wird, kann dies jedoch zumeist erst bei der Installation des Operatorgraphen mit konkreten Implementierungen erfolgen. Denn selbst physisch äquivalente Datenströme können sich bezüglich der Eigenschaft Endzeitstempel-Monotonie unterscheiden. Es bietet sich daher an, die Endzeitstempel-Monotonie als Metadatum der konkreten Operatorinstanzen zu verwalten, welche die Datenströme produzieren. Datenquellen können dabei die Endzeitstempel-Monotonie in vielen Fällen garantieren. Nachfolgende Operatorimplementierungen können von ihren Quellen das entsprechende Metadatum abfragen und dann anhand ihrer eigenen Eigenschaften berechnen, für welche ihrer Ausgabeströme sie selber Endzeitstempel-Monotonie garantieren können.

Während eine positive Entscheidung über die Endzeitstempel-Monotonie der Ausgabeströme eines physischen Operators auf Grund einer Definition auf Basis der in der physischen Ausgabe enthaltenen Stromelemente nicht möglich ist, kann gegebenenfalls eine negative Entscheidung getroffen werden, nämlich dann, wenn der Ausgabestrom zwei Stromelemente $(e_1, [t_{s_1}, t_{e_1}))$ und $(e_2, [t_{s_2}, t_{e_2}))$ mit $t_{s_1} < t_{s_2}$ und $t_{e_1} > t_{e_2}$ enthalten kann. Da dies im Allgemeinen bei der Verbundoperation der Fall ist, können deren Ausgabeströme nur selten als Endzeitstempel-monoton garantiert werden, zum Beispiel dann, wenn die Gültigkeitsintervalle in einem der Eingabeströme alle die Länge eins

haben. Während der Verbund also von Endzeitstempel-monotonen Eingabeströmen profitiert, kann er sie als seine Ausgabe zumeist nicht liefern.

Spätes Löschen

Natürlich können auch physische Datenströme, für die nicht im Voraus garantiert werden kann, dass sie Endzeitstempel-monoton sind, sich zur Laufzeit entsprechend verhalten. Zudem gibt es Fälle, in denen die Folge der Endzeitstempel zwar weitgehend, aber nicht komplett monoton ist. Um auch diese Fälle optimieren zu können, kann man den vorherigen Ansatz leicht modifizieren. Wendet man ihn unverändert auch auf nicht Endzeitstempel-monotone Eingabeströme an, so entfernt die Löschstruktur immer nur Elemente am Anfang der FiFo-Warteschlange, bis diese das zeitliche Löschprädikat nicht mehr erfüllen. Somit werden keine Elemente verfrüht gelöscht. Vielmehr verbleiben jedoch gegebenenfalls weiter hinten in der Warteschlange Elemente, die bereits entfernt werden könnten. Dies stellt insofern ein Problem dar, dass nachfolgende Anfragen an die SweepArea nun auch Elemente als Ergebnisse liefern, die eigentlich schon entfernt sein müssten. Grund dafür ist, dass das Nichterfülltsein der Entfernungsprädikate aus Gleichung 9.3 beziehungsweise 9.10 als Voraussetzung wegfällt. Dieses Problem kann aber dadurch gelöst werden, dass bei Anfragen zusätzlich explizit überprüft wird, dass sich die Gültigkeitsintervalle des anfragenden und des gefundenen Elementes schneiden.

Insgesamt senkt diese Variante des späten Löschens also die Kosten für das Entfernen aus der Löschstruktur, erhöht aber durch den zusätzlichen Intervallschnitttest die Kosten des Anfrageprädikates und durch die verspätet gelöschten Elemente den Speicherverbrauch. Da die Kosten für den Heap als Löschstruktur logarithmisch mit der Anzahl der Elemente steigen, lohnt der Einsatz des späten Löschens bei SweepAreas mit wenigen Elementen nicht. Anderenfalls kann sie auch nur dann ihre Stärken ausspielen, wenn keine Elemente die Ordnung auf den Endzeitstempeln grob verletzen, da sonst ihre Nachteile überwiegen.

Zeitliches Partitionieren

In [GÖ05] wird vorgeschlagen, Statusstrukturen nach dem Endzeitstempel zu partitionieren und diese je nach Löschstrategie zu sortieren. Die Suche nach zu entfernenden Elementen beschränkt sich damit auf die vorderen Partitionen. Der in dieser Arbeit für die zeitbezogene Sortierung von Löschstrukturen (siehe 9.3.5) eingesetzte Heap hat jedoch ebenfalls die Eigenschaft, dass Operationen darauf im Wesentlichen die Elemente vergleichen, die kurz vor dem Ablauf stehen.

10.2.2. Zeitlicher Fortschritt

Die meisten Optimierungsmaßnahmen beschäftigen sich mit Problemen, die auf Grund hoher Last, also insbesondere hohen Datenraten und langen Gültigkeitsintervallen, entstehen. Allerdings kann auch eine extrem niedrige Datenrate eines physischen Datenstromes ein Problem darstellen. Dies trifft insbesondere auch für die Verbundoperation zu. Das Löschen aus der SweepAreas zu einem Eingabestrom erfolgt nämlich nur dann, wenn

im anderen Eingabestrom ein Stromelement eintrifft. Liefert eine Eingabe also lange Zeit keine Stromelemente, die andere aber häufig, so müssen die von dieser gelieferten Stromelemente lange Zeit gespeichert werden und verursachen so hohen Speicherbedarf. Allerdings treten in der Praxis durchaus solche Ströme auf, die oft lange Zeit gar keine Stromelemente liefern, beispielsweise solche, deren Stromelemente Ausnahmesituationen oder Alarme ausdrücken.

Die grundsätzliche Ursache für das Problem ist, dass der zeitliche Fortschritt der Startzeitstempel für einen Operator nur dann erkennbar ist, wenn auch gültige Stromelemente mit einem dieser Startzeitstempel existieren und eintreffen. Als Lösung bietet es sich daher an, einen Mechanismus zu schaffen, der es einem Strom innerhalb eines DSMS erlaubt, sein logisches zeitliches Fortschreiten unabhängig von Stromelementen anzuzeigen.[SW04a, JMSS05] Die zeitlichen Fortschrittsinformationen werden dabei üblicherweise als *Heartbeats* bezeichnet. Wird auf diese Art der zeitliche Fortschritt eines Stromes bis zu einem Zeitpunkt $\check{t}$ angezeigt, so dürfen in diesem nachfolgend keine Stromelemente mit Startzeitstempel kleiner als $\check{t}$ mehr eintreffen. Die in dieser Arbeit vorgestellten Verbundimplementierung können mit Hilfe dieser Information dieselben Löschoperationen auf der gegenüberliegenden SweepArea durchführen, die das Eintreffen eines Stromelementes mit Startzeitstempel $\check{t}$ erlaubt hätte.

Während die Datenquellen im System zeitlichen Fortschritt beispielsweise anhand externer Informationen ermitteln und weiterleiten können, müssen physische Operatoren anhand ihrer Eingabeströme und Berechnungen entscheiden, welchen zeitlichen Fortschritt sie garantieren können. Dies entspricht im Wesentlichen der Frage, bis zu welchem Zeitstempel Ergebnisse weitergeleitet werden können.

Beim in 9.3.2 erläuterten ersten Algorithmus zur Verbundberechnung entspricht der zeitliche Fortschritt dem Fortschritt des Zeitpunktes, bis zu dem die Eingabewarteschlangen verarbeitet wurden. Meldungen über zeitlichen Fortschritt reihen sich sozusagen in die jeweilige Eingabewarteschlange ein. Allerdings sind nur solche von echtem Interesse, die am Ende der Warteschlange stehen. Denn für alle Zeitpunkte zwischen den Startzeitstempeln von Stromelementen in einer Eingabewarteschlange ist dem Algorithmus der zeitliche Fortschritt ohnehin bekannt. Dies erlaubt zusätzliche Entfernungsschritte, die im Algorithmus so zunächst nicht vorgesehen sind. Sie sind dann interessant, wenn sich die beiden Eingaberaten stark unterscheiden.

Beim in 9.3.3 erläuterten zweiten Algorithmus entspricht er dem Zeitpunkt, bis zu dem Ergebnisse aus der Ausgabewarteschlange weitergeleitet werden dürfen. Analog zu dessen Bestimmung empfiehlt es sich dabei, lediglich konservativ approximativ vorzugehen, da eine exakte Berechnung zusätzlichen Aufwand bedeuten würde.

Damit die Weiterleitung des zeitlichen Fortschritts nicht ihrerseits erheblichen Aufwand verursacht, ist es zudem wichtig, nicht jeden einzelnen theoretisch feststellbaren Fortschritt zu melden. Dazu kann man beispielsweise eine Anzahl von Zeiteinheiten festlegen, um die der Strom mindestens fortschreiten muss, wenn man das zeitliche Fortschreiten anzeigt. Mit diesem Parameter kann dann das Verhältnis zwischen Kosten und Nutzen reguliert werden. Seine Wahl hängt aber auch von der konkreten Anwendung ab, da er beispielsweise auch Auswirkungen darauf hat, wie eine Senke im System einem Benutzer Informationen präsentieren kann.

10.2.3. Scheduling

Da für die Ausführung eines DSMS nur endliche CPU-Ressourcen zur Verfügung stehen, wird mit Hilfe verschiedenster Optimierungen versucht, den Ressourcenbedarf zu verringern. Zusätzlich wird auch vielfach die Frage betrachtet, wie die CPU-Ressourcen zur Laufzeit optimal eingesetzt werden sollten.[CHK+07, CCZ+03, JC04, VN02, VNB03] Bei der aktiven Datenverarbeitung ist zumeist eine Vielzahl an Threads gleichzeitig aktiv, die auf zumeist mehreren verfügbaren Prozessoren ausgeführt werden. Dabei kommen für das Scheduling verschiedene Optimierungsziele in Frage, wie beispielsweise die Minimierung der Verzögerung zwischen Eintreffen eines Eingabeelementes und Produktion der zugehörigen Ergebnisse oder die Minimierung des maximal benötigten Speichers.

Allerdings kann ein DSMS nur dann funktionieren, wenn im längerfristigen Mittel mehr CPU-Ressourcen zur Verfügung stehen, als zur Ausführung aller aktiven Anfragen benötigt werden, da sonst bei andauernder Überlast zwangsweise der Speicherverbrauch durch Puffern von Eingabeelementen kontinuierlich ansteigt, was dann zwingend dazu führt, dass das System kollabiert oder Maßnahmen ergreifen muss, welche die Last reduzieren. Stehen andererseits im Mittel genug CPU-Ressourcen zur Verfügung, so spielt das Scheduling vorwiegend bei vorübergehender Überlast eine wichtige Rolle.

Die Verbundoperation kann insbesondere dann für vorübergehende Lastschübe sorgen, wenn die Selektivität im Mittel gering ist, aber einzelne Stromelemente besonders viele Verbundpartner finden und demnach Ergebnisse produzieren. Lässt man das DSMS beim Eintreffen eines solchen Elementes zunächst alle Ergebnisse komplett in möglicherweise teuren nachfolgenden Operatoren weiterverarbeiten, bevor der Verbund das nächste Eingabeelement akzeptiert, so kann dies zu einem Rückstau von Elementen führen, der sich gegebenenfalls auch auf andere Pfade des Anfragegraphen auswirkt. Daher kann es in solchen Fällen empfehlenswert sein, die Ergebnisse des Verbundes in eine Warteschlange einzufügen, die dann von einem eigenen Thread weiterverarbeitet wird. Auf diese Art kann die Warteschlange abgearbeitet werden, während der Verbund bereits nachfolgende Eingabeelemente verarbeitet.

Die Verbundoperation selber kann ebenfalls ein Problem für das Scheduling darstellen, und zwar, wenn die Verarbeitung eines Stromelementes lange Zeit in Anspruch nehmen kann. Eine Konstellation, in der dies vorkommen kann, besteht, wenn ein Eingabestrom extrem viele und der andere nur wenige Elemente liefert. Ein Stromelement aus der zweiten Eingabe fragt dann eine große SweepArea an, was in Abhängigkeit von der dort verwendeten Anfragestruktur hohen Aufwand verursachen kann. Die Wahl der Anfragestruktur hängt dabei wiederum wesentlich vom Verbundprädikat ab. Allgemein kann natürlich ein kompliziertes Verbundprädikat die Verarbeitung innerhalb des Verbundes langwierig gestalten. Für Eingaben, deren Elemente im Verbund gegebenenfalls aufwendige Berechnungen auslösen, kann es daher sinnvoll sein, sie vor dem Verbund in einer Warteschlange zu puffern und diese in einem eigenen Thread abzuarbeiten.

Die Zahl der zur Abarbeitung zusätzlich eingefügter Warteschlangen eingesetzten Threads sollte natürlich nicht Überhand nehmen, weshalb es sich wiederum empfiehlt, unter einer begrenzten Zahl von Threads die Aufgabe der Abarbeitung der Warteschlangen aufzuteilen. Für Details zum Thema DSMS-Scheduling sei an dieser Stelle auf [CHK+07]

verwiesen.

10.2.4. Selbstverbund

Für den durchaus praxisrelevanten Spezialfall, dass beide Eingabedatenströme des binären Verbundes identisch sind, kann dessen Verarbeitung in vielen Fällen erheblich optimiert werden. Bei der Ausführung einer der allgemeinen Varianten würden nämlich für jedes Element des Eingabedatenstromes sämtliche Schritte zweimal durchgeführt, einmal für das Eintreffen von links und einmal für das von rechts. Der Unterschied läge lediglich in den beteiligten SweepAreas, wobei beide immer dieselben Elemente enthalten, bis auf die Ausnahme, dass beim zweiten Anfragen das Element selber bereits in die angefragte SweepArea eingefügt ist.

Beim Einsatz von zweistufigen SweepAreas können sich beide daher auf jeden Fall dieselbe Löschstruktur teilen. Die zu löschenden Elemente müssen dann nur einmal ermittelt und aus beiden Anfragestrukturen entfernt werden.

Doch auch die Anfragestrukturen lassen sich in vielen Fällen zusammenführen. Dazu muss lediglich eine gewählt werden, die das Verbundprädikat aus der Perspektive beider Seiten unterstützen kann. Dies ist trivialerweise bei allen symmetrischen Prädikaten der Fall. Aber auch die SweepAreas für ein allgemeines Prädikat, die ohnehin alle Paare prüfen, können das Prädikat natürlich ein zweites Mal mit vertauschten Parametern testen. Bei SweepAreas für Äquivalenzklassenverbünde überträgt sich dieses Prinzip auf die einzelnen Klassen, wodurch diese SweepAreas ihre gute Performanz erhalten.

Im Falle des Zusammenführens der Anfragestrukturen müssen diese also zu einem Element alle Gegenstücke liefern, die damit von einer der Seiten das Verbundprädikat erfüllen. Für alle solchen Paare wird dann in der richtigen Orientierung die Ergebnisfunktion aufgerufen. Zusätzlich muss noch das anfragende Element einmalig mit sich selbst auf Erfüllung des Verbundprädikates geprüft und gegebenenfalls auch daraus ein Ergebnis produziert werden. Als Alternative könnte man das Element auch abweichend von der üblichen Reihenfolge vor der Anfrage in die SweepArea einfügen, müsste dann allerdings eine entsprechende Duplikateliminierung leisten.

Eine weitere wichtige Optimierung im Falle des Selbstverbundes wird dadurch ermöglicht, dass bei diesem die Eingabeelemente ohne weiteres Zutun global sortiert nach Startzeitstempeln eintreffen. Dies ermöglicht des Einsatz von Algorithmus 7 ohne Verwendung der Eingabewarteschlangen.

10.3. Mehrdimensionaler Verbund

Die Verbundoperation auf physischen Datenströmen verbindet zwei physische Datenströme, indem sie Paare aus je einem Element der Eingabeströme bildet und mit Hilfe einer Ergebnisfunktion daraus ein Ergebnis produziert, falls sich die Gültigkeitsintervalle der Stromelemente in einem Paar schneiden und die darin enthaltenen Elemente das Verbundprädikat erfüllen. Diese Grundidee lässt sich analog zum Vorgehen in DBMS für mehr als zwei Eingabeströme erweitern, wobei aus Paaren Tupel werden und Verbundprädikat sowie Ergebnisfunktion diese entsprechend als Definitionsbereich erhalten.

Der auf diese Weise entstehende mehrdimensionale Verbund hat nicht nur für sich eine Berechtigung, sondern kann auch zur Optimierung der Berechnung mehrerer binärer Verbünde benutzt werden.

10.3.1. Definition

Die mehrdimensionale Verbundoperation wird aus Gründen der Übersichtlichkeit hier direkt auf physischen Datenströmen definiert. Die Definition auf logischen Datenströmen kann direkt daraus hergeleitet werden.

Definition 10.6 (Mehrdimensionale Verbundoperation auf physischen Datenströmen)**.** Seien $1 < n \in \mathbb{N}$, $\mathcal{T}_1, \ldots, \mathcal{T}_n$ und $\mathcal{T}$ Typen, P ein Prädikat auf $\Omega_{\mathcal{T}_1} \times \ldots \times \Omega_{\mathcal{T}_n}$ und $r : \Omega_{\mathcal{T}_1} \times \ldots \times \Omega_{\mathcal{T}_n} \to \Omega_{\mathcal{T}}$ eine Ergebnisfunktion. Der *allgemeine mehrdimensionale Verbund über physischen Datenströmen* $\bowtie^p_{P,r}\colon \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n} \to \mathbb{S}^p_{\mathcal{T}}$ ist definiert durch

$$\bowtie^p_{P,r}(S_1, \ldots, S_n) := \omega\Big(f : (\Omega_{\mathcal{T}} \times I_{\mathbb{T}}) \to \mathbb{N}, f((e,i)) = \sum_{\substack{e_1 \in \Omega_{\mathcal{T}_1}, \ldots, e_n \in \Omega_{\mathcal{T}_n}, i_1 \in I_{\mathbb{T}}, \ldots, i_n \in I_{\mathbb{T}}: \\ P(e_1,\ldots,e_n) \wedge e = r(e_1,\ldots,e_n) \wedge i = i_1 \cap \ldots \cap i_n}} \bar{\omega}(S_1)(e_1, i_1) \cdot \ldots \cdot \bar{\omega}(S_n)(e_n, i_n)\Big)$$

Die Definition stellt eine Verallgemeinerung des allgemeinen Verbundes über physischen Datenströmen dar, wie das folgende Korollar zeigt.

Korollar 10.2. *Der allgemeine Verbund über physischen Datenströmen ist ein Spezialfall des allgemeinen mehrdimensionalen Verbundes über physischen Datenströmen.*

Beweis. Man wähle $n := 2$. □

10.3.2. Einsatzmöglichkeiten

Der allgemeine mehrdimensionale Verbund über physischen Datenströmen kann nicht nur dann eingesetzt werden, wenn er von vorne herein vorgesehen war, sondern er kann auch zu Optimierungszwecken eingesetzt werden. Die Grundlage dafür liefert das folgende Lemma.

Lemma 10.2. *Seien $k \in \mathbb{N}$ und $\mathcal{T}^e_1, \ldots, \mathcal{T}^e_k$ sowie $\mathcal{T}^a$ Typen und $g : \mathbb{S}^p_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}^e_k} \to \mathbb{S}^p_{\mathcal{T}^a}$ ein physischer Operatorgraph, der ein Baum ist und nur aus den physischen Operatoren allgemeiner Verbund über physischen Datenströmen und mehrdimensionaler Verbund über physischen Datenströmen besteht. Dann ist g physisch äquivalent zu einem physischen Operatorgraphen $\bar{g} : \mathbb{S}^p_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}^e_k} \to \mathbb{S}^p_{\mathcal{T}^a}$, der lediglich aus einem mehrdimensionalen Verbund über physischen Datenströmen besteht.*

Beweis. Da die Aussage die physische Äquivalenz fordert, kann zum Beweis nicht die Schnappschuss-Reduzierung auf entsprechende Ergebnisse aus der erweiterten relationalen Algebra genutzt werden. Der Beweis erfolgt daher per struktureller Induktion über die Baumstruktur von g.

Induktionsanfang: Die atomaren Bestandteile von g sind der allgemeine Verbund über physischen Datenströmen und der mehrdimensionale Verbund über physischen Datenströmen. Mit Korollar 10.2 sind beide trivialerweise äquivalent zu einem mehrdimensionalen Verbund über physischen Datenströmen.

Induktionsvoraussetzung: Alle Teilbäume von g mit weniger Knoten als g sind äquivalent zu einem physischen Operatorgraphen der lediglich aus einem mehrdimensionalen Verbund über physischen Datenströmen besteht.

Induktionsschritt: Man betrachte die Wurzel w des Anfragebaumes g, die wegen Korollar 10.2 o.B.d.A. ein mehrdimensionaler Verbund über physischen Datenströmen ist. Dann existieren ein $n \in \mathbb{N}$, Typen $\mathcal{T}_1, \ldots, \mathcal{T}_n$, ein Prädikat P auf $\Omega_{\mathcal{T}_1} \times \ldots \times \Omega_{\mathcal{T}_n}$ und eine Ergebnisfunktion $r : \Omega_{\mathcal{T}_1} \times \ldots \times \Omega_{\mathcal{T}_n} \to \Omega_{\mathcal{T}^a}$, so dass w den mehrdimensionalen Verbund $\bowtie^p_{P,r}: \mathbb{S}^p_{\mathcal{T}_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_n} \to \Omega_{\mathcal{T}^a}$ über physischen Datenströmen $S_1, \ldots, S_n$ berechnet.

Nach Induktionsvoraussetzung existieren zudem $m_1, \ldots, m_n \in \mathbb{N}$, Typen $\mathcal{T}_1^{s_1}, \ldots, \mathcal{T}_{m_1}^{s_1}$, $\ldots, \mathcal{T}_1^{s_n}, \ldots, \mathcal{T}_{m_n}^{s_n}$, sowie für $i \in \{1, \ldots, n\}$ jeweils ein Prädikat P_i auf $\Omega_{\mathcal{T}_1^{s_i}} \times \ldots \times \Omega_{\mathcal{T}_{m_i}^{s_i}}$ und eine Ergebnisfunktion $r_i : \Omega_{\mathcal{T}_1^{s_i}} \times \ldots \times \Omega_{\mathcal{T}_{m_i}^{s_i}} \to \Omega_{\mathcal{T}_i}$, so dass S_i vom mehrdimensionalen Verbund über physischen Datenströmen $\bowtie^p_{P_i,r_i}: \mathbb{S}^p_{\mathcal{T}_1^{s_i}} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_{m_i}^{s_i}} \to \Omega_{\mathcal{T}_i}$ über physischen Datenströmen $S_1^{s_i}, \ldots, S_{m_i}^{s_i}$ berechnet wird. Sind $S_1^e, \ldots S_k^e$ die Eingabeströme von g, so existiert zudem eine bijektive Abbildung zwischen $\{S_1^{s_1}, \ldots, S_{m_1}^{s_1}, \ldots, S_1^{s_n}, \ldots, S_{m_n}^{s_n}\}$ und $\{S_1^e, \ldots S_k^e\}$. Da man die Eingaben und Eingabetypen von g beliebig durchnummerieren kann, kann man o.B.d.A. annehmen, dass $S_1^{s_1} = S_1^e, \ldots, S_{m_n}^{s_n} = S_k^e$ gilt.

Seien für $x \in \{0, \ldots, n\}$ $\check{m}_x$ definiert durch $\check{m}_x := \sum_{y=1}^{x} m_y$

Man definiert nun das Prädikat $\bar{P}$ auf $\Omega_{\mathcal{T}_1^e} \times \ldots \times \Omega_{\mathcal{T}_k^e}$ durch $\bar{P}(e_1, \ldots, e_k) := P_1(e_1, \ldots, e_{m_1}) \wedge \ldots \wedge P_n(e_{k-m_n+1}, \ldots, e_k) \wedge P(r_1(e_1, \ldots, e_{m_1}), \ldots, r_n(e_{k-m_n+1}, \ldots, e_k))$ und die Ergebnisfunktion $\bar{r} : \Omega_{\mathcal{T}_1^e} \times \ldots \times \Omega_{\mathcal{T}_k^e} \to \Omega_{\mathcal{T}^a}$ durch $\bar{r}(e_1, \ldots, e_k) := r(r_1(e_1, \ldots, e_{m_1}), \ldots, r_n(e_{k-m_n+1}, \ldots, e_k))$.

Dann gilt für den mehrdimensionalen Verbund über physischen Datenströmen $\bowtie^p_{\bar{P},\bar{r}}$: $\mathbb{S}^p_{\mathcal{T}_1^e} \times \ldots \times \mathbb{S}^p_{\mathcal{T}_k^e} \to \Omega_{\mathcal{T}^a}$:

$$
\begin{aligned}
&\bowtie^{p}_{\bar{P},\bar{r}}(S_1,\ldots,S_k) = \\
&\omega\Big(f:(\Omega_{\mathcal{T}}\times I_{\mathbb{T}})\to\mathbb{N}, f((e,i)) = \\
&\qquad\sum_{\substack{e_1\in\Omega_{\mathcal{T}^e_1},\ldots,e_k\in\Omega_{\mathcal{T}^e_k}, i_1\in I_{\mathbb{T}},\ldots,i_k\in I_{\mathbb{T}}:\\ \bar{P}(e_1,\ldots,e_k)\wedge e=\bar{r}(e_1,\ldots,e_k)\wedge i=i_1\cap\ldots\cap i_k}}\ \prod_{j=1}^{k}\bar{\omega}(S_j)(e_j,i_j)\Big) = \\
&\omega\Big(f:(\Omega_{\mathcal{T}}\times I_{\mathbb{T}})\to\mathbb{N}, f((e,i)) = \\
&\qquad\sum_{\substack{e_1\in\Omega_{\mathcal{T}^e_1},\ldots,e_k\in\Omega_{\mathcal{T}^e_k}, i_1\in I_{\mathbb{T}},\ldots,i_k\in I_{\mathbb{T}}:\\ P_1(e_1,\ldots,e_{m_1})\wedge\ldots\wedge P_n(e_{k-m_n+1},\ldots,e_k)\wedge\bar{P}(r_1(e_1,\ldots,e_{m_1}),\ldots,r_n(e_{k-m_n+1},\ldots,e_k))\wedge\\ e=r(r_1(e_1,\ldots,e_{m_1}),\ldots,r_n(e_{k-m_n+1},\ldots,e_k))\wedge\\ i=i_1\cap\ldots\cap i_k}}\ \prod_{j=1}^{k}\bar{\omega}(S_j)(e_j,i_j)\Big) = \\
&\omega\Big(f:(\Omega_{\mathcal{T}}\times I_{\mathbb{T}})\to\mathbb{N}, f((e,i)) = \\
&\qquad\sum_{\substack{e_1\in\Omega_{\mathcal{T}^e_1},\ldots,e_k\in\Omega_{\mathcal{T}^e_k},\\ i_1\in I_{\mathbb{T}},\ldots,i_k\in I_{\mathbb{T}}:\\ P(e_1,\ldots,e_k)\wedge e=r(e_1,\ldots,e_k)\wedge\\ i=i_1\cap\ldots\cap i_k}}\ \prod_{j=1}^{k}\ \sum_{\substack{\hat{e}_1\in\Omega_{\mathcal{T}^e_{1+\check{m}_{j-1}}},\ldots,\hat{e}_{m_j}\in\Omega_{\mathcal{T}^e_{\check{m}_j}},\\ \hat{i}_1\in I_{\mathbb{T}},\ldots,\hat{i}_{m_j}\in I_{\mathbb{T}}:\\ P_j(\hat{e}_1,\ldots,\hat{e}_{m_j})\wedge e_j=r_j(\hat{e}_1,\ldots,\hat{e}_{m_j})\wedge\\ i_j=\hat{i}_1\cap\ldots\cap\hat{i}_{m_j}}}\ \prod_{l=1+\check{m}_{j-1}}^{\check{m}_j}\bar{\omega}(S_l)(\hat{e}_l,\hat{i}_l)\Big) = \\
&\omega\Big(f:(\Omega_{\mathcal{T}}\times I_{\mathbb{T}})\to\mathbb{N}, f((e,i)) = \\
&\qquad\sum_{\substack{e_1\in\Omega_{\mathcal{T}^e_1},\ldots,e_k\in\Omega_{\mathcal{T}^e_k}, i_1\in I_{\mathbb{T}},\ldots,i_k\in I_{\mathbb{T}}:\\ P(e_1,\ldots,e_k)\wedge e=r(e_1,\ldots,e_k)\wedge i=i_1\cap\ldots\cap i_k}}\ \prod_{j=1}^{k}\bar{\omega}\Big(\omega\Big(f:(\Omega_{\mathcal{T}^e_j}\times I_{\mathbb{T}})\to\mathbb{N}, f((\hat{e},\hat{i})) = \\
&\qquad\qquad\sum_{\substack{\hat{e}_1\in\Omega_{\mathcal{T}^e_{1+\check{m}_{j-1}}},\ldots,\hat{e}_{m_j}\in\Omega_{\mathcal{T}^e_{\check{m}_j}},\hat{i}_1\in I_{\mathbb{T}},\ldots,\hat{i}_{m_j}\in I_{\mathbb{T}}:\\ P_j(\hat{e}_1,\ldots,\hat{e}_{m_j})\wedge\hat{e}=r_j(\hat{e}_1,\ldots,\hat{e}_{m_j})\wedge\hat{i}=\hat{i}_1\cap\ldots\cap\hat{i}_{m_j}}}\ \prod_{l=1+\check{m}_{j-1}}^{\check{m}_j}\bar{\omega}(S_l)(\hat{e}_l,\hat{i}_l)\Big)(e_j,i_j)\Big)\Big) = \\
&\omega\Big(f:(\Omega_{\mathcal{T}}\times I_{\mathbb{T}})\to\mathbb{N}, f((e,i)) = \\
&\qquad\sum_{\substack{e_1\in\Omega_{\mathcal{T}^e_1},\ldots,e_k\in\Omega_{\mathcal{T}^e_k}, i_1\in I_{\mathbb{T}},\ldots,i_k\in I_{\mathbb{T}}:\\ P(e_1,\ldots,e_k)\wedge e=r(e_1,\ldots,e_k)\wedge i=i_1\cap\ldots\cap i_k}}\ \prod_{j=1}^{k}\bar{\omega}(\bowtie^{p}_{P_j,r_j}(S_{1+\check{m}_{j-1}},\ldots,S_{\check{m}_j})(e_j,i_j))\Big) = \\
&\bowtie^{p}_{P,r}(\bowtie^{p}_{P_1,r_1}(S_1,\ldots,S_{m_1}),\ldots,\bowtie^{p}_{P_n,r_n}(S_{k-m_n+1},\ldots,S_k))
\end{aligned}
$$

□

Auf Basis dieses Lemmas können Teilgraphen eines Anfragegraphen, die eine Baumstruktur haben und nur aus Verbünden bestehen, durch einen einzigen mehrdimensiona-

len Verbund ersetzt werden. Ob dies sinnvoll ist, hängt natürlich vom Algorithmus, der den mehrdimensionalen Verbund berechnet, und insbesondere dessen Kosten ab, die in der Folge entwickelt werden.

Zuvor sei noch auf eine Feinheit beim praktischen Einsatz der in Lemma 10.2 angegebenen Transformation hingewiesen. Es kann nämlich vorkommen, dass in der Praxis die Ergebnisfunktion eines Verbundes nicht den kompletten in der Verbunddefinition geforderten Definitionsbereich abdeckt, sondern nur auf der Teilmenge definiert ist, für die auch das Verbundprädikat erfüllt ist. Das stellt normalerweise kein Problem dar, da die Ergebnisfunktion nur mit solchen Parametern aufgerufen wird, die zuvor gegen das Verbundprädikat getestet wurden. In der Transformation werden jedoch für das Verbundprädikat sowohl die Verbundprädikate, als auch die Ergebnisfunktionen der darunterliegenden Verbünde benutzt. Dabei muss daher beachtet werden, dass die Ergebnisfunktionen nur dann aufgerufen werden, wenn zuvor die zugehörigen Verbundprädikate erfüllt waren. Anderenfalls ist das gemeinsame Verbundprädikat ohnehin nicht erfüllbar.

Die Umkehrung von Lemma 10.2 gilt ebenso, wie das folgende Lemma zeigt, dessen Aussage noch etwas stärker formuliert ist.

Lemma 10.3. *Seien $k \in \mathbb{N}$ und $\mathcal{T}^e_1, \ldots, \mathcal{T}^e_k$ sowie $\mathcal{T}^a$ Typen und $g : \mathbb{S}^p_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}^e_k} \to \mathbb{S}^p_{\mathcal{T}^a}$ ein physischer Operatorgraph, der lediglich aus einem mehrdimensionalen Verbund über physischen Datenströmen besteht. Dann ist g physisch äquivalent zu einem physischen Operatorgraphen $\bar{g} : \mathbb{S}^p_{\mathcal{T}^e_1} \times \ldots \times \mathbb{S}^p_{\mathcal{T}^e_k} \to \mathbb{S}^p_{\mathcal{T}^a}$, der ein Baum ist und nur aus allgemeinen Verbünden über physischen Datenströmen besteht.*

Beweis. Sei P auf $\Omega_{\mathcal{T}^e_1} \times \ldots \times \Omega_{\mathcal{T}^e_n}$ das Verbundprädikat und $r : \Omega_{\mathcal{T}^e_1} \times \ldots \times \Omega_{\mathcal{T}^e_n} \to \Omega_{\mathcal{T}^a}$ die Ergebnisfunktion des mehrdimensionalen Verbundes in g.

Beweis durch Induktion über n: Induktionsanfang $n = 2$: Für $n = 2$ entspricht der mehrdimensionale Verbund nach Korollar 10.2 dem binären allgemeinen Verbund, g ist also bereits ein Baum mit nur einem Knoten, der ein allgemeiner Verbund ist.

Induktionsvoraussetzung: Ein mehrdimensionaler Verbund über physischen Datenströmen mit n Eingabeströmen ist physisch äquivalent zu einem Baum aus allgemeinen Verbünden über physischen Datenströmen.

Induktionsschritt ($n \to n + 1$): Man betrachte die Wurzel w des Anfragebaumes g, die ein mehrdimensionaler Verbund über $n + 1$ Eingabeströmen der Typen $\mathcal{T}^e_1, \ldots, \mathcal{T}^e_{n+1}$ mit einem Verbundprädikat P auf $\Omega_{\mathcal{T}_1} \times \ldots \times \Omega_{\mathcal{T}_{n+1}}$ und einer Ergebnisfunktion $r : \Omega_{\mathcal{T}_1} \times \ldots \times \Omega_{\mathcal{T}_{n+1}} \to \Omega_{\mathcal{T}^a}$ ist. Sei $\mathcal{T}^i$ ein Typ mit $\Omega_{\mathcal{T}^i} = \Omega_{\mathcal{T}_1} \times \ldots \times \Omega_{\mathcal{T}_n}$, $\tilde{P}$ ein Prädikat auf $\Omega_{\mathcal{T}^i} \times \Omega_{\mathcal{T}_{n+1}}$ mit $\tilde{P}((e_1, \ldots, e_n), e_{n+1}) := P(e_1, \ldots, e_{n+1})$ und $\tilde{r} : \Omega_{\mathcal{T}^i} \times \Omega_{\mathcal{T}_{n+1}} \to \Omega_{\mathcal{T}^a}$ eine Ergebnisfunktion mit $\tilde{r}((e_1, \ldots, e_n), e_{n+1}) := r(e_1, \ldots, e_{n+1})$. Weiterhin sei $\check{P}$ ein Prädikat auf $\Omega_{\mathcal{T}_1} \times \ldots \times \Omega_{\mathcal{T}_n}$ mit $\check{P}(e_1, \ldots, e_n) := \top$ und $r : \Omega_{\mathcal{T}_1} \times \ldots \times \Omega_{\mathcal{T}_n} \to \Omega_{\mathcal{T}^i}$ eine Ergebnisfunktion mit $r(e_1, \ldots, e_n) := (e_1, \ldots, e_n)$. Dann gilt:

$$
\begin{aligned}
&\bowtie^{p}_{P,r}(S_1,\ldots,S_{n+1}) = \\
&\omega\Big(f:(\Omega_{\mathcal{T}}\times I_{\mathbb{T}})\to\mathbb{N}, f((e,i)) = \\
&\qquad \sum_{\substack{e_1\in\Omega_{\mathcal{T}^e_1},\ldots,e_{n+1}\in\Omega_{\mathcal{T}^e_{n+1}}, i_1\in I_{\mathbb{T}},\ldots,i_{n+1}\in I_{\mathbb{T}}:\\ P(e_1,\ldots,e_{n+1})\wedge e=r(e_1,\ldots,e_{n+1})\wedge i=i_1\cap\ldots\cap i_{n+1}}} \prod_{j=1}^{n+1}\bar{\omega}(S_j)(e_j,i_j)\Big) = \\
&\omega\Big(f:(\Omega_{\mathcal{T}}\times I_{\mathbb{T}})\to\mathbb{N}, f((e,i)) = \\
&\qquad \sum_{\substack{e_1\in\Omega_{\mathcal{T}_1},\ldots,e_{n+1}\in\Omega_{\mathcal{T}_{n+1}},\\ \bar{i}\in I_{\mathbb{T}}, i_{n+1}\in I_{\mathbb{T}}:\\ P(e_1,\ldots,e_{n+1})\wedge e=r(e_1,\ldots,e_{n+1})\wedge\\ i=\bar{i}\cap i_{n+1}}} \Bigg(\sum_{\substack{i_1\in I_{\mathbb{T}},\ldots,i_n\in I_{\mathbb{T}}:\\ \bar{i}=i_1\cap\ldots\cap i_n}} \prod_{j=1}^{n}\bar{\omega}(S_j)(e_j,i_j)\Bigg)\cdot\bar{\omega}(S_{n+1})(e_{n+1},i_{n+1})\Big) = \\
&\omega\Big(f:(\Omega_{\mathcal{T}}\times I_{\mathbb{T}})\to\mathbb{N}, f((e,i)) = \\
&\qquad \sum_{\substack{\bar{e}\in\Omega_{\mathcal{T}^i},e_{n+1}\in\Omega_{\mathcal{T}_{n+1}},\\ \bar{i}\in I_{\mathbb{T}}, i_{n+1}\in I_{\mathbb{T}}:\\ \bar{P}(\bar{e},e_{n+1})\wedge e=\bar{r}(\bar{e},e_{n+1})\wedge\\ i=\bar{i}\cap i_{n+1}}} \Bigg(\sum_{\substack{e_1\in\Omega_{\mathcal{T}^e_1},\ldots,e_n\in\Omega_{\mathcal{T}^e_n},\\ i_1\in I_{\mathbb{T}},\ldots,i_n\in I_{\mathbb{T}}:\\ \bar{e}=(e_1,\ldots,e_n)\wedge\\ \bar{i}=i_1\cap\ldots\cap i_n}} \prod_{j=1}^{n}\bar{\omega}(S_j)(e_j,i_j)\Bigg)\cdot\bar{\omega}(S_{n+1})(e_{n+1},i_{n+1})\Big) = \\
&\omega\Big(f:(\Omega_{\mathcal{T}}\times I_{\mathbb{T}})\to\mathbb{N}, f((e,i)) = \\
&\qquad \sum_{\substack{\bar{e}\in\Omega_{\mathcal{T}^i},e_{n+1}\in\Omega_{\mathcal{T}_{n+1}},\bar{i}\in I_{\mathbb{T}},i_{n+1}\in I_{\mathbb{T}}:\\ \bar{P}(\bar{e},e_{n+1})\wedge e=\bar{r}(\bar{e},e_{n+1})\wedge i=\bar{i}\cap i_{n+1}}} \bar{\omega}(\omega(f:(\Omega_{\mathcal{T}^i}\times I_{\mathbb{T}})\to\mathbb{N}, f((\hat{e},\hat{i})) = \\
&\qquad\qquad \sum_{\substack{e_1\in\Omega_{\mathcal{T}^e_1},\ldots,e_n\in\Omega_{\mathcal{T}^e_n},i_1\in I_{\mathbb{T}},\ldots,i_n\in I_{\mathbb{T}}:\\ \check{P}(e_1,\ldots,e_n)\wedge\hat{e}=\check{r}(e_1,\ldots,e_n)\wedge\hat{i}=i_1\cap\ldots\cap i_n}} \prod_{j=1}^{n}\bar{\omega}(S_j)(e_j,i_j)))(\bar{e},\bar{i})\cdot\bar{\omega}(S_{n+1})(e_{n+1},i_{n+1})\Big) = \\
&\omega\Big(f:(\Omega_{\mathcal{T}}\times I_{\mathbb{T}})\to\mathbb{N}, f((e,i)) = \\
&\qquad \sum_{\substack{\bar{e}\in\Omega_{\mathcal{T}^i},e_{n+1}\in\Omega_{\mathcal{T}_{n+1}},\bar{i}\in I_{\mathbb{T}},i_{n+1}\in I_{\mathbb{T}}:\\ \tilde{P}(\bar{e},e_{n+1})\wedge e=\tilde{r}(\bar{e},e_{n+1})\wedge i=\bar{i}\cap i_{n+1}}} \bar{\omega}(\bowtie^{p}_{\check{P},\check{r}}(S_1,\ldots,S_n))(\bar{e},\bar{i})\cdot\bar{\omega}(S_{n+1})(e_{n+1},i_{n+1})\Big) = \\
&\bowtie^{p}_{\tilde{P},\tilde{r}}(\bowtie^{p}_{\check{P},\check{r}}(S_1,\ldots,S_n),S_{n+1})
\end{aligned}
$$

Dabei ist $\bowtie^{p}_{\tilde{P},\tilde{r}}$ ein physischer binärer allgemeiner Verbund, und $\bowtie^{p}_{\check{P},\check{r}}$ lässt sich nach Induktionsvoraussetzung physisch äquivalent zu einem Baum aus allgemeinen Verbünden über physischen Datenströmen umformen. □

10.3.3. Implementierung

An dieser Stelle wird nun zunächst aufgezeigt, wie der mehrdimensionale Verbund implementiert werden kann.

Motivation

Zur Entwicklung eines geeigneten Algorithmus bietet es sich natürlich an, die Algorithmen für den binären Fall um weitere Eingaben und SweepAreas zu erweitern. Allerdings müssen dafür die Details neu betrachtet werden, da einige davon sich im mehrdimensionalen Fall komplizierter verhalten als im binären Spezialfall.

Zeitlicher Fortschritt Der zeitliche Fortschritt wird wie im binären Fall durch denjenigen Eingabestrom vorgegeben, der am spätesten eintrifft, also vom Minimum der auf den Eingabeströmen zuletzt eingetroffenen Startzeitstempel. Bis zu diesem Zeitpunkt können bei Verwendung von Eingabepuffern die Eingaben verarbeitet werden, und bis zu diesem Zeitpunkt können bei Verwendung eines Ausgabepuffers die Ergebnisse weitergeleitet werden, da keine mit kleinerem Startzeitstempel mehr entstehen können.

Löschen Mit analoger Argumentation können Elemente in den SweepAreas, deren Endzeitstempel kleiner oder gleich dem Minimum der Startzeitstempel der zuletzt aus den Eingabeströmen verarbeiteten Stromelemente ist, nicht mehr zu einem Ergebnis beitragen und daher aus ihrer SweepArea entfernt werden. Der letzte Startzeitstempel im zur SweepArea gehörigen Strom kann bei der Bildung dieses Minimums allerdings unberücksichtigt bleiben, da ja die Möglichkeiten betrachtet werden, mit dem Element aus der SweepArea ein Ergebnis zu bilden. Darin zeigt sich auch wieder die Analogie zum binären Fall, wo der letzte Startzeitstempel im anderen Strom eben auch das Minimum aller letzter Startzeitstempel in allen anderen Strömen ist.

Einfügen Eintreffende Stromelemente werden wie im binären Fall in die zu ihrer Eingabe zugehörige SweepArea eingefügt. Somit wird jedes Stromelement nur einmal und nur so lange wie nötig im Verbund gespeichert.

Anfragen Bezüglich der Bedingung, dass sich die Gültigkeitsintervalle der beteiligten Stromelemente schneiden müssen, übertragen sich die Ergebnisse aus dem binären Fall nur bei Verwendung der Eingabepufferung. Durch das vor dem Anfragen durchgeführte Löschen ist dann sichergestellt, dass das Gültigkeitsintervall angefragter, also noch in einer SweepArea befindliche Elemente, weder komplett vor dem des Anfragenden liegen kann (sonst wäre es zuvor gelöscht worden) noch komplett dahinter (sonst würde es später verarbeitet). Anfragen können also allein anhand des eigentlichen Verbundprädikates erfolgen.

Im Falle der Verwendung eines Ausgabepuffers, also bei Verarbeitung der Stromelemente in der Reihenfolge ihres Eintreffens, kann es schon im binären Fall vorkommen,

dass das Gültigkeitsintervall des anfragenden Elementes komplett vor dem des angefragten liegt, was sich in Gleichung 9.10 zeigt. Da im mehrdimensionalen Fall der Startzeitstempel des anfragenden Stromelementes nicht mehr allein für den Zeitpunkt, bis zu dem zuvor aus den SweepAreas gelöscht wird, verantwortlich ist, kann dessen Gültigkeit auch nach dem Ende des Gültigkeitsintervalls des angefragten Elementes beginnen. Daher muss neben dem Verbundprädikat auch immer die Schnittbedingung an die Gültigkeitsintervalle überprüft werden.

Die wesentlichste Änderung gegenüber dem binären Fall ergibt sich bei dem wertbasierten Teil der Anfrage an die SweepAreas. Trifft ein Stromelement ein, so müssen die SweepAreas aller anderen Eingaben nach passenden Elementen durchsucht werden. Im allgemeinsten Fall liegt jedoch nur das n-stellige Verbundprädikat P vor. Um gegen dieses testen zu können, müssten alle Tupel berechnet werden, die aus dem eingetroffenen Stromelement und denen in den anderen SweepAreas unter Berücksichtigung der Schnittbedingung an die Gültigkeitsintervalle gebildet werden können. Die Menge solcher Tupel kann jedoch sehr groß werden. Daher ist es anzustreben, analog zum binären Fall durch wertbasierte Anfragestrukturen die Zahl der Tests gegen das Verbundprädikat zu verringern. Insbesondere ist anzustreben, schon vor dem Aufbau kompletter Tupel einige Kombinationen ausschließen zu können.

Als Beispiel diene ein ternärer Verbund ($n = 3$) bei Benutzung von Eingabewarteschlangen (nach dem Löschen schneiden sich also die Gültigkeitsintervalle aller verbliebener Elemente). Es sei gerade ein Stromelement e_1 von Eingabe 1 eingetroffen und die SweepArea zu Eingabe i enthalte nach dem Löschen x_i Elemente. Die einfachste Lösung wäre nun, aus dem eingetroffenen Element und denen in den anderen SweepAreas alle x_2x_3 Kombinationen zu bilden und dann gegen das Prädikat P zu testen. Schon günstiger wäre es, nur die x_2 Kombinationen aus e_1 und den Elementen aus der zweiten SweepArea zu bilden, und dann über eine wertbasierte Anfragestruktur, die an P angepasst ist, für jedes Element e_2 jeweils die dritte SweepArea anzufragen. Dabei werden dann die Elemente z gesucht, so dass $P(e_1, e_2, z)$ gilt. Allerdings muss man auch nicht alle Elemente aus der zweiten SweepArea betrachten. Alle Elemente y, für die es keinen Wert $\tilde{z}$ vom Typ des dritten Eingabestromes gibt, für den $P(e_1, y, \tilde{z})$ erfüllt wäre, brauchen nicht für Anfragen an die dritte SweepArea verwendet werden. Konkretisiert für $P(x, y, z) := x > z > y$ ergibt sich im Beispiel also beim Eintreffen von e_1 folgendes Anfrageverhalten: Zuerst werden alle Elemente e_2 in der zweiten SweepArea gesucht, für die $e_1 > e_2$ gilt. Für jedes solche Paar (e_1, e_2) werden dann die Elemente e_3 in der dritten SweepArea gesucht, so dass $e_1 > e_3 > e_2$ gilt. Die daraus gebildeten Tripel (e_1, e_2, e_3) erfüllen das Verbundprädikat und werden als Ergebnisse gemeldet.

Im Beispiel wird beim Eintreffen eines Stromelementes der ersten Eingabe zunächst die SweepArea zur zweiten Eingabe angefragt, dann die zur dritten. Ebenso könnte man zunächst die zur dritten anfragen, und dabei alle Elemente e_3 ermitteln, für die $e_1 > e_3$ gilt, und danach die zur zweiten. Die Wahl der Anfragereihenfolge stellt also einen Parameter des Verfahrens dar.[VNB03, GWYL07] Prinzipiell kann diese sogar für jedes eintreffende Stromelement neu gewählt werden. Allerdings hängen auch die verwendeten Prädikate, zum Beispiel bezüglich ihrer Stelligkeit, von der jeweiligen Anfragereihenfolge ab, wobei jede der n Eingaben Anfragen mit $(n - 1)$ Prädikaten verursacht. Bei einem n-dimensionalen Verbund gibt es für jede der n Eingaben $(n - 1)!$ Möglichkeiten, die

Anfragereihenfolge festzulegen. Daher ist es schon ab $n > 4$ zweifelhaft, ob es Sinn macht, alle diese Prädikate bereitzuhalten. Für den hier vorgestellten Algorithmus wird daher die Anfragereihenfolge für jede Eingabe festgelegt, und dafür werden jeweils die benötigten Prädikate vorgehalten. Zur Optimierung kann diese, wie in 10.3.5 erläutert, gegebenenfalls zur Laufzeit geändert werden.

Für einen n-dimensionalen Verbund über Eingabeströmen der Typen $\mathcal{T}_1, \ldots, \mathcal{T}_n$ sei nun $o : \{1, \ldots, n\} \times \{1, \ldots, n\} \to \{1, \ldots, n\}$ die Funktion, für welche die Folge $o(i, 2), \ldots, o(i, n)$ die Anfragereihenfolge für Eingabe i angibt. Zusätzlich gelte $\forall i \in \{1, \ldots, n\} : o(i, 1) = i$, wodurch ausgedrückt wird, dass das anfragende Element den Anfang beim Aufbau der Ergebnistupel bildet. Die Anfrageprädikate an die SweepAreas können damit mathematisch angegeben werden. Ist dann ein Element x_i aus Eingabe i eingetroffen, wurden bereits k SweepAreas mit $0 \leq k < n-1$ angefragt und wird die aus x_i und den Ergebnissen gebildete Kombination $x_{o(i,1)}, \ldots, x_{o(i,k+1)}$ betrachtet, so werden bei der Anfrage an die SweepArea zu Eingabe $o(i, k+2)$ die Elemente $x_{o(i,k+2)} \in \Omega_{\mathcal{T}_{o(i,k+2)}}$ gesucht, für die gilt:

$$\exists (x_{o(i,k+3)}, \ldots, x_{o(i,n)}) \in \Omega_{\mathcal{T}_{o(i,k+3)}} \times \ldots \times \Omega_{\mathcal{T}_{o(i,n)}} : P(x_1, \ldots, x_n) \tag{10.1}$$

Das entspricht also genau den Elementen $x_{o(i,k+2)}$, durch die das ausgehend von x_i gebildete teilweise berechnete Ergebnis noch zu einem Tupel ausgebaut werden kann, welches das Verbundprädikat erfüllt.

Modifikation des binären Algorithmus

Die Umsetzung der Modifikationen vom binären zum mehrdimensionalen Verbundalgorithmus wird hier am Beispiel des in 9.3.3 vorgestellten Algorithmus mit Ausgabepuffer demonstriert.

Datenstruktur 9: SweepArea: Erweiterung für mehrdimensionale Verbünde

```
1  ...
2  Index i;
3  Anfragereihenfolge O;
4  ...
5  SweepArea(Index i, Prädikat p_query, Anfragereihenfolge o, Prädikat p_remove) {
6     Legt eine neue SweepArea mit den übergebenen Parametern an;
7  }
8  ...
9  Iterator query(Elemente e_1, ..., e_n) {
10    Liefert einen Iterator über (im Falle ∃k ≠ i : e_k = ⊥ eine Obermenge) alle Elemente t in der
      SweepArea, für die P_query(e_1, ..., e_n) erfüllbar ist, wenn e_i ← t und alle e_k mit e_k = ⊥ beliebig
      belegt werden können;
11 }
12 ...
```

SweepAreas Da der abstrakte Datentyp SweepArea in der in 9.2 vorgestellten Variante die im mehrdimensionalen Fall benötigte Anfragefunktionalität nicht unterstützt, muss er geeignet wie in Datenstruktur 9 angegeben modifiziert werden.

Beim Anlegen einer SweepArea können damit zusätzlich die Anfragereihenfolge und der Index der Eingabe, zu der sie gehört, übergeben werden. Die benötigte Anfragefunktionalität wird durch eine zweite Variante der Funktion query bereitgestellt, die angelehnt an Bedingung 10.1 spezifiziert ist. Diese wurde jedoch dahingehend abgeschwächt, dass die Erfüllbarkeit des Verbundprädikats P nur dann exakt geprüft werden muss, wenn $e_1, \ldots, e_n$ komplett belegt werden kann. In den Fällen mit $\exists k \neq i : e_k = \perp$ hingegen wird nur eine konservative Approximation der Lösung zu Bedingung 10.1 gefordert. Denn eine exakte Überprüfung der Bedingung würde gegebenenfalls eine erschöpfende Suche über den Wertebereich der Typen der Variablen erfordern, die durch die Existenzquantoren gebunden sind. Dies könnte jedoch sehr ineffizient oder bei unbegrenzten Wertebereichen sogar unmöglich sein.

Algorithmus Zur Umsetzung des Konzeptes, ausgehend von einem eingetroffenen Stromelement sukzessive Ergebnistupel aufzubauen, indem Anfragen an eine SweepArea gestellt und das Zwischenergebnis jeweils mit allen Ergebnissen weiter ausgebaut wird, bietet sich ein rekursives Vorgehen an. Algorithmus 10 zeigt die Funktion SucheErgebnisse, die dieses leistet. In einer konkreten Implementierung sollte allerdings aus Performanzgründen eine entrekursivierte Variante Verwendung finden, da dies viele Zuweisungen einspart.

Unter Verwendung von SucheErgebnisse kann nun Algorithmus 11 zur Berechnung des mehrdimensionalen Verbundes formuliert werden.

Der Algorithmus verarbeitet die eintreffenden Stromelemente weiterhin nach dem Prinzip der in Algorithmus 2 vorgestellten Funktion VerarbeiteElement. Zunächst wird es in die eigene SweepArea eingefügt. Dann wird der aus ihm resultierende zeitliche Fortschritt zum Löschen aus den anderen SweepAreas genutzt (optional könnte man hier auch aus der eigenen SweepArea löschen). Zum Löschen wird ein Hilfselement l erzeugt, von dem das Löschprädikat P_{remove} aber nur den Startzeitstempel benutzt, weshalb die Wahl des Wertes und des Endzeitstempels irrelevant sind.

Algorithmus 10: SucheErgebnisse

Input : SweepArea $S_1, \ldots,$ SweepArea S_n, Element $e_1, \ldots,$ Element e_n, Anfragereihenfolge o, Index i, Index j
Output : Elementetupelmenge E

```
1 E ← ∅;
2 if j ≤ n then
3 |   R ← S_o(i,j).query(e_1, ..., e_n);
4 |   foreach r ∈ R do
5 |   |   e_o(i,j) ← r;
6 |   |   E ← E ∪ SucheErgebnisse(S_1, ..., S_n, e_1, ..., e_n, o, i, j + 1);
7 else E ← {(e_1, ..., e_n)};
```

Algorithmus 11: Mehrdimensionaler Verbundalgorithmus

Input : physischer Eingabestrom $S_1, \ldots,$ physischer Eingabestrom S_n, Verbundprädikat P, Ergebnisfunktion r, Anfragereihenfolge o
Output : physischer Ausgabestrom S_r

1 $P_{query}((e_1, i_1), \ldots, (e_1, i_n)) := P(e_1, \ldots, e_n) \wedge i_1 \cap \ldots \cap i_n \neq \emptyset$;
2 $P_{remove}((e, [t_s, t_e)), (\hat{e}, [\hat{t}_s, \hat{t}_e))) := t_e \leq \hat{t}_s$;
3 $r'((e_1, i_1), \ldots, (e_n, i_n)) := (r(e_1, \ldots, e_n), i_1 \cap \ldots \cap i_n)$;
4 SweepArea $A_1 \leftarrow$ SweepArea$(1, P_{query}, o, P_{remove})$;
5 $\ldots$;
6 SweepArea $A_n \leftarrow$ SweepArea$(n, P_{query}, o, P_{remove})$;
7 $PriorityQueue_{t_s} Q \leftarrow \emptyset$;
8 $m_1 \leftarrow -\infty; \ldots m_n \leftarrow -\infty$;
9 **foreach** $s := (e, [t_s, t_e)) \hookleftarrow S_i$ **do**
10 S_i.insert(e);
11 $m_i \leftarrow t_s$;
12 $m \leftarrow min(m_1, \ldots, m_n)$;
13 $l \leftarrow (e, [m, m))$;
14 **foreach** $k \in \{1, \ldots, n\} \backslash \{i\}$ **do**
15 S_k.remove(l, i);
16 $Z \leftarrow$ SucheErgebnisse$(A_1, \ldots, A_n, \underbrace{\perp, \ldots, \perp}_{i-1}, (e, [t_s, t_e)), \underbrace{\perp, \ldots, \perp}_{n-i}, i, 2)$;
17 **foreach** $z \in Z$ **do**
18 $Q.enQueue(r'(z))$;
19 **while** $\neg Q.isEmpty()$ **do**
20 $(e', [t'_s, t'_e)) \leftarrow Q.peek()$;
21 **if** $t'_s \leq m$ **then**
22 $Q.deQueue() \hookrightarrow S_r$;
23 **else** break ;
24 **while** $\neg Q.isEmpty()$ **do**
25 $Q.deQueue() \hookrightarrow S_r$;

Nach Einfügen und Löschen erfolgt dann die Ergebnisproduktion mit Hilfe der Funktion SucheErgebnisse. Auf die Tupel, die das Verbundprädikat erfüllen, wird die Ergebnisfunktion r' angewendet, die sich aus r und der Schnittbedingung an die Gültigkeitsintervalle ergibt. Die fertig berechneten Ergebnisse werden dann analog zu Algorithmus 9.3.3 so lange in einer Ausgabewarteschlange gepuffert, bis der zeitliche Fortschritt ihre Weiterleitung zulässt oder alle Eingabestromelemente eingetroffen sind.

Korrektheit

Die Korrektheit des Verfahrens basiert analog zum binären Fall auf ähnlichen Überlegungen wie beim Sort-Merge-Join. Zunächst einmal wird durch die Wahl der Löschbedingung sichergestellt, dass kein Element gelöscht wird, solange es noch zu Ergebnissen beitragen könnte. Jedes Ergebnis basiert auf dem Zusammentreffen von je einem Stromelement pro

Eingabestrom, die gemeinsam die Verbundbedingung erfüllen, die, gegebenenfalls unter Ausnutzung der Löschbedingung, überprüft wird, weshalb keine inkorrekten Ergebnisse entstehen. Die Verbundbedingung wird im Zuge der Ergebniskonstruktion bei der Anfrage an die letzte SweepArea jeweils exakt getestet, die Zwischenergebnismengen sind jeweils konservative Approximationen. Die Vollständigkeit der Menge der produzierten Stromelemente wird sichergestellt, da das zuletzt eintreffende benötigte Stromelement die anderen in den SweepAreas durch Anfragen auffindet. Unzulässige Duplikate treten nicht auf, da jede Kombination erst beim Eintreffen des letzten beteiligten Stromelementes entstehen kann. Die zeitliche Ordnung des Ausgabestromes wird bei Verwendung von Eingabepuffern wieder durch die global sortierte Verarbeitung garantiert. Bei Verwendung eines Heaps als Ausgabepuffer stellt dieser die Ordnung sicher, da Ergebnisse erst weitergeleitet werden, wenn keine in der zeitlichen Ordnung vor ihnen stehenden mehr produziert werden können.

10.3.4. Wahl der SweepAreas

Wie schon im binären Fall kommt mit Blick auf die Effizienz des Verbundalgorithmus der Wahl der verwendeten SweepAreas eine Schlüsselrolle zu. Die prinzipielle Überlegung, dass das Löschen durch eine Organisation der Elemente nach Endzeitstempeln und das Anfragen durch eine Organisation nach Werten besonders effizient unterstützt werden, büßt dabei im mehrdimensionalen Fall kaum an Gültigkeit ein. Bei den Anfragen kommt zwar nun bei Verwendung des Algorithmus mit Ausgabepuffer die vollständige Überprüfung der Schnittbedingung an die Gültigkeitsintervalle hinzu, bei im Wesentlichen synchronem Eintreffen der Eingabeströme ist diese aber fast immer erfüllt. Daher bleibt die Organisation der Elemente nach Werten für Anfragen die Strategie der Wahl. Es bietet sich also an, auch im mehrdimensionalen Fall das in 9.2.4 vorgestellte Konzept zweistufiger SweepAreas einzusetzen.

Als zeitbezogene Löschstruktur eignet sich dabei unverändert die Organisation der Elemente in einem Min-Heap bezüglich ihrer Endzeitstempel. Abweichend davon kommen analog zum binären Fall aber auch die in 10.2.1 vorgestellten vereinfachten Löschstrukturen als Optimierung in Betracht.

Die Wahl der wertbezogenen Anfragestruktur hängt, wie schon in 9.3.5 für den binären Fall diskutiert, wesentlich vom Verbundprädikat P ab. Für spezielle Verbundprädikate können dabei besonders effiziente Anfragestrukturen benutzt werden, während allgemeinere Prädikate im Allgemeinen höheren Aufwand verursachen. Da für den mehrdimensionalen Verbund die Signatur der Anfragefunktion query geändert wurde, muss natürlich die Schnittstelle der Anfragestrukturen ebenfalls entsprechend modifiziert werden.

Allgemeines Prädikat Für ein Verbundprädikat P ohne spezielle Eigenschaften gibt es wiederum eine allgemein einsetzbare Anfragestruktur. Da ohne Zusatzinformation letztlich alle Elemente untersucht werden müssen, liefert diese in query immer alle enthaltenen Elemente zurück, falls im übergebenen Tupel $(e_1, \ldots, e_n)$ ein weiterer Wert außer dem an der Position der gerade angefragten SweepArea gleich $\perp$ ist. Anderenfalls wird diese letzte Position jeweils mit den Elementen in der Anfragestruktur belegt und

das Element nur dann zurückgegeben, wenn das Verbundprädikat P für das gebildete Tupel erfüllt ist. Das durch Verwendung dieser Anfragestruktur entstehende Verfahren entspricht dem mehrdimensionalen Nested-Loops-Join, da es aus einem eintreffenden Stromelement und den bereits in den SweepAreas vorhandenen Elementen alle möglichen Kombinationen bildet und gegen das Verbundprädikat testet.

Im Falle der Verwendung im Verbundalgorithmus mit Ausgabewarteschlange kommt zusätzlich zum Verbundprädikat noch die Überprüfung der Schnittbedingung an die Gültigkeitsintervalle hinzu. Hier können also in jedem Schritt diejenigen Elemente nicht zurückgeliefert werden, deren Gültigkeitsintervalle sich nicht mit dem Schnitt der Gültigkeitsintervalle der übergebenen Elemente $e_i \neq \perp$ schneidet. Da dies das einzige echte Filterkriterium ist, könnte man sogar die Elemente in der eigentlich wertbasierten Anfragestruktur nach Gültigkeitsintervall organisieren. Da die Schnittbedingung im Normalfall aber fast immer erfüllt ist, ist der dafür gegenüber der ansonsten einsetzbaren einfachen Listenstruktur benötigte Mehraufwand nicht gerechtfertigt.

Spezielle Prädikate Für den Einsatz spezieller Anfragestrukturen für spezielle Verbundprädikate gilt es nun zu beachten, dass die Anfragestrukturen im Gegensatz zum binären Fall nicht nur in einer, sondern in bis zu $n-1$ verschiedenen Konstellationen angefragt werden können, und zwar abhängig davon, an welcher Stelle der Anfragereihenfolgen zu den anderen Eingabeströmen diese auftauchen. Das Problem der Wahl der SweepAreas sollte daher wenn möglich gemeinsam mit der Wahl der Anfragereihenfolge und nicht isoliert davon betrachtet werden.

Anfragen, bei denen nur noch eine Komponente des Tupels $e_1, \ldots, e_n$ mit $\perp$ belegt ist und das Ergebnis exakt sein muss, sind natürlich besonders gut durch geeignete Anfragestrukturen unterstützbar. Der Schlüssel zu effizienter Verbundverarbeitung ist es jedoch, bereits die vorherigen Anfragen, die nur konservativ beantwortet werden müssen, mit möglichst optimaler, also kleiner Ergebnismenge zu beantworten. Dies muss zudem effizient geschehen, also ohne Testen aller möglichen Belegungen der an den Existenzquantoren gebundenen Variablen. Extrem hilfreich dafür sind notwendige Bedingungen für die Erfüllbarkeit von P, die sich ohne Kenntnis aller $s_1, \ldots, s_n$ entscheiden lassen. Solche Bedingungen sind immer dann leicht zu finden, wenn sich P als Konjunktion von Prädikaten darstellen lässt, von denen einige auf einer echten Teilmenge $\tilde{S} \subset \{s_1, \ldots, s_n\}$ auswertbar sind. Der Spezialfall von Prädikaten mit $|\tilde{S}| = 1$ bedeutet dabei, dass eine notwendige Bedingung an die Elemente eines Eingabestromes vorliegt, damit diese am Verbund teilnehmen können. Elemente, die diese Bedingung nicht erfüllen, sollten schon vor dem Verbund oder beim Eintreffen im Verbund direkt ausgefiltert werden.

Besonders interessant sind hingegen Prädikate mit $|\tilde{S}| = 2$, die also auf nur zwei der Variablen ausgewertet werden können, beispielsweise s_x und s_y. Für Stromelemente aus Eingabe x könnte dann das entsprechende Prädikat $P_{xy}(s_x, s_y)$ bei Anfragen an die SweepArea zu Eingabe y direkt dazu verwendet werden, Elemente als mögliche Verbundpartner auszuschließen. Ist die Anfragestruktur zu Eingabe y zudem so organisiert, dass sie zu gegebenen s_x das Auffinden von Elementen s_y mit $P_{xy}(s_x, s_y)$ effizient unterstützt, so kann der Suchraum bei der rekursiven Ergebnisproduktion gegebenenfalls effizient stark reduziert werden.

Zur systematischen Ausnutzung dieser vorteilhaften Anfrageart kann man einen Graphen betrachtet, dessen Knoten den Quellen des Verbundes entsprechen. Eine ungerichtete Kante wird zwischen zwei Knoten zu Eingaben x und y gezogen, falls eine Darstellung von P als Konjunktion ein binäres Prädikat $P_{xy}(s_x, s_y)$ enthält. Wird die Anfragestruktur der SweepArea zu Eingabe x zudem so gewählt, dass sie Anfragen bezüglich $P_{xy}(s_x, s_y)$ effizient unterstützt, so wird zusätzlich eine gerichtete Kante vom Knoten zu Eingabe y zu dem zu x eingezogen. Mit so gewählten Anfragestrukturen kann man nun die Anfragereihenfolgen für die SweepAreas günstig wählen. Dazu betrachtet man die Anfragereihenfolge zu einer Eingabe i als Weg durch den Graphen, der bei Knoten i beginnt, der nicht immer Kanten verfolgen muss und der alle Knoten genau einmal besucht. Besonders günstig ist dabei das Verfolgen gerichteter Kanten, nicht ganz so günstig aber immer noch gut das Verfolgen ungerichteter Kanten. Am ungünstigsten sind Sprünge zu einem anderen Knoten. Die gewählten Pfade werden natürlich dann am besten, wenn das Problem der Wahl der Anfragestruktur gemeinsam mit dem der damit wählbaren Pfade betrachtet wird. Weitere Faktoren sind die Selektivitätseigenschaften der Prädikate, die erwarteten mittleren Größen der SweepAreas und die Anfragekosten der Anfragestrukturen, die zu einer Gewichtung der Kanten und der Bedeutung der Pfade herangezogen werden könnten. Da entscheidende Faktoren dieses Modells zum Zeitpunkt der Instanziierung des Verbundes im System auf Grund fehlender Informationen über die Eingabeströme nicht bekannt oder schwierig einzuschätzen sind, ist eine anwendungsspezifische Wahl der Strukturen und Anfragereihenfolge durch Systementwickler einer automatischen Optimierung oftmals überlegen.

Insbesondere ist auch die Umformung von P in eine Konjunktion von Prädikaten mit möglichst wenigen Variablen ein nicht immer einfaches Problem. Wird der mehrdimensionale Verbund gemäß Lemma 10.2 zur Ausführung eines ursprünglich aus binären Verbünden bestehenden Anfrageplans eingesetzt, so ist P nach Konstruktion aber ohnehin eine Konjunktion von Prädikaten, die bis auf eines jeweils von mindestens einem Eingabestrom nicht abhängen. Bei der Umformung des Prädikates P ist es zudem hilfreich, weitere Prädikate zu finden, die von mindestens zwei aber möglichst wenigen Eingabeströmen abhängen und von P impliziert werden. Durch Hinzufügen solcher Prädikate zur Konjunktion wird nämlich die Semantik nicht verändert, aber es ergeben sich zusätzliche Möglichkeiten zum frühen Ausfiltern von Tupeln bei der Konstruktion von Verbundergebnissen.

Die zusätzlich zum wertbasierten Verbundprädikat im Falle des Verbundalgorithmus mit Ausgabewarteschlange zu überprüfende Schnittbedingung an die Gültigkeitsintervalle lässt sich ideal aufteilen, da sie partiell für eine beliebige Teilmenge der Gültigkeitsintervalle gelten muss, die sich ebenfalls immer schneiden müssen. Sie gilt also insbesondere für beliebige Paare von Stromelementen verschiedener Eingaben.

10.3.5. Optimierung

Zum Abschluss dieses Kapitels wird noch die Optimierung des mehrdimensionalen Verbundes diskutiert.

Dynamische Anfrageorganisation

Wie schon in 10.1.5 für die dynamische Planmigration erläutert, empfiehlt es sich, die bei der Integration einer neuen Anfrage ins System getroffenen Entscheidungen während deren langer Laufzeit gelegentlich zu überprüfen, da neue Statistiken über die Eingabeströme und Selektivitäten vorliegen können und sich diese Faktoren zudem immer wieder ändern können. Nach Lemma 10.3 ist ein mehrdimensionaler Verbund physisch äquivalent zu einem Baum aus binären Verbünden, der nach Korollar 10.1 historisch kontextfrei und verzögerungsfrei ist, weshalb die dynamische Planmigration unter Verwendung der Referenzpunktmethode dazu verwendet werden kann, ihn gegen einen mit der gleichen Begründung ebenfalls historisch kontextfreien und verzögerungsfreien physisch äquivalenten mehrdimensionalen Verbund oder Baum aus Verbünden auszutauschen. Da jedoch auch diese Form der Planmigration vorübergehend die Systemlast erhöht, stellt sich die Frage, ob eine Optimierung auch einfacher ermöglicht werden kann.

Die Anfragereihenfolgen, anhand derer die Funktion SucheErgebnisse beim Eintreffen eines Stromelementes die SweepAreas zu den anderen Eingaben durchläuft, wird im grundlegenden Algorithmus 11 als Parameter für die gesamte Laufzeit festgelegt und dabei wie vorstehend erläutert auf die Anfrageprädikate der SweepAreas abgestimmt. Die Wahl dieser Parameter kann allerdings zur Laufzeit zwischen der Verarbeitung zweier Stromelemente jederzeit geändert werden. Im Extremfall könnte man für jedes einzelne Stromelement bei dessen Eintreffen die darauf anzuwendende Anfragereihenfolge neu bestimmen, ähnlich wie dies bei den Eddies[AH00] geschieht. Die Erfahrungen damit zeigen jedoch, dass ein so massiver Optimierungsaufwand in keinem vernünftigen Verhältnis zu den erzielten Ressourceneinsparungen steht. Eine Neubestimmung der Anfragereihenfolgen und Prädikate in regelmäßigen Abständen oder als Reaktion auf merkliche Änderungen der Eigenschaften der Eingabeströme stellt jedoch eine sinnvolle Optimierung dar.

Dabei ist jedoch zu beachten, dass die Wahl der Anfragestrukturen in den SweepAreas wiederum wesentlichen Einfluss darauf hat, wie effizient für bestimmte Anfrageprädikate die zugehörigen Ergebnisse geliefert werden können. Das Optimierungspotential kann also weiter gesteigert werden, indem auch ein Austausch der Anfragestrukturen zur Laufzeit ermöglicht wird. Die einfachste Möglichkeit dazu bietet das Verschieben der Elemente von der alten in die neue Speicherstruktur zwischen der Verarbeitung zweier Stromelemente durch den Verbund.

Da dies jedoch bei SweepAreas mit vielen Elementen längere Zeit dauern kann, in der die Verarbeitung blockiert wäre, sollte die Migration besser kontinuierlich erfolgen. Dazu kann die Idee der Planmigration auf Speicherstrukturen übertragen werden. Sollen jedoch gleich mehrere Speicherstrukturen ausgetauscht werden, so ist es auch möglich, per Planmigration auf einen zweiten, mit den neuen Strukturen, Prädikaten und Anfragereihenfolgen versehenen mehrdimensionalen Verbund überzugehen.

Allerdings haben die SweepAreas die Eigenschaft, dass ihre Operationen bezüglich eines darin gespeicherten Elementes logisch unabhängig von der Anwesenheit anderer sind. Daher können auch zwei SweepAreas für eine Eingabe parallel betrieben werden, wenn Löschen und Anfragen jeweils gegen beide erfolgt während eintreffende Elemente in genau eine von beiden eingefügt werden. Auf diese Art kann eine zweite SweepArea

installiert werden, in die fortan eingefügt wird, und später die erste entfernt werden, sobald sie leer ist. Dieses einfachere Migrationsverfahren ist bei gleichzeitiger Änderung der Anfragereihenfolgen aber nur anwendbar, wenn alte und neue Reihenfolgen von jeweils beiden Versionen der SweepAreas unterstützt werden.

Zusätzlich zum dynamischen Variieren der Anfragereihenfolgen wird in [BMWM05] für mehrdimensionale Verbundoperatoren vorgeschlagen, innerhalb des Operators dynamisch Zwischenergebnisse in Caches zu materialisieren und damit auf Kosten erhöhten Speicherverbrauchs die Kosten bestimmter Anfragereihenfolgen durch Zugriff auf die Caches zu verringern. Durch das zusätzliche Anlegen von SweepAreas könnte dieses Konzept adaptiert werden.

Ausgabecharakteristik

Um den mehrdimensionalen Verbund im Kontext des in 10.1.3 vorgestellten Kostenmodells anwenden zu können, ist es besonders wichtig, dessen Ausgabecharakteristik anzugeben. Dabei erweist sich Lemma 10.3 und insbesondere die Konstruktion im zugehörigen Beweis als extrem hilfreich, da sie aufzeigt, wie der mehrdimensionale Verbund in einen physisch äquivalenten linkstiefen Baum aus binären Verbünden umgewandelt werden kann. Somit genügt es, die Ausgabecharakteristik dieses Baumes mit Hilfe der in 10.1.4 angegebenen Abschätzungen hochzurechnen.

Für den Längenparameter $l(n)$ eines n-dimensionalen Verbundes erhält man auf diese Art

$$l(n) = \frac{\prod_{i=1}^{n} l_i}{\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j}$$

Dies stimmt für $n = 2$ mit der Formel $l = \frac{l_1 l_2}{l_1 + l_2}$ für den binären Verbund überein und setzt sich induktiv wie folgt fort:

$$\begin{aligned}
l(n+1) &= \frac{l(n) l_{n+1}}{l(n) + l_{n+1}} \\
&= \frac{\frac{\prod_{i=1}^{n} l_i}{\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j} l_{n+1}}{\frac{\prod_{i=1}^{n} l_i}{\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j} + l_{n+1}} \\
&= \frac{\frac{\prod_{i=1}^{n+1} l_i}{\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j}}{\frac{\prod_{i=1}^{n} l_i}{\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j} + \frac{(\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j) l_{n+1}}{\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j}} \\
&= \frac{\prod_{i=1}^{n+1} l_i}{\prod_{i=1}^{n} l_i + \sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n+1} l_j} \\
&= \frac{\prod_{i=1}^{n+1} l_i}{\sum_{i=1}^{n+1} \prod_{j=1, j\neq i}^{n+1} l_j}
\end{aligned}$$

Für den Abstandsparameter $d(n)$ eines n-dimensionalen Verbundes ergibt sich

$$d(n) = \frac{1}{\sigma} \frac{\prod_{i=1}^{n} d_i}{\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j}$$

Auch dies entspricht für $n = 2$ der Formel $d = \frac{d_1 d_2}{\sigma(l_1+l_2)}$ im binären Fall. Bei der induktiven Fortsetzung ist zu beachten, dass die Konstruktion das Verbundprädikat P alleine an der Wurzel des Baumes aus binären Verbünden auswertet, während alle Verbünde darunter das Prädikat $\top$ mit $\sigma_\top = 1$ erhalten.

$$\begin{aligned}
d(n+1) &= \frac{1}{\sigma} \frac{d(n) d_{n+1}}{l(n) + l_{n+1}} \\
&= \frac{1}{\sigma} \frac{\frac{1}{\sigma_\top} \frac{\prod_{i=1}^{n} d_i}{\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j} d_{n+1}}{\frac{\prod_{i=1}^{n} l_i}{\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j} + l_{n+1}} \\
&= \frac{1}{\sigma} \frac{\frac{\prod_{i=1}^{n+1} d_i}{\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j}}{\frac{\prod_{i=1}^{n} l_i}{\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j} + \frac{(\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j) l_{n+1}}{\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} l_j}} \\
&= \frac{1}{\sigma} \frac{\prod_{i=1}^{n+1} d_i}{\prod_{i=1}^{n} l_i + \sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n+1} l_j} \\
&= \frac{1}{\sigma} \frac{\prod_{i=1}^{n+1} d_i}{\sum_{i=1}^{n+1} \prod_{j=1, j\neq i}^{n+1} l_j}
\end{aligned}$$

Abschließend ergibt sich der Granularitätsparameter $g(n)$ eines n-dimensionalen Verbundes aus der Formel $g = ggT(g_1, g_2)$ im binären Fall wegen $ggT(x, y, z) = ggT(ggt(x, y), z)$ zu

$$g(n) = ggT(g_1, \ldots, g_n)$$

Kostenformeln

Um auch die Kosten des n-dimensionalen Verbundes im Kostenmodell berücksichtigen zu können, werden zudem noch dessen Kostenformeln benötigt. Analog zu den binären Verbundalgorithmen wird angenommen, dass die Datenströme im Wesentlichen der globalen zeitlichen Ordnung folgend verarbeitet werden. Auch die zu Grunde liegenden Annahmen, Parameter und Funktionen seien analog zum in 10.1.4 erläuterten binären Fall gewählt.

Speicherverbrauch Der Speicherverbrauch ergibt sich daher wieder alleine aus dem der SweepAreas, in denen ein Stromelement aus Eingabe i nach wie vor mindestens l_i Zeiteinheiten gespeichert bleibt. Sein Löschen erfolgt nun, sobald in allen anderen

Strömen mindestens ein Element eingetroffen ist, also $\frac{max_{j\in\{1,\ldots,n\}\setminus\{i\}}(d_j)}{2}$ Zeiteinheiten später. Der Speicherverbrauch ergibt sich also zu

$$M = \sum_{i=1}^{n} V_{SA_i}(\frac{1}{d_i}(l_i + \frac{max_{j\in\{1,\ldots,n\}\setminus\{i\}}(d_j)}{2}), v_i)$$

CPU-Belastung Mit dieser Größenschätzung für die SweepAreas ergeben sich Einfügekosten für die SweepArea zu Eingabe i von $K^{einf}_{SA_i}(\frac{1}{d_i}(l_i + \frac{max_{j\in\{1,\ldots,n\}\setminus\{i\}}(d_j)}{2}))$.

Beim Entfernen aus den SweepAreas zu Eingabe j, $j \neq i$, geht nun auch diese Größenschätzung ein. Zur Bestimmung des Verhältnisses von Einfüge- und Löschrate muss nun das gewichtete Mittel der Eingaberaten der anderen Eingabeströme $\frac{1}{\sum_{k=1,k\neq j}^{n}\frac{1}{d_k}}$ mit der Eingaberate von Strom j ins Verhältnis gesetzt werden, woraus sich $K^{entf}_{SA_j}(\frac{1}{d_j}(l_j + \frac{max_{k\in\{1,\ldots,n\}\setminus\{j\}}(d_k)}{2}), \frac{1}{d_j\sum_{k=1,k\neq j}^{n}\frac{1}{d_k}})$ ergibt. Der Term $\frac{1}{d_j\sum_{k=1,k\neq j}^{n}\frac{1}{d_k}}$ kann dabei verhältnismäßig klein werden, weshalb sich die Frage ergibt, ob es sich lohnt, eine Löschoperation auszulösen, obgleich im Mittel nur so wenige Elemente gelöscht werden. Allerdings wird das vorherige Durchführen einer Löschoperation bei den nachfolgenden Anfragen zur Vereinfachung der Anfrageprädikate benötigt. Zudem ist die Kostenformel hier stark von den Annahmen des Kostenmodells beeinflusst. Bei Einsatz eines Ausgabepuffers findet in der Praxis selten exakt global sortierte Verarbeitung statt, weshalb seltener ein zeitlicher Fortschritt im Operator eintritt. Dadurch werden dann viele Löschschritte gar nicht erst unternommen.

Die Größen der SweepAreas nach dem Löschen, $\frac{l_j}{d_j}$, geht nun in die Berechnung der Anfragekosten ein. Bei der Verarbeitung eines Stromelementes aus Eingabe i wird anhand der Anfragereihenfolge o zuerst die SweepArea $o(i,2)$ angefragt, wobei Kosten von $K^{anfr}_{SA_{o(i,2)}}(\frac{l_{o(i,2)}}{d_{o(i,2}})$ entstehen. Die Zahl der Ergebnisse hängt dabei von einer Selektivität $\sigma_{i,o(i,2)}$ ab. Dementsprechend werden dann $\sigma_{i,o(i,2)}\frac{l_{o(i,2)}}{d_{o(i,2)}}$ Anfragen an die SweepArea $o(i,3)$ gestellt, mit Kosten von $\sigma_{i,o(i,2)}\frac{l_{o(i,2)}}{d_{o(i,2)}} \cdot K^{anfr}_{SA_{o(i,3)}}(\frac{l_{o(i,3)}}{d_{o(i,3)}})$. Analog fortgesetzt ergeben sich Kosten von $\sum_{k=2}^{n}\prod_{j=2}^{k-1}\sigma_{i,o(i,j)}\frac{l_{o(i,j)}}{d_{o(i,j)}} \cdot K^{anfr}_{SA_{o(i,k)}}(\frac{l_{o(i,k)}}{d_{o(i,k)}})$. Die Möglichkeit des vorzeitigen Abbruchs der Anfragefolge bleibt hier unberücksichtigt, da die dafür benötigten Wahrscheinlichkeiten mit den Möglichkeiten des Modells nicht zu bestimmen sind.

Die Kosten für die Ergebnisproduktion lassen sich analog zum binären Fall am einfachsten mit Hilfe der Ausgaberate d zu $\frac{C_r}{d}$ bestimmen. Alle anderen zuvor genannten Kosten treten jeweils mit der Rate $\frac{1}{d_i}$ des jeweiligen Stromes auf.

Insgesamt ergibt sich also eine CPU-Belastung von

$$\sum_{i=1}^{n} \frac{1}{d_i}(K_{SA_i}^{einf}(\frac{1}{d_i}(l_i + \frac{max_{j\in\{1,\ldots,n\}\setminus\{i\}}(d_j)}{2}))+ K_{SA_i}^{entf}(\frac{1}{d_i}(l_i + \frac{max_{k\in\{1,\ldots,n\}\setminus\{i\}}(d_k)}{2}), \frac{1}{d_i \sum_{k=1,k\neq i}^{n} \frac{1}{d_k}})+ \sum_{k=2}^{n}\prod_{j=2}^{k-1} \sigma_{i,o(i,j)} \frac{l_{o(i,j)}}{d_{o(i,j)}} \cdot K_{SA_{o(i,k)}}^{anfr}(\frac{l_{o(i,k)}}{d_{o(i,k)}})) + \frac{C_r}{d}$$

Spezialfall

Lässt sich für $n > 2$ das Verbundprädikat P eines n-dimensionalen Verbundes mit Hilfe von binären Prädikaten $P_{i,j}$, $\{i,j\} \subset \{1,\ldots,n\}$ so umschreiben, dass $P(e_1,\ldots,e_n) \Leftrightarrow \bigwedge_{\{i,j\}\subseteq\{1,\ldots,n\}} P_{i,j}(s_i,s_j)$ und $\forall\{i,j,k\} \subseteq \{1,\ldots,n\} : P_{i,j}(s_i,s_j) \wedge P_{i,k}(s_i,s_k) \Rightarrow P_{j,k}(s_j,s_k)$ gilt, so kann der mehrdimensionale Verbundalgorithmus erheblich vereinfacht werden. Trifft nämlich ein Stromelement e aus einer Eingabe l ein, so kann man für alle $m \in \{1,\ldots,n\}\setminus\{l\}$ jeweils die Menge R_m von Elementen in den SweepAreas zu Eingabe m mit $P_{l,m}$ bestimmen. Mit $R_l := \{e\}$ gilt dann $\forall(e_1,\ldots,e_n) \in R_1 \times \ldots \times R_n : P(e_1,\ldots,e_n)$. Zudem gibt es keine $e_1,\ldots,e_n$ mit $P(e_1,\ldots,e_n)$ derart, dass ein e_i mit $i \neq l$ zwar in der SweepArea zu Eingabe l, aber nicht in R_l liegt.

In diesem Spezialfall können alle SweepAreas so ausgelegt werden, dass sie lediglich Anfragen mit binären Prädikaten von Elementen aus den anderen Eingaben erlauben. Beim Eintreffen eines Stromelementes werden dann nacheinander oder sogar parallel alle SweepAreas zu den anderen Eingaben entsprechend angefragt und so die Mengen R_m bestimmt. Durch Kreuzproduktbildung kann dann die Menge von Ergebnissen $R_1 \times \ldots \times R_n$ leicht bestimmt werden.

Diese Vorgehensweise hat mehrere Vorteile. Zunächst werden nur binäre Prädikate verwendet, wofür es oftmals gute Speicherstrukturen gibt. Viel wichtiger ist aber, dass für jedes eintreffende Stromelement nur maximal $n-1$ Anfragen gestellt werden, unabhängig davon, wie viele Ergebnisse produziert werden. Ist eine der Mengen R_m leer, so kann das weitere Anfragen abgebrochen werden.

Die CPU-Kosten für das Anfragen der SweepAreas ändern sich daher zu folgendem Term. Dabei wird analog zum allgemeinen Fall die Möglichkeit des vorzeitigen Abbruchs des Anfragens vernachlässigt.

$$\sum_{i=1}^{n} \frac{1}{d_i} \sum_{k=1,k\neq i}^{n} K_{SA_k}^{anfr}(\frac{l_k}{d_k})$$

Obgleich die Voraussetzungen für diese Optimierung zunächst sehr stark erscheinen, sind sie in der Praxis doch oftmals erfüllt. Sie gelten nämlich beispielsweise dann, wenn die

Menge aller möglicher Eingabestromelemente so in Äquivalenzklassen aufgeteilt werden kann, dass $P(e_1, \ldots, e_n)$ genau dann erfüllt ist, wenn alle e_i in derselben Äquivalenzklasse liegen. Dies wiederum ist beispielsweise dann gegeben, wenn für relationale Tupel ein Gleichverbund bezüglich eines gemeinsamen Attributes berechnet werden soll. Allgemein umfasst der Spezialfall das Einsatzgebiet von Hash-Joins. Entsprechend erfolgt auch die Ergebnisproduktion des M-Joins nach diesem Schema.[VNB03]

11. Progressive-Merge-Join

Betrachtet man die Entwicklungshistorie von Verbundalgorithmen, so beginnt diese mit einer langen Zeit, in der Verfahren für klassische persistente Datenbankmanagementsysteme entwickelt wurden. Darunter fallen insbesondere die in 9.1.2 beschriebenen klassischen Verbundtechniken. Optimierungsziel war dabei fast immer eine möglichst geringe Gesamtlaufzeit. Erst wesentlich später begann die Entwicklung nicht blockierender Algorithmen, also solcher, die bereits vor der kompletten Konsumierung einer der Eingaben Ergebnisse produzieren. Das wesentliche Ziel dabei ist es, bereits möglichst früh Teile des Ergebnisses zu liefern. Motiviert wurden diese Verfahren, von denen einige in 9.1.3 vorgestellt werden, zumeist für zwei Anwendungen. Zum einen soll dem Benutzer während der langen Laufzeit eines aufwendigen Verbundes eine Vorschau auf die Ergebnisse erlaubt werden. Zum anderen soll in verteilten Umfeldern, die zunehmend an Bedeutung gewonnen haben, bereits möglichst viel der Berechnungsarbeit erledigt werden, während die Eingaben noch über Netzwerke bezogen werden. Als dann Verbundalgorithmen für potentiell unbegrenzte Datenströme entwickelt werden sollten, bot es sich natürlich an, solche nicht blockierenden Verfahren auf die Datenstromverarbeitung anzupassen. Die in Kapitel 9 vorgestellten Verbundalgorithmen für Datenströme basieren dabei wie in 9.3.1 erläutert auf dem Prinzip der Familie der Ripple-Joins, die wiederum wesentlich auf der Familie der klassischen Index-Joins, insbesondere dem Hash-Join, basiert.

Ebenfalls bereits in 9.1.2 vorgestellt wurde der klassische Sort-Merge-Join. Auch dessen Grundidee wurde im Laufe der Zeit weiterverfolgt. So entstand mit dem Progressive-Merge-Join, der in 11.1 beschrieben wird, ein sortierbasierter Verbundalgorithmus, der frühe Ergebnisse liefert. Es bot sich daher an, auch dieses Verbundverfahren auf Datenströme hin zu adaptieren, was Thema dieses Kapitels ist.

In 11.1 wird dazu zunächst eine Zusammenfassung bezüglich des klassischen Progressive-Merge-Join gegeben. In 11.2 wird dann erläutert, wie dieser, zunächst weiterhin über Multimengen operierend, für datengetriebene Verarbeitung erweitert werden kann. In 11.3 wird diese datengetriebene Variante dann zur Verbundoperation auf physischen Datenströmen ausgebaut. Teile der in diesem Kapitel präsentierten Ergebnisse wurden bereits in [CHKS05] veröffentlicht.

11.1. Progressive-Merge-Join (PMJ)

Wie bereits in 9.1.3 und der Einleitung dieses Kapitels angedeutet, behebt der *Progressive-Merge-Join* (PMJ) [DSTW02, DSTW03] für die passive Datenverarbeitung das Problem, dass der externe Sort-Merge-Join blockierend ist, weil er in seiner ersten Phase die Eingaben nacheinander komplett sortiert.

Beim externen Sort-Merge-Join wird als externer Sortieralgorithmus im Allgemeinen der externe Merge-Sort eingesetzt. Zur Sortierung füllt dieser den Hauptspeicher mit Elementen einer Eingabe, die er sortiert und als sortierte Teilfolge (auch *Run* genannt) auf den Externspeicher zurückschreibt. Dieser Vorgang wird wiederholt, bis die komplette Eingabe in Form sortierter Runs auf dem Externspeicher liegt (Run-Generierung). Diese Runs werden dann zu längeren Runs zusammengeführt, indem jeweils möglichst viele von ihnen synchronisiert bezüglich der für den Verbund verwendeten globalen Ordnung gelesen und direkt als längerer Run wieder herausgeschrieben werden. Den begrenzenden Faktor bildet dabei die Zahl der Externspeicherseiten, auch *Fan-In* genannt, die gleichzeitig effizient gelesen werden können. Dieser Vorgang wird so lange wiederholt, bis nur noch ein Run übrig bleibt, der die komplette Eingabe in sortierter Form enthält. Analog dazu wird dann die zweite Eingabe sortiert. Danach wird der in 9.1.2 beschriebene Merge durchgeführt.

Eine gängige Optimierung dieses Verfahrens ist es, den jeweils letzten Merge-Schritt beim Sortieren nicht mehr durchzuführen, sondern die vorher beim Sortieren generierten Runs aller Eingaben nun gemeinsam der globalen Ordnung folgend zu verarbeiten und dabei direkt den Merge-Schritt des Merge-Joins durchzuführen.[NP85] Damit der Fan-In dafür ausreicht, muss die Zahl der Runs allerdings geeignet gesteuert werden. Die wesentliche Idee des PMJ ist nun die Übertragung der Idee dieser Optimierung auch auf die vorangegangenen Merge-Schritte des Sortierens, also die generelle Verschmelzung der ähnlich ablaufenden Merge-Schritte des externen Sortierens und des Merge-Joins.

Dabei nutzt man aus, dass die für den Join benutzte globale Ordnung auch jeweils als lokale Ordnung zum Sortieren der Eingaben benutzt wird. Für einen Merge-Schritt des Sortierens wird jetzt generell nur ein Teil des Hauptspeichers für eine Eingabe genutzt, während gleichzeitig im Rest des Hauptspeichers ein Merge-Schritt für die anderen Eingaben durchgeführt wird. Die Verarbeitung erfolgt dabei in der durch die globale Ordnung vorgegebenen Reihenfolge, die Verarbeitung aller Eingaben erfolgt also verzahnt. Dabei ist dann immer wie beim Merge-Schritt des Merge-Joins die Situation gegeben, dass Elemente aller Eingaben der globalen Ordnung folgend verarbeitet werden. Dadurch kann dieser Schritt hierbei nebenbei zusätzlich durchgeführt werden. Insgesamt erhält man einen Merge-Schritt (*Multiple-Merge*), der mehrere Runs pro Eingabe verarbeitet und daraus deren kompletten Join und je einen sortierten Run pro Eingabe erzeugt.

Abschließend kann die Idee der Verzahnung von Sortierung und Join-Berechnung auch auf die Run-Generierung übertragen werden. Der Hauptspeicher wird dabei ebenfalls je zu einem Teil mit Elementen aller Eingaben gefüllt, die dann jeweils zunächst sortiert werden. Bevor sie jedoch als Runs nach Eingabe getrennt auf den Externspeicher geschrieben werden, wird noch der Join dieser sortierten Teile der Eingaben berechnet und als Ergebnis weitergeleitet. Die Strategie zur Aufteilung des Hauptspeichers unter den Eingaben, also die Entscheidung wie viele Elemente welcher Eingabe jeweils gelesen werden, ist dabei ein Parameter des Verfahrens. Mögliche Strategien werden ausführlich in [Cam02] diskutiert.

Die sich insgesamt ergebende Struktur des Generierung und des Verschmelzens von Runs wird als *Merge-Baum* bezeichnet. Da der Sort-Merge-Join alle seine Ergebnisse erst im letzten Schritt produziert, spielt bei ihm die Reihenfolge der vorangehenden Schritte eine untergeordnete Rolle. Beim PMJ hingegen beeinflusst diese Reihenfolge maßgeblich

die Produktion früher Ergebnisse. Da beim Bilden des Verbundes im Allgemeinen umso mehr Ergebnisse produziert werden können, je mehr Eingabeelemente beteiligt sind, liefern die Merge-Schritte zum Verschmelzen von Runs mehr Ergebnisse als die zur Run-Generierung. Die Multiple-Merge-Schritte liefern zudem um so mehr Ergebnisse, je länger die beteiligten Runs sind. Die Struktur des Merge-Baumes und der Reihenfolge seiner Abarbeitung bestimmt also wesentlich den zeitlichen Verlauf der Produktion früher Ergebnisse.[DSTW03, Cam02]

Prinzipiell berechnet der letzte Merge-Schritt, bei dem alle Elemente aller Eingaben in je einem Run enthalten sind, das komplette Joinergebnis, da er dem Merge-Schritt des wie oben beschrieben optimierten Sort-Merge-Joins entspricht. Es ergibt sich aber das Problem auftretender Duplikate, da manche Joinergebnisse schon in vorherigen Merge-Schritten als frühe Ergebnisse identifiziert und weitergeleitet wurden. Um das zu vermeiden, könnte man alle Joinergebnisse erst im letzten Schritt herausgeben, was jedoch die Idee früher Ergebnisse zunichtemachen würde. Man möchte Ergebnisse möglichst früh produzieren, also sofort bei deren erstem Auftreten. Dieses erfolgt, wenn die beteiligten Eingabeelemente beim Sortieren gemeinsam im Hauptspeicher vorlagen oder wenn sie zum ersten Mal gemeinsam an einem Merge-Schritt beteiligt sind. In beiden Fällen werden sie in Runs gespeichert, die im selben Schritt erzeugt werden. Man stellt nun sicher, dass derart zusammengehörige Runs in den folgenden Merge-Schritten nur gemeinsam verarbeitet werden. Um ein Ergebnis als Duplikat zu identifizieren, muss man dann nur feststellen, ob die beteiligten Elemente aus zusammengehörigen Runs kommen. Dieses Vorgehen gestattet eine effiziente Duplikateliminierung und garantiert korrekte und vollständige Ergebnisse des PMJ.[DSTW02, Cam02]

11.2. Datengetriebener PMJ

In diesem Abschnitt wird nun zunächst erläutert, wie der PMJ vom bedarfsgesteuerten Verbundalgorithmus zum datengetriebenen Verbundalgorithmus für Multimengen modifiziert werden kann.

11.2.1. Speicheraufteilung

Während der bedarfsgesteuerte PMJ über eine Strategie selber entscheidet, wie viele Elemente welcher Eingabe er in den Hauptspeicher liest, müssen im datengetriebenen Fall die Elemente in der Reihenfolge verarbeitet werden, wie sie eintreffen. Gibt man einen Speicherbereich vor und füllt ihn so lange mit eintreffenden Elementen der verschiedenen Eingaben, bis er voll ist, so kann dies allerdings auch als Strategie zur Verteilung des Speichers unter den Eingaben betrachtet werden. Diese Verteilung kann dabei natürlich auch sehr asymmetrisch ausfallen, was gegebenenfalls zu einer geringeren Zahl früher Ergebnisse als bei optimaler Verteilung führt. Dies ist jedoch direkte Folge der Vorgabe, die Eingaben in der Reihenfolge zu verarbeiten, in der sie eintreffen.

11.2.2. Rungenerierung

Sind nun so viele Elemente eingetroffen, dass der vorgegebene Speicherbereich gefüllt ist, so gilt es nun, zum einen den Verbund der Teilfolgen im Speicher zu berechnen, und zum anderen diese als sortierte Runs auf den Externspeicher auszulagern. Den Verbund der im Hauptspeicher eingetroffenen Teile der Eingaben könnte man nun mit einem beliebigen Hauptspeicher-Verbundalgorithmus berechnen. Da die Teilfolgen aber ohnehin sortiert werden müssen, um sie als Runs auf den Externspeicher zu schreiben, und da der Algorithmus normalerweise nur für Joinprädikate angewendet wird, die einen effizienten sortierbasierten Join unterstützen, führt man zuerst die Sortierung durch. Danach wird der Join der sortierten Runs mit Hilfe eines Merge-Joins (siehe 9.2.3) unter Verwendung von SweepAreas berechnet. Abschließend werden die Runs für die Weiterverarbeitung auf den Externspeicher verlagert.

Ein Parameter bei diesem Vorgehen ist die Wahl eines geeigneten Sortierverfahrens. Berücksichtigt man das sukzessive Eintreffen der Elemente, so bietet sich Insertion-Sort als Möglichkeit an. Dieses Verfahren hat jedoch leider ein sehr ungünstiges Verhalten im schlechtesten Fall. Als weiteres inkrementelles Sortierverfahren bietet sich, insbesondere im Umfeld des externen Sortierens, *Replacement-Selection*[Knu73] an. Das Verfahren liefert im Mittel Runs, die doppelt so groß wie der benutzte Teil des Hauptspeichers sind. Die in [MPDSLP10] vorgestellte Weiterentwicklung *Two-way Replacement Selection* liefert sogar noch längere Runs, falls die Eingaben bereits sortierte Teilfolgen enthalten. Eine Adaption der Verfahren für den PMJ gestaltet sich jedoch schwierig. Zunächst muss man pro Eingabe einen Heap, oder im Falle von Two-way Replacement Selection sogar zwei Heaps und einen Zusatzpuffer, verwalten. Beim Eintreffen eines neuen Elements verarbeitet man das kleinste Element aller Eingaben, wobei man es als potentiellen Joinpartner einsetzt und dann herausschreibt. Das kleinste Element gehört aber nichts zwangsläufig zur selben Eingabe wie das eingetroffene Element. Folglich müsste man zur Laufzeit Speicher zwischen den Heaps umverteilen. Hinzu kommt noch die Problematik, dass Replacement-Selection empfindlich auf Elemente verschiedener Größe reagiert.[LG98] Obwohl auch dieser ein ungünstiges Verhalten im schlechtesten Fall aufweist, hat sich in der Praxis Quicksort[Hoa62] als gute Wahl für das Sortierverfahren erwiesen. Zu dessen Verwendung werden eintreffende Elemente zunächst an eine ihrer Eingabe zugeordnete Liste angehängt. Bei Erreichen der vorgegebenen Speichergrenze werden diese Listen dann in situ sortiert. Eine Auswertung verschiedener Sortieralgorithmen bezüglich ihrer Verwendbarkeit in DBMS findet sich in [Gra06].

Während des Sortiervorgangs können jedoch weitere zu verarbeitende Elemente eintreffen. Aus diesem Grund werden bereits beim Erreichen eines bestimmten Anteils der Speichergrenze neue Listen zur Aufnahme der Eingabeelemente angelegt. Die Sortierung und Verarbeitung der alten Listen wird in einem separaten Thread vorgenommen. Um die Auslagerung sicher zu stellen und so nicht an die Speichergrenze zu stoßen, ist der Thread mit einer hohen Priorität versehen.

Dieses Vorgehen hat zudem den Vorteil, dass es adaptiv bezüglich verschiedener und möglicherweise stark schwankender Datenraten der Eingaben ist. Stößt man an die Speichergrenze, so sortiert man und schreibt Runs auf den Externspeicher. Dadurch werden für Eingaben mit aktuell höherer Datenrate auch im Verhältnis längere Runs

erzeugt. Die Tatsache, dass somit für jede Eingabe gleich viele Runs erzeugt werden, stellt eine gute Voraussetzung für den Mergeschritt des PMJ dar.

11.2.3. Merge

Für den Fall endlicher Datenströme wäre eine Strategie, die Mergephase des passiven PMJs anzuwenden, nachdem alle Elemente der Eingaben in Runs auf dem Externspeicher vorliegen. Mergeschritte in höheren Ebenen des Mergebaumes erzeugen jedoch wegen der größeren Anzahl beteiligter Eingabeelemente mehr frühe Ergebnisse als solche in tieferen Ebenen. Somit eignet sich, wie schon beim klassischen PMJ, ein Postorder-Durchlauf eines Mergebaumes mit mindestens zwei Ebenen besser zur Produktion früher Ergebnisse als ein Levelorder-Durchlauf.[DSTW03]

Der Fan-In ist analog zum passiven Fall der wesentliche limitierende Faktor für die Anzahl der Runs, die in einem Schritt verarbeitet werden können. Die konkrete Wahl des Fan-In hängt vom für den Joinoperator verfügbaren Hauptspeicher ab. Ein Teil dessen muss dabei für die Sortierschritte der eintreffenden Elemente reserviert bleiben. Ist die Anzahl der maximal verarbeitbaren Runs noch nicht erreicht, kann sich unter Umständen ein Mergeschritt trotzdem schon lohnen. Im Hinblick auf schwankende Datenraten der beteiligten Ströme können sich nämlich Phasen geringer Systemlast ergeben.[UF00] Führt man in diesen Mergeschritte durch, so produziert man zum einen Ergebnisse und reduziert zum anderen die Zahl der Runs (und dadurch auch den verbleibenden Sortieraufwand). Um dies zu ermöglichen, wird der Mergeschritt ebenfalls in einem separaten Thread realisiert. Dieser erhält eine geringere Priorität als der für die Sortierung initialer Runs, da die Elemente auf dem Externspeicher liegen und keine Ergebnisse verloren gehen, auch wenn längere Zeit kein Mergeschritt durchgeführt wird. Die Ergebnisse werden lediglich später erzeugt, was in Phasen hoher Systemlast sogar von Vorteil sein kann, da die Joinergebnisse so erst später weiterverarbeitet werden müssen.

Beim Durchführen einer Mergephase ist zudem zu beachten, dass diese soweit möglich auch komplett durchgeführt werden kann, und nicht aufgrund auftretender Speicherknappheit abgebrochen werden muss. Dies ist zwar im Notfall möglich, da die Seiten zum Einlesen der Runs freigegeben und lediglich noch aktuelle Seitennummer und Position darin gespeichert werden können. Zusätzlich benötigen jedoch auch die Datenstrukturen des Join-Rahmenwerks, die SweepAreas, Hauptspeicher. Dieser Bedarf steigt im Gegensatz zum Speicherbedarf für die Seiten zum Lesen der Runs mit der Höhe des Mergeknotens im Merge-Baum. Daher sollten Joins auf höheren Ebenen gegebenenfalls mit geringerem Fan-In und auch nur in Situationen durchgeführt werden, in denen dem DSMS hinreichend Speicher zur Verfügung steht.

Es bietet sich eine Vielzahl an Möglichkeiten, die Anzahl der für einen Mergeschritt zu verwendenden Runs zu wählen. Allerdings sollten Runs verschiedener Eingaben, die gemeinsam auf den Externspeicher geschrieben wurden, auch gemeinsam wieder eingelesen werden, um die vorgestellte Technik zur Duplikateliminierung verwenden zu können. Um die konkrete Wahl flexibel zu gestalten, kann man die Auswahl der zu verschmelzenden Runs einer Strategie überlassen, die als zusätzlicher Parameter an den Algorithmus übergeben wird. Eine konkrete Strategie legt demnach für jede Mergephase fest, welche Runs verarbeitet werden.

Als eine mögliche Strategie bieten sich an, so viele Merge-Schritte durchzuführen, wie es die Ressourcen zulassen. Bei dieser *Greedy*-Strategie führt der Merge-Thread kontinuierlich Merge-Schritte aus, falls zu verschmelzende Runs zur Verfügung stehen. Diese Strategie beeinträchtigt im Kontext eines Systems allerdings andere Anfragen.

Um den Aufwand zu minimieren, kann man die Zahl der Merge-Schritte maximal verringern, indem man Merges nur mit maximalem Fan-In ausführt. Diese *MaxFanIn*-Strategie verzögert jedoch gegebenenfalls die Ergebnisproduktion erheblich und sollte nur in Umgebungen eingesetzt werden, in denen die Ressourcen sehr knapp sind.

Im Einsatz in einem DSMS empfiehlt sich ein Mittelweg, bei dem ein verfügbarer Teil der Gesamtressourcen eingesetzt wird, um zusätzliche Merge-Schritte zu ermöglichen, bevor der maximale Fan-In erreicht wird. Abbildung 11.1[CHKS05] zeigt einen solchen Mergebaum, bei dem an der linken Flanke kontinuierlich Ergebnisse produziert werden, während die Ebenen darunter lediglich so tief ausgelegt sind, dass die zur Verfügung stehenden Ressourcen ausreichen.

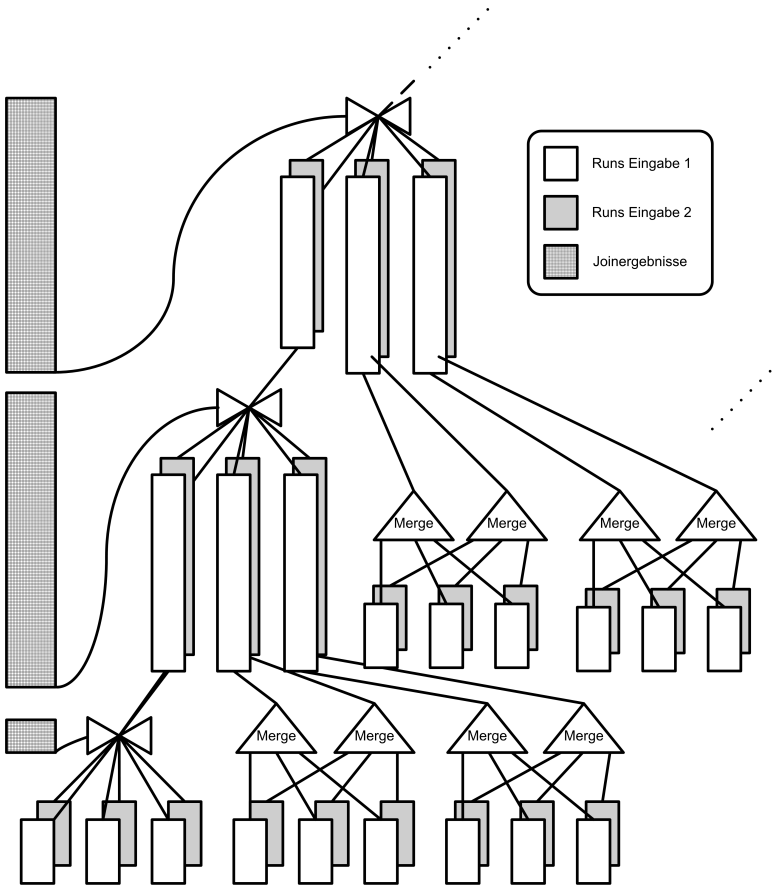


Abbildung 11.1.: Mergebaum beim PMJ

11.2.4. Limits

Der in diesem Abschnitt vorgestellte PMJ für Multimengen kann als Externspeicheralgorithmus in den meisten Fällen schon wesentlich größere Multimengen datengetrieben joinen als reine Hauptspeicherverfahren. Dennoch hat auch er seine Grenzen. Die Länge der einzelnen Runs steigt proportional zur Anzahl der bereits konsumierten Elemente, und je länger die Runs werden, umso mehr Ergebnisse sind bei deren Join zu erwarten. Entsprechend werden auch die Teilströme, die bei der sortierten Verarbeitung in den SweepAreas gepuffert werden müssen, immer länger. Daher können die SweepAreas der Merge-Schritte irgendwann nicht mehr komplett im Hauptspeicher gehalten werden. Zwar kann auch dieses Problem noch gelöst werden, indem man als SweepAreas ebenfalls (teilweise) Externspeicherstrukturen einsetzt, allerdings reduziert dies merklich die Performanz und damit die Rate, mit der Elemente verarbeitet werden können. Das Verfahren eignet sich deswegen nur für endliche Datenströme, deren Länge aber vorher nicht bekannt sein muss. Im nächsten Abschnitt wird daher eine Variante des PMJ vorgestellt, die über den in 2.3.3 definierten physischen Datenströmen operiert und damit durch den Einsatz von Gültigkeitsintervallen einen sortierbasierten Verbund über potentiell unbegrenzten Datenströmen erlaubt.

11.3. Temporaler PMJ (TPMJ)

Eines der wesentlichen Ziele dieser Arbeit ist es, die wichtigsten Mechaniken zur Verbundberechnung aus DBMS auf die Berechnung von Verbünden über physischen Datenströmen zu übertragen. Die in 9.3 vorgestellten Algorithmen liefen je nach Wahl der SweepAreas dabei die Entsprechungen zu Verfahren mit verschachtelten Schleifen und zur Verwendung von Hashing. In diesem Abschnitt wird nun die Übertragung der dritten wesentlichen Mechanik, der sortierbasierten Verbundberechnung, vorgestellt.

11.3.1. Grundlegendes Design

Auf den ersten Blick ist der PMJ als Verbundoperator, der zur Produktion früher Ergebnisse konstruiert wurde, prädestiniert zur Berechnung der in Definition 8.6 definierten Verbundoperation auf physischen Datenströmen. Als Ordnungskriterium für den sortierbasierten Algorithmus drängt sich dabei die Zeitachse auf. Da die Eingabedatenströme aber ohnehin nach Startzeitstempeln sortiert eintreffen, sind sie bezüglich dieser Ordnung bereits lokal sortiert, die Sortierschritte und die damit verbundenen frühen Merge-Schritte wären also hinfällig. Benötigt würde lediglich noch der letzte Merge-Schritt, in dem dann alle Ergebnisse direkt aus den Eingabedatenströmen berechnet werden. Eine konsequente Adaption dieses Schrittes auf die datengetriebene Verarbeitung ergibt dann aber letztendlich einen Algorithmus, der sich nicht merklich von dem in 9.3.2 vorgestellten ersten Algorithmus zur Berechnung des Verbundes über Datenströmen unterscheidet. Die Wahl dieser Ordnung bringt also keinen Fortschritt.

Zu einer sinnvollen Anwendung muss man daher die Sortierung auf die Wertkomponente der Elemente der physischen Datenströme beziehen. Bezüglich dieser aber treffen

die Elemente im Allgemeinen nicht sortiert ein, eine echte Sortierung muss also stattfinden. Allerdings wird der Algorithmus dadurch nicht von der Notwendigkeit befreit, seinen Ausgabedatenstrom nach Startzeitstempeln sortiert auszugeben. Diese steht im Widerspruch zu der Tatsache, dass der PMJ in jedem Mergeschritt die Ergebnisse lokal sortiert nach der globalen Ordnung auf den beteiligten Werten ausgibt und insgesamt die Ergebnisreihenfolge im Allgemeinen keiner Ordnung folgt. Diese widerstrebenden Ordnungsanforderungen müssen beim Design des *Temporalen PMJ* (TPMJ) berücksichtigt werden. Verbundalgorithmen, die Daten auf den Externspeicher auslagern, haben jedoch generell das Problem, dass die Anforderung an die nach Zeitstempeln sortierte Ausgabe das Weiterleiten von Ergebnissen verzögern kann, wie auch die Betrachtungen in [FH07] zeigen.

Gegenüber dem PMJ muss natürlich auch zusätzlich sicher gestellt werden, dass bei der Überprüfung der Joinbedingung die Schnittbedingung an die Gültigkeitsintervalle der beteiligten Elemente geprüft wird und dass bei der Ergebnisproduktion der entsprechende Schnitt dem Ergebnis als Gültigkeitsintervall zugewiesen wird.

Schließlich muss insbesondere noch sichergestellt werden, dass die temporale Ordnung ihre Aufgabe erfüllt, es nämlich ermöglicht, Elemente nicht unbegrenzt im Status des Operators halten zu müssen, sondern sie nach vollständiger Produktion der zugehörigen Ergebnisse entfernen zu können.

11.3.2. Algorithmus

Ausgabeordnung

Die grundlegende Idee für die Lösung des Ordnungsproblems liefert eine Erkenntnis, die beim PMJ noch ein Problem darstellte: In jedem Mergeschritt werden alle Ergebnisse erzeugt, die sich aus den am Merge beteiligten Elementen berechnen lassen. Dabei werden auch Ergebnisse erneut gefunden, die bereits in vorangegangenen Schritten identifiziert wurden. Der Algorithmus muss daher sicherstellen, dass diese mehrmals produzierten Ergebnisse nur je einmal ausgegeben werden (Duplikateliminierung). Während normalerweise beim PMJ solche Duplikate im Sinne der frühen Ergebnisproduktion jeweils bei ihrem ersten Auftreten ausgegeben und danach jeweils ausgefiltert werden, kann man das wiederholte Auftreten der Ergebnisse auch dazu ausnutzen, diese gerade dann zu produzieren, wenn das im Sinne einer nach Startzeitstempeln sortierten Ausgabeordnung sinnvoll ist.

Dazu nutzt man die Tatsache aus, dass alle Ergebnisse unter anderem an der linken Flanke des Mergebaumes produziert werden, da es zu einer beliebigen Menge an Eingabeelementen immer einen Knoten auf der linken Flanke gibt, der diese alle verarbeitet. Nun unterteilt man die Zeitachse vollständig in disjunkte Intervalle $[s_i, e_i)$ und ordnet diese den Knoten an der linken Flanke geeignet zu. In den jeweiligen Mergeschritten sollen dann genau die Ergebnisse produziert werden, deren Startzeitstempel in dem zugeordneten Intervall liegen. Die disjunkte Unterteilung stellt dabei sicher, dass keine Duplikate entstehen.

Um ein vollständiges Ergebnis zu gewährleisten, müssen im jeweiligen Mergeschritt natürlich auch alle entsprechenden Ergebnisse produziert werden können. Der Teilbaum

des Merge-Baumes unterhalb eines Knotens k der linken Flanke beinhaltet dabei für jeden Eingabestrom S_i jeweils alle Elemente bis zu einem letzten Element $(e_i^k, [t_{s_i}^k, t_{e_i}^k))$. Somit enthält er auf Grund der zeitlichen Ordnungsbedingung an physische Eingabeströme insbesondere für jeden Strom S_i alle Elemente mit einem Startzeitstempel, der echt kleiner $t_{s_i}^k$ ist. Damit enthält er dann auch alle Elemente aller n Eingabeströme, deren Startzeitstempel echt kleiner als $min_{i=1}^n(t_{s_i}^k)$ sind. Um ein Ergebnis mit Startzeitstempel t zu produzieren, muss der Startzeitstempel aller beteiligten Eingabeelemente kleiner oder gleich t sein. Daher können an dem Knoten der linken Flanke alle Ergebnisse produziert werden, deren Startzeitstempel im Intervall $[-\infty, min_{i=1}^n(t_{s_i}^k))$ liegen. Dieses Intervall ist zudem das größtmögliche, für das diese Aussage möglich ist, da ein Ergebnis mit Startzeitstempel $min_{i=1}^n(t_{s_i}^k)$ aus n Eingabeelementen mit eben diesem Startzeitstempel entstehen kann, die noch nicht Teil des Mergebaumes sein müssen. Um Ergebnisse weiterhin so früh wie möglich zu erzeugen, wählt man daher für den Knoten 1 links unten im Merge-Baum das entsprechende Intervall $[-\infty, min_{i=1}^n(t_{s_i}^1))$, und für alle anderen Knoten k der linken Flanke darüber das entsprechende Intervall abzüglich der Intervalle der Knoten darunter, also $[min_{i=1}^n(t_{s_i}^{k-1}), min_{i=1}^n(t_{s_i}^k))$. Insbesondere erreicht dies die gewünschte disjunkte Unterteilung der kompletten Zeitachse, wenn man im Falle endlicher Ströme abweichend für den letzten Knoten, also die Wurzel des Merge-Baumes, $+\infty$ als Ende setzt.

Da die linke Flanke des Mergebaumes von unten nach oben verarbeitet wird, genügt es nun, jeweils die in einem Mergeschritt produzierten Ergebnisse zu sammeln und dann nach Gültigkeitsintervallen sortiert auszugeben. Damit ist bereits eine globale Sortierung der Ausgabe nach Zeitintervallen sichergestellt. Sollte ein Merge-Schritt dabei sehr viele Ergebnisse produzieren und der Hauptspeicher knapp sein, kann das Sortieren der Ergebnisse nach Startzeitstempeln auch durch externes Sortieren vorgenommen werden.

Statusbereinigung

Nun gilt es noch, den temporalen Fortschritt auszunutzen, um Elemente wieder aus dem Status des Operators, also aus den in Verarbeitung befindlichen Runs, entfernen zu können. Da ein Eingabeelement mit Gültigkeitsintervall $[t_s, t_e)$ an keinem Ergebnis beteiligt sein kann, dessen Startzeitstempel größer oder gleich t_e ist, können während des Merges in Knoten k alle Elemente mit $t_e \leq min_{i=1}^n(t_{s_i}^k)$ verworfen werden. Die Vollständigkeit der Ergebnismenge bleibt dabei erhalten.

Struktur des Merge-Baumes

Die vorstehenden Modifikationen gegenüber dem PMJ ändern jedoch auch grundlegende Eigenschaften, die für den Aufbau des Merge-Baumes wesentlich waren. Zum einen werden Ergebnisse überhaupt nur noch an der linken Flanke produziert. Der Einsatz des Merge-Schrittes, der zugleich Joinergebnisse produziert, macht daher abseits der linken Flanke keinen Sinn mehr. Vielmehr ist der Rest des Baumes lediglich noch dazu da, der linken Flanke hinreichend lange nach Werten sortierte Runs der Eingabedatenströme zuzuführen. Da dabei kein Join mehr stattfindet, können die reinen Merge-Schritte dort aber auch wieder getrennt nach Eingaben erfolgen, da dadurch ein höherer Fan-In möglich

wird. Dadurch sind weniger Merge-Schritte erforderlich, oder es werden zumindest weniger Seiten gleichzeitig und damit weniger Hauptspeicherressourcen benötigt.

Zum einen geht die Eigenschaft, die alle Mergejoinverfahren vom externen Sortieren übernommen haben, nämlich dass die Menge der Ausgaberuns eines Mergeschrittes jeweils die Vereinigung der Eingaberuns ist, verloren. Damit macht eine balancierte Baumstruktur keinen Sinn mehr. Vielmehr muss nur noch sichergestellt werden, dass an der linken Flanke jeweils nur so viele Runs verarbeitet werden müssen, dass genügend Hauptspeicher für die SweepAreas und die Seiten zum Laden der Runs (Fan-In) verfügbar ist. Die neu zu generierenden Runs müssen also gar nicht immer länger werden. Dies würde vielmehr sogar dazu führen, dass die zur Ergebnisbestimmung verwendeten Intervalle $[min_{i=1}^{n}(t_{s_i}^{k-1}), min_{i=1}^{n}(t_{s_i}^{k}))$ immer länger würden, wodurch der Zeitraum vom Eintreffen der Elemente bis zur Produktion der zugehörigen Ergebnisse im Mittel immer länger würde. Daher sollte unter jedem Knoten der linken Flanke rechts ein etwa gleich großer Merge-Baum hängen.

Wahl der SweepAreas

Da es sich beim TPMJ wie beim klassischen ebenso wie dem datengetriebenen PMJ und dem Sort-Merge-Join (siehe 9.1.2) um einen sortierbasierten Verbund auf Basis einer wertbasierten Sortierung handelt, könnte man als SweepAreas die für diese entwickelten Strukturen verwenden. Diese müssten dazu dahingehend modifiziert werden, dass sie statt der Werte nun Stromelemente mit Gültigkeitsintervall speichern, und für wertbasiert identifizierte Verbundergebnisse zusätzlich die Schnittbedingung an die beteiligten Intervalle testen.

Betrachtet man als einfaches Beispiel jedoch die für den häufig benötigten Gleichverbund und andere Äquivalenzverbünde zu verwendende SweepArea, so muss diese lediglich Elemente einer Äquivalenzklasse ansammeln, diese alle ausgeben und komplett geleert werden können (siehe Tabelle 9.2). Hierfür eignet sich also fast jede beliebige Speicherstruktur. Zusätzlich ist im Gegensatz zu anderen Verbundalgorithmen für Datenströme beim TPMJ die Schnittbedingung an die Gültigkeitsintervalle nicht so häufig ohnehin erfüllt. Zur Erhöhung der Effizienz bietet es sich hier also an, eine SweepArea zu verwenden, welche die Stromelemente nach Gültigkeitsintervallen verwaltet und Intervallanfragen ermöglicht, zum Beispiel analog zur für den wertbasierten Intervallverbund auf Datenströmen verwendeten SweepArea auf Basis eines Intervallbaumes[Ede80].

Generell kann man die Schnittbedingung an die Gültigkeitsintervalle wie eine zusätzliche normale Verbundbedingung betrachten. Soll wertbasiert ein Intervallverbund berechnet werden, so wird dieser durch das Hinzufügen der Zeitachse als weitere Dimension somit zum räumlichen Verbund für achsenparallele Rechtecke in der Ebene.

11.3.3. Optimierung

An dieser Stelle werden noch einige Optimierungen vorgestellt, durch die der (T)PMJ weiter verbessert werden kann.

Mehrdimensionaler (T)PMJ

Wie schon in 10.3 diskutiert, kann die Verwendung eines mehrdimensionalen Verbundes an Stelle eines Baumes aus binären Verbünden erhebliche Vorteile bringen. Ähnliche Überlegungen werden in [Cam02] bezüglich des klassischen PMJ vorgestellt. Diese lassen sich entsprechend auch auf den datengetriebenen PMJ und die Merge-Schritte an der linken Flanke des TPMJ übertragen. Daher wurde die vorstehende Diskussion dieser Algorithmen unabhängig von der Zahl der zu joinenden Eingaben formuliert. Diese profitieren dabei insbesondere auch von der Möglichkeit, die Wahl der SweepAreas und der Anfragereihenfolgen zwischen den einzelnen Merge-Schritten problemlos modifizieren zu können. Der TPMJ hat damit eine gute Möglichkeit, sich kontinuierlich an sich ändernde Stromcharakteristika anzupassen.

Aufgabe der Ausgabeordnung

Die durch die in 2.4.1 motivierte Datenstromalgebra bedingte Bedingung, dass der von einem Operator produzierte Ausgabedatenstrom nach Startzeitstempeln sortiert produziert werden muss, kann in manchen Anwendungsfällen gelockert werden. Dies ist insbesondere dann der Fall, wenn der Verbundoperator die Wurzel eines Anfragegraphen darstellt und die Anfrage nicht als Unteranfrage dienen soll. Sollen die Ergebnisse nämlich beispielsweise lediglich präsentiert oder archiviert werden, ist gegebenenfalls keine Sortierung nach Zeitstempeln erforderlich. Während der Mehraufwand für diese Sortierung bei herkömmlichen Datenstromverbundalgorithmen noch moderat ist, kann gerade der TPMJ enorm von der Aufgabe der temporalen Ausgabeordnung profitieren, da er seine Ergebnisse dann nicht mehr nach der ursprünglich wertbasiert sortiert erfolgenden Ausgabeproduktion umsortieren muss, sondern direkt ausgeben kann.

Instanzübergreifende Ressourcenaufteilung

Alle bisherigen Überlegungen bezogen sich jeweils auf Funktionsweise und Optimierung einzelner Instanzen der Verbundoperatoren. Da übliche Operatoren von DSMS ihren Status komplett im Hauptspeicher halten und eintreffende Elemente umgehend verarbeiten, ist die Optimierung jeder einzelnen Instanz wichtig für das Gesamtsystem. Der TPMJ weist jedoch zwei Besonderheiten auf, die zusätzliche Optimierungen erlauben, wenn man mehrere Instanzen betrachtet, die im System gleichzeitig vorhanden sind, und zwar unabhängig davon, ob sie zur gleichen oder verschiedenen Anfragen gehören. Beim TPMJ besteht die unmittelbare Verarbeitung nämlich lediglich aus dem Puffern in den Listen zur Generierung initialer Runs. Die weitere Verarbeitung erfolgt dann nicht mehr direkt datengetrieben, sondern wird von zusätzlichen Threads gesteuert. Zudem benötigt der TPMJ nicht immer etwa konstant viel Speicher und CPU-Last, sondern der Verbrauch dieser für DSMS besonders wichtigen Ressourcen steigt immer dann kurzzeitig enorm an, wenn ein Merge-Schritt des Sortierens und insbesondere wenn ein Merge-Schritt mit Join an der linken Flanke stattfindet.

Daher bietet es sich an, die Entscheidung über die Durchführung solcher Merge-Schritte nicht den einzelnen Instanzen des TPMJ zu überlassen, sondern sie zentral zu

steuern. Da bei einem Merge-Schritt umso mehr Runs verarbeitet werden können, je mehr Speicher zur Verfügung steht, und da viele Varianten von Externspeicher noch an Performanz einbüßen, wenn viele Lese- und Schreiboperationen an verschiedenen Stellen gleichzeitig stattfinden, liegt es sogar nahe, global nur jeweils einen Merge-Schritt gleichzeitig stattfinden zu lassen. Wie schon beim einzelnen TPMJ ist es aber weiterhin empfehlenswert, zum Zwecke der früheren Ergebnisproduktion verfügbare Ressourcen zu nutzen, um zusätzliche Merge-Schritte auch dann schon auszuführen, wenn der maximal mögliche Fan-In noch nicht ausgeschöpft ist.

11.3.4. Einsatzgebiete

Zuletzt bleibt die Frage, wann sich der Einsatz des TPMJ lohnt. Das ist sicher nicht der Fall, falls die SweepAreas bei den normalen Verbundalgorithmen für temporale Datenströme durchgängig in den zur Verfügung stehenden Hauptspeicher passen, da Externspeicherzugriffe selbst beim Einsatz relativ schneller Externspeicher wie Solid State Disks immer noch sehr teuer sind.

Ein weiteres wichtiges Kriterium ist, dass die nun wertbasierten SweepAreas der Mergeschritte jederzeit gemeinsam in den Hauptspeicher passen sollten, da zusätzliche Externspeicherzugriffe in den SweepAreas den Algorithmus extrem ausbremsen. Eine wichtige Voraussetzung dafür ist, dass das Verbundprädikat für einen sortierbasierten Verbund geeignet sein muss. Kreuzprodukte oder allgemeine Theta-Verbünde profitieren nicht von einer Sortierung und machen daher als Einsatzgebiet für den (T)PMJ keinen Sinn.

Zusätzlich ist der TPMJ für Anwendungen ungeeignet, die es erfordern, dass die Zeit zwischen dem Eintreffen von Eingabeelementen und der Produktion der daraus berechenbaren Verbundergebnisse sehr kurz ist, da es gegebenenfalls recht lange dauern kann, bis die Eingabeelemente ihren Weg über die Eingabepuffer, Runs auf dem Externspeicher und die linke Flanke des Merge-Baumes bis in den Knoten gefunden haben, in dem sie in Ergebnisse eingehen. Anwendungen, die eine hohe sogenannte *Lifeliness* erfordern, also dass die Ausgabe sehr schnell auf Änderungen in der Eingabe reagiert, sind daher kein geeignetes Einsatzgebiet für den TPMJ. In solchen Fällen sollten daher Techniken wie die Reduktion von Zeitfensterlängen eingesetzt werden, um rein hauptspeicherbasierte Verbundalgorithmen einsetzen zu können.[CKSV06]

Analog ist der TPMJ für Verbünde ungeeignet, bei denen die Eingaberaten so hoch sind, dass eine Externspeichernutzung schon dadurch ausscheidet, dass die eintreffenden Elemente nicht so schnell herausgeschrieben werden könne, wie sie eintreffen.

Sind aber alle Voraussetzungen erfüllt, ermöglicht der TPMJ die Berechnung von Verbünden, die mit reinen Hauptspeicheralgorithmen nicht möglich sind, weil die Länge der Gültigkeitsintervalle zu große Hauptspeicheranforderungen stellen würde.

Teil IV.

Experimente

12. Einleitung

Nach den ersten Teilen, in denen die zentralen Themen dieser Arbeit behandelt werden, dient dieser Teil nun der experimentellen Auswertung der darin behandelten Konzepte. Kapitel 13 beschäftigt sich dabei mit den in Teil II thematisierten Metadaten, während Kapitel 14 Experimente zu den in Teil III behandelten Verbundoperationen beinhaltet. Die Zuordnung der Experimente zu diesen Kapiteln folgt dabei dem thematischen Schwerpunkt, denn Metadaten werden natürlich auch im Kontext von Verbundoperationen erhoben. Insbesondere wurde das in Teil II präsentierte Rahmenwerk für dynamische Metadaten auch zur Messung sämtlicher Laufzeitmetadaten der Experimente in Kapitel 14 verwendet. Zunächst werden in diesem Kapitel jedoch noch die für die Experimente verwendeten Soft- und Hardwareumgebungen beschrieben.

12.1. Implementierungsumgebung

Forschung ist ein inkrementeller Prozess. Die gesamte Teildisziplin der praktischen Informatik benötigt für ihre Existenz jahrhundertelange Vorarbeiten, welche zur Entwicklung von Computern führten. Aber auch diese Disziplin ist bereits so weit fortgeschritten, dass neue Entwicklungen immer auf existierenden aufbauen. So profitiert das noch verhältnismäßig junge Gebiet der Datenstromverarbeitung, in dem sich diese Arbeit bewegt, insbesondere von jahrzehntelangen Arbeiten im Bereich der Datenbanksysteme. Aber auch andere Teildisziplinen der Informatik wie verteilte oder parallele Systeme oder die theoretische Informatik beeinflussen die Arbeiten in der Datenstromforschung. Für diese bietet es sich daher an, auf etablierte Grundlagen wie die in den folgenden Unterabschnitten beschriebenen Softwarebibliotheken aufzubauen.

12.1.1. XXL

Eines der wesentlichen Ziele bei der Entwicklung der freie Softwarebibliothek *eXtensible and fleXible Library* (XXL)[BDS00, BBD+01, CHK+03] ist es, für das Forschungsgebiet der Datenbanksysteme diesen Prozess der inkrementellen Forschung zu unterstützen. Forscher sollen bei der Entwicklung neuer Verfahren nicht mehr dazu gezwungen sein, mit ihrer Implementierung von vorne beginnen zu müssen. Vielmehr versucht XXL einen Baukasten von etablierten Techniken bereit zu stellen, die ihrerseits modular aufgebaut und flexibel parametrisierbar sind. Dadurch können sich Wissenschaftler bei der Entwicklung neuer Techniken auf gerade diese neuen Aspekte fokussieren und für etablierte Grundlagen auf bestehende Implementierungen zurückgreifen.

Diese Herangehensweise hat zudem den Vorteil, dass experimentelle Vergleiche mit zuvor entwickelten Verfahren stark von ihr profitieren. Neue Verfahren können erheblich schneller und mit weniger Aufwand implementiert werden. Gleiches gilt für

Vergleichsverfahren, insbesondere natürlich dann, wenn diese bereits als Teil der Bibliothek zur Verfügung stehen. Und so durchgeführte Experimente liefern besonders faire und aussagekräftige Vergleiche, da diese so durchgeführt werden können, dass sich die Implementierung gerade nur im zu untersuchenden Aspekt unterscheidet, während der Rest identisch aus der Bibliothek verwendet wird.

Zur Implementierung von XXL wurde die Programmiersprache Java gewählt, die sehr beliebt und daher vielen Entwicklern in Forschung und Entwicklung bekannt ist. Diese Wahl bietet zudem den Vorteil der plattformübergreifenden Ausführbarkeit, der als Nachteil aber auch eine erhöhte Abstraktion von der Hardware und damit geringeren Einfluss auf deren Ansteuerung mit sich bringt.

12.1.2. PIPES

Für die Datenstromforschung in der Arbeitsgruppe Datenbanksysteme am Fachbereich Mathematik und Informatik der Philipps-Universität Marburg ist es daher nahe liegend, auf das in derselben Gruppe entwickelte XXL aufzubauen. Dazu wurde die Forschungsbibliothek *Public Infrastructure for Processing and Exploring Streams* (PIPES)[KS04] entwickelt, die zum einen XXL benutzt und zum anderen dessen Philosophie folgt. Dementsprechend entstanden die Implementierungen für das Forschungsprojekt *Anfrageverarbeitung aktiver Datenströme* und auch alle für diese Arbeit entstandenen Implementierungen in oder zumindest mit PIPES.

Dies bringt in vieler Hinsicht Vorteile. Durch die Implementierung als Teil der Bibliothek werden die dabei entwickelten Komponenten vielfach benutzt und somit getestet. Dadurch werden nicht nur eventuelle Implementierungsfehler leichter entdeckt, sondern insbesondere auch die Konzepte und die Performanz eingehend bezüglich ihrer Praxistauglichkeit geprüft. So wurden im Rahmen dieser Arbeit entwickelte Implementierungen nicht nur für die in diesem Teil präsentierten Experimente benutzt, sondern auch für weitere Arbeiten im Rahmen des Forschungsprojekts (siehe 15.3).

Zudem zeigt die Einbindung der entwickelten Algorithmen und Verfahren in die Bibliothek, dass das Ziel erreicht wurde, keine Insellösungen zu entwickeln, sondern Ergebnisse die zur Anwendung in einem DSMS geeignet sind. Denn während PIPES zunächst eine Bibliothek von Datenstromverarbeitungstechniken darstellt, wird es durch Auswahl und Kombination enthaltener Komponenten zu einem voll funktionierenden System.

12.2. Ausführungsumgebung

Die Experimente in diesem Teil wurden zum großen Teil auf einem PC mit einem mit 3 GHz getakteten *Intel Core 2 Duo E6850* Prozessor und 6 GB RAM unter *Java 7* ausgeführt. Als Betriebssystem kam dabei *Microsoft Windows Vista Enterprise 64bit* mit Service Pack 2 zum Einsatz. Die bereits vorab veröffentlichten Experimente wurden unter *Microsoft Windows XP* auf Systemen mit *Intel Pentium* Prozessoren bei 2,3 bis 2,8 GHz mit 1 GB RAM durchgeführt. Obwohl diese Systeme über für DSMS verhältnismäßig wenig Hauptspeicher verfügen, sind sie für die präsentierten Ergebnisse völlig hinreichend, da

sie das Abbilden realistischer Anwendungsfälle ermöglichen. Größere Hardware wird in der Anwendungspraxis zumeist erst für die vertikale Skalierung auf eine hohe Zahl parallel auszuführender Anfragen benötigt.

Dabei gilt es zusätzlich zu beachten, dass die *Java Virtual Machine* insbesondere älterer Java-Versionen bei hohen Hauptspeichergrößen zu für die Latenz eines DSMS nachteiligen längeren Speicherbereinigungszyklen neigt. Dieser zusätzlichen Anforderung kann aber durch Verwendung von Speicherbereichen außerhalb des Java-Heaps Rechnung getragen werden, was wiederum mit entsprechend modifizierten SweepArea-Implementierungen erfolgen kann. Da es sich hierbei um ein spezifisches Problem von Java handelt, das zudem in aktuelleren Java-Versionen auf Grund darin verfügbarer kontinuierlicher Speicherbereinigungsstrategien nur noch bedingt auftritt, wurde es nicht in die allgemeine Diskussion der in dieser Arbeit besprochenen Verfahren einbezogen.

13. Metadaten

Bei der Entwicklung des in Teil II vorgestellten Rahmenwerkes zur Bereitstellung dynamischer Metadaten wurden mehrere Ziele verfolgt. Es soll in der Lage sein, bei Bedarf präzise Metadaten zu liefern, es soll dabei möglichst wenig Zusatzaufwand verursachen und es soll flexibel konfigurierbar sein. In diesem Kapitel wird nun anhand diverser Experimente aufgezeigt, dass diese Ziele erreicht wurden.

In 13.1 wird zunächst untersucht, inwieweit das Erfassen dynamischer Metadaten Zusatzaufwand erzeugt. Dabei wird auch aufgezeigt, wie die Möglichkeit zur dynamischen Aktivierung und Deaktivierung diesen Mehraufwand reduzieren kann. In 13.2 wird dann untersucht, wie die Parametrisierung des Rahmenwerkes die Genauigkeit der Metadaten beeinflusst. Zudem wird der Einfluss der eingesetzten Techniken zur Absicherung von negativem Effekten der Nebenläufigkeit analysiert. In 13.3 werden Techniken zur Weiterverarbeitung der Metadaten untersucht. Abschließend wird in 13.4 aufgezeigt, wie dynamische Metadaten zur Validierung des in 10.1 vorgestellten Kostenmodells eingesetzt werden können.

13.1. Durchsatz in Abhängigkeit von Metadatennutzung

Datenstromraten und Speicherverbrauch von Operatoren sind wichtige Laufzeitmetadaten, die Einblick in die Vorgänge innerhalb eines DSMS bieten. In diesem Abschnitt wird daher anhand dieser Beispiele untersucht, welchen Einfluss deren Messung, die unter Verwendung verschiedener Typen von Metadatahandlern erfolgt, auf den Aufwand typischer Operatoren hat, die ihrerseits moderaten Aufwand verursachen.

13.1.1. Filter mit Messung der Ein- und Ausgaberate

Im ersten Experiment wird der Durchsatz eines einfachen Filters in Abhängigkeit davon, zu welchem Zeitpunkt die Bereitstellung der Metadaten *Eingaberate* und *Ausgaberate* aktiviert wird, untersucht. Dabei handelt es sich um Metadaten mit periodischer Aktualisierung (5.2.2).

Dazu werden jeweils die Zahlen 0 bis 100 Millionen in einen Filter geleitet, der nur gerade Zahlen akzeptiert. In einem Lauf werden die genannten Metadaten durchgängig aktiviert, in einem komplett deaktiviert. In den anderen Läufen werden sie teilweise aktiviert, je einmal bis zur und ab der Hälfte der Eingabedaten und einmal zwischen 25% und 75% der Eingabe. Gemessen wurde jeweils die Zeit, die zur Verarbeitung benötigt wurde, und zwar in Schritten von 500.000 Elementen.

Die Ergebnisse sind in Abbildung 13.1 dargestellt. Wie zu erwarten wurde der höchste Durchsatz erzielt, wenn keine Metadaten aktiviert sind. Im Gegensatz dazu benötigte

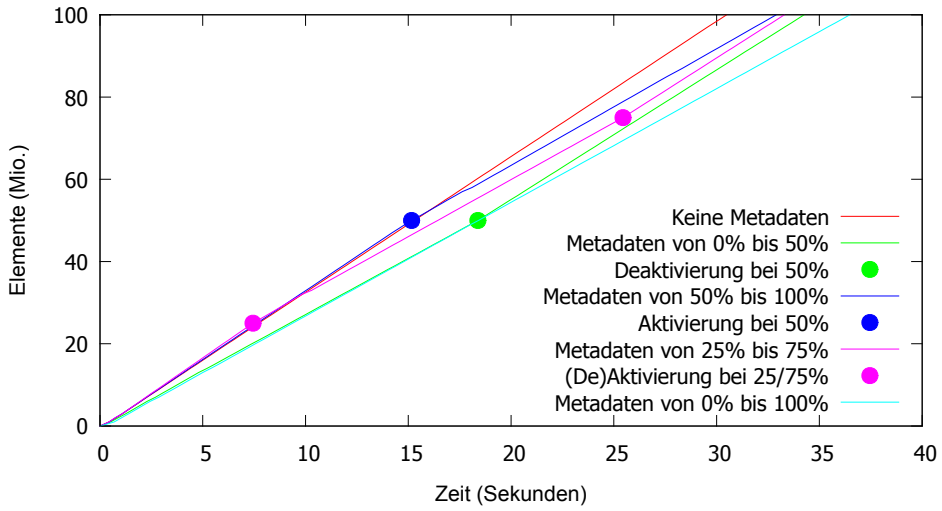


Abbildung 13.1.: Durchsatz eines Filters in Abhängigkeit von Ratenmessung

der Lauf mit durchgängig aktivierten Metadaten sechs Sekunden länger. Dieser Laufzeitzuwachs von etwa 20% erscheint zunächst moderat. Allerdings entfällt ein guter Teil des Gesamtaufwandes des Experimentes gar nicht auf den Filter. Ein Vergleichslauf ohne Filter benötigt etwa 15,6 Sekunden. Der Filter ist also nur für knapp die Hälfte des Gesamtaufwandes verantwortlich. Die Kosten des Filters erhöhen sich durch Aktivierung der Metadaten also um gut 40%. Dies bestätigt die Erwartung, dass der zur Bereitstellung selbst einfacher Metadaten benötigte Mehraufwand im Verhältnis zum Verarbeitungsaufwand eines Operators nicht vernachlässigbar ist (4.2.1).

Der Lauf, bei dem die Metadaten erst nach Verarbeitung der Hälfte der Eingabe angefordert wurden, hat die zweitbeste Gesamtlaufzeit. Er ist schneller als die beiden Läufe, bei denen die Metadaten zu Beginn beziehungsweise zwischenzeitlich für die gleiche Zahl an Eingabeelementen angefordert wurden. Dies erklärt sich dadurch, dass der zum Bereitstellen der Metadaten gestartete Thread nach der Deaktivierung noch kurz weiter läuft, bis er erkennt, dass er nicht länger benötigt wird. Dennoch zeigen die Läufe, dass eine nur zwischenzeitliche Aktivierung der Metadaten den Gesamtdurchsatz weniger verringert als eine komplette Aktivierung. Zudem ist am Ansteigen der Steigung nach Deaktivierung der Metadaten deutlich erkennbar, dass sich der Durchsatz jeweils nach Abschalten der Metadaten wieder erholt.

In diesem Experiment wurden Metadaten untersucht, die vergleichsweise hohen Aufwand verursachen. Zum einen erfordert die Ratenmessung die Erfassung jedes einströmenden beziehungsweise ausströmenden Elementes, zum anderen ist eine periodische Aktualisierung nötig, um diese korrekt zu erfassen (siehe 5.2.2).

13.1.2. Vereinigung mit Messung der Größe der Datenstruktur

In einem weiteren Experiment werden zum Vergleich Metadaten untersucht, die mit bedarfsgesteuerter Aktualisierung (siehe 5.2.1) auskommen und daher keinen Aufwand pro Element verursachen.

Dazu werden zwei bezüglich der Zeitstempel um 10.000 Elemente gegeneinander verschobene Datenströme von jeweils 100.000.000 Elementen durch eine temporale Vereinigung geschickt. Währenddessen sind die Metadaten *Heapgröße* und *Heapspeichergröße* aktiviert. Da die Aktivierung allein lediglich Handler bereitstellt und sonst keinen Aufwand verursacht, werden die Metadaten mit Hilfe eines zusätzlichen Threads im Abstand von 10, 100 beziehungsweise 1.000 Millisekunden abgefragt. Zum Vergleich wird ein weiterer Lauf ohne Metadaten durchgeführt.

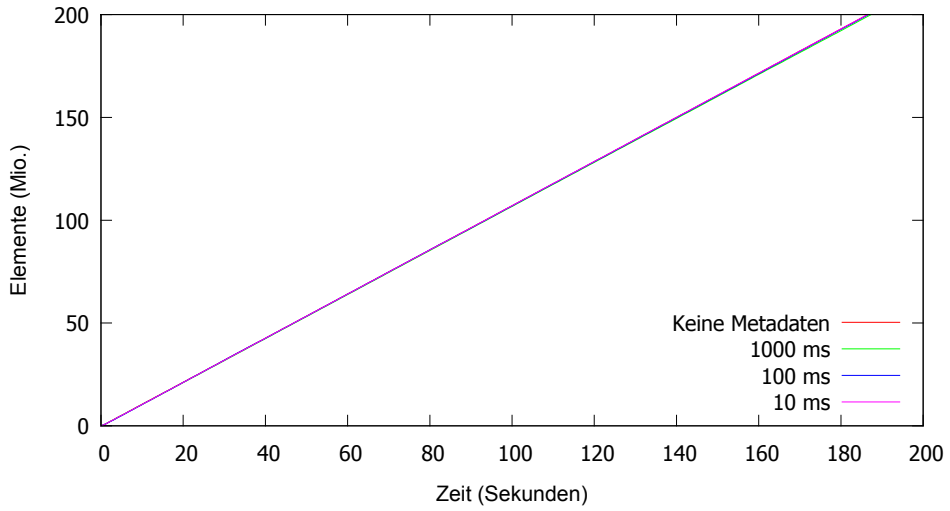


Abbildung 13.2.: Durchsatz einer Vereinigung in Abhängigkeit von Speicherverbrauchsmessung

Die Ergebnisse sind in Abbildung 13.2 dargestellt. Die Größe der Heaps der Vereinigung muss ohnehin immer mitgeführt werden, unabhängig davon, ob diese als Metadatum bereitgestellt wird. Ein Zugriff erfordert also lediglich das Auslesen dieser Größe. Da selbst bei Aktualisierung einmal pro 10 Millisekunden zwischen zwei Zugriffen auf die Metadaten mehr als 10.000 Elemente verarbeitet wurden, verursachte der Metadatenzugriff wie erwartet keinen wesentlichen Mehraufwand, der sich im Experiment in einer längeren Laufzeit hätte äußern müssen. Dargestellt wird ein zusammengehöriger Satz von Läufen. In weiteren Läufen zeigte sich, dass schon geringfügige Beeinträchtigungen durch andere Prozesse oder durch die Prozessor- und Cachearchitekturen bedingte Effekte sogar dazu führen können, dass Läufe mit häufigerem Metadatenzugriff schneller sind als solche mit seltenerem.

13.2. Genauigkeit periodisch aktualisierter Metadaten

Die Selektivität von Selektionen und Verbünden spielt sowohl in DBMS als auch in DSMS eine besonders wichtige Rolle für die Kosten von Anfrageplänen, da sie die Größe von Ergebnissen beziehungsweise die Ausgaberaten wesentlich beeinflusst. Bei der Datenstromverarbeitung ist es daher wichtig, Selektivitäten zur Laufzeit effizient untersuchen zu können. Dieses dynamische Metadatum wird daher in der in diesem Abschnitt vorgestellten Serie von Experimenten verwendet. Diese dienen sowohl der Untersuchung der Genauigkeit von periodisch aktualisierten Metadaten (siehe 5.2.2), als auch der des Einflusses des Multithreadings auf den Overhead, der durch Bereitstellung der Metadaten entsteht.

Dazu wird ein Strom von 200.000.000 Elementen durch einen Filter geschoben und die *Selektivität* als Metadatum mit periodischer Aktualisierung gemessen. Der Strom ist dabei so gewählt, dass die Selektivität zunächst kontinuierlich von 0 auf 1 anwächst und danach wieder auf 0 zurück geht. Unabhängig von der Aktualisierungsrate wird der Wert des Metadatums für die Messung in allen Experimenten im Takt von 100 Millisekunden abgetastet.

Zur besseren Vergleichbarkeit wurden alle Experimente jeweils fünfmal wiederholt. Hier werden die jeweils schnellsten Läufe diskutiert. Die längeren Läufe dauerten aber jeweils maximal drei Prozent länger.

13.2.1. Selektivität eines Filters

In diesem Experiment wird die Selektivität mit den Standardeinstellungen von *PIPES* gemessen. Die periodische Aktualisierung erfolgt dabei im Takt von 500 Millisekunden.

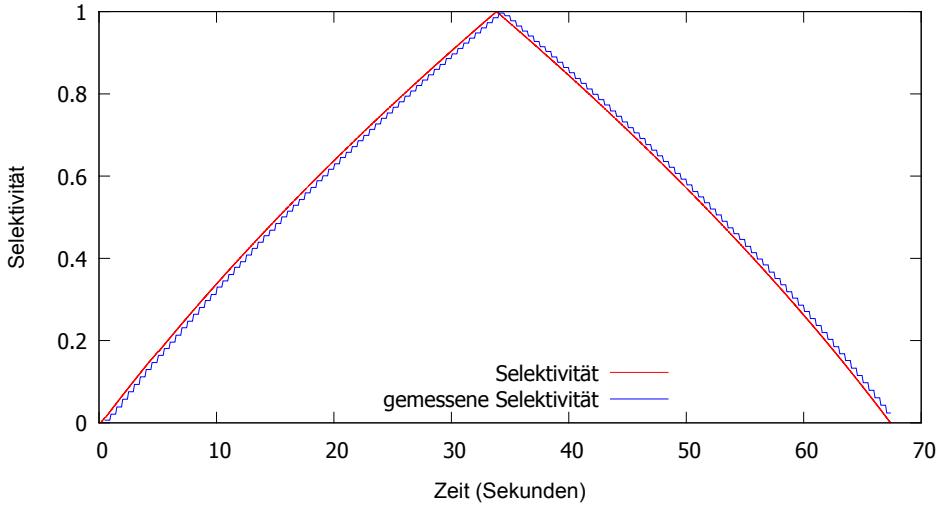


Abbildung 13.3.: Messung der Filterselektivität mit Standardeinstellungen

Die Ergebnisse sind in Abbildung 13.3 dargestellt. Als Referenzkurve ist die an der

Eingabe produzierte Selektivität dargestellt. Diese Kurve verläuft nicht exakt linear ansteigend und dann abfallend, sondern hat jeweils eine leichte Ausbeulung nach oben. Dies rührt daher, dass bei niedriger Selektivität viele Elemente im Filter verworfen werden, während sie bei hoher Selektivität durch die Folgeverarbeitung zusätzlichen Aufwand verursachen.

Wie erwartet approximiert die gemessene Selektivität die Referenzkurve gut, allerdings mit einer geringen Verzögerung. Diese rührt im Wesentlichen daher, dass die Zähler jeweils für einen Zeitraum von 500 Millisekunden laufen, bis die periodische Aktualisierung daraus ein Metadatum berechnet. Danach kann es nochmal bis zu 100 Millisekunden dauern, bis dies auch für die Darstellung abgetastet wird. Dennoch sind die gemessenen Metadaten sicherlich geeignet, valide Erkenntnisse über die tatsächliche Selektivität zu gewinnen.

Führt man dasselbe Experiment ohne Messung der Selektivität durch, so ergibt sich eine Gesamtlaufzeit von knapp unter 59 Sekunden. Es zeigt sich also wieder deutlich der Overhead der Metadatenbereitstellung (siehe 4.2.1). In den folgenden Experimenten wird daher untersucht, wie sich diese Problematik verringern lässt.

13.2.2. Verringerte Aktualisierungsrate

Eine nahe liegende Vorgehensweise zur Verringerung des Aufwandes ist die Verringerung der Aktualisierungsrate.

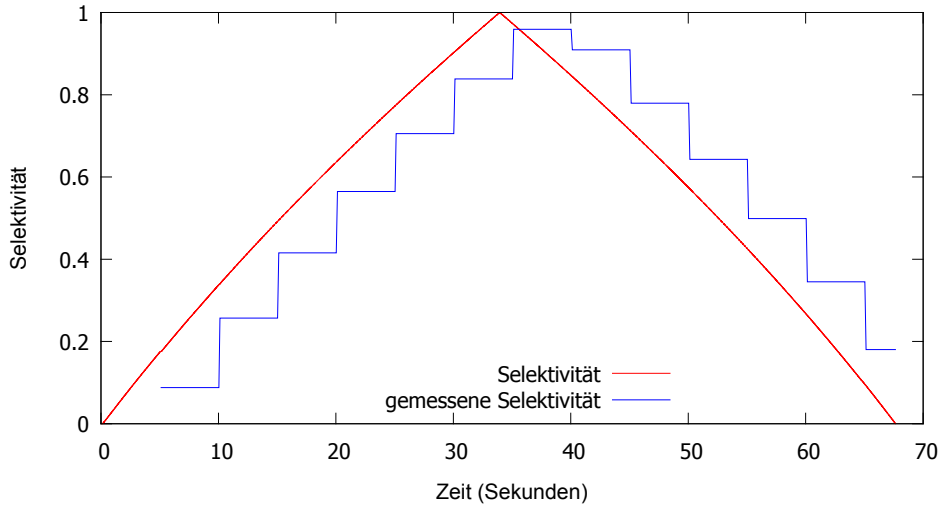


Abbildung 13.4.: Messung der Filterselektivität mit verringerter Aktualisierungsrate

Abbildung 13.4 zeigt den Verlauf der gemessenen Selektivität bei einem Aktualisierungstakt von 5 Sekunden. Wie erwartet dauerte es dabei länger, bis die gemessene Selektivität der zuvor tatsächlichen entsprach. Je nach Anwendungsszenario kann jedoch auch eine derartige Auflösung noch hinreichend sein. Allerdings verringert sich die

Gesamtlaufzeit nicht. Dies liegt darin begründet, dass schon bei einer Rate von zwei Hertz pro Aktualisierung mehr als 44.000 Elemente verarbeitet wurden. Der Aufwand für die periodische Aktualisierung ist also im Vergleich zum Aufwand pro Element vernachlässigbar.

13.2.3. Temporäre Aktivierung

Der Aufwand zur Bereitstellung von Metadaten kann dadurch verringert werden, dass sie nicht dauerhaft, sondern lediglich temporär aktiviert werden. In diesem Experiment wird die Selektivität daher nicht durchgängig, sondern im Abstand von zehn Sekunden für jeweils eine Sekunde gemessen.

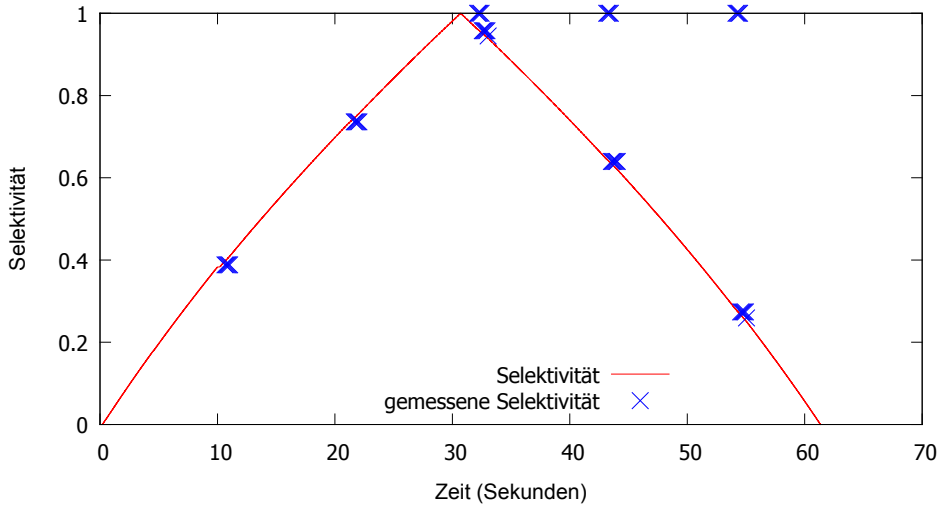


Abbildung 13.5.: Zeitweise Messung der Filterselektivität

Abbildung 13.5 zeigt, dass die Selektivität bereits jeweils kurz nach der Aktivierung wieder den korrekten Wert gut approximierte. Lediglich für das jeweils erste Bezugsintervall ergaben sich falsche Werte. Die extremen Werte dort liegen darin begründet, dass der Datengenerator die Testdaten in Blöcken zu 10.000 Elementen erzeugt, wobei diejenigen, die den Filter passieren, alle am Ende erzeugt werden. Liegen Aktivierung der Selektivitätsmessung und Aktualisierung daher nahe beieinander, so wurde gegebenenfalls kurzzeitig eine Selektivität von 1 gemessen.

Darin zeigt sich, dass eine Verkürzung der Bezugsintervalle generell ermöglichen würde, kurzzeitige Schwankungen der Selektivität besser zu erfassen. Allerdings kommt es stark auf die Anwendung an, ob das sinnvoll ist. Soll die gemessene Selektivität zum Zwecke der Optimierung verwendet werden, so sollte eher vermieden werden, auf kurzzeitige Ausreißer zu reagieren.

An der im Vergleich zu den Läufen mit dauerhaft aktivierter Selektivitätsmessung deutlich reduzierten Laufzeit zeigt sich, dass die temporäre Deaktivierung wie erwartet

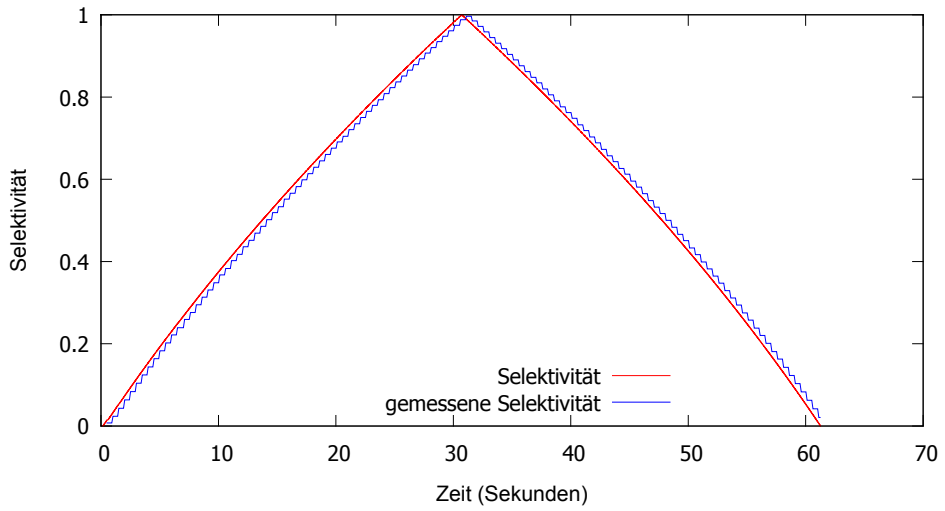


Abbildung 13.6.: Messung der Filterselektivität bei Absicherung gegen Lock des Filteroperators

den Gesamtaufwand reduzierte. Allerdings werden deutlich weniger als 10/11 der Laufzeitverlängerung eingespart, da auch das Aktivieren und Deaktivieren Aufwand verursacht und zudem den Filter jeweils kurzzeitig lahmlegt.

13.2.4. Alternatives Locking

Die Messung der Selektivität erfolgt mittels Dekoration des Filterprädikates, das heißt das Filterprädikat wird durch eines ersetzt, welches das ursprüngliche benutzt und zusätzlich Zähler für Aufrufe und Erfülltsein des Prädikates führt. Dadurch entfällt die Prüfung pro Element, ob das Zählen aktiviert ist.

Bei periodischem Aktualisieren der daraus abgeleiteten Selektivität müssen nun beide Zähler ausgelesen und auf 0 gesetzt werden. Da keinerlei Garantien für kontinuierlichen Datenfluss bestehen, muss die periodische Aktualisierung durch einen gesonderten Thread erfolgen (siehe 5.2.2). Somit greifen aber sowohl die Threads, die Daten durch den Anfragegraphen schieben, als auch der Metadatenaktualisierungsthread konkurrierend auf die Zähler zu. Dabei können unerwünschte Effekte wie *Lost Updates* auftreten. Geht so beispielsweise das Zurücksetzen eines Zählers verloren, so wird die Selektivität im nachfolgenden Zeitraum etwa halbiert (falls der Aufrufzähler nicht zurückgesetzt wird) oder verdoppelt. Solche groben Fehler sollten natürlich verhindert werden. Daher wurden Zugriffe auf die Zähler in den vorstehenden Experimenten über eine separate Semaphore abgesichert. In den folgenden Experimenten soll nun untersucht werden, ob es effizientere Methoden gibt, die Probleme durch konkurrierende Zugriffe zu beheben.

13.2.5. Nutzung des Operatorlocks

PIPES ermöglicht generell Multithreading bei der Datenstromverarbeitung. Dazu werden unter anderem alle Operatoren durch ein Lock dagegen abgesichert, dass mehrere Threads gleichzeitig versuchen, damit Daten zu verarbeiten. Dies schützt auch das Inkrementieren der Zähler für die Selektivitätsmessung, da diese als Teil der Verarbeitung von einströmenden Elementen erfolgt. In diesem Experiment wird nun untersucht, wie es sich auswirkt, wenn der Zugriff auf die Zähler statt über eine separate Semaphore ebenfalls über das Lock des Operators abgesichert wird.

Abbildung 13.6 zeigt, dass die Selektivität nach wie vor korrekt erfasst wird, die Gesamtlaufzeit aber deutlich sinkt, da nun der Zusatzaufwand pro verarbeitetem Element für die separate Semaphore entfällt.

Dies zeigt, dass die Absicherung der Metadaten gegen konkurrierende Aktivitäten wenn möglich durch Locks erfolgen sollte, die ohnehin verwendet werden. Allerdings ist dies aus Gründen möglicher Verklemmungen nicht immer möglich. Die Operatorlocks werden von Threads, die Elemente durch den Operatorgraphen schieben, in Richtung des Datenflusses gesperrt. Gegenläufige operatorübergreifende Metadatenaktualisierungen dürfen dabei keinesfalls die Locks in gegenläufiger Richtung sperren.

Verringerte Aktualisierungsrate bei Nutzung des Operatorlocks

Auch dieses Experiment wurde mit verringerter Aktualisierungsrate wiederholt.

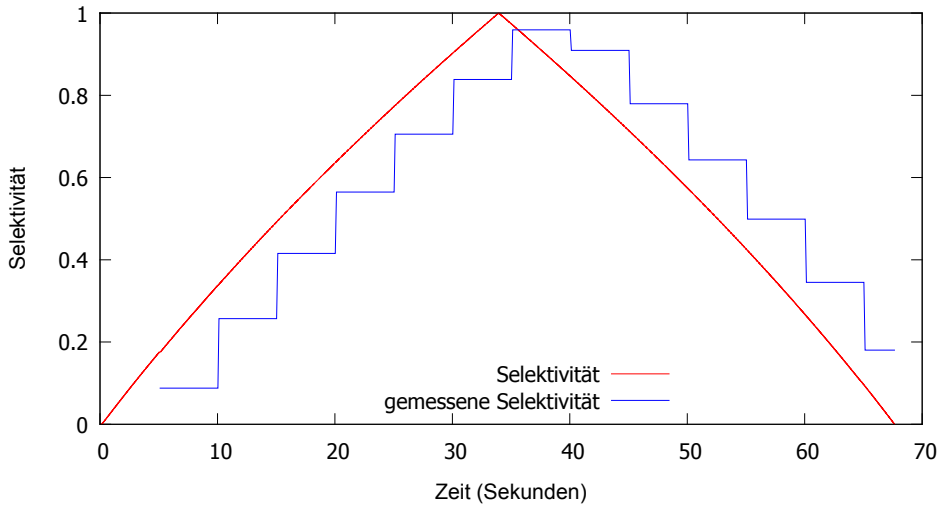


Abbildung 13.7.: Messung der Filterselektivität mit verringerter Aktualisierungsrate

Abbildung 13.7 zeigt erneut eine deutlich verringerte Gesamtlaufzeit im Vergleich zur Messung mit separater Semaphore. Allerdings zeigte sich erneut, dass die Verringerung der Aktualisierungsrate lediglich die Qualität der gemessenen Selektivität verringert,

während keine merklichen Einsparungen an Aufwand gegenüber der vergleichbaren Messung mit höherer Aktualisierungsrate vorliegen.

Vermeidung des Lockings

Ist man bereit, kleinere Fehler bei der Selektivitätsmessung in Kauf zu nehmen, so kann man auf die Verwendung eines Locks auch verzichten. Die Idee ist es, die beiden Zähler als Array zu bündeln und dieses bei der periodischen Auswertung auszutauschen. Dadurch wird sichergestellt, dass die Möglichkeit des groben Fehlers durch inkonsistent verloren gegangene Aktualisierungen entfällt. Allerdings können immer noch einzelne Zählereignisse verloren gehen.

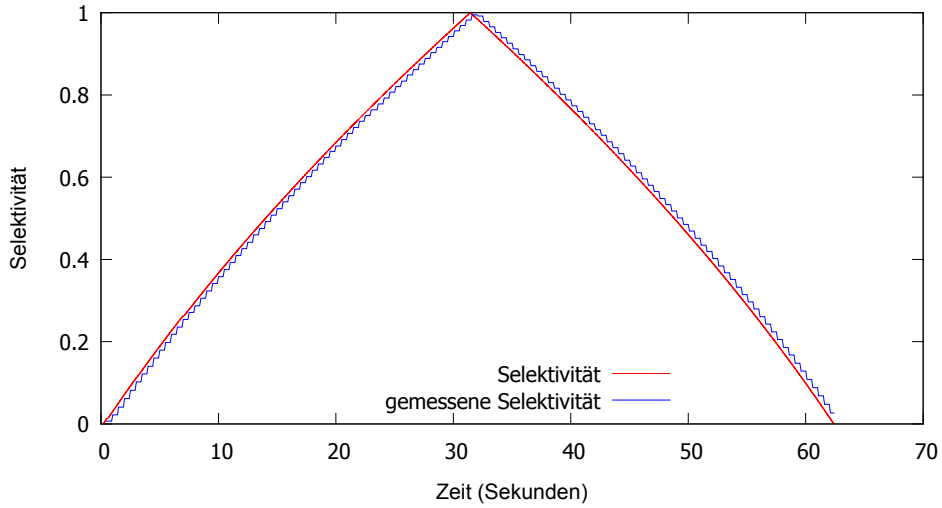


Abbildung 13.8.: Messung der Filterselektivität ohne Locks

In Abbildung 13.8 sind die Ergebnisse der Messung aufgetragen. Es ist kein Qualitätsunterschied gegenüber den vorangegangenen sicheren Messungen erkennbar. Allerdings zeigte sich, dass der nun pro verarbeitetem Element anfallende kleine Zusatzaufwand für die Benutzung des Arrays sich insgesamt auf die Laufzeit schlechter auswirkt als das Holen des Operatorlocks pro periodischer Aktualisierung.

Obwohl bei Metadaten Fehler in gewissem Umfang durchaus akzeptabel sein können, birgt nicht abgesicherte Nebenläufigkeit im Allgemeinen hohe Risiken und sollte nur mit höchster Vorsicht eingesetzt werden.

13.3. Weiterverarbeitung von Metadaten

Um Entscheidungen auf Basis von Metadaten zu treffen, ist es zumeist nicht ratsam, diese allein auf einer Momentaufnahme basieren zu lassen. Insbesondere bietet es sich an, diese

vorher zu aggregieren. *PIPES* bietet dazu zwei Mechanismen an, die in diesem Abschnitt untersucht werden sollen.

In den Experimenten wird jeweils ein Datenstrom benutzt, der im Filter eine sinusförmig um 0,5 schwankende Selektivität ergibt.

13.3.1. Einfache Aggregation von Metadaten

In diesem Experiment wird die in 6.2.2 vorgestellte einfache Aggregation von Metadaten benutzt.

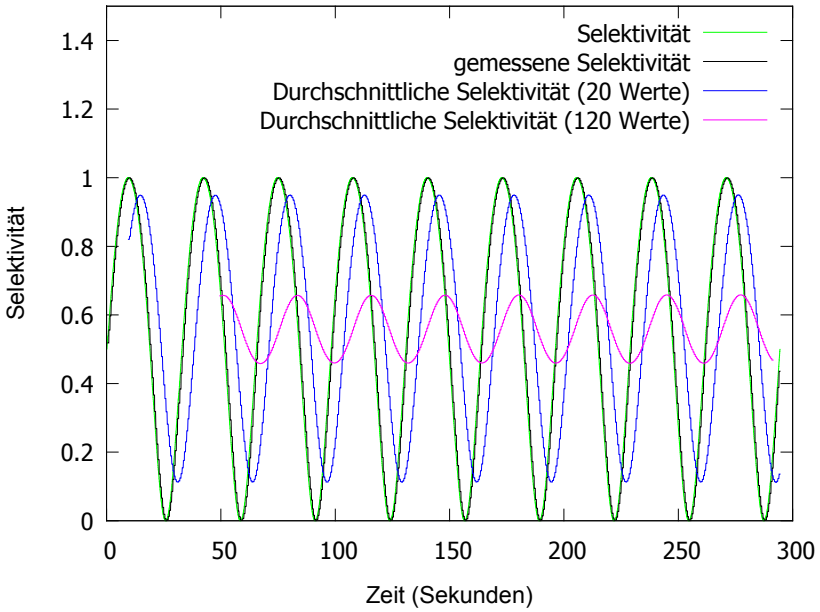


Abbildung 13.9.: Aggregierte Metadaten

Abbildung 13.9 zeigt neben der verwendeten Selektivität auch das direkt ausgelesene Metadatum Selektivität. Wie in den vorstehenden Experimenten lief dies der tatsächlichen Selektivität nach, wobei die Verzögerung durch die längere Laufzeit des Experimentes kaum noch zu erkennen ist.

Zwei weitere Kurven zeigen die mittels zählbasierter Fenster aggregierte Selektivität, wobei das Aggregat erst nach Füllung des kompletten Fensters als gültig konfiguriert wurde. Es wurde der Mittelwert der jeweils letzten 20 und letzten 100 Werte der gemessenen Selektivität aggregiert. Bei 2 Hertz Aktualisierungsrate entspricht das zehn Sekunden beziehungsweise einer Minute Fenstergröße. Wie erwartet schwanken die Aggregate mit größerem Fenster weniger und erfassen damit die mittlere Selektivität besser.

13.3.2. Metadatenstromverarbeitung

In einem weiteren Experiment wird die in 6.2.3 vorgestellte Alternative genutzt, Metadaten als Datenstrom zur Verfügung zu stellen. Die Benutzung dieser Variante erfordert

mehr Konfigurationsaufwand, bietet aber erweiterte Möglichkeiten. Wie im vorherigen Experiment werden Mittelwerte der Selektivität über verschiedene Fensterlängen ermittelt. Zudem wird die Möglichkeit genutzt, durch ein Kreuzprodukt mit folgender Abbildung die absolute Differenz des Stroms mit der kleinsten und der größten Fensterlänge zu ermitteln.

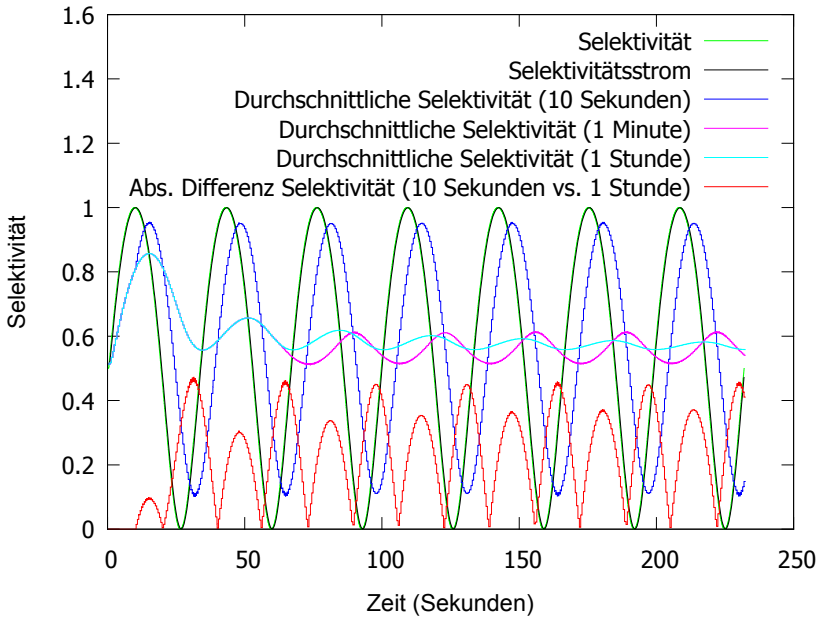


Abbildung 13.10.: Metadatenstromverarbeitung

Abbildung 13.10 zeigt wiederum die verwendete Selektivität, diesmal jedoch den Datenstrom mit dem Selektivitätswerten. Da sich diese unmittelbar aus dem Metadatum ergeben, zeigt sich analog vom vorherigen Experiment eine sehr gute Messung der Selektivität mit geringem Nachlauf.

Ebenfalls analog zum vorstehenden Experiment ergeben sich für die Fensterlängen zehn Sekunden und eine Minute zunehmend bessere Approximationen der mittleren Selektivität von 0,5. Der Strom mit dem Fenster von einer Stunde, was in Anbetracht der Laufzeit gleichbedeutend mit dem Mittelwert über alle gemessenen Selektivitäten ist, wich am Ende kaum noch davon ab.

Im Unterschied zum vorherigen Experiment wurden die Aggregate von Beginn an gemessen, wodurch sich die typischen Anlaufeffekte zeigen. Zunächst erfassten alle drei gleitenden Fenster dieselben Werte und ergaben daher die gleichen Mittelwerte. Die bezüglich eines Fensters aggregierte Selektivität weicht erst nach dessen Füllung von den Aggregaten bezüglich der jeweils größeren ab, also wenn zum ersten Mal Werte aus dem Fenster fallen.

Der Absolutbetrag der Differenz zwischen den Werten des bezüglich des kleinsten beziehungsweise größten Fensters berechneten Mittelwertes ist daher zunächst Null. Später erfüllt die Kurve gut die Intention, mit akzeptabler Verzögerung zu erkennen, dass die Selektivität maßgeblich vom längerfristigen Mittel abweicht.

13.4. Metadaten zur Validierung des Kostenmodells

Das bei der kostenbasierten Optimierung (siehe 10.1) eingesetzte Kostenmodell ist approximativ, da ein exaktes Kostenmodell Berechnungen der Komplexität der abzuschätzenden Aufgabe erfordern würde. Daher ist es wichtig zu validieren, dass das Kostenmodell für reale Szenarien gute Voraussagen für die tatsächlichen Kosten liefert. Dies war essentieller Bestandteil der Arbeiten in [CKSV08]. Dazu wurden die Schätzungen des Kostenmodells mit den realen Werten verglichen, die mit Hilfe dynamischer Metadaten gemessen werden. An dieser Stelle wird beispielhaft eines dieser Experimente vorgestellt.

Der Parameter g für die Granularität wirkt sich besonders auf die Effizienz des Aggregationsoperators aus. Eine geringere Granularität vermindert dabei die Zahl der Zeitpunkte, an denen sich Aggregate ändern können, und somit den Aufwand zu deren Berechnung und Weiterverarbeitung.

Für das Experiment wurden Echtdaten von Messungen mittels Induktionsschleifen aus dem *Freeway Service Patrol Project* verwendet, die in einer Granularität von 1/60 Sekunden vorlagen. Für die Stunden zwischen vier und sieben der Daten eines Tages wurde die Granularität durch Runden der Zeitstempel auf 10 Sekunden reduziert. Da in einem solchen Intervall zumeist mehrere Fahrzeuge eine Induktionsschleife passieren ($g > l$), sind anhand des Kostenmodells eine Reduzierung des Speicherverbrauchs der Aggregation sowie der Ausgaberate zu erwarten.

Abbildung 13.11 zeigt, dass wie erwartet sowohl der Speicherverbrauch als auch die Ausgaberate im Zeitraum mit verringerter Granularität deutlich absinken. Der nach Kostenmodell erwartete Speicherverbrauch wurde dabei mit Hilfe zur Laufzeit ermittelter Metadaten einmal pro Stunde (Applikationszeit) neu berechnet. Die Abbildung zeigt, dass der gemessene Speicherverbrauch gut dieser Voraussage entspricht.

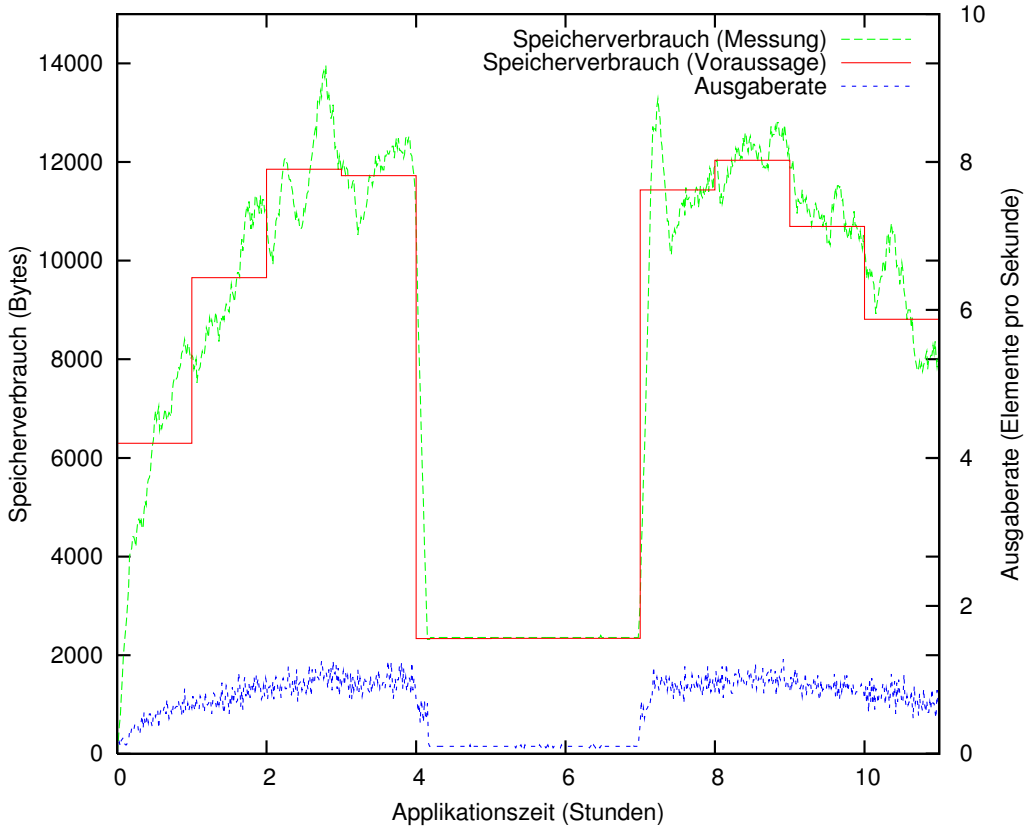


Abbildung 13.11.: Metadaten zur Validierung des Speicherverbrauchs eines Verbundes

14. Verbundoperationen

Thema dieses Kapitels sind die im Kernteil III dieser Arbeit diskutierten Verbundoperationen über physischen Datenströmen. Dabei werden insbesondere die Parametrisierung und Optimierung der vorgestellten Verfahren untersucht. Für einen experimentellen Vergleich des hier benutzten Intervallansatzes mit dem Positiv/Negativ-Ansatz sei an dieser Stelle auf [Krä07] verwiesen.

Dieses Kapitel ist wie folgt aufgebaut: In 14.1 werden zunächst verschiedene SweepAreas verglichen. In 14.2 wird dann aufgezeigt, dass die Verbundoperationen zur Laufzeit das mittels des Kostenmodells vorhergesagte Verhalten zeigen. In 14.3 wird demonstriert, dass der hybride Algorithmus den Verfahren, auf denen er basiert, tatsächlich überlegen ist. Danach werden in 14.4 die Vorteile des mehrdimensionalen Verbundes gegenüber binären Verbundbäumen aufgezeigt. Abschließend liefert 14.5 Experimente zum TPMJ.

14.1. SweepAreas

Die Performanz der in 9 vorgestellten Verbundalgorithmen hängt wesentlich von der Wahl geeigneter SweepAreas ab. In den folgenden Experimenten werden daher wichtige Anwendungsszenarien bezüglich der Wahl der SweepAreas ausgewertet. Dabei wird zunächst insbesondere das Anfrageverhalten untersucht, bevor in einem weiteren Experiment der Schwerpunkt auf das Löschverhalten gelegt wird.

14.1.1. Vergleich zwischen Nested-Loops-Join und Hash-Join

In diesem Experiment wird der Performanzgewinn untersucht, den der Einsatz von hash–basierten SweepAreas gegenüber SweepAreas liefert, bei denen bei Anfragen alle enthaltenen Elemente auf Erfülltsein des Verbundprädikates untersucht werden.

Dazu wird der Gleichverbund zweier Ströme mit jeweils 100.000 Elementen bestehend aus ganzen Zahlen berechnet. Die Länge der Gültigkeitsintervalle wird jeweils gleichverteilt zwischen einer Millisekunde und einer Sekunde Applikationszeit gewählt. Die Verbünde werden jeweils so schnell wie möglich ausgeführt, um anhand der Laufzeit den Aufwand vergleichen zu können. Dabei wird jedoch sichergestellt, dass die Quellen weitgehend synchron arbeiten, um Effekte durch zu großen temporalen Versatz zu vermeiden.

Alle Verbünde werden auf diese Weise je 100 mal ausgeführt. Im n-ten Durchlauf werden dabei in beiden Strömen Zufallszahlen zwischen 1 und n verwendet. Im ersten ergibt sich somit eine Selektivität von 100%, die dann bis zum letzten linear auf 1% absinkt.

Die hash–basierten SweepAreas sind zweistufig ausgelegt, wobei die Anfragen hash–basiert erfolgen. Das temporale Löschen erfolgt normalerweise über einen Heap, der

die Elemente nach Endzeitstempel ordnet. Als alternatives Verfahren wird zusätzlich FIFO-Löschen eingesetzt, bei dem Elemente in der Reihenfolge entfernt werden, in der sie eingefügt wurden. Da über einen Heap bei Anfragen auch problemlos iteriert werden kann, wird als nicht hash–basierte SweepArea eine integriert implementierte eingesetzt und somit wie in 9.3.5 diskutiert auf das zweistufige Design verzichtet.

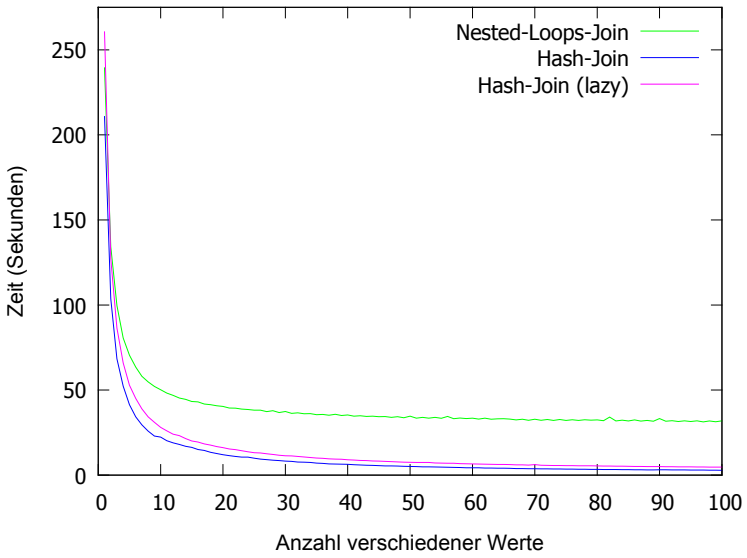


Abbildung 14.1.: Vergleich zwischen Nested-Loops-Join und Hash-Join

Abbildung 14.1 zeigt die sich ergebenden Laufzeiten in Abhängigkeit von der Zahl verschiedener Werte. Die Laufzeiten beinhalteten auch den Aufwand für die Quellen, der jedoch in einer separaten Messung als vergleichsweise gering identifiziert wurde. Da bei hoher Selektivität mehr Ergebnisse erzeugt werden müssen, sank der Aufwand für alle Verfahren wie zu erwarten mit der Selektivität.

Für hohe Selektivitäten müssen alle SweepAreas bei jeder Anfrage jeweils einen großen Teil der enthaltenen Elemente liefern, was unabhängig vom Verfahren hohen Aufwand verursacht. Erst bei sinkender Selektivität zeigt sich der Vorteil der hash–basierten SweepArea, die jeweils nur einen Teil der enthaltenen Elemente für den Verbund berücksichtigen muss. Der Performanzgewinn ist allerdings geringer als der Faktor Selektivität, da die Elemente für das Löschen nach Endzeitstempeln hier zweistufig verwaltet werden.

Obwohl das FIFO-Löschen weniger Löschaufwand verursacht, ist es insgesamt weniger effizient, da bei den Anfragen zusätzliche Elemente aufgefunden und danach auf Grund fehlender zeitlicher Überlappung wieder verworfen werden. Angesichts des zusätzlich noch erhöhten Speicherverbrauchs ist FIFO-Löschen in diesem Szenario also klar unterlegen.

14.1.2. Löschverfahren bei zweistufigen SweepAreas

In einem weiteren Szenario wird daher untersucht, ob das FIFO-Löschen besser abschneidet, falls die Variabilität der Gültigkeitsdauern geringer ist. Dazu wird das vorherige Experiment dahingehend modifiziert, dass die maximale Gültigkeitsdauer bei einer Sekunde Applikationszeit verbleibt, die gleichverteilte zufällige Abweichung nach unten aber schrittweise zwischen 0 und 1000 begrenzt wird. Der Extremfall der Begrenzung auf 1000 entspricht also der Verteilung im vorherigen Experiment, der andere Extremfall entspricht einem gleitenden Fenster, also dem Fall, für den das FIFO-Löschen ohnehin optimal ist (siehe auch 10.2.1). Zusätzlich wird die Selektivität bei 1% fixiert und die Anzahl der Elemente verfünffacht.

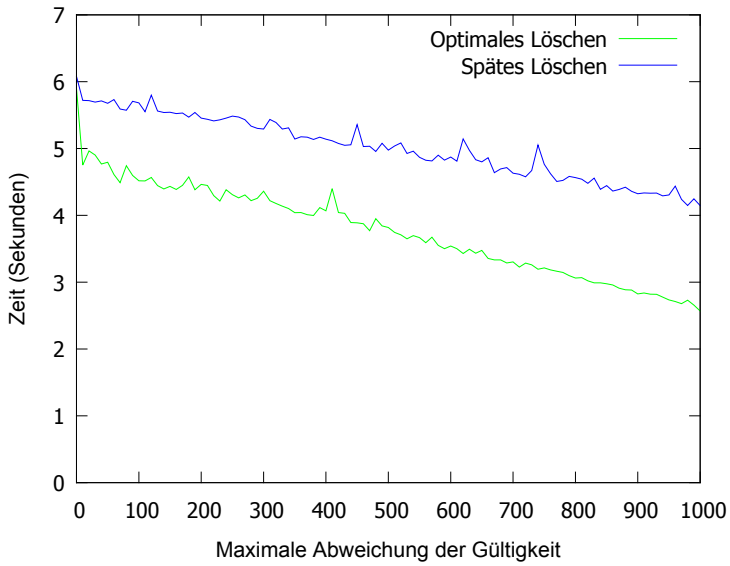


Abbildung 14.2.: Vergleich von Löschverfahren bei zweistufigen SweepAreas

Abbildung 14.2 zeigt, dass schon bei kleinen Abweichungen von der fixen Fensterlänge das FIFO-Löschen die Laufzeit erhöht, der geringere Löschaufwand also nicht den höheren Anfrageaufwand kompensiert. Da in diesen Fällen beim FIFO-Löschen zudem ein höherer Speicherverbrauch entsteht, ist das optimale Löschen zu präferieren. Das FIFO-Löschen sollte lediglich zum Einsatz kommen, wenn auf Grund von Metadaten garantiert werden kann, dass alle Elemente des zugehörigen Eingabestroms die gleiche Gültigkeitsdauer aufweisen.

14.2. Kostenmodell

Dieses Experiment, das in vereinfachter Form auch für [CKSV08] durchgeführt wurde, dient der Validierung der Speicherverbrauchsabschätzung für den Verbundoperator in Abhängigkeit vom Parameter l, der sich direkt aus der Fenstergröße von vor dem Verbundoperator platzierten gleitenden Zeitfenstern ergibt.

Dazu werden gleichzeitig vier Operatorbäume installiert, die sich lediglich in der Behandlung der Zeitfenster der beiden Eingabeströme unterscheiden. In dreien sind die Zeitfenster jeweils konstant bei 20, 40 beziehungsweise 60 Sekunden. Im vierten sind zunächst ebenfalls Fenster von 60 Sekunden aktiv, die dann jedoch nach 100 Sekunden auf 20 Sekunden reduziert werden. Bei 200 Sekunden wird die Fenstergröße dann auf 40 Sekunden erhöht.

Für dieses Experiment werden Sensordatenströme aus dem Produkt *I-Plant* verwendet, über denen ein Anti-Band-Verbund berechnet wird. Dieser dient dazu, die Geschwindigkeiten zweier simulierter Industrie-Walzen zu vergleichen und Alarm zu schlagen, wenn diese zu weit voneinander abweichen. Größere Zeitfenster erlauben dabei die Überwachung eines größeren zeitlichen Kontextes auf Kosten erhöhten Speicherbedarfs und erhöhter CPU-Last. Systemzeit und Applikationszeit laufen bei diesem Experiment weitgehend synchron.

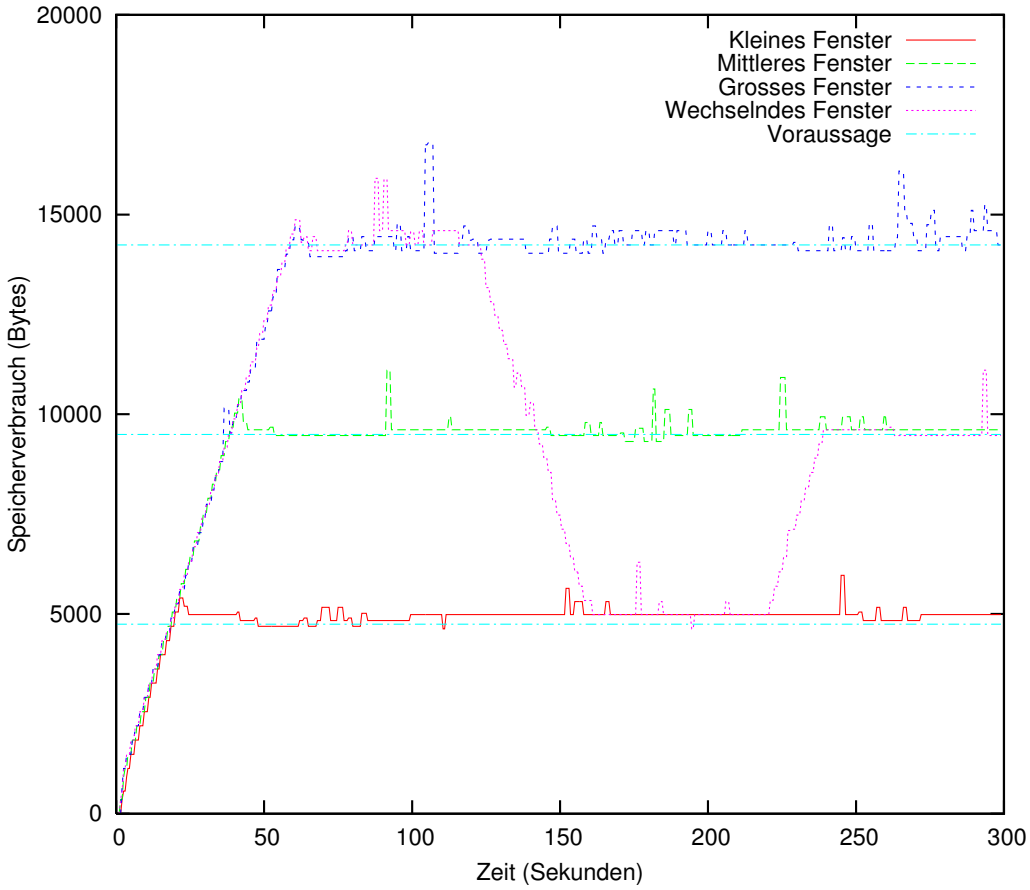


Abbildung 14.3.: Metadaten zur Validierung des Kostenmodells

Abbildung 14.3 zeigt, dass sich nach der Anlaufphase der Speicherverbrauch der drei Verbundoperatoren über den Fenstern mit konstanter Größe wie erwartet nah bei den durch das Kostenmodell vorausgesagten Werten befand. Auch der Verbund mit

sich ändernden Fenstergrößen nahm jeweils diese Speicherverbräuche an. Nach der Verringerung der Fenstergröße bei 100 Sekunden dauerte es die neue Fenstergröße von 20, bis bei 120 der Speicherverbrauch sank. Nach Verdrängung der letzten Elemente mit der alten Fenstergröße 60 nach 100 + 60 = 160 Sekunden hatte der Verbund nun etwa den Speicherverbrauch des Operators, der konstant über den Fenstern mit Größe 20 operierte. Nach der Vergrößerung des Fensters zeigte sich der duale Effekt. Nach Ablauf der kleineren Fenstergröße von 20 bei 200 + 20 = 220 Sekunden begann der Anstieg des Speicherverbrauchs, der dann nach Ablauf der größeren Fenstergröße bei 200 + 40 = 240 Sekunden endete.

14.3. Hybrider Algorithmus

Dieses Experiment untersucht, ob der in 9.3.4 vorgestellte hybride Algorithmus zur Berechnung des Verbunds über temporalen Datenströmen wie erwartet den in 9.3.2 und 9.3.3 vorgestellten Algorithmen, auf denen er beruht, überlegen ist.

Dazu wird ein Natürlicher Verbund über zwei Strömen von ganzen Zahlen berechnet. Dabei senden beide Eingabeströme jeweils im Abstand von einer Millisekunde Elemente mit einer Gültigkeit von einer Sekunde. Einer der Ströme wird dabei um 1,5 Sekunden verzögert, da wie in 9.3.4 erläutert die Vorteile des hybriden Algorithmus vornehmlich in diesem Fall zu Tage treten, während sich die Algorithmen anderenfalls im Wesentlichen ähnlich verhalten.

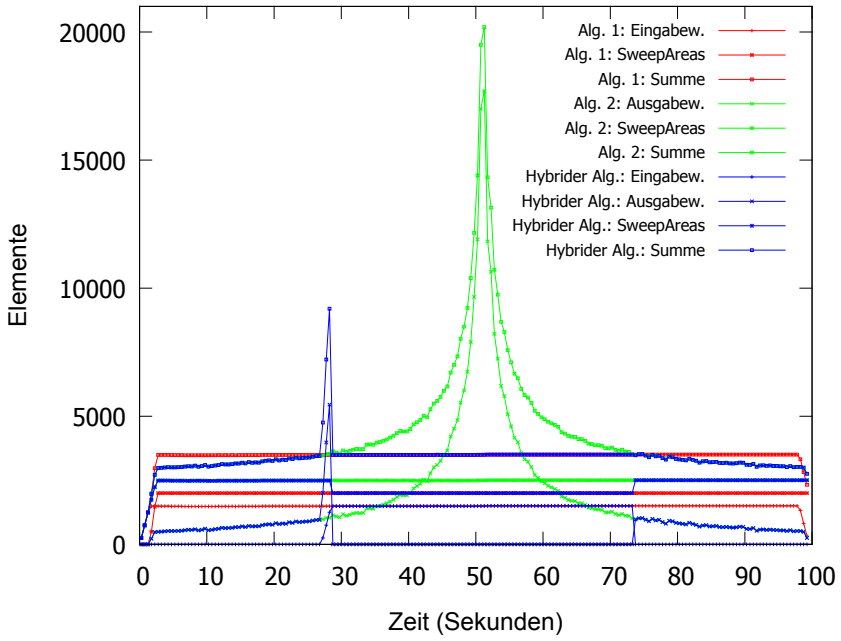


Abbildung 14.4.: Vergleich des Speicherverbrauchs von Verbundalgorithmen für Datenströme

Während des Experiments wird die Selektivität des Verbundes variiert, indem der Bereich, aus dem die verwendeten Zufallszahlen gleichverteilt gezogen werden, verändert wird. Beginnend mit einem Bereich von 1000 Zahlen wird die Obergrenze linear abgesenkt, bis der Bereich nur noch 20 Zahlen umfasst, und dann linear wieder auf den Ursprungswert erhöht. Zu Beginn findet ein Element also im 1000 Elemente fassenden Zeitfenster der Gegenseite im Mittel einen Verbundpartner, zur Hälfte der Laufzeit hingegen 50. Dann liegt also ein expansiver Verbund vor.

Abbildung 14.4 zeigt den Verlauf des Speicherverbrauchs der drei Verbundalgorithmen, jeweils aufgeschlüsselt nach SweepAreas, Ein- und Ausgabewarteschlangen (soweit vorhanden) und Gesamtsumme. Da Ein- und Ausgabeelemente gleich groß sind, ist jeweils nur die Zahl der Elemente angegeben.

Durch die konstanten Raten und Gültigkeitsintervalllängen blieb der Speicherverbrauch der SweepAreas wie erwartet weitgehend gleich. Der erste Algorithmus synchronisiert die Ströme vor den SweepAreas und muss daher auf beiden Seiten eine Fensterlänge von je 1000 Elementen speichern. Dazu kommt die Eingabewarteschlange, die 1500 Elemente des früher eintreffenden Stromes puffern muss. Der zweite Algorithmus hingegen puffert den früher eintreffenden Strom komplett in der zugehörigen SweepArea, die dadurch eine Größe von 2500 Elementen erreicht. Die andere SweepArea hingegen bleibt leer, da Elemente aus dem anderen Strom beim Eintreffen sofort komplett verarbeitet werden. Betrachtet man allein diese konstanten Anteile, so muss der zweite Algorithmus 1000 Elemente weniger speichern. Er benötigt jedoch eine Ausgabewarteschlange, deren Größe von der Zahl der Verbundergebnisse abhängt. Die Messung bestätigte diese Erwartungen.

Sobald die Selektivität hinreichend angestiegen war, benötigte der zweite Algorithmus mehr Speicher als der erste. Der hybride Algorithmus schaltete an dieser Stelle auf die Benutzung einer Eingabewarteschlange um. Dabei entstand kurzzeitig ein erhöhter Speicherverbrauch, während beide Warteschlagen aktiv waren. Sobald das Umschalten abgeschlossen war, verhielt sich der hybride Algorithmus nun aber wie der erste Algorithmus. Insgesamt zeigte er, abgesehen von dieser kurzen Umschaltphase, jedoch durchgängig das Speicherverhalten des jeweils besseren Vergleichsalgorithmus. Der bei geringen Selektivitäten gute zweite Algorithmus hingegen benötigte in der Spitze ein Vielfaches an Speicher.

14.4. Mehrdimensionaler Verbund

Ein weiteres Experiment dient der Evaluation des in 10.3 vorgestellten mehrdimensionalen Verbundes. Dazu wird wie in den vorstehenden Experimenten ein Gleichverbund über Strömen von Zufallszahlen berechnet, in diesem Fall aber über drei davon. In allen Strömen ist der Abstand zwischen den Elementen mit fünf Millisekunden und die Gültigkeit der Elemente mit fünf Sekunden fest gewählt. Die Zufallszahlen aller Eingabeströme werden gleichverteilt gewählt; zu Beginn des Experimentes im ersten Strom zwischen 1 und 1000 und in den beiden anderen zwischen 1 und 100. Jeweils nach 20.000 Elementen in jedem Strom wird derjenige Strom mit dem erweiterten Wertebereich gewechselt.

Als Vergleichsverfahren zum ternären Mehrdimensionalen Verbund werden die drei möglichen Bäume aus binären Verbünden ausgeführt. Jeder dieser Bäume hat jedoch den

Nachteil, dass er zeitweise die beiden Ströme mit je 100 verschiedenen Werten im unteren Verbund verarbeitet und somit in dieser Phase sehr viele Zwischenergebnisse produziert. Beim mehrdimensionalen Verbund hingegen wird die Anfragereihenfolge der SweepAreas so angepasst, dass von den beiden anderen Strömen jeweils zunächst der Strom mit dem erweiterten Wertebereich angefragt wird. Auf die in 10.3.5 vorgeschlagene Optimierung für in binäre Teilprädikate zerlegbare Verbundprädikate wurde im Experiment verzichtet, um die Eigenschaften des mehrdimensionalen Verbundes unabhängig vom Vorhandensein des Spezialfalles zu untersuchen.

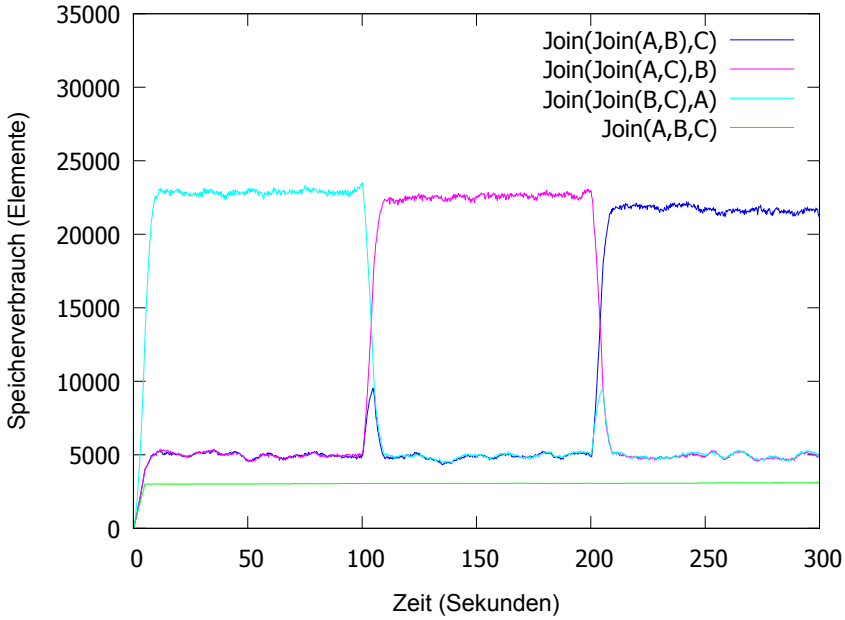


Abbildung 14.5.: Speicherverbrauch des mehrdimensionalen Verbundes im Vergleich zu Bäumen aus binären Verbünden

Abbildung 14.5 zeigt den Verlauf des Speicherverbrauchs der vier Varianten. Der mehrdimensionale Verbund speicherte dabei nur die benötigten Fenster über den Eingabeströmen und hatte somit durchgängig den geringsten Speicherverbrauch. Die binären Verbundbäume hingegen mussten jeweils zusätzlich im oberen Verbund die Ergebnisse des unteren speichern und hatten somit zusätzlichen Speicherbedarf. Dieser stieg wie erwartet jeweils in der Phase stark an, in der die Selektivität des unteren Verbundes hoch war.

Abbildung 14.6 zeigt den Verlauf der Anfragekosten. Dabei wurden die Kosten der Hashfunktion und des Verbundprädikates berücksichtigt und gleichgewichtet angesetzt. Analog zum Speicherverbrauch erzeugten die binären Verbundbäume jeweils dann besonders hohe Kosten, wenn sie den ungünstigen Verbund unten im Baum berechnen mussten. In den beiden anderen Phasen verursachten sie jeweils leicht geringere Kosten als der mehrdimensionale Verbund. Dieser zeigte wie erwartet durchgängig niedrige Kosten. Lediglich beim Übergang zwischen den Phasen stiegen auch seine Kosten an, da für die Dauer von jeweils fünf Sekunden in allen Eingabeströmen die Zahlen von 1 bis

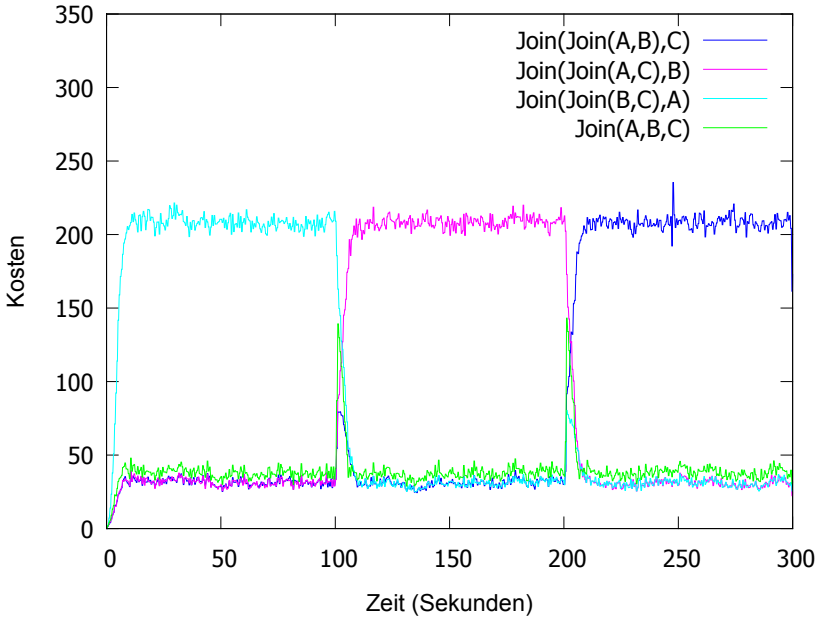


Abbildung 14.6.: Kosten des mehrdimensionalen Verbundes im Vergleich zu Bäumen aus binären Verbünden

100 häufig vertreten waren.

Insgesamt bestätigt sich die Erwartung, dass der mehrdimensionale Verbund den niedrigsten Speicherverbrauch hat, dafür aber leicht höhere Kosten verursacht als die optimale Kombination aus binären Verbünden. Durch seine hohe Flexibilität bei der Anfragereihenfolge der SweepAreas kann er jedoch ohne Techniken wie dynamische Planmigration schnell an geänderte Bedingungen wie schwankende Selektivitäten angepasst werden und verursacht dabei deutlich weniger Kosten als ungünstige Bäume aus binären Verbünden.

14.5. TPMJ

In den abschließenden Experimenten wird noch der Mehrdimensionale Temporale Progressive-Merge-Join als das sortierbasierte Verfahren zur Berechnung des temporalen Verbundes über Datenströmen untersucht.

14.5.1. Merge-Strategien

In diesem Experiment wird ein temporaler Intervall-Verbund über vier Eingabeströmen berechnet. Das temporale Verbundprädikat verlangt also neben dem Schnitt der Gültigkeitsintervalle auch einen Schnitt aller beteiligten Wertintervalle. Alle Elemente haben eine Gültigkeit von zehn Sekunden, wobei in jedem Eingabestrom die Startzeitstempel pro Element um eine Millisekunde erhöht werden und die Elemente auch in diesem Abstand gesendet werden. Die Wertintervalle haben alle die gleiche Länge von

75, wobei für jeden Eingabestrom eine andere Permutation der Zahlen von 1 bis 100.000 als Sequenz der unteren Intervallgrenzen gewählt wird.

Wie im vorstehenden Experiment wird ein mehrdimensionaler Verbund mit einer Kombination aus binären verglichen. Neben dem quaternären Verbund dient in diesem Experiment ein linkstiefer Baum aus binären Verbünden als Vergleichsverfahren. Da alle Eingabeströme vergleichbare Eigenschaften haben und die Selektivität der binären Verbünde niedrig ist, wird auf andere Kombinationen verzichtet. Allen verwendeten Instanzen des TPMJ ist insgesamt 1 MB Speicher zur Generierung der initialen Runs zugeteilt.

Sowohl der quaternäre Verbund als auch der linkstiefe Baum werden in zwei Varianten ausgeführt, wobei die eine jeweils die *Greedy*-Strategie und die andere die *MaxFanIn*-Strategie zur Steuerung der Mergeschritte verwendet.

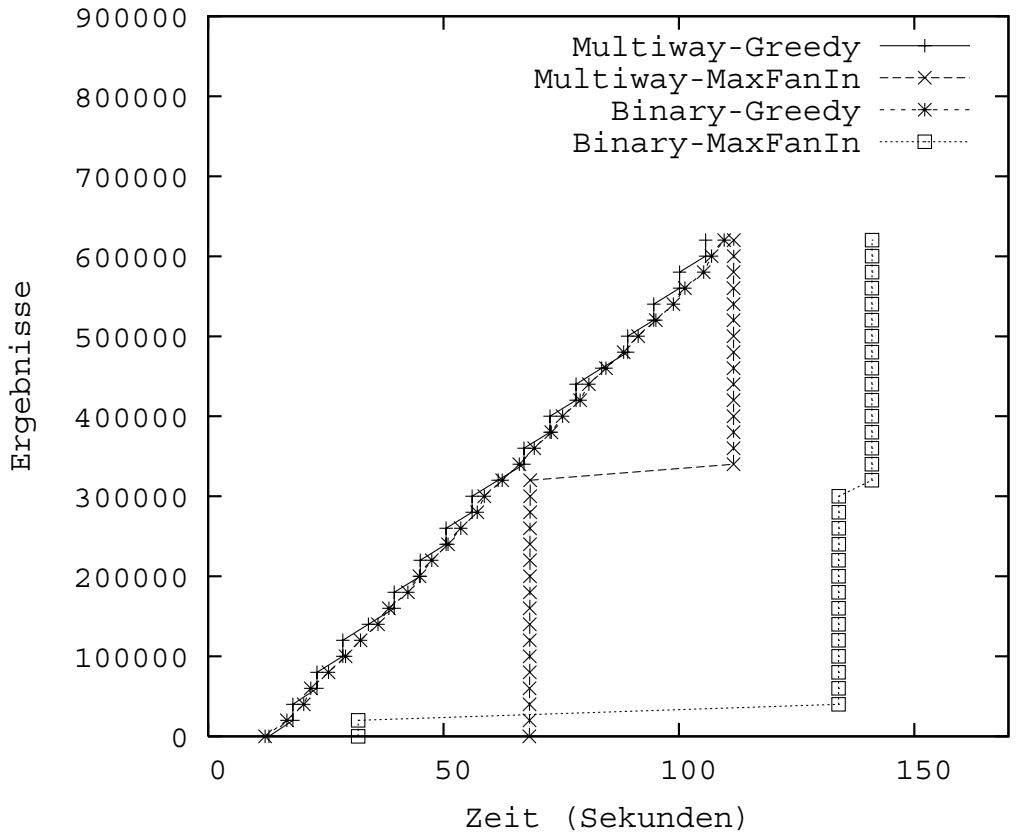


Abbildung 14.7.: Ergebnisproduktion des TPMJ in Abhängigkeit von der Merge-Strategie

Abbildung 14.7 zeigt die Zahl der produzierten Ergebnisse in Abhängigkeit von der Laufzeit.

Alle Varianten können erst Ergebnisse produzieren, wenn nach zehn Sekunden die ersten Elemente ungültig werden. Die beiden Varianten, welche die Greedy-Strategie einsetzen, produzieren danach kontinuierlich Ergebnisse, wobei die Zahl der Ergebnisse nahe am Optimum liegt. Nach Ablauf der Gültigkeit der letzten Eingabeelemente

produzieren sie somit zeitnah die letzten Ergebnisse. Der mehrdimensionale Verbund produziert seine Ergebnisse durchgängig etwas früher als der binäre Verbundbaum.

Bei der MaxFanIn-Strategie dauert es sehr lange, bis Ergebnisse produziert werden, da sehr viele Runs generiert werden können. Im Falle des binären Verbundbaumes führt dies sogar dazu, dass die Mehrzahl der Ergebnisse erst produziert wird, nachdem die Eingabeströme keine Daten mehr senden. Der mehrdimensionale Verbund hingegen kann zumindest die letzten Ergebnisse ähnlich schnell berechnen wie Varianten mit Greedy-Strategie.

	Binäre Verbünde	Mehrdimensionaler Verbund
Greedy-Strategie	492.240	107.214
MaxFanIn-Strategie	227.872	42.004

Tabelle 14.1.: Externspeicherseitenzugriffe in Abhängigkeit von Verbundstruktur und Merge-Strategie

Tabelle 14.1 zeigt die Zahl der Externspeicherseitenzugriffe in Abhängigkeit von der Struktur der Verbundbaumes und der verwendeten Merge-Strategie. Die MaxFanIn-Strategie benötigte wie erwartet jeweils deutlich weniger Zugriffe als die Greedy-Strategie. Allerdings erhöhte im Falle der binären Verbundbäume der zur Materialisierung von Zwischenergebnissen benötigte Zusatzaufwand die Zahl der Externspeicherseitenzugriffe so sehr, dass der mehrdimensionale Verbund sogar bei Einsatz der Greedy-Strategie noch weniger Zugriffe benötigte als die binären Verbünde mit MaxFanIn-Strategie.

Insgesamt zeigt sich hier deutlich, dass der Einsatz des mehrdimensionalen Verbundes teure Externspeicherzugriffe einspart. Die Wahl der Strategie hingegen sollte in Abhängigkeit von den verfügbaren Ressourcen erfolgen, da die frühere Ergebnisproduktion bei Einsatz der Greedy-Strategie mit deutlich erhöhtem Aufwand einhergeht.

14.5.2. Adaptivität

Im nächsten Experiment wird daher untersucht, wie beim mehrdimensionalen Verbund mit Strategiewechseln auf sich ändernde Bedingungen reagiert werden kann. Zu diesem Zweck wird das vorherige Experiment dahingehend modifiziert, dass nun nach jeweils 20 Sekunden der Abstand zwischen den Elementen zwischen einer und zehn Millisekunden gewechselt wird. In den Phasen mit hohen Datenraten wird die MaxFanIn-Strategie verwendet, während in den Phasen mit niedrigen Datenraten die Greedy-Strategie aktiviert wird.

Abbildung 14.8 zeigt den Verlauf der Eingabe- und Ausgaberatenraten sowie den Speicherverbrauch. Immer wenn dieser die Rungröße von 1 MB erreichte, wurde ausgelagert und der Speicherverbrauch sank deutlich ab. Ergebnisse wurden jeweils beim Strategiewechsel hin zur Greedy-Strategie erzeugt. Während der Ergebnisberechnung stieg der Speicherverbrauch des Verbundes dabei über die 1 MB an, die zur Rungenerierung benötigt wurden, da dann die SweepAreas des Merge-Schrittes zusätzlichen Speicher belegten.

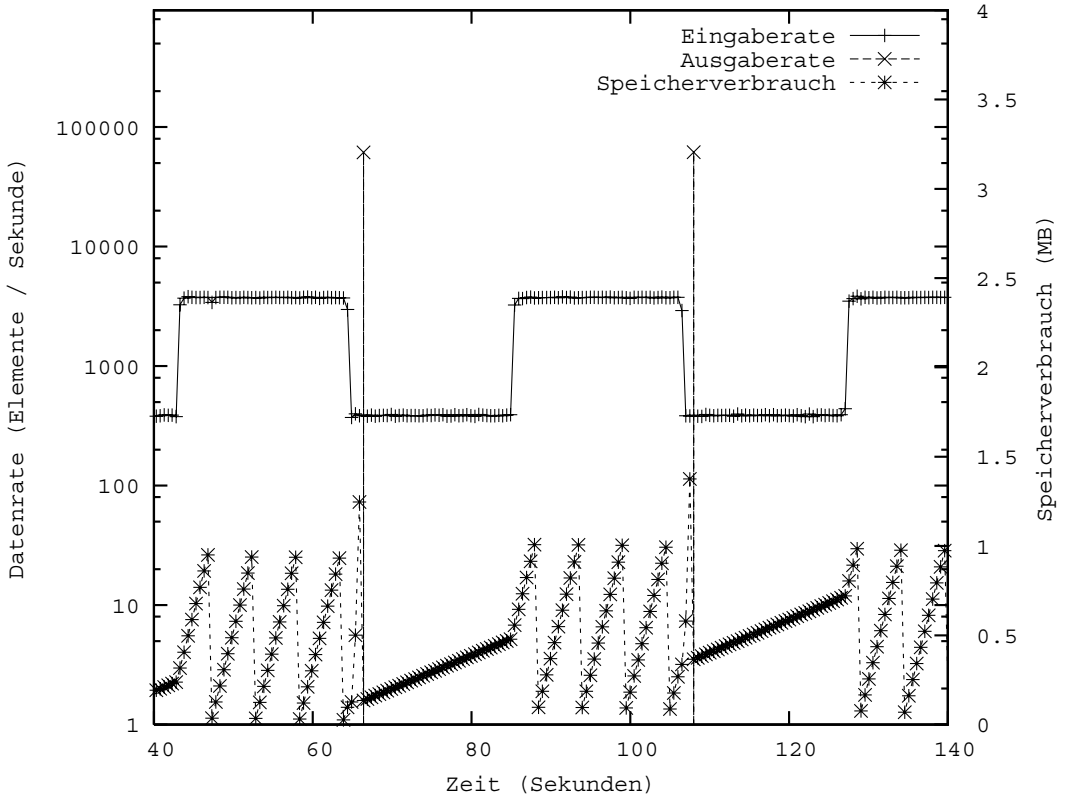


Abbildung 14.8.: Adaptiver Speicherverbrauch beim TPMJ

Teil V.

Verwandte Arbeiten

15. Einleitung

Dieser Teil der Arbeit behandelt verwandte Arbeiten. Nachdem in der Einleitung verwandte Arbeiten aus dem Forschungsumfeld dieser Arbeit diskutiert werden, werden in Kapitel 16 verwandte Arbeiten zum in Teil II behandelten Thema Metadaten diskutiert. Danach widmet sich Kapitel 17 Arbeiten bezüglich Verbundoperationen, die das Thema von Teil III sind.

Wie bereits in 1.5 erwähnt wurden einige grundlegende Ergebnisse zu den zentralen Themen dieser Arbeit bereits vorab auf Konferenzen veröffentlicht. In [CKS07] wurde bereits das Rahmenwerk zur Verwaltung von dynamischen Metadaten bei der Datenstromverarbeitung vorgestellt und in [CHKS05] wurde der TPMJ als der sortierbasierte Verbundalgorithmus für physische Datenströme eingeführt. Beide Themen werden jedoch in dieser Arbeit detaillierter dargestellt.

Die Forschung zu dieser Arbeit erfolgte zu großen Teilen in der Arbeitsgruppe Datenbanksysteme am Fachbereich Mathematik und Informatik der Philipps-Universität Marburg. Dementsprechend wurde sie natürlich von anderen Forschungsarbeiten dieser Gruppe beeinflusst, die in den nächsten Abschnitten diskutiert werden.

15.1. XXL und PIPES

Die Validierung der entwickelten Konzepte und die Experimente zu dieser Arbeit erfolgten wie bereits in 12.1.2 erläutert unter Verwendung und Erweiterung des Forschungsprototypen PIPES, der in [KS04] vorgestellt wird. Dieser wiederum basiert auf der Forschungsbibliothek XXL, die in [BDS00, BBD+01] präsentiert wird. Diese Arbeit wurde in [CHK+03] fortgeschrieben.

15.2. Progressive-Merge-Join

Ebenfalls unter wesentlicher Beteiligung dieser Arbeitsgruppe entwickelt wurde der Progressive-Merge-Join, der in [DSTW02, DSTW03] vorgestellt wurde. In [Cam02] wurden diese Arbeiten erweitert. Der Algorithmus wird in diesem Teil in 17.2.2 behandelt und in dieser Arbeit in 11.1 ausführlicher vorgestellt.

15.3. Anfrageverarbeitung aktiver Datenströme

Wesentliche Teile dieser Arbeit entstanden im Rahmen des von der Deutschen Forschungsgemeinschaft (DFG) geförderten Forschungsprojektes *Anfrageverarbeitung aktiver Datenströme*, das ebenfalls von der Arbeitsgruppe Datenbanksysteme am Fachbereich

Mathematik und Informatik der Philipps-Universität Marburg bearbeitet wurde. Aus diesem Forschungsprojekt ist eine Vielzahl an Veröffentlichungen hervorgegangen.

Die bereits in 1.5 aufgelisteten Arbeiten [CHKS03, CHKS04, CHK+06, CKSV06, KYC+06, CHK+07, HKRS07, CKSV08] stehen in engem Zusammenhang mit dieser Arbeit. Wichtige Grundlagen zu dieser Arbeit werden zudem in [KS05, YKPS07, HKRS08, KS09] besprochen. Weitere Ergebnisse des Forschungsprojektes wurden in [BHS05, HKSZ05, HS05, HS06b, HS06a, HS06c, HS06d, HS07, KSPL06] veröffentlicht. Aus dem Projekt gingen zudem bereits mehrere Dissertationen [Krä07, Hei07, Rie08] hervor.

16. Metadaten

Obwohl Metadaten eine wichtige Bedeutung für DSMS haben und daher sicher nicht nur in kommerziellen Systemen, sondern auf Grund ihrer Bedeutung zur Analyse von Verfahren gerade auch in Forschungsprototypen Verwendung finden, findet sich verhältnismäßig wenig Literatur zu diesem Thema. Lediglich [CKS07], das die Grundlagen der in dieser Arbeit vorgestellten Techniken beschreibt, konzentriert sich auf das Thema Metadatenverwaltung für DSMS.

Im näheren Umfeld des Themas findet sich dennoch eine Vielzahl an relevanten Arbeiten, die in diesem Kapitel vorgestellt werden. Zunächst widmet sich 16.1 speziellen Typen von Metadaten. In 16.2 werden dann verfügbare Informationen über die Metadatenverwaltung in Forschungsprototypen präsentiert. Abschließend werden in 16.3 noch einige weitere Arbeiten im Bereich Metadaten vorgestellt.

16.1. Spezielle Metadaten

In diesem Abschnitt werden mit Punctuations und k-Constraints zwei spezielle Arten von Metadaten vorgestellt, die bei der Optimierung von Datenstromanfragen Verwendung finden können.

16.1.1. Punctuations

Punctuations[TM02, TMSF03] werden bereits in [BBD+02] erwähnt und sind eines der ersten Konzepte, die zur Lösung des Problems, potentiell unbegrenzte Datenströme mit endlichen Ressourcen verarbeiten zu wollen, vorgestellt wurden. *Punctuation*s sind dabei zusätzliche Informationen, die zwischen den Elementen eines Datenstromes in den Datenfluss eingefügt werden. In der ursprünglichen Definition handelt es sich dabei um Prädikate über dem Wertebereich der Datenstromelemente, und die Bedeutung einer Punctuation ist, dass dieses Prädikat für kein später im Strom folgendes Element mehr erfüllt sein wird. Da Punctuations somit Aussagen über nachfolgende Daten im Strom treffen, werden sie vielfach auch den Metadaten zugerechnet. In der Praxis beschränkt man sich dabei zumeist auf spezielle Prädikate, wie beispielsweise die charakteristischen Prädikate für Mengen relationaler Tupel, die sich als Kreuzprodukt von Teilmengen der Wertebereiche der Attribute ergeben. Die Idee der Punctuations ist es nun, mit Hilfe der Prädikate den Status statusbehafteter Operatoren bereinigen zu können. Beispielsweise kann eine Gruppierung mit Aggregation die Aggregate für eine Gruppe ausgeben und dann alle Informationen über die Gruppe verwerfen, sobald eine Punctuation die Information liefert, dass kein weiteres Element mehr eintreffen kann, das zu der Gruppe gehören würde.[LMT+05]

Punctuations stellen somit eine alternative Strategie zur Statusbereinigung auf Basis einer temporalen Semantik dar. Vielfach wurde der Begriff der Punctuation auch auf Ströme mit Zeitinformation wie Gültigkeitsintervallen ausgeweitet.[SW04a, JMSS05] In dem in dieser Arbeit verwendeten Intervallansatz (siehe 2.3.3) beinhaltet jedes Stromelement dabei implizit eine nachfolgende Punctuation, die besagt, dass keine Stromelemente mit kleinerem Startzeitstempel mehr eintreffen werden. Analoges gilt für die Stromelemente im Positiv/Negativ-Ansatz. *Heartbeats*[SW04a], also Träger von Information über den zeitlichen Fortschritt (siehe 10.2.2), können dabei als Spezialfall solcher Punctuations aufgefasst werden.[SW04a, JMSS05] In [LMT+05] werden zeitbasierte Punctuations bei der Berechnung fensterbasierter Aggregate über zeitlich nicht komplett sortiert eintreffenden Datenströmen eingesetzt.

Wertbasierte Punctuations haben jedoch das Problem, dass sie im Allgemeinen nicht garantieren, dass bei der Verarbeitung potentiell unbegrenzter Datenströme der Status immer bereinigt werden kann. Für Verbundoperationen kann dies nicht einmal für den wichtigen Spezialfall des Gleichverbundes allgemein gewährleistet werden.[LCT+06] Für komplexere Verbundprädikate stellt zudem die Berechnung von Punctuations, die der Verbund an Folgeoperatoren weitergeben könnte, ein extrem schwieriges Problem dar. Aus diesen Gründen werden wertbasierte Punctuations in dieser Arbeit nicht näher untersucht. Punctuations wurden dennoch, wie in 17.3.5 beschrieben, auch vielfach[TM02, TMSF03, DRH03, DMRH04, Din08] zur Optimierung von Verbünden eingesetzt.

16.1.2. k-Constraints

Während Punctuations zur Laufzeit der Anfrage in den Datenströmen als eine Art dynamische Metadaten auftreten, sind die sogenannten *k-Constraints*[BSW04] statische Metadaten. Wie klassische Constraints in DBMS den Raum der möglichen Instanzen von Relationen einschränken, schränken k-Constraints das Spektrum der möglichen Datenströme ein. Neben aus der Datenbankwelt bekannten Constraints wie Fremdschlüsseleigenschaften werden in [BSW04] aber auch stromübergreifende Einschränkungen vorgeschlagen, wie beispielsweise, dass bei einem Fremdschlüsselverbund zwischen zwei Elementen mit Schlüssel s über beide am Verbund beteiligten Ströme maximal k Elemente eintreffen dürfen. Viele der vorgeschlagenen Einschränkungen sind nicht mit dem in dieser Arbeit verwendeten Datenstrommodell vereinbar, das insbesondere die Unabhängigkeit des Ergebnisses von stromübergreifenden Eintreffreihenfolgen erfordert.

16.2. Systeme

Im Rahmen der Datenstromforschung wurde eine Reihe von Forschungsprototypen entwickelt, von denen die wesentlichen in diesem Abschnitt bezüglich ihrer Metadatenkonzepte diskutiert werden. Leider sind diesbezüglich allerdings in den meisten Fällen nur wenige Informationen in der Literatur verfügbar.

16.2.1. Aurora und Borealis

Aurora speichert seine Metadaten in einem zentralen Katalog, und zwar in einer *Berkeley DB*[OBS99] Datenbank.[ACC+03] Dieser wird insbesondere auch benutzt, um für die QoS benötigte statistische Informationen wie Selektivitäten und Verarbeitungskosten zu verwalten. Die Knoten im Anfrageplan sind dafür verantwortlich, diese statistischen Informationen im Katalog zu verwalten.[Bal06] Das auf Aurora aufbauende *Borealis* verwaltet seine Metadaten ebenfalls zentral in diesem „Local Catalog", der von allen Komponenten zugegriffen werden kann.[AAB+05a] Borealis stellt dabei eine verteilte Erweiterung von Medusa dar, die es nötig machte, die Metadatenkataloge der beteiligte Medusa-Systeme global zugreifbar und änderbar zu machen.[BBC+04]

Beide Systeme benutzen diese Metadaten insbesondere zur Steuerung der QoS.[ZSC+03] Um auf Überlast im System reagieren zu können, vertrauen sie auf die Technik des *Load Shedding*, bei dem Eingabeelemente direkt verworfen werden, um die Systemlast zu reduzieren.[TCZ+03]

16.2.2. Calder

Calder[VLP06], eine Erweiterung des *dQUOB*[PS03] Projektes, ist ein verteiltes System zur Verarbeitung von Datenströmen. An Calder können dabei auch Metadatenanfragen gestellt werden.[VLP06] Es verwaltet statische Metadaten wie Zugriffsrechte als Schlüssel/Wert-Paare, unterstützt auf die gleiche Art aber auch benutzerdefinierte Metadaten.[LVP06a] Ein Dienst überwacht zusätzlich die Änderung wichtiger Metadaten zur Laufzeit.[LVP06a] Liu et. al. weisen in ihrem Ausblick in [LVP06b] darauf hin, dass sie in Echtzeit gewonnene Metadaten bezüglich der Performanz bei der Verarbeitung von Datenströmen für ein Muss halten.

16.2.3. Cape und D-Cape

Cape[RDS+04] und *D-Cape*[SLJR05] benutzen verschiedene Arten von Metadaten. Klassische statische Metadaten wie Schemainformationen und Fremdschlüsseleigenschaften werden bei der Anfrageübersetzung und zur Optimierung eingesetzt.[RDS+04] Eine „Statistics Gatherer" genannte, zentrale Komponente überwacht die in Ausführung befindlichen Pläne und aktualisiert in regelmäßigen Abständen Statistiken.[SLJR05] Diese werden auch dazu genutzt, sogenannte *SPJ-Anfragen*, die nur aus Selektionen, Projektionen und Joins bestehen, zur Laufzeit zu reoptimieren.[Din08] Die Systeme benutzen auch wertbasierte Punctuations.

16.2.4. Cayuga

Cayuga[BDG+07], ein System auf Basis nichtdeterministischer Automaten, versucht seinen Hauptspeicherverbrauch mit Hilfe von *Garbage Collection* zu begrenzen.[DGP+07] Speicher wird gegebenenfalls direkt vom Betriebssystem angefordert, wobei Auslagerung auf den Externspeicher in Kauf genommen wird.

16.2.5. Gigascope

Auch *Gigascope*[CJSS03a] gehört zu den Systemen, die Metadaten in einem zentralen Katalog verwalten.[CJSS03b] Punctuations kommen in Form des Spezialfalls Heartbeats zum Einsatz.[JMSS05]

16.2.6. Niagara und NiagaraCQ

Niagara[NDM+01] und das Teilsystem *NiagaraCQ*[CDTW00] dienen zur Beantwortung kontinuierlicher Anfragen über XML-Dokumenten aus dem Internet. Dabei kommen wiederum Punctuations zum Einsatz.[Tuc05] Beispielsweise werden wertbasierte Punctuations benutzt, um die Größe der Statusstrukturen beim symmetrischen Hash-Join endlich zu halten.[TMSF03]

16.2.7. Nile und Nile-PDT

Das auf *Nile*[HMA+04] aufbauende Nile-PDT[AAB+05b] beobachtet, welche Ströme vermehrt zu Ergebnissen beitragen. Quellen, die zu vielen Ergebnissen beitragen, werden dann häufiger abgefragt als andere, was zur Begrenzung der Systemlast dient.[AAB+05b]

16.2.8. STREAM

Alle Komponenten von *STREAM*[MWA+03] erben von einer gemeinsamen Oberklasse, welche für jede Instanz eine *Control Table* beinhaltet, in der Schlüssel/Wert-Paare gespeichert werden. Diese werden unter anderem auch dazu genutzt, statische und dynamische Metadaten wie Statistiken über die bereits verarbeiteten Elemente zu speichern.[MWA+03] Dieses Konzept weist somit Ähnlichkeiten zu der in PIPES als grundlegende Struktur verwendeten `CompositeMetaData` (siehe 4.2) auf. Die gewonnenen Statistiken werden beispielsweise zur dynamischen Bestimmung von Reihenfolgen zur Auswertung von Filtern verwendet, wobei ein zentraler *Profiler* zum Einsatz kommt.[BMM+04]

STREAM verwendet Heartbeats und sieht diese im in 16.1.1 erläuterten Sinne auch als Metadaten an.[ABW06]

16.2.9. StreamGlobe

StreamGlobe[SKK04, KSKR05], ein Peer-to-Peer System zur Verarbeitung von XML-Datenströmen, verwaltet Metadaten mit Hilfe einer Weiterentwicklung des verteilten Systems *ObjectGlobe*[BKK+00, BKK+01, KKKK02] (siehe 16.3.1).[SK04] Auf den Peers werden die Metadaten, darunter statische wie Schemainformationen zentral verwaltet.[SKK04] Dynamische Metadaten wie Stromraten werden auf jedem Peer von einem Service kontinuierlich gesammelt und aktualisiert.[KSKR05] Zusätzlich verwaltet das System Metadaten bezüglich der Fähigkeiten der Peers und der für die Kommunikation verfügbaren Bandbreiten.[SK04]

16.2.10. Stream Mill

Laut [ZTZ06] stellt das *Stream Mill* System[BTW+06, TMZ08] unter anderem Funktionalität zum Überwachen der Performanz bereit.

16.2.11. TelegraphCQ

Das auf *Telegraph*[SMFH01] aufbauende *TelegraphCQ*[KCC+03] verwaltet statische Metadaten in einem Katalog, um verschiedene Arten von Datenquellen, darunter lokale wie Dateien und entfernte wie Sensoren, zu verwalten.[CCD+03] TelegraphCQ setzt mit *Eddies*[AH00], *Flux*[SHCF03] und *Juggle*[RRH99] verschiedene Techniken ein, um die Verarbeitungsreihenfolge, die Eingabeelemente durchlaufen, zur Laufzeit dynamisch zu ändern.[CCD+03] Dafür ist es hilfreich zu messen, welche Komponenten wie viele Ressourcen verbrauchen und wie viele Elemente sie verarbeiten konnten, weshalb solche Statistiken bei Flux am Anfang einer Phase für eine vorgegebene Zeit erhoben werden.[SHCF03]

16.3. Weitere Arbeiten

In diesem Abschnitt werden abschließend noch einige Arbeiten aus dem Umfeld Metadaten und Datenströme vorgestellt.

16.3.1. MDV

ObjectGlobe[BKK+00, BKK+01] ist ein verteiltes System zur Bearbeitung von Anfragen über Diensten im Internet. Da solche Dienste fortlaufenden Änderungen unterworfen sein können, benötigt ObjectGlobe eine umfangreiche dynamische Metadatenverwaltung, wozu mit *MDV* ein verteiltes Metadaten-Managementsystem als Teilkomponente entwickelt wurde.[KKKK02] MDV verwendet eine aus drei Schichten bestehende Architektur. Ein Backbone, der im Wesentlichen aus einem verteilten DBMS besteht, speichert die Metadaten. Lokale Verzeichnisse registrieren sich beim Backbone anhand einer Regelsprache und replizieren darüber Teile der Datenbank, die für ihre Aufgaben von Interesse sind. Diese lokalen Verzeichnisse beantworten dann die Anfragen von Client-Systemen. Der Fokus von MDV liegt somit zwar auf der dynamischen Verwaltung und regelbasierten Verteilung von Metadaten, allerdings für ein verteiltes System und Metadaten, die sich erheblich langsamer ändern als die von Operatoren eines DSMS. Ein DBMS ist jedoch zur Verwaltung dynamischer Metadaten eines DSMS aus denselben Gründen ungeeignet, aus denen es dies zur Verarbeitung von dessen Daten ist.

16.3.2. Multimediadatenströme

In [PCR01] wird die kontinuierliche Einbettung von Metadaten in Multimediadatenströme diskutiert, wobei sich jedoch sowohl die Art der Daten als auch der Metadaten fundamental von den in dieser Arbeit untersuchten unterscheiden.

16.3.3. Datenqualität

[KDH+07] beschäftigt sich mit einer speziellen Form von Metadaten, nämlich Qualitätsinformationen über Datenströme, wie beispielsweise die Messgenauigkeit von Sensoren, die einen Wert geliefert haben. Die Qualitätsmetadaten werden zur Reduzierung des Verarbeitungsaufwandes über mehrere Stromelemente aggregiert und als Qualitätsmetadaten zusammen mit den Ergebnissen präsentiert oder in Datenbanken archiviert. Die Propagierung der Qualitätsmetadaten entlang des Datenflusses durch den Operatorgraphen eines DSMS anhand einer Qualitätsmetadatenalgebra wird dabei unter zukünftige Arbeiten geführt. Dies wäre ein interessanter Anwendungsfall für das in dieser Arbeit vorgestellte Rahmenwerk zur Metadatenverarbeitung.

Semantik von DSMS

Sowohl von Seiten der Forschung als auch von Seiten kommerzieller Anbieter hat sich auf dem Gebiet der Datenstromverarbeitung eine starke Vielfalt an Syntax- und Semantikvorschlägen für Anfragesprachen und deren Auswertung entwickelt. Um Nutzern in dieser Vielfalt eine Orientierung zu geben, wurde mit *SECRET*[BDD+10] ein Modell zur Beschreibung des Verhaltens von DSMS entwickelt.

17. Verbundoperationen

Verbundoperationen stellen auf Grund ihrer besonderen Bedeutung sowohl für die Fähigkeiten als auch für den Ressourcenverbrauch von Datenverarbeitungssystemen ein beliebtes Forschungsgebiet dar. Sie wurden daher nicht nur generell für klassische relationale Datenbanken, sondern auch für spezialisiertere Anwendungsfälle wie temporale, räumliche oder verteilte Datenbanksysteme näher untersucht. Einige der dabei entwickelten Konzepte haben die in dieser Arbeit vorgestellten Algorithmen zur Berechnung des Verbundes über Datenströmen im Intervallansatz stark beeinflusst. Dieses Kapitel zu verwandten Arbeiten im Bereich Verbundoperationen beschäftigt sich daher zunächst mit solchen aus dem Bereich der Datenbanken.

Erst danach werden verwandte Arbeiten aus dem Bereich der Datenstromverarbeitung diskutiert. Dabei ist zu beobachten, dass in diesem Gebiet das Problem der Verbundoperation vielfach isoliert definiert und dann behandelt wird. Die dabei verwendeten Definitionen sind vielfach nicht oder nur eingeschränkt dafür geeignet, die vorgestellten Verfahren im Rahmen einer Datenstromalgebra zu verwenden, und eignen sich somit insbesondere weniger zum Einsatz in einem DSMS mit deklarativer Anfragesprache.

Diejenigen Algorithmen, die implizit oder explizit für die Verwendung in einer Datenstromalgebra geeignet sind, gehen zumeist von einer dem Positiv/Negativ-Ansatz folgenden oder ähnlichen physischen Repräsentation der Datenströme aus und sind daher für den in dieser Arbeit verwendeten Intervallansatz nicht oder nur modifiziert einsetzbar. Dennoch spielen Arbeiten zu Verbundalgorithmen für Datenströme für diese Arbeit natürlich eine wichtige Rolle und werden daher hier ebenfalls besprochen.

Dieses Kapitel ist wie folgt strukturiert: In 17.1 wird zunächst der Spezialfall von Verbundoperationen für temporale Datenbanken diskutiert, bevor in 17.2 solche für allgemeine Datenbanken behandelt werden. In 17.3 werden abschließend Verbundoperationen für Datenströme besprochen.

17.1. Temporale Datenbanken

Die Forschung zur Datenstromverarbeitung nahm erst mit der breiten Verfügbarkeit schneller Datennetze und von Systemen mit erheblichen Hauptspeicherkapazitäten an Fahrt auf. Das grundsätzliche Problem, potentiell unbegrenzte Datenströme mit endlichen Speicherkapazitäten, insbesondere endlichem Hauptspeicher, verarbeiten zu müssen, wird dabei wie in dieser Arbeit zumeist über eine Begrenzung der Gültigkeit von Stromelementen mittels einer temporalen Semantik gelöst. Mit ähnlichen Fragestellungen hat sich die Datenbankforschung schon lange zuvor im Rahmen der Arbeiten zu temporalen Datenbanken beschäftigt. Daher behandelt dieser Abschnitt verwandte Arbeiten aus diesem Bereich. Auch [TCG+93] gibt einen guten Überblick über dieses Forschungsgebiet.

17.1.1. Temporale Semantik

In [CT85] findet eine zu $\mathbb{N}$ isomorphe diskrete Zeitdomäne Verwendung. [DS93] beschäftigt sich mit der Repräsentation von Zeit in temporalen Datenbanken, darunter auch der von Gültigkeitsintervallen.

Die Prinzipien der Schnappschuss-Äquivalenz und Schnappschuss-Reduzierung wurden bereits in [Gad86] von Gadia für temporale Datenbanken vorgestellt. In [JSS94] wurden sie dazu benutzt, um verschiedene Konzepte temporaler Datenbanken in Einklang zu bringen.

Bereits in [JM80] wurde eine Klassifikation temporaler Datenbanken vorgenommen, die neben der Möglichkeit historischer Anfragen auch die Möglichkeit von Rollbacks berücksichtigt. Während Rollbacks in DBMS zur Standardfunktionalität zählen, sehen DSMS zumeist nicht vor, einmal verarbeitete Elemente zu widerrufen.[GÖ03b]

In [Sno87] wird bereits hervorgehoben, dass die gesonderte Betrachtung der Gültigkeit als spezielles Attribut von zentraler Bedeutung ist. Snodgrass weist darauf hin, dass es für eine Semantik auf Basis von Schnappschüssen unerlässlich ist, den Zugriff der Benutzer auf diese Attribute zu limitieren und stattdessen ihre korrekte Verwendung durch das System zu garantieren:[Sno87]

> „Since the additional temporal attributes are an artifact of embedding a temporal relation in a snapshot one, users must be constrained in how they use these attributes. The query language must be designed so that temporal attributes are used correctly.“

Snodgrass unterscheidet bereits zwischen Zeitpunkten, zu denen Elemente gültig beziehungsweise ungültig werden, und denjenigen, zu denen Einfügen und Löschen in der Datenbank effektiv werden. Diese Unterscheidung spiegelt sich in DSMS durch geeignetes Setzen der Gültigkeitsintervalle anhand von Applikations- oder Systemzeit wider.

17.1.2. Temporale Verbünde

Bereits in [CC87] werden der Theta-Verbund, der Gleichverbund und der Natürliche Verbund für temporale Datenbanken auf Basis einer temporalen Semantik mit geschlossenen Gültigkeitsintervallen über einer zu $\mathbb{N}$ isomorphen Zeitdomäne definiert. Zusätzlich wird dort auch ein Verbund spezifiziert, der auf einer Seite ein Wert-Attribut eines Tupels als Zeitstempel interpretiert und dieses mit allen Tupeln der Gegenseite joint, die zu diesem Zeitpunkt gültig sind. Auf Grund fehlender Garantien bezüglich des zeitlichen Fortschrittes auf dem Zeitstempel-Wert lässt sich dieser Verbund allerdings nicht unmodifiziert im Rahmen der in 2.4 vorgestellten Stromalgebra verwenden.

In [SG89] werden ebenfalls ein Gleichverbund und auch ein äußerer Gleichverbund für temporale Datenbanken auf Basis des Schnittes von Gültigkeitsintervallen vorgestellt, allerdings ohne entsprechende Einbettung in eine temporale Algebra. Der Gleichverbund trägt den Namen „Event Join“, während heute Datenstromverarbeitung vielfach unter dem Namen *Complex Event Processing* vermarktet wird.

Ein temporaler Verbund auf Basis des Schnittes von Zeitintervallen und eines davon unabhängigen wertbasierten Verbundprädikates wird für temporale Datenbanken zudem auch in [GS91] definiert. In [LOT94] wird für temporale Datenbankverbünde mit Gültigkeitsintervallen vorgeschlagen, diese als Punkte in der Ebene aufzufassen und dann partitionierende räumliche Verbundalgorithmen anzuwenden.

In [LM93] wird ein breites Spektrum an verschiedenen temporalen Verbünden für temporale Datenbanken mit halboffenen Gültigkeitsintervallen vorgestellt. Darunter befindet sich auch eine mehrdimensionale Variante, die analog zum in 10.3 vorgestellten mehrdimensionalen Verbund für Datenströme ist, der sich aus der Schnappschuss-Reduzierung ergibt. Andere dort definierte Varianten widersprechen dieser Bedingung, könnten aber auch über Datenströmen berechnet werden, darunter die Variante, die das Überlappen der beteiligten Gültigkeitsintervalle nicht paarweise fordert, sondern lediglich für nebeneinanderliegende Intervalle in einer Kette durch alle beteiligten temporalen Relationen. Weitere Varianten, wie der $before - join$, bei dem ein Gültigkeitsintervall vor dem anderen liegen muss, lassen sich über potentiell unendlichen Datenströmen nicht mit endlichem Speicherverbrauch berechnen.

[GJSS05] gibt einen sehr guten Überblick über Verbundoperatoren für temporale Datenbanken. Gao et. al. geben dazu zunächst eine Klassifizierung in Algorithmen auf Basis von Nested-Loops, Indexierung, Partitionierung und Sort-Merge an, und unterteilen die letzten beiden dann jeweils nochmals anhand des Partitionierungs- beziehungsweise Sortierkriteriums. Sie ordnen bestehende Verfahren in diese Klassen ein und ergänzen Algorithmen für bisher leere Klassen. Die Autoren schließen Algorithmen, die einen Index oder eine andere Vorsortierung der Daten erfordern, die für die Datenstromverarbeitung mit ihrer zeitlichen Sortierung von besonderem Interesse wären, jedoch von ihren weiteren Betrachtungen aus. Einen wesentlichen Teil der Arbeit bildet ein umfassender experimenteller Vergleich der Algorithmen, der sich allerdings auf den Gleichverbund beschränkt, der mit allen genannten Techniken berechnet werden kann. Als Sieger gehen dabei ausschließlich partitionierende Algorithmen hervor. Die Autoren weisen jedoch darauf hin, dass sich die Verhältnisse unter Nebenbedingungen wie Sortierbedingungen für Ein- und Ausgaben leicht umkehren können. Zudem eignen sich partitionierende Algorithmen schlechter für Verbundprädikate, die im Gegensatz zum wichtigen Spezialfall Gleichverbund keine Äquivalenzklassenverbünde (siehe 9.3.5) darstellen.

17.1.3. Stromverarbeitung

Während bei der Datenstromverarbeitung ein zeitlich geordnetes Eintreffen der Daten im Allgemeinen zu den Anforderungen gehört, ist das bei temporalen Datenbanken nicht der Fall, falls nicht automatisch Systemzeitstempel verwendet werden. Leung und Muntz weisen in [LM93] jedoch darauf hin, dass dieses gleichwohl in vielen realen Anwendungsszenarien gegeben ist. Daher können die Daten vielfach in temporaler Ordnung zugegriffen werden, gegebenenfalls auch über genau dafür erstellte Indizes. Unter Ausnutzung dieser Tatsache präsentieren sie nun Algorithmen, die derart zugreifbare Relationen nur einmalig geordnet lesen. Diese Technik bezeichnen sie bereits als Stromverarbeitung.

Dabei wird bereits die Tatsache thematisiert, dass eine derartige Verarbeitung mit

endlichen Hauptspeicherressourcen auskommen muss, um den Status des Algorithmus während der Verarbeitung zu halten. Anders als beim Temporalen PMJ für Datenströme können diese Algorithmen sich jedoch zu Nutze machen, dass die Daten ohnehin in einer Datenbank, also auf Externspeicher, vorliegen. Die in [LM93] präsentierten temporalen Verbundalgorithmen nutzen zudem aus, dass sie als Datenbankalgorithmen bedarfsgesteuert arbeiten können. Dies ermöglicht beim sortierten Lesen mehrerer Relationen eine global sortierte Verarbeitung. Daher ähneln sie dem in 9.3.2 vorgestellten ersten Algorithmus, der für die datengetriebene Verarbeitung Warteschlangen vorschaltet, um die global sortierte Verarbeitung zu ermöglichen. Leung und Muntz betonen unter Verweis auf [SC75, SAC+79] bereits, dass die Ergebnisse dabei wieder nach Startzeitstempeln sortiert produziert werden und sich daher gut für die Weiterverarbeitung in Folgeoperatoren eignen, die diese Eigenschaft voraussetzen.

Für die Kostenanalyse verwenden Leung und Muntz in [LM93] mit der durchschnittlichen Einfügerate und der durchschnittlichen Gültigkeitsintervalllänge bereits zwei Charakteristika, die in ähnlicher Weise in den in 10.1.2 beschriebenen Stromcharakteristika Verwendung finden. Sie weisen auch darauf hin, dass entsprechende Metadaten von Bedeutung sind und durch Anwendung von Operatoren auf eine Relation einer temporalen Datenbank Ausgaben mit signifikant anderen Werten für diese Charakteristika entstehen können.

In [SSJ94] wird ein Natürlicher Verbund für temporale Datenbanken auf Basis temporaler Partitionierung vorgestellt. Das dabei gewählte Partitionierungsmodell benutzt analog zur in 11.3.2 beschriebenen Strategie zur Duplikateliminierung eine vollständige Unterteilung der Zeitachse in aneinander angrenzende Intervalle.

17.2. Verbundoperationen für Datenbanken

Neben Verbundoperationen für temporale Datenbanken haben natürlich auch solche für andere Datenbanken Einfluss auf die Entwicklung von Verbundoperationen über Datenströmen.

17.2.1. Räumliche Verbünde

Bei der Entwicklung von Datenstromverbünden gilt es insbesondere, die ausgezeichnete Zeitachse zu beachten. Neben den temporalen gibt es aber auch noch andere Achsen, die in der Datenbankforschung besondere Beachtung finden, insbesondere die räumlichen. Dieser Unterabschnitt stellt daher Verfahren zur Berechnung des räumlichen Verbundes vor. Diese sind zudem noch dadurch von besonderem Interesse, weil auch in diesem Kontext Sweep-Verfahren und entsprechende Datenstrukturen mit Analogien zu den in dieser Arbeit verwendeten SweepAreas entwickelt wurden.

Spatial Hash-Join

In [LR96] wird ein Rahmenwerk für *Spatial Hash Joins* vorgestellt, mit dessen Hilfe räumliche Verbünde auf Basis eines Hash-Verbundes berechnet werden können. Das

grundsätzliche Problem, dass sich partitionierende Verbundverfahren normalerweise lediglich für Äquivalenzklassenverbünde (siehe 9.3.5) eignen, wird dadurch gelöst, dass Elemente dupliziert und mehreren Partitionen zugeordnet werden. Da dieses Duplizieren im Falle eines Hauptspeicherverfahrens lediglich ein Duplizieren von Referenzen auf die konkreten Objekte erfordert, ist eine Verwendung dieses Ansatzes auch für räumliche Verbünde über Datenströmen interessant.

Partition Based Spatial-Merge Join

Analog dazu wird auch beim *Partition Based Spatial-Merge Join*[PD96] der Datenraum in disjunkte Rechtecke unterteilt, denen dann die Elemente zugeordnet werden, deren minimal umgebende Rechtecke diese schneiden. Das Verbundergebnis wird dann partitionsweise mit Hilfe eines SweepLine-Verfahrens im Hauptspeicher berechnet. Die Partitionsgrößen werden dabei so gewählt, dass dieses möglich ist.

Scalable Sweeping-Based Spatial Join

Beim *Scalable Sweeping-Based Spatial Join*[APR+98] (SSSJ) hingegen wird die Ebene nicht in Rechtecke, sondern lediglich in Streifen unterteilt, wobei jedes Rechteck bezüglich der Aufteilung mehrere Streifen überdecken kann und in maximal zwei weiteren endet. Nach dem Sortieren beider Eingabemengen nach der Anfangskoordinate bezüglich der anderen Achse werden die Rechtecke nun traversiert und dabei den Streifen zugeordnet. Zusätzlich werden bereits alle Ergebnisse bestimmt, bei denen mindestens eine Streifenüberdeckung beteiligt ist. Alle anderen Ergebnisse werden dann rekursiv für die einzelnen Streifen berechnet. Zum Berechnen des Verbundes verwenden die Autoren bereits austauschbare SweepAreas und einen allgemeinen Algorithmus mit dem Ausführungsmuster Einfügen, Löschen und Anfragen. Wird der in dieser Arbeit vorgestellte TPMJ ebenfalls zur Berechnung eines räumlichen Verbundes genutzt, so lassen sich die Optimierungen aus [APR+98] übertragen, wobei wie beim PMJ üblich der Merge-Schritt mit der Verbundberechnung verwoben wird. Allerdings weisen die Autoren bereits darauf hin, dass in den meisten realistischen Szenarien die Aufteilung in Streifen gar nicht benötigt wird, da schon die sortierte Verarbeitung bezüglich einer Achse den Speicherverbrauch für die SweepAreas hinreichend eingrenzt, um sie im Hauptspeicher halten zu können.

17.2.2. Verbundalgorithmen mit frühen Ergebnissen

Während Verbundalgorithmen für DBMS zunächst vornehmlich auf eine möglichst geringe Gesamtlaufzeit optimiert wurden, wurden später Verfahren entwickelt, die bereits möglichst früh Teile des Ergebnisses liefern. Als Motivation dafür diente zum einen die Verfügbarkeit früher Ergebnisse, insbesondere die Möglichkeit diese bereits parallel zur Verarbeitung des Verbundes weiterverarbeiten zu können[WA90] oder daraus früh Schätzungen bezüglich des Gesamtergebnisses ableiten zu können[HH99, DSTW02, JDA+06]. Zum anderen motivierten Szenarien, in denen die Eingabeelemente beispielsweise durch langsame Netzwerke nicht in der Rate zugegriffen werden können, in der man sie

verarbeiten könnte, zur Nutzung der Wartezeit zur früheren Ergebnisproduktion.[UF01] Da diese Verfahren einen wichtigen Schritt auf dem Weg zur Verbundberechnung über potentiell unbegrenzten Datenströmen darstellen, werden im Folgenden einige der so entstandenen Algorithmen vorgestellt.

Pipelining Hash Join

Beim *Pipelining Hash Join*[WA91] (oft auch als *Symmetric Hash Join* bezeichnet) wird im Gegensatz zum klassischen Hash Join für beide Eingaben eine Hashtabelle angelegt. Beim Eintreffen eines Elementes wird zunächst die Hashtabelle der anderen Eingabe nach Ergebnissen durchsucht, bevor das Element in die Hashtabelle zur eigenen Eingabe eingefügt wird. Dieses symmetrische Grundprinzip liegt sehr vielen hash–basierten Verbundalgorithmen für sowohl DBMS als auch DSMS zu Grunde.

Double Pipelined Hash Join

Beim *Double Pipelined Hash Join* wurde das Konzept des Pipelining Hash Join dahingehend erweitert, dass Externspeicher zum Auslagern von Teilen der Hashtabellen genutzt wird, falls diese nicht mehr in den Hauptspeicher passen.[IFF+99]

XJoin und MJoin

Beim *XJoin*[UF01] werden zusätzliche Wartezeiten, die durch Verzögerungen beim Beziehen der Eingabeelemente entstehen, ausgenutzt, um bereits ausgelagerte Partitionen der Hashtabellen wieder einzulesen und durch Join mit im Speicher befindlichen Elementen zusätzliche frühe Ergebnisse zu produzieren. Die restlichen Ergebnisse werden nach dem Eintreffen aller Eingabeelemente durch Joinen der ausgelagerten Partitionen produziert. Beim *MJoin*[VNB03] wurde dieses Konzept auf mehrdimensionale Verbünde übertragen. Zu den beiden Algorithmen siehe auch 9.1.3.

Progressive-Merge-Join

In [DSTW02] wurde mit dem *Progressive-Merge-Join* (PMJ) der erste Verbundalgorithmus mit frühen Ergebnissen vorgestellt, der nicht auf den Spezialfall des Gleichverbundes spezialisiert war. In [DSTW03] wurde diese Arbeit um eine theoretische Analyse erweitert, die insbesondere auch Fälle höherer Mergebäume betrachtet. [Cam02] erweiterte das Konzept auf mehrdimensionale Verbünde und stellte weitere Optimierungen für die erste Mergephase vor. In [GJSS05] wird vorgeschlagen, den PMJ auch zur Produktion früher Ergebnisse für Verbünde in temporalen Datenbanken einzusetzen. Wegen der Bedeutung für diese Arbeit wird der PMJ in 11.1 näher diskutiert.

Sort-Merge-Shrink Join

Der *Sort-Merge-Shrink Join*[JDA+05, JDA+06] (SMS Join) ist eine Variante des PMJ, die zusätzlich im Hauptspeicher einen Hash-Ripple-Join einsetzt. Der Algorithmus ist damit durchgängig daraufhin optimiert, jederzeit das Ergebnis einer im Operatorbaum auf den

Verbund folgenden Aggregation korrekt abschätzen zu können, um dem Benutzer die Möglichkeit zu geben, den Verbund vorzeitig abbrechen zu können. Voraussetzung dafür ist dabei allerdings, dass die Eingaben so eintreffen, dass jedes Präfix eine Stichprobe der jeweiligen Eingaben ist. Da dies in Datenstromszenarien nicht der Fall ist, sind derartige Optimierungen dort nicht anwendbar. Die Idee dahinter spiegelt sich für Datenströme über das sogenannte *Random Drop Load Shedding*[TCZ+03] wieder, bei dem statt der kompletten Eingaben nur eine Stichprobe verarbeitet wird.

Hash-Merge Join

Der *Hash-Merge Join*[MLA04] kombiniert die Ideen des XJoins und des Progressive-Merge-Joins. Im Hauptspeicher findet das Konzept des XJoins Anwendung. Vor dem Auslagern von Hashbuckets auf den Externspeicher werden diese dann allerdings sortiert. Beim Wiedereinlesen zum Zwecke der Produktion weiterer früher Ergebnisse und des vollständigen Ergebnisses wird für die einzelnen Hashbuckets wie in den Merge-Phasen des PMJ vorgegangen. Als partitionierendes Verfahren bleibt der Hash-Merge Join damit bezüglich der verwendbaren Verbundprädikate limitiert.

Rate-based Progressive Join

Der *Rate-based Progressive Join*[TYP+05] (RPJ) stellt eine Weiterentwicklung des Hash-Merge Joins dar. Zum einen wird eine andere Auslagerungsstrategie verfolgt, die darauf basiert, solche Elemente auszulagern, für die die Wahrscheinlichkeit am geringsten ist, dass sie zu Ergebnissen beitragen. Diese Strategie erfordert daher die kontinuierliche Erfassung umfangreicher statistischer Metadaten. Zusätzlich werden beim RPJ im Falle von Verzögerungen in den Eingaben auch bereits Verbünde ausgelagerter Partitionen berechnet, falls dies mehr Ergebnisse verspricht, als ausgelagerte Partitionen mit Hauptspeicherpartitionen zu verbinden.

Early Hash Join

Weil die Produktion früher Ergebnisse beim Verbund endlicher Eingaben häufig nur auf Kosten einer höheren Gesamtlaufzeit zu erreichen ist, wurde mit dem *Early Hash Join*[Law05] das Ziel verfolgt, zur Laufzeit die frühe Ergebnisproduktion zugunsten einer dadurch geringeren Restlaufzeit zurückfahren oder ganz einstellen zu können. Zusätzlich wurden Verbesserungen zur Duplikateliminierung und für Fremdschlüsselverbünde vorgestellt, die darauf beruhen, dass Elemente, die ihren Verbundpartner gefunden haben, nicht weiter berücksichtigt werden müssen.

Partitioned expanding Ripple Join (PR-Join)

Der *Partitioned expanding Ripple Join* (*PR-Join*)[CGN10] stellt eine Weiterentwicklung des Hash-Ripple-Joins dar, bei der höhere Raten früher Ergebnisse erreicht werden sollen. Zudem sollen wie beim Progressive-Merge-Join oder dem Sort-Merge-Shrink Join für zufällig sortierte Eingaben statistische Garantien bezüglich der Repräsentativität der

Ergebnisse gegeben werden. Der *Partitioned expanding Ripple Join* liest dazu zuvor erzeugte Hashpartitionen früh erneut ein, um deren Verbund zu berechnen. Durch die Nutzung von Solid State Disks werden die Kosten dieses Schritts relativ zum Einlesen neuer Daten verringert.

Double/Multi Index NEstedloops Reactive join

Der *Double Index NEstedloops Reactive join* (*DINER*)[BVKD09] baut wiederum auf dem Rate-based Progressive Join auf, bringt aber mit dem Nested-Loops-Join (siehe 9.1.2) eine weitere klassische Verbundtechnik zur Anwendung. Die Eingabeelemente werden nun im Speicher nicht mehr in einer Hashtabelle, sondern in einem sortierten Hauptspeicherindex abgelegt, der in drei Bereiche aufgeteilt ist. Die Auslagerung erfolgt dann aus dem Bereich, der die wenigsten frühen Ergebnisse zu liefern verspricht. Beim Blockieren der Eingaben werden nun ausgelagerte Blöcke mit Hilfe des Nested-Loops-Joins verarbeitet, wobei die Blöcke sortiert ausgelagert werden und innerhalb der Blöcke ein Merge-Join verwendet wird. Der Algorithmus bleibt im Gegensatz zu hash–basierten Verfahren nicht auf Äquivalenzklassenverbünde beschränkt, sondern kann auch einen Band-Verbund berechnen.

Der *Multiple Index NEstedloops Reactive join* (*MINER*)[BVKD10] erweitert das Prinzip von DINER auf mehrdimensionale Verbünde. Zur effizienten Überprüfung von Bedingungen auf verschiedenen Attributen werden Tupel einer Eingabe dabei gegebenenfalls in mehreren Indizes referenziert. Bei der Auslagerung aus dem Bereich, der wenige Ergebnisse zu liefern verspricht, werden Referenzen auf die dort referenzierten Tupel dann auch aus den anderen Bereichen entfernt. Für eintreffende Tupel werden, ähnlich zur in 10.3.5 vorgestellten dynamischen Anfragereihenfolge, die Indizes in der Reihenfolge ansteigender Selektivität angefragt, um Anfragesequenzen möglichst früh abbrechen zu können. Analog werden beim Blockieren Blöcke mit Hilfe des mehrdimensionalen Nested-Loops-Joins verarbeitet, wobei die Eingaben nach und nach als äußere Relation gewählt werden.

Eddies und SteMs

Eddies[AH00] wurden für das System *Telegraph*[SMFH01] entwickelt, das Datenbankanfragen über verteilten Systemen verarbeiten können soll. Um auf Änderungen in der Geschwindigkeit, mit der Eingaben empfangen werden können, und auf die Charakteristiken der eintreffenden Daten möglichst dynamisch reagieren zu können, wird beim Einsatz von Eddies kein fester Anfragegraph installiert. Vielmehr werden die eintreffenden Elemente dynamisch zu den einzelnen Operatoren geleitet, wobei die Pfade anhand statistischer Auswertung des bisherigen Verhaltens immer wieder neu bestimmt werden. Im Falle von Verbundoperationen muss dabei, in Abhängigkeit von den verwendeten Verbundalgorithmen, sichergestellt werden, dass diese dennoch korrekt arbeiten. Die dazu eingesetzten Momente der Symmetrie schränken dabei gegebenenfalls die Wahl der Pfade signifikant ein. In [Des04] wird untersucht, welchen Einfluss die sehr feingranulare Wahl der Operatorreihenfolge bei Eddies auf die Performanz hat.

State Modules[RDH03], kurz *SteMs*, wurden entwickelt, um die Flexibilität von Eddies weiter zu erhöhen. Es handelt sich dabei um Datenstrukturen, die, ähnlich wie die in 9.2 beschriebenen SweepAreas, den Zugriff auf die Elemente einer Eingabe des Verbundes abstrahieren. Innerhalb eines Verbundes steuert ein Eddy nun die Reihenfolge, in der eintreffende Elemente die SteMs anfragen. Auf diese Art kann unter anderem eine mehrdimensionale Variante des Symmetric Hash Join realisiert werden.

Telegraph wurde später zur Verarbeitung kontinuierlicher Anfragen zu *Telegraph-CQ*[CCD+03] (siehe dazu auch 16.2.11) weiterentwickelt, wobei Eddies und SteMs weiterhin zum Einsatz kommen.

In [CC08] wird eine Weiterentwicklung namens *Trained Eddies*(*Teddies*) vorgeschlagen, bei der Batches von Elementen verarbeitet werden und somit die Wahl der Verarbeitungsreihenfolge weniger feingranular abläuft als bei Eddies.

17.2.3. Mehrdimensionale Optimierung

In [LKM10] wird vorgeschlagen, Bäume aus binären Verbünden, die auf dem Symmetric Hash Join basieren, global zur Produktion früher Ergebnisse zu optimieren. Die Motivation dafür bietet die Erkenntnis, dass viele frühe Ergebnisse in einem der Verbünde unten im Baum nicht zwingend zu vielen Ergebnissen an der Wurzel beitragen. Als Lösung werden daher bei vollem Speicher Elemente ausgelagert, bei denen zu erwarten ist, dass sie wenig zum globalen Ergebnis beitragen. Zudem wird die Verteilung des Hauptspeichers unter allen Verbünden global optimiert, statt jedem einen festen Teil zuzuweisen. Wird einem Verbund zeitweise ein Großteil des Hauptspeichers zugewiesen, so kann dieser besonders viele frühe Ergebnisse produzieren. Die in 11.3.3 vorgeschlagene Optimierung für DSMS, die benötigten Ressourcen für den PMJ global zu verteilen, stellt eine Übertragung dieses Prinzips auf die Stromverarbeitung und zugleich eine Verallgemeinerung auf mehrere parallel verarbeitete Anfragen dar.

17.2.4. Skew

Bei sortierbasierten externen Verbünden kann es zu Problemen führen, wenn einzelne Werte an extrem vielen Ergebnissen teilhaben, da dann besonders viel Hauptspeicher für den Merge benötigt wird, was es gegebenenfalls nötig macht, die Statusstrukturen teilweise auf den Externspeicher auszulagern. In [LGS02] wird erläutert, wie man diesen *Skew* genannten Effekt beim Sort-Merge Join handhabt.

17.2.5. Planmigration

Obwohl die Bedeutung der Planmigration auf Grund der erheblich längeren Laufzeiten von Anfragen für DSMS erheblich höher ist als für DBMS, wurden auch für die Migration von Anfrageplänen in DBMS Verfahren vorgeschlagen. In [EFP06] und [EPFL10] wird für Hash Join, Nested-Loops-Join, Merge Join, Index Nested-Loops Join und Symmetric Hash Join erläutert, wie deren Ausführung an bestimmten Punkten unterbrochen werden kann, zu denen sie bestimmte Teilmengen des Gesamtergebnisses produziert haben, um sie dann durch einen anderen Operator zu ersetzen, mit dessen Hilfe dann die noch fehlenden

Ergebnisse berechnet werden. Als Motivation dient analog zu der in DSMS die Tatsache, dass sich die Wahl des Optimierers für eine Implementierung des Verbundoperators anhand zur Laufzeit gemessener Statistiken als suboptimal erweisen kann.

17.3. Verbundoperationen für Datenströme

Eine gute Übersicht zu den Grundlagen der Datenstromverarbeitung liefert [GÖ03b]. Die Theorie zur Datenstromalgebra auf Basis von Schnappschuss-Reduzierbarkeit und Intervallansatz, die den in dieser Arbeit betrachteten Verbundalgorithmen zu Grunde liegt, wird in [KS05] und [Krä07] detailliert beschrieben. Diese Arbeiten liefern jeweils auch eine Übersicht bezüglich alternativer Ansätze wie dem Positiv/Negativ-Ansatz[ABW06, GHM+07]. Eine Übersicht zu frühen Arbeiten zur Verbundverarbeitung über Datenströmen findet sich in [XY07].

Input-Triggered Approach

In [GHM+07] diskutieren die Autoren Varianten zur Optimierung des Positiv/Negativ-Ansatzes zur Berechnung allgemeiner Anfragen über Datenströmen. In einer *Input-Triggered Approach* genannten Variante verwenden sie dabei Stromelemente mit zwei Zeitstempeln, wobei der zweite als Ablaufzeitstempel verwendet wird und dem Endzeitstempel des Intervallansatzes entspricht. Zusätzlich werden jedoch auch negative Elemente verwendet und dann gezeigt, dass sich diese weitgehend wegoptimieren lassen, wobei nur der zeitliche Fortschritt noch propagiert werden muss, was der Heartbeat-Optimierung im Intervallansatz entspricht. Dabei ergibt sich ein Verbundalgorithmus für Gleichverbünde, der dem in dieser Arbeit diskutierten ersten Algorithmus zur Berechnung des Verbundes über Datenströmen in der Variante mit hash–basierten SweepAreas ähnelt. Dabei wird zusätzlich bei jedem Element in den Statusstrukturen gespeichert, wie oft es an einem Verbund teilgenommen hat. Beim Eintreffen negativer Elemente wird der zeitliche Fortschritt nur dann propagiert, wenn das zugehörige positive Element an mindestens einem Verbund teilgenommen hat.

CJOIN

[SFL05] beschäftigt sich mit dem Problem des Verbunds von Sensordatenströmen mit unterschiedlichen Abtastraten und Charakteristiken. Beim Algorithmus *CJOIN* werden das Interpolieren der Ströme mit geringeren Raten oder das Reduzieren der Ströme mit höheren Raten auf die Elemente mit zeitlich passenden Verbundpartnern als Lösungsmöglichkeiten verfolgt. Dabei wird berücksichtigt, dass Interpolation nur für bestimmte Klassen von Sensordaten sinnvoll möglich ist.

Die in dieser Arbeit verfolgte Verbundsemantik trägt dem Problem indirekt dadurch Rechnung, dass Sensorwerte in Strömen mit geringerer Rate längere Gültigkeitsintervalle hätten und die Gültigkeitsintervalle der Verbundergebnisse als Schnitt der Gültigkeitsintervalle der beteiligten Elemente genau wiedergeben, in welchem zeitlichen Kontext das Verbundergebnis Gültigkeit hat. Dem Zuweisen geeigneter Gültigkeitsintervalle an den Quellen kommt dabei eine zentrale Bedeutung zu.

PTJoin

[MWL06] widmet sich dem Problem des Verbundes über Strömen, in denen die Applikationszeitpunkte der Elemente nicht genau bestimmt, sondern lediglich bezüglich einer über Histogramme modellierten Verteilung abgeschätzt werden können. Der vorgeschlagene Algorithmus *PTJoin* erzeugt Ergebnisse nur dann, wenn die beteiligten Elemente mit einer Mindestwahrscheinlichkeit alle innerhalb eines vorgegebenen Zeitfensters gültig waren.

EM-SWJoin

Farag und Hammad beschäftigen sich in [FH07] mit der Fragestellung, wie ein Verbund über einem Zeitfenster über Datenströmen berechnet werden kann, wenn der Hauptspeicher nicht für die Speicherung der Fenster genügt. Wie in dieser Arbeit fordern sie dabei, dass die Eingabedatenströme lediglich lokal sortiert nach Zeitstempeln eintreffen müssen und die Ausgabe des Verbundes wiederum nach Zeitstempeln sortiert erfolgen soll. Die Autoren schlagen dazu den Algorithmus *External-Memory Sliding Window Join*(*EM-SWJoin*) vor, der auf den Konzepten des MJoins basiert und einen hash–basierten mehrdimensionalen Verbund anhand eines über alle Eingaben gemeinsamen Verbundattributes berechnet.

Eintreffende Elemente werden dabei gespeichert und mit dem Hauptspeicheranteil der entsprechenden Partitionen der anderen Eingaben gejoint. Ist der Hauptspeicher voll, so wird die größte Hauptspeicherpartition ausgelagert. Liefern die Eingaben gerade keine Elemente, so werden Verbünde mit Externspeicherpartitionen berechnet. Sowohl in den Hauptspeicher-, als auch in den Externspeicherpartitionen werden Elemente gelöscht, die aus dem Zeitfenster gefallen sind. Ergebnisse werden, wie bei dem in dieser Arbeit vorgestellten Algorithmus 8 zunächst gespeichert und dann nach Zeitstempeln sortiert herausgegeben, wenn der globale zeitliche Fortschritt bei der Verarbeitung dies erlaubt.

Im Gegensatz zum in 11.2 vorgestellten Datengetriebenen PMJ geht der EM-SWJoin von Eingabedatenströmen aus, die nur einen Zeitstempel enthalten, und wendet ein gemeinsames Zeitfenster auf alle Ströme an, wodurch er nur bedingt innerhalb einer Stromalgebra einsetzbar ist. Durch den Einsatz eines partitionierenden Verfahrens unterliegt er zudem den entsprechenden Beschränkungen bezüglich verwendbarer Verbundprädikate.

(Low Latency) Handshake Join

Der in [TM11] vorgestellte *Handshake Join* wurde speziell zur parallelen Ausführung von Datenstromverbünden entwickelt. Insbesondere soll der Algorithmus auch Field-Programmable Gate Arrays (FPGAs) oder Massively Parallel Processor Arrays (MPPAs) zur Joinberechnung effizient nutzen können. Die wesentliche Idee des Verfahrens ist, dass sich zwei Elemente während ihrer Gültigkeit innerhalb von zeit- oder wertbasierten Fenstern an einer Stelle begegnen müssen, um joinen zu können. Dazu werden die Ströme asynchron gegenläufig über die beteiligen Auswertungsknoten geschoben, wobei sichergestellt wird, dass sich Elemente nicht auf dem Kommunikationswege begegnen und somit verpassen könnten, indem gesendete Elemente bis zu einer asynchronen

Empfangsbestätigung noch am Sender verbleiben. Auf den Auswertungsknoten werden eintreffende Elemente gegen die vorhandenen der anderen Eingabe auf Erfülltsein der Joinbedingung geprüft. Erfolgt dieser Schritt ohne Ausnutzung von Indexierung gegen alle Elemente, so prüft der Handshake Join insgesamt jedes mögliche Paar von Elementen und eignet sich somit für beliebige Joinprädikate. Allerdings bringt der Algorithmus mit sich, dass Elemente die Fenster in derselben Reihenfolge verlassen, in der sie diese betreten haben, was im Kontext der in dieser Arbeit verwendeten Semantik nachteilig sein kann.

In [RTG14] wird der Algorithmus zum *Low Latency Handshake Join* weiterentwickelt, um das Problem anzugehen, dass beim Ursprungsverfahren eine längere Verzögerung auftreten kann, bis sich zwei Elemente begegnen und joinen. Daher wird beim Low Latency Handshake Join jedem Element ein Heimatknoten zugeordnet. Elemente, die diesen noch nicht erreicht haben, wandern möglichst schnell dorthin, wo sie dann verbleiben. Zusätzlich werden sie jedoch zum Anfragen weiterhin schnell weitergereicht, um möglichst schnell alle Knoten zu erreichen und somit alle Ergebnisse mit zuvor eingetroffenen Verbundpartnern zu bilden. Zusätzlich können beim Low Latency Handshake Join zeitbasierte Punctuations erzeugt werden, wodurch er sich in Kombination mit einer Ausgabewarteschlange analog zu 9.3.3 zur Berechnung einer nach Zeitstempeln sortierten Ausgabe eignet.

17.3.1. Mehrdimensionale Verbundverarbeitung

In diesem Unterabschnitt werden Arbeiten diskutiert, die sich auch mit der Berechnung des Verbundes über mehr als zwei Eingaben beschäftigen.

CellJoin

Der *CellJoin*[GYB07, GBY09] wurde speziell entwickelt, um Datenstromverbünde besonders effizient mit Hilfe eines *Cell Processor* berechnen zu können. Dieser besteht neben einem Allzweckprozessor aus acht zusätzlichen 128-Bit Coprozessoren, die zwar nur einen eingeschränkten Befehlssatz unterstützen, dafür aber besonders effizient auf den Hauptspeicher zugreifen und mehrere Berechnungen parallel ausführen können.

Um nun neu eintreffende Elemente gegen das Zeitfenster des anderen Stromes auf Erfülltsein der Verbundbedingung zu prüfen, werden die Fenster in Partitionen geteilt und auf die Coprozessoren verteilt. Bei hohen Eingaberaten werden dabei auch mehrere neu eingetroffene Elemente einer Eingabe gemeinsam verarbeitet. Im Falle von Gleichverbünden werden die Fenster zusätzlich nach Hashwerten partitioniert und nur die Partitionen mit möglichen Verbundpartnern an die Coprozessoren übermittelt. Die Partitionen werden im Hauptspeicher spaltenorientiert abgelegt, um den Coprozessoren nur die für das Verbundprädikat nötigen Attribute übermitteln zu müssen. Die Konstruktion der finalen Ergebnisse übernimmt asynchron der Allzweckprozessor. Bei der Berechnung von mehrdimensionalen Verbünden geht der CellJoin für das in der Abfragereihenfolge erste Fenster bei der Lastverteilung analog vor, berechnet dann aber weitere Schritte auf dem jeweils selben Coprozessor.

Der CellJoin geht davon aus, dass Elemente die Zeitfenster in derselben Reihenfolge verlassen, in der sie sie betreten, und überprüft die Zeitfensterbedingung daher nur in der jeweils ältesten Partition eines Fensters (und gegebenenfalls Hashwertes). Haben alle Elemente in einer Partition das Fenster verlassen, so wird die Partition verworfen.

Hybrid NLJ-Hash Join

Mit dem *Hybrid NLJ-Hash Join*[GÖ03a] und diversen Vorstufen stellen Golab und Özsu erste Algorithmen für mehrdimensionale Verbünde über Zeitfenstern vor. Diese dürfen dabei bereits verschiedener Größe sein, aber müssen alle relativ zum jeweils aktuellen Zeitpunkt liegen. Neu eintreffende Elemente werden dabei zunächst gesammelt und dann in Batches zum Anfragen der Fenster verwendet. Die Autoren untersuchen dabei bereits verschiedene Anfragereihenfolgen und Zugriffsmethoden, wobei sich der Einsatz von Hashing wenig überraschend dem kompletten Lesen der Fenster in verschachtelten Schleifen als überlegen erweist. Beim Hybrid NLJ-Hash Join kommt daher Hashing für alle Zugriffe zum Einsatz, bei denen das Verbundprädikat dies erlaubt. Die Autoren liefern zudem ein Kostenmodell und eine Heuristik zur Auswahl der Joinreihenfolge.

W-Join und FEW-Join

In [HAE03] schlagen die Autoren einen Algorithmus *W-Join* vor, der über mehreren Eingabedatenströmen mit Zeitstempeln operiert und die paarweise Definition von Zeitfensterbedingungen zwischen den Strömen erlaubt. Als Anwendungsszenarien für diese Art von Zeitfenstern werden insbesondere Sensordatenströme bezüglich bewegter Objekte genannt. Der Algorithmus kann sowohl mit verschachtelten Schleifen, als auch mit hash–basierter Partitionierung operieren.

Als Weiterentwicklung wird der Algorithmus *FEW-Join* vorgeschlagen, der Elemente erst verarbeitet, wenn alle Eingaben dessen Zeitstempel erreicht haben. Dadurch kann in einigen Fällen analog zum in 9.3.2 vorgestellten Algorithmus die Überprüfung der Zeitfensterbedingung entfallen, weil diese bereits durch das vorherige Entfernen anderer Elemente garantiert ist.

Adaptive Caching

In [BMWM05] wird die Idee der Hauptspeicherverarbeitung des XJoins und des MJoins auf die Verarbeitung von Datenströmen im Positiv/Negativ-Ansatz übertragen und erweitert. Die wesentliche Idee dabei ist, dass in Bäumen aus XJoins und MJoins Zwischenergebnisse in den Status der Operatoren über den Blättern immer materialisiert vorliegen, während ein MJoin gar keine Zwischenergebnisse puffert.

Die Autoren schlagen ein hybrides Verfahren *Adaptive Caching* (*A-Caching*) vor, das alle Eingaben eines mehrdimensionalen Verbundes verarbeitet und für jede eine unterschiedliche Anfragereihenfolge an die Status der anderen erlaubt. Zusätzlich werden dynamisch Caches angelegt, die Teilergebnisse materialisieren und somit die Berechnungskosten von Anfragefolgen reduzieren können. Dabei sind auch Kombinationen möglich, die nicht der Materialisierung in einem Baum aus XJoins und MJoins entsprechen.

Nach einer initialen Optimierung überwacht der Algorithmus fortlaufend die erwarteten Kosten und Nutzen solcher Caches und erstellt und entfernt diese dynamisch. Die Caches garantieren dabei nur, dass sie soweit vorhanden im Falle eines Treffers die korrekte Ergebnismenge enthalten und können somit während der normalen Verarbeitung kontinuierlich gefüllt und bei Speicherknappheit einfach entfernt werden. Da die Caches hash–basiert arbeiten, eignet sich Adaptive Caching im Wesentlichen für die mehrdimensionalen Verbundprädikate, die zu guten Teilen aus Gleichverbünden auf bestimmten Attributen bestehen. Zudem geht der Algorithmus von sequentieller, global sortierter Verarbeitung der Eingaben aus und kann daher im Gegensatz zu Bäumen aus Operatoren nicht von Multithreading profitieren.

17.3.2. Optimierung

In [VN02] stellen Viglas und Naughton erste Untersuchungen zur Optimierung der Datenstromverarbeitung an. Sie erkennen, dass klassische Kostenmodelle aus der Datenbankwelt nicht unverändert übertragen werden können, und schlagen als neuen Faktor für Kostenmodelle die Stromraten vor. Für Verbünde untersuchen sie bereits die Unterschiede zwischen Berechnung durch verschachtelte Schleifen oder Partitionierung mit Hilfe von Hashverfahren. Allerdings betrachten die Autoren noch keine Zeitfenster, sondern gehen davon aus, dass der Verbund unter allen Elementen zweier Ströme berechnet werden soll.

Kostenmodell für binäre Verbünde

Kang, Naughton und Viglas untersuchen in [KNV03] ausführlich binäre Strom-Verbünde über Eingabedatenströmen. Sie stellen ein Kostenmodell vor, das ähnlich zu dem in dieser Arbeit vorgestellten bereits Einfüge-, Anfrage und Löschkosten in den Datenstrukturen des Verbundes berücksichtigt. Dazu werden Stromraten und Fenstergrößen berücksichtigt, wobei die Zeitfenster relativ zum Zeitpunkt des Eintreffens der Elemente beim System betrachtet werden und Elemente die Fenster immer in derselben Reihenfolge betreten und verlassen. Als Datenstrukturen für die Status des Verbundes werden neben Listen auch Hashtabellen, B+Bäume und T-Bäume untersucht und analysiert, wobei auch deren asymmetrische Wahl untersucht wird.

Zusätzlich betrachten die Autoren die Fälle, in denen CPU-Ressourcen oder Hauptspeicher nicht ausreichen, um den Verbund komplett zu berechnen. Sie schlagen vor, in diesen Fällen die Anfragen beziehungsweise das Speichern von Elementen derart zu beschränken, dass eine möglichst große Teilmenge der Ausgabe zu erwarten ist.

Im Ausblick sehen Kang, Naughton und Viglas ein Kostenmodell für Verbünde, das Teil eines Kostenmodells für komplette Anfragepläne ist und wie es in dieser Arbeit betrachtet wird, als interessante Richtung für zukünftige Forschung an.

XGreedyJoin

In [GC06] untersuchen Gomes und Choi für zählbasierte Fenster die Wahl des optimalen buschigen Baums aus binären Verbundoperatoren. Sie gehen davon aus, dass die Reihen-

folge der Verbünde, also der Blätter im Verbundbaum, vorgegeben ist, und berechnen die optimale Baumstruktur dann mit Hilfe dynamischer Programmierung. Die Selektivitäten der Verbünde werden dabei anhand umfangreicher Gleichverteilungsannahmen abgeschätzt. Neben der erschöpfenden Suche in den Fenstern werden auch hash–basierte Verfahren betrachtet, wobei die Gleichverteilungsannahmen jedoch noch weitreichender sind.

In [GC08] erweitern die Autoren diese Arbeit und untersuchen dieselbe Fragestellung für beliebige Reihenfolgen der Verbünde. Sie zeigen zunächst, dass das sich daraus ergebende Problem NP-hart ist und schlagen dann zunächst einen Algorithmus *OptDP* vor, der wiederum auf dynamischer Programmierung beruht, aber sehr hohen Aufwand hat. Daher schlagen die Autoren mit dem *XGreedyJoin* einen weiteren Algorithmus vor, der einen Baum binärer Verbünde für n Eingabeströme in $O(n^3)$ vorschlägt. Zusätzlich regen die Autoren an, auf sich zur Laufzeit ändernde Bedingungen mit Planmigration zu reagieren und geben Verfahren zur Abschätzung der Selektivitäten auf Basis von Statistiken über die Zahl disjunkter Werte einzelner Verbundattribute an. Dabei treffen sie jedoch weiterhin umfassende Gleichverteilungsannahmen.

Ausnutzung von Änderungsmustern

In [GÖ05] betrachten Golab und Özsu die Fragestellung, wie sich Muster in den Zeitstempeln zur Optimierung innerhalb einer Datenstromalgebra ausnutzen lassen. Treffen Elemente beispielsweise nicht nur nach Start-, sondern auch nach Endzeitstempeln sortiert ein, so können, wie in 10.2.1 beschrieben, vereinfachte Löschstrukturen verwendet werden. Golab und Özsu definieren verschiedene Klassen von Änderungsmustern und rechnen diese als statische Metadaten durch ihre Datenstromalgebra hoch, die von global sortierter Verarbeitung der Eingabedatenströme ausgeht. Anhand dieser Daten werden dann physische Operatoren und Datenstrukturen ausgewählt, die effizienter arbeiten als solche, die für beliebige Eingabeströme geeignet sind.

Window-Oblivious Join

In diversen Anwendungsszenarien von DSMS wird der Verbund unter anderem dazu eingesetzt, zwei Ströme von Elementen über einen Gleichverbund ihrer Schlüssel zu joinen, wobei die Schlüssel in den Strömen mehr oder weniger monoton ansteigend auftreten. In [WTZ07] wird dieses Problem näher spezifiziert, wobei Parameter für die Verzögerung des Auftretens von Schlüsseln zwischen den Strömen sowie für mögliche Abweichungen von der Ordnung innerhalb der Ströme definiert werden. Die Autoren schlagen dann einen Algorithmus *Window-Oblivious Join* (*WO-Join*) vor, der für gegebene Obergrenzen dieser Parameter den Verbund berechnet.

17.3.3. Approximative Verbundverarbeitung

In diesem Unterabschnitt werden Arbeiten zu Verbundoperationen diskutiert, bei denen als Reaktion auf beschränkte Ressourcen die Anforderungen an die Vollständigkeit des Ergebnisses gelockert wurden.

Semantic Load Shedding

In [DGR03] schlagen Das, Gehrke und Riedewald bezüglich der approximativen Berechnung von Verbünden die Technik des *Semantic Load Shedding* vor, bei der eine benutzerdefinierte Ähnlichkeitsfunktion das berechnete Ergebnis mit dem kompletten vergleicht. Dies steht im Gegensatz zum *Random Load Shedding*, bei dem zufällige Elemente verworfen werden. In ihren Betrachtungen für Datenströme konzentrieren sich die Autoren allerdings auf das häufig verwendete Kriterium der größten Teilmenge des kompletten Ergebnisses, die erreicht werden soll. Dazu schlagen sie vor, Statistiken darüber zu erfassen, mit welcher Wahrscheinlichkeit ein Element zum Ergebnis beiträgt, und dann die Elemente mit den geringsten Wahrscheinlichkeiten zu verwerfen. Gegebenenfalls sollen die Wahrscheinlichkeiten dabei zusätzlich mit der noch verbleibenden Lebensdauer der Elemente im Zeitfenster gewichtet werden.

UNIFORM und PPS-CLUSTER

In [SW04b] stellen Srivastava und Widom grundlegende Überlegungen für den Fall an, dass zwar hinreichend CPU-Ressourcen, aber zu wenig Hauptspeicher für das Ausführen binärer Gleichverbünde über Datenströmen zur Verfügung steht. Sie liefern theoretische Überlegungen die zeigen, dass im Allgemeinen beim Verwerfen von Elementen weder eine maximale, noch eine repräsentative Teilmenge des kompletten Ergebnisses erreicht werden kann. Srivastava und Widom schlagen zusätzlich ein Modell vor, bei der die Wahrscheinlichkeit eines Elementes, am Verbund teilzunehmen, nicht von dessen Wert, sondern von dessen Zeitstempel abhängt. Für Verbünde über Stichproben der Fenster schlagen die Autoren zwei Algorithmen *UNIFORM* und *PPS-CLUSTER* vor.

JAQPOT

In [MRW11] setzen Mukherji und Rundensteiner die Arbeit aus [KNV03] (siehe 17.3.2) bezüglich des Falles, dass nicht genügend Ressourcen zur Verfügung stehen, um den Verbund komplett zu berechnen, fort. Sie konzentrieren sich dabei auf den Fall, dass zwar ausreichend Hauptspeicher, aber nicht genug CPU-Last verfügbar ist und untersuchen ausschließlich Bäume aus binären Verbünden. Dabei macht es keinen Sinn, das ursprüngliche Verfahren auf die einzelnen Verbünde anzuwenden, da dann gegebenenfalls unten im Baum hohe Ausgaberaten generiert werden, die oben wieder verworfen werden müssen. Die Autoren führen das Problem bei globaler Optimierung auf ein Rucksack-Problem zurück und schlagen einen Greedy-Algorithmus vor, der dieses für n Eingaben in $O(n\ log(n))$ löst.

Die Autoren gehen wiederum davon aus, Eingabedatenströme mit Zeitstempeln zu verarbeiten, und untersuchen zusätzlich einen Parameter, über den der Benutzer bestimmen soll, wie lange zwischen dem Eintreffen eines Elementes und dessen Auftreten in einem Ergebnis maximal vergehen darf. Um dies dann garantieren zu können, schlagen sie vor, Elemente aus den einzelnen Eingaben hinreichend oft zu verarbeiten, um Elemente aus den gegenüberliegenden Status rechtzeitig zu verdrängen.

Den kombinierten Algorithmus, der unter Einhaltung dieser Nutzervorgaben maximal

viele Ausgabeelemente produziert, nennen Mukherji und Rundensteiner *JAQPOT*. Leider beantworten sie ebenso wenig wie Kang, Naughton und Viglas die Frage, in welchen Fällen eine maximale Teilmenge des vollständigen Ergebnisses für den Nutzer optimal ist. Dies ist insbesondere problematisch, da diesem nicht transparent wird, welche Art von Ergebnissen fehlen.

GrubJoin

Auch Gedik, Wu, Yu und Liu schlagen in [GWYL05] einen binären Verbund vor, der bei ausreichendem Hauptspeicher aber zu hoher CPU-Last einen Teil der Verbundergebnisse berechnet. Sie konzentrieren sich dabei auf den Spezialfall von Verbünden über mengenwertigen Attributen. In [GWYL07] erweitern die Autoren ihre Arbeit auf mehrdimensionale Verbünde und allgemeinere Verbundprädikate. Der Algorithmus *GrubJoin* adaptiert dabei in regelmäßigen Abständen, welche Teile der Statusfenster über den Strömen er für den Verbund berücksichtigt. Als Kriterium wird dabei wie beim Load Shedding üblich deren Beitrag zu möglichst vielen Verbundergebnissen gewählt.

17.3.4. Verteilte Verbundverarbeitung

Während sich die Betrachtungen zu Verbundoperationen in dieser Arbeit auf die lokale Ausführung von Verbünden konzentrieren, beschäftigen sich diverse Arbeiten mit der Frage, wie sich Verbünde verteilt ausführen lassen. Dies ist insbesondere in Szenarien von Interesse, in denen die zu joinenden Ströme ebenfalls verteilten Ursprungs sind. In [AcJZ05] wird darüber hinaus davon ausgegangen, dass Ströme von verschiedenen Knoten gespeist werden und sich die Verteilung der gelieferten Werte dabei unterscheidet. Unter Ausnutzung dieser Verteilungen wird nun versucht, das Problem so zu partitionieren, dass die einzelnen Partitionen möglichst nahe am Ursprung der entsprechenden Daten berechnet werden.

Auch der *Distributed Hash Table Join*[PAPV08, PAPV09b, PAPV09a] wurde für die verteilte Berechnung von Verbünden über Datenströmen entwickelt. Dabei werden die Daten gehasht, und die Anfrageverarbeitung wird anhand des Teile und Herrsche-Prinzips mittels der Hashfunktion über verschiedene Knoten verteilt. Das Verfahren ist approximativ und setzt unter anderem synchrone Systemzeit auf allen beteiligten Knoten voraus.

In [PLS+06] wird die allgemeinere Fragestellung betrachtet, wie die Operatoren eines verteilt arbeitenden DSMS über die Knoten verteilt werden können. Dabei werden auch Synergien durch das gemeinsame Nutzen von Teilergebnissen und Reoptimierung zur Laufzeit berücksichtigt.

Pipelined State Partitioning

Einen zum Handshake Join verwandten Ansatz verfolgen Wang und Rundensteiner in [WR09] zur parallelen Berechnung mehrdimensionaler Datenstromverbünde mit beliebigen Verbundprädikaten über Eingabedatenströmen. Sie stellen das Verfahren *Pipelined State Partitioning* (*PSP*) vor, bei dem die Verarbeitungsknoten einen Ring bilden

und die Zeitfenster über den Ring partitioniert werden. Eintreffende Elemente werden dann zum einen auf dem zu ihrer Partition ihres Zeitfensters gehörigen Knoten abgelegt und durchlaufen zusätzlich als Anfrageelemente den Ring. Treffen sie dabei auf Elemente, mit denen eine Chance auf Erfüllen des Verbundprädikates besteht, so werden partielle Ergebnisse gebildet, die zur Anfrage der Partitionen der andere Ströme wiederum den Ring durchlaufen. Dabei kann jeder Eingabe eine eigene Anfragereihenfolge zugeordnet werden.

Verteilter Verbund für Sensordaten

In [MJIG10] wird das Problem betrachtet, den Verbund zweier Datenströme, die jeweils aus mehreren Sensoren gespeist werden, verteilt zu berechnen. Um dabei zu verhindern, dass jedes Element des einen Stromes mit jedem des anderen joinen könnte, wird vorausgesetzt, dass die Ströme in Partitionen zerfallen und Verbundergebnisse nur aus Elementen zusammengehöriger Partitionen gebildet werden können.

Verteilter Theta-Verbund

In [EEVK14] wird ein Rahmenwerk zur adaptiven Verteilung des Theta-Verbundes vorgestellt. Die Eingabeströme werden dabei zufällig auf n beziehungsweise m Partitionen verteilt. Jede der $m \cdot n$ Kombinationen dieser Partitionen wird dann auf einem Rechner mit einem beliebigen lokalen Verbundalgorithmus verarbeitet. Durch Erhebung dynamischer Metadaten wird das Verhältnis von n zu m zur Laufzeit dynamisch adaptiert und Daten entsprechend neu verteilt. Temporäre Redundanz verhindert dabei Verarbeitungspausen. Das Rahmenwerk zielt explizit nicht auf Verbünde über Zeitfenstern ab, sondern soll alle historischen Daten verarbeiten. Dabei ist allerdings zu erwarten, dass ohne das Hinzufügen zusätzlicher Rechner das Verfahren degeneriert. Allerdings könnte es durch den Einsatz von fensterbasierten lokalen Algorithmen und das Hinzunehmen zusätzlicher Metadaten wohl zu einem fensterbasierten Verfahren ausgebaut werden.

17.3.5. Punctuations

Die bereits in 16.1.1 diskutierten Punctuations, in Datenströme eingebettete Metadaten, können natürlich auch dazu verwendet werden, Verbünde zu optimieren. In [TMSF03] werden wertbasierte Punctuations als generelles Konzept zur Lösung des Problems vorgeschlagen, statusbehaftete Operatoren über potentiell unbegrenzten Datenströmen berechnen zu können.

Werden ausschließlich Punctuations zum Entfernen von Elementen aus dem Status von Verbundoperatoren verwendet, so stellt sich die Frage, wie garantiert werden kann, dass auch jedes Element wirklich entfernt werden kann. In [LCT+06] wird diskutiert, wie dies garantiert werden kann.

Dem *Metadata-aware Stream Join* (*MJoin*)[DRH03] erweitert zur hash–basierten Berechnung von 1:n Gleichverbünden bezüglich eines Attributes das Konzept des X-Joins um die Verwendung wertbasierter Punctuations bezüglich des Verbundattributes. Der

PJoin[DMRH04] setzt dies für hash–basierte binäre Gleichverbünde bezüglich eines Attributes fort. Die Punctuations geben jeweils an, dass Werte des Attributes, die ein in der Punctuation spezifiziertes Prädikat erfüllen, fortan nicht mehr im Strom vorkommen können. Die Algorithmen nutzen diese Information dann zum Löschen von Elementen aus den Status. Der PJoin generiert, soweit möglich, auch Punctuations in seinen Ausgabestrom.

Mit dem *PWJoin*[DR04] wird diese Idee auf Zeitfensterverbünde über Eingabedatenströmen übertragen. Dabei werden sowohl bei der Ausnutzung der Punctuations zur Statusbereinigung, als auch beim Weiterleiten von Punctuations auch Kombinationen aus zeitlichem Fortschritt und Punctuations ausgenutzt.

In [LCT+06] wird erläutert, wie das Konzept auf mehrdimensionale Verbundoperatoren mit verschiedenen Verbundattributen erweitert werden kann.

In [LTS+08] wird vorgeschlagen, die zeitliche Ordnung auf Datenströmen als Bedingung aufzugeben und stattdessen ganz auf Punctuations zu setzen. Ähnlich zum Window-Oblivious Join (siehe 17.3.2) wird der Fortschritt auf Datenströmen hier als Fortschritt auf Attributen definiert, wobei diese ab einer entsprechenden Punctuation keine Werte kleiner als in der Punctuation vorgegeben annehmen dürfen. Dies stellt die Übertragung des Konzepts, auf strikte Zeitstempelordnung zu verzichten und per Punctuations minimale Zeitstempel, die noch auftreten zu können, zu versenden, von den Zeitstempeln auf Attribute dar.

17.3.6. Weitere Arbeiten

In [MK09] wird die dynamische Reoptimierung von Verbundbäumen in Datenbanksystemen diskutiert. Dabei kommt Simulation zum Einsatz, um die Effizienz potentieller Ausführungspläne zu evaluieren.

In [BBD05] wird ein dynamischer Reoptimierungsprozess für generelle Datenbankanfragen vorgeschlagen, bei dem während der Anfrageausführung bessere Konfidenzintervalle für geschätzte Parameter bestimmt werden und auf deren Basis eine Reoptimierung durchgeführt wird.

In [GELA10] wir ausführlich die Übertragung des Konzeptes von Sichten aus der Welt der Datenbanken in die der Datenstromverarbeitung diskutiert.

Teil VI.

Diskussion und Ausblick

18. Diskussion

Die zunehmende Technifizierung der verschiedensten Lebensbereiche in Verbindung mit der globalen Vernetzung über das Internet hat den über das Computerzeitalter fortwährenden Prozess des kontinuierlichen Anwachsens der anfallenden Datenmengen weiter beschleunigt. Parallel dazu hat in der sich entwickelnden Informationsgesellschaft das schnelle Gewinnen von Informationen aus diesen Daten enorm an Bedeutung gewonnen. Datenbanksysteme als etablierte Technik zur Verwaltung und Analyse von Daten haben sich für einige solcher Anwendungen als weniger geeignet erwiesen. Daher etablieren sich ergänzend Datenstrommanagementsysteme, da sie die Verarbeitung von Datenströmen in quasi Echtzeit ermöglichen.

Die Nutzbarkeit und Zugänglichkeit dieser Technologie hängt wesentlich von einer wohldefinierten Semantik und der Verfügbarkeit einer guten Anfragesprache ab. Die Übertragung der Konzepte der erweiterten relationalen Algebra und der zugehörigen beliebten Anfragesprache SQL von Datenbanken auf die Datenstromverarbeitung hat sich dabei in Forschung und Praxis bewährt. Gleiches gilt für das konsequent datengetriebene Verarbeitungsparadigma und die Verwendung von Gültigkeitsintervallen zur internen Repräsentation der zeitlichen Komponente der Semantik, die es ermöglicht, potentiell unbegrenzte Datenströme mit endlichen Ressourcen zu verarbeiten. Diese Ansätze wurden daher auch für diese Arbeit gewählt.

Die Mächtigkeit eines DSMS bei der Anfrageausführung und damit die Möglichkeiten darauf aufbauender Anwendungen hängt wesentlich von der Verfügbarkeit von Operationen ab, die Daten aus verschiedenen Datenströmen miteinander verknüpfen. Die wichtigste solche Operation ist die Verbundoperation. Daher spielt es für ein DSMS eine wichtige Rolle, diese effizient zu unterstützen. Die Bereitstellung der dafür benötigten Techniken ist daher Thema dieser Arbeit.

Durch die Zurückführung der Semantik der Datenstromverarbeitung mit Gültigkeitsintervallen auf die der erweiterten relationalen Algebra mit Hilfe der Schnappschuss-Reduzierung wird die Semantik der Verbundoperation sowohl über der abstrakten logischen als auch der in der praktischen Umsetzung verwendeten physischen Repräsentation von Datenströmen formal definiert. Alle in dieser Arbeit vorgestellten Verbundalgorithmen erfüllen die sich daraus ergebenden Anforderungen an Vollständigkeit und Korrektheit des Ergebnisses. Mit den äußeren Verbünden wird zusätzlich noch eine ebenfalls aus Datenbanken bewährte Erweiterung des Konzeptes der Verbundoperation behandelt.

Bezüglich der Implementierung der Verbundoperation über Datenströmen wurde insbesondere auch die Übertragbarkeit von bewährten Konzepten aus der Datenbankforschung untersucht. Da die Datenströme innerhalb des Systems nach Startzeitstempeln der Gültigkeitsintervalle sortiert verarbeitet werden, hat sich dabei insbesondere die Übertragung des für sortierbasierte Datenbankverbünde entwickelten Verfahrens bewährt,

bezüglich beider Eingaben jeweils eine Datenstruktur, genannt SweepArea, zu unterhalten und dann zur Verarbeitung jedes Elementes drei Schritte durchzuführen: Das Bereinigen der Datenstruktur zur anderen Eingabe zum Einsparen von Ressourcen, das Anfragen derselben Struktur zum Auffinden von Ergebnissen und das Speichern des Elementes in der Datenstruktur zur eigenen Eingabe.

Durch die klare Abgrenzung dieses Basisalgorithmus von der Implementierung der abstrakten Datenstruktur SweepArea kann das Verfahren durch die Wahl derer konkreter Implementierungen noch stark variiert werden. Diese wiederum können ihrerseits wieder aus DBMS bewährte Konzepte umsetzen. Einfache Strukturen, die bei Anfragen jedes enthaltene Element mit dem anfragenden gegen das Verbundprädikat prüfen, bilden dabei das Gegenstück zu DBMS-Verbünden mittels verschachtelter Schleifen. Analog zu diesen verursachen sie relativ hohen Aufwand, sind aber essentiell, da sie die wichtige Eigenschaft haben, mit beliebigen Verbundprädikaten einsetzbar zu sein, und damit unter anderem die Anwendungen Theta-Verbund und Kreuzprodukt abdecken. Komplexere SweepAreas strukturieren die Daten und bilden damit das Gegenstück zu Index-Verbünden aus DBMS. Damit lässt sich ein breites Spektrum an spezielleren Verbundprädikaten abdecken, darunter insbesondere auch verschiedene Arten räumlicher Verbünde. Einen wichtigen Spezialfall bildet die Indexierung mit Hilfe von Hash-Tabellen. Mit so aufgebauten SweepAreas können Äquivalenzklassenverbünde besonders effizient unterstützt werden. Darunter fallen insbesondere die extrem wichtigen Spezialfälle Gleichverbund und Natürlicher Verbund.

Eine weitere wesentliche Anforderung an die SweepAreas ist die Unterstützung des Löschens auf Grund der temporalen Semantik nicht mehr benötigter Daten. In dieser Arbeit wird mit der zweistufigen SweepArea ein Konzept präsentiert, um dieses Problem orthogonal zu dem effizienter Anfragen zu betrachten. Auf diese Art wird eine hohe Flexibilität bei der Implementierung neuer SweepAreas erreicht. Es werden aber auch Optimierungen vorgestellt, die es ermöglichen, wichtige Spezialfälle noch effizienter zu gestalten. Die verschiedenen Varianten ermöglichen zudem eine Prioritätensetzung bezüglich der Ziele der Reduktion des Speicherverbrauchs und der der CPU-Benutzung.

Da alle in dieser Arbeit vorgestellten Verbundimplementierungen für die Verwendung auf Basis einer datengetriebenen Operatoralgebra gedacht sind, erfüllen sie die dafür notwendige Bedingung, auch ihre Ausgabe nach Startzeitstempeln der Gültigkeitsintervalle sortiert zu produzieren. Das dazu erforderliche Synchronisieren bezüglich des zeitlichen Fortschrittes der Eingaben kann dabei sowohl am Eingang als auch am Ausgang des Verbundoperators geschehen, wobei beide Vorgehensweisen unterschiedliche Vorteile haben. Daher wird in dieser Arbeit ein hybrider Algorithmus eingeführt, der deren Vorteile in sich vereinigt.

Obwohl ein sortierbasiertes Verfahren das Konzept für diese Verbundimplementierungen liefert, basieren diese doch nicht auf einer nach den Werten der Daten sortierten Verarbeitung. Mit dem Temporalen Progressive-Merge-Join wird in dieser Arbeit daher ein zusätzliches Verfahren präsentiert, das eine solche Verarbeitung umsetzt und dabei das zur Produktion früher Ergebnisse bei der Verbundberechnung in Datenbanken entwickelte Verfahren Progressive-Merge-Join auf die Datenstromverarbeitung überträgt. Der Temporale Progressive-Merge-Join erfüllt zudem die Aufgabe, Verbünde berechnen zu können, deren Status nicht im Hauptspeicher gehalten werden kann. Damit ist auch dieses

Problem für sehr viele Verbundkriterien gelöst, da sortierbasierte Verbundverfahren ein weites Spektrum davon abdecken.

Alle in dieser Arbeit vorgestellten Verbundalgorithmen sind nicht auf die Verarbeitung von zwei Eingabedatenströmen beschränkt, sondern können auch mehrdimensionale Verbünde berechnen. Im Gegensatz zu Bäumen aus binären Verbünden sparen sie dabei wenn günstig die Materialisierung von Zwischenergebnissen in SweepAreas ein. Der dadurch entstehende Mehraufwand bei der Verarbeitung der Elemente kann dabei oftmals durch die feingranulare Justierbarkeit der Anfragereihenfolgen kompensiert werden.

Für den effizienten Einsatz der Vielzahl an Verbundverfahren und deren Parametrisierung ist es wichtig, den Ressourcenverbrauch einer der verschiedenen für eine konkrete Anfrage in Frage kommenden Varianten abschätzen zu können. Zu diesem Zweck wird in dieser Arbeit ein detailliertes Kostenmodell beschrieben, das hinreichend abstrahiert, um es mit geringem Berechnungsaufwand nutzen zu können, und dennoch präzise Kostenabschätzungen erlaubt.

Da bei der Wahl des Ausführungsplans bei der Installation einer neuen kontinuierlichen Anfrage auf Grund fehlender Informationen bezüglich der zu verarbeitenden Datenströme oftmals noch keine optimale Entscheidung getroffen werden kann, werden Mechanismen zur Reoptimierung zur Laufzeit benötigt. Mit der dynamischen Planmigration wird in dieser Arbeit eine geeignete Technik diskutiert. Die Voraussetzungen für deren Einsatz werden klar aufgezeigt, ebenso dass diese im Falle von Modifikationen an Verbundbäumen erfüllt sind. Dabei wird mit der weiterentwickelten Referenzpunktmethode sichergestellt, dass die Auswirkungen der Migration auf die Kosten nachfolgender Operatoren stark begrenzt werden.

Der effiziente Einsatz der Verfahren zur Optimierung und Reoptimierung, insbesondere auch unter Einsatz des Kostenmodells, sowie die richtige Parametrisierung der Verfahren und deren Steuerung erfordern Informationen bezüglich der zu verarbeitenden Daten und des jeweils aktuellen Systemverhaltens. Als besonders wichtiges Beispiel sind dabei Selektivitäten zu nennen, die erheblichen Einfluss auf das Verhalten von Verbünden haben. Da diverse solcher Metadaten in einem DSMS während der langen Laufzeit der kontinuierlichen Anfragen Änderungen unterworfen sein können und viele überhaupt erst zur Laufzeit sinnvoll erhoben werden können, sind effiziente Mechanismen zur Erhebung dynamischer Metadaten von essentieller Bedeutung für ein DSMS im Allgemeinen, insbesondere aber auch für die Verbundverarbeitung. Daher liegt ein Schwerpunkt dieser Arbeit auf dem Thema Metadaten.

Während die Verwendung dynamischer Metadaten somit hilft Ressourcen zu sparen, benötigt die Erhebung der Metadaten zunächst einmal selbst Ressourcen. Daher ist es wichtig, diese nicht nach dem Streuschussprinzip zu erheben, sondern gezielt für den jeweiligen Einsatzzweck. Das in dieser Arbeit vorgestellte Rahmenwerk ermöglicht es daher, dynamische Metadaten bei Bedarf anzufordern, diese dann zu nutzen und sie dann wieder freizugeben, wenn sie nicht mehr benötigt werden. Dabei wird sichergestellt, dass der für die jeweiligen Metadaten benötigte Aufwand nur während der Zeit anfällt, in der diese nutzbar sind.

Während der Bereitstellung der Metadaten wird durch unterschiedliche Aktualisierungsmechanismen der Aufwand weiter reduziert. Wenn sinnvoll möglich, werden dynamische Metadaten nur beim Zugriff bedarfsgesteuert erzeugt. Metadaten, die auf

Grund eines zeitlichen Bezuges zur konsistenten Erhebung eine gesteuerte Auswertung benötigen, werden mittels periodischer Aktualisierung unterstützt. Hängt die Änderung eines Metadatums zwingend von bestimmten Ereignissen, insbesondere von der Änderung eines anderen Metadatums ab, so werden Änderungen gezielt nur dann ausgelöst, wenn diese Bedingung erfüllt ist.

Das Rahmenwerk stellt dabei zudem sicher, dass sämtliche Metadaten konsistent erhoben werden, auch für den Fall des parallelen Zugriffs durch mehrere Nutzer, wobei redundante Berechnungen vermieden werden. Wird ein Metadatum ganz oder teilweise aus anderen bestimmt, so übernimmt das Rahmenwerk automatisch nicht nur die hierarchische Berechnung, sondern auch die Anforderung und Freigabe. Allerdings wird nicht nur die Benutzung der dynamischen Metadaten so komfortabel wie möglich gestaltet, sondern durch Abstraktion der Bereitstellungsmechanismen und der Abhängigkeitsverwaltung auch die Unterstützung zusätzlicher Metadaten bei der Implementierung von DSMS-Komponenten.

Damit eignet sich das Rahmenwerk neben dem Einsatz in der Steuerung und Überwachung von DSMS zur Laufzeit auch sehr gut zur experimentellen Evaluation von Techniken in der Datenstromforschung. Es kam daher auch bei der experimentellen Auswertung der in dieser Arbeit vorgestellten Techniken zum Einsatz.

19. Ausblick

Der Fokus dieser Arbeit liegt auf der datengetriebenen Berechnung der Verbundoperation über Datenströmen im Intervallansatz und auf dazu hilfreichen Techniken. Darauf aufbauend eröffnen sich sowohl im Bereich Metadaten als auch im Bereich Verbundverarbeitung weitere interessante Themen für zukünftige Arbeiten.

Metadaten werden in dieser Arbeit auf der Metaebene eingesetzt, nämlich zur Analyse und Optimierung des Systemverhaltens. Allerdings können Metadaten auch aus der Anwendungssicht von Interesse sein. Modelliert beispielsweise ein Verbundprädikat eine Fehlerbedingung, indem es das unerwünschte Zusammentreffen zweier Ereignisse erkennt, so sind die einzelnen so erzeugten Alarme für den Anwender von Bedeutung. Erhöht sich die Selektivität des Verbundes, so bedeutet dies, dass vermehrt Fehler auftreten, was auf ein tiefer liegendes Problem hindeuten kann. Das Metadatum könnte also auch von Interesse für den Anwender sein, nicht nur für Administratoren und das System selbst. Über die Möglichkeit, Metadaten ihrerseits als Datenquellen zur Verfügung zu stellen, ist deren Verwendung als Daten im selben System möglich.

In der Datenstromforschung wurden neben der Übertragung der Operatoren der erweiterten relationalen Algebra auch eine Vielzahl weiterer Operatoren entwickelt, darunter solche zur Mustererkennung auf Abfolgen von Stromelementen oder zu Vorhersagen über das zukünftige Verhalten der Ströme. Die Anwendung dieser Techniken auch auf die Metadaten wurde beispielsweise schon zu Optimierungszwecken eingesetzt[HKRS08], eröffnet jedoch auch darüber hinaus weitere interessante Möglichkeiten.

Bei den Verbünden ist diese Arbeit klar auf den Intervallansatz fokussiert und es wird insbesondere der Fall betrachtet, dass beide Eingaben diesem folgen. Allerdings hat dieser in manchen Anwendungsfällen den Nachteil, dass für einzelne Datenströme eine Festlegung des Endzeitstempels durch die Applikation zum Zeitpunkt des Eintretens eines Ereignisses noch nicht möglich ist. In solchen Fällen hat die Verwendung des Positiv/Negativ-Ansatzes gegebenenfalls Vorteile. Während die Verbundoperation für diesen Ansatz recht gut erforscht ist, wären detailliertere Betrachtungen zu einem Verbund zwischen einem Strom im Positiv/Negativ-Ansatz mit einem im Intervallansatz interessant. Ein solcher könnte in einem hybriden System, das beide Ansätze unterstützt, der vorherigen Konvertierung eines der Ströme überlegen sein.

Ähnliches gilt für das Nebeneinander von Datenströmen und persistierten Daten. In der Praxis ist es häufig hilfreich, Datenströme mit Daten aus Datenbanken oder anderen Systemen, die Daten über längere Zeit speichern, zu verknüpfen. Dies kann man beispielsweise relational als Verbund zwischen einem Datenstrom und einer Datenbanktabelle modellieren. Die Semantik auf Basis der Schnappschuss-Reduzierung ist dabei jedoch nur anwendbar, wenn auch die Datenbank Gültigkeitsinformationen zur Verfügung stellen kann, was üblicherweise nicht der Fall ist. Weitergehende Lösungen mit wohldefinierter Semantik wären daher wünschenswert.

Die in 10.2.4 vorgestellte Optimierung für den Selbstverbund ist unmodifiziert nur anwendbar, wenn beide Eingabeströme identisch sind. Für diverse Anwendungsfälle sind aber auch Verbünde von Strömen von Interesse, bei denen die eine Eingabe eine modifizierte Variante der anderen darstellt. Als wichtiges Beispiel ist eine Modifikation der Gültigkeitsintervalle durch Addieren eines Deltas auf alle Zeitstempel zu nennen. Da die Optimierung des Selbstverbundes bis zu der Hälfte des Ressourcenverbrauchs einsparen kann, wäre eine Ausweitung ihrer Verwendbarkeit auf zusätzliche Anwendungsfälle ein weiteres interessantes Gebiet für zukünftige Arbeiten.

Literaturverzeichnis

[AAB+05a] Abadi, Daniel J. ; Ahmad, Yanif ; Balazinska, Magdalena ; Çetintemel, Ugur ; Cherniack, Mitch ; Hwang, Jeong-Hyon ; Lindner, Wolfgang ; Maskey, Anurag ; Rasin, Alex ; Ryvkina, Esther ; Tatbul, Nesime ; Xing, Ying ; Zdonik, Stanley B.: The Design of the Borealis Stream Processing Engine. In: *CIDR*, 2005, S. 277–289

[AAB+05b] Ali, Mohamed H. ; Aref, Walid G. ; Bose, Raja ; Elmagarmid, Ahmed K. ; Helal, Abdelsalam ; Kamel, Ibrahim ; Mokbel, Mohamed F.: NILE-PDT: A Phenomenon Detection and Tracking Framework for Data Stream Management Systems. In: *VLDB*, 2005, S. 1295–1298

[ABW06] Arasu, Arvind ; Babu, Shivnath ; Widom, Jennifer: The CQL Continuous Query Language: Semantic Foundations and Query Execution. In: *VLDB Journal* 15 (2006), Nr. 2, S. 121–142

[ACC+03] Abadi, Daniel J. ; Carney, Don ; Cetintemel, Ugur ; Cherniack, Mitch ; Convey, Christian ; Lee, Sangdon ; Stonebraker, Michael ; Tatbul, Nesime ; Zdonik, Stan: Aurora: A New Model and Architecture for Data Stream Management. In: *VLDB Journal* 12 (2003), Nr. 2, S. 120–139

[ACG+04] Arasu, Arvind ; Cherniack, Mitch ; Galvez, Eduardo F. ; Maier, David ; Maskey, Anurag ; Ryvkina, Esther ; Stonebraker, Michael ; Tibbetts, Richard: Linear Road: A Stream Data Management Benchmark. In: *Proc. of the VLDB Endowment (PVLDB)*, 2004, S. 480–491

[AcJZ05] Ahmad, Yanif ; Çetintemel, Ugur ; Jannotti, John ; Zgolinski, Alexander: Locality Aware Networked Join Evaluation. In: *ICDE Workshops*, 2005, S. 1183

[AH00] Avnur, Ron ; Hellerstein, Joseph M.: Eddies: Continuously Adaptive Query Processing. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2000, S. 261–272

[APR+98] Arge, Lars ; Procopiuc, Octavian ; Ramaswamy, Sridhar ; Suel, Torsten ; Vitter, Jeffrey S.: Scalable Sweeping-Based Spatial Join. In: *Proc. of the VLDB Endowment (PVLDB)*, 1998, S. 570–581

[AVL62] Adel'son-Vel'skii, G.M. ; Landis, E.M.: An algorithm for the organization of information. In: *Soviet Math. Dokl.* 3 (1962), S. 1259–1263

[Bal06] Balazinska, Magdalena: *Fault-Tolerance and Load Management in a Distributed Stream Processing System*, Massachusetts Institute of Technology, Diss., 2006

[Bay72] Bayer, Rudolf: Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms. In: *Acta Inf.* 1 (1972), S. 290–306

[BBC+04] Balakrishnan, Hari ; Balazinska, Magdalena ; Carney, Donald ; Çetintemel, Ugur ; Cherniack, Mitch ; Convey, Christian ; Galvez, Eduardo F. ; Salz, Jon ; Stonebraker, Michael ; Tatbul, Nesime ; Tibbetts, Richard ; Zdonik, Stanley B.: Retrospective on Aurora. In: *VLDB J.* 13 (2004), Nr. 4, S. 370–383

[BBD+01] Bercken, Jochen van den ; Blohsfeld, Björn ; Dittrich, Jens-Peter ; Krämer, Jürgen ; Schäfer, Tobias ; Schneider, Martin ; Seeger, Bernhard: XXL - A Library Approach to Supporting Efficient Implementations of Advanced Database Queries. In: *Proc. of the VLDB Endowment (PVLDB)*, 2001. – ISBN 1–55860–804–4, S. 39–48

[BBD+02] Babcock, Brian ; Babu, Shivnath ; Datar, Mayur ; Motwani, Rajeev ; Widom, Jennifer: Models and Issues in Data Stream Systems. In: *Symp. on Principles of Database Systems (PODS)*, 2002. – ISBN 1–58113–507–6, S. 1–16

[BBD05] Babu, Shivnath ; Bizarro, Pedro ; DeWitt, David: Proactive Re-Optimization. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2005, S. 107–118

[BBDM03] Babcock, Brian ; Babu, Shivnath ; Datar, Mayur ; Motwani, Rajeev: Chain: Operator Scheduling for Memory Minimization in Data Stream Systems. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2003, S. 253–264

[BDD+10] Botan, Irina ; Derakhshan, Roozbeh ; Dindar, Nihal ; Haas, Laura M. ; Miller, Renée J. ; Tatbul, Nesime: SECRET: A Model for Analysis of the Execution Semantics of Stream Processing Systems. In: *PVLDB* 3 (2010), Nr. 1, S. 232–243

[BDG+07] Brenna, Lars ; Demers, Alan J. ; Gehrke, Johannes ; Hong, Mingsheng ; Ossher, Joel ; Panda, Biswanath ; Riedewald, Mirek ; Thatte, Mohit ; White, Walker M.: Cayuga: a high-performance event processing engine. In: *SIGMOD Conference*, 2007, S. 1100–1102

[BDS00] Bercken, Jochen van den ; Dittrich, Jens-Peter ; Seeger, Bernhard: javax.XXL: A prototype for a Library of Query processing Algorithms. In: Chen, Weidong (Hrsg.) ; Naughton, Jeffrey F. (Hrsg.) ; Bernstein, Philip A. (Hrsg.): *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2000, S. 588

[BE77] Blasgen, Mike W. ; Eswaran, Kapali P.: Storage and Access in Relational Data Bases. In: *IBM Systems Journal* 16 (1977), Nr. 4, S. 362–277

[BHS05] Blohsfeld, Björn ; Heinz, Christoph ; Seeger, Bernhard: Maintaining Nonparametric Estimators over Data Streams. In: *Proc. of the Conf. on Database Systems for Business, Technology, and the Web (BTW)*, 2005, S. 385–404

[BKK+00] Braumandl, Reinhard ; Keidl, Markus ; Kemper, Alfons ; Kossmann, Donald ; Kreutz, Alexander ; Pröls, Stefan ; Seltzsam, Stefan ; Stocker, Konrad: ObjectGlobe: Ubiquitous Query Processing on the Internet. In: *TES*, 2000, S. 247–268

[BKK+01] Braumandl, Reinhard ; Keidl, Markus ; Kemper, Alfons ; Kossmann, Donald ; Seltzsam, Stefan ; Stocker, Konrad: ObjectGlobe: Open Distributed Query Processing Services on the Internet. In: *IEEE Data Eng. Bull.* 24 (2001), Nr. 1, S. 64–70

[BMM+04] Babu, Shivnath ; Motwani, Rajeev ; Munagala, Kamesh ; Nishizawa, Itaru ; Widom, Jennifer: Adaptive Ordering of Pipelined Stream Filters. In: *SIGMOD Conference*, 2004, S. 407–418

[BMWM05] Babu, Shivnath ; Munagala, Kamesh ; Widom, Jennifer ; Motwani, Rajeev: Adaptive Caching for Continuous Queries. In: *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, 2005, S. 118–129

[BSW04] Babu, Shivnath ; Srivastava, Utkarsh ; Widom, Jennifer: Exploiting k-Constraints to Reduce Memory Overhead in Continuous Queries over Data Streams. In: *ACM Transactions on Database Systems (TODS)* 29 (2004), Nr. 3, S. 545–580

[BTW+06] Bai, Yijian ; Thakkar, Hetal ; Wang, Haixun ; Luo, Chang ; Zaniolo, Carlo: A Data Stream Language and System Designed for Power and Extensibility. In: *Proc. of the Int. Conf. on Information and Knowledge Management (CIKM)*, 2006, S. 337–346

[BVKD09] Bornea, Mihaela A. ; Vassalos, Vasilis ; Kotidis, Yannis ; Deligiannakis, Antonios: Double Index NEsted-Loop Reactive Join for Result Rate Optimization. In: *ICDE*, 2009, S. 481–492

[BVKD10] Bornea, Mihaela A. ; Vassalos, Vasilis ; Kotidis, Yannis ; Deligiannakis, Antonios: Adaptive Join Operators for Result Rate Optimization on Streaming Inputs. In: *IEEE Trans. Knowl. Data Eng.* 22 (2010), Nr. 8

[Cam02] Cammert, Michael: *Frühe Ergebnisse bei Verbundoperationen*. Marburg, Philipps-Universität, Diplomarbeit, 2002

[CC87] Clifford, James ; Croker, Albert: The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In: *ICDE*, 1987, S. 528–537

[CC08] Claypool, Kajal T. ; Claypool, Mark: Teddies: Trained Eddies for Reactive Stream Processing. In: *DASFAA*, 2008, S. 220–234

[CCD+03] Chandrasekaran, Sirish ; Cooper, Owen ; Deshpande, Amol ; Franklin, Michael J. ; Hellerstein, Joseph M. ; Hong, Wei ; Krishnamurthy, Sailesh ; Madden, Samuel ; Raman, Vijayshankar ; Reiss, Frederick ; Shah, Mehul A.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In: *Proc. of the Conf. on Innovative Data Systems Research (CIDR)*, 2003

[CCKN01] Cai, Jin yi ; Chakaravarthy, Venkatesan T. ; Kaushik, Raghav ; Naughton, Jeffrey F.: On the Complexity of Join Predicates. In: *PODS*, 2001

[CCZ+03] Carney, Donald ; Cetintemel, Ugur ; Zdonik, Stan ; Rasin, Alex ; Cerniak, Mitch ; Stonebraker, Michael: Operator Scheduling in a Data Stream Manager. In: *Proc. of the VLDB Endowment (PVLDB)*, 2003, S. 838–849

[CDTW00] Chen, Jianjun ; DeWitt, David J. ; Tian, Feng ; Wang, Yuan: NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2000, S. 379–390

[CGN10] Chen, Shimin ; Gibbons, Phillip B. ; Nath, Suman: PR-join: a non-blocking join achieving higher early result rate with statistical guarantees. In: *SIGMOD Conference*, 2010, S. 147–158

[Cha98] Chaudhuri, Surajit: An Overview of Query Optimization in Relational Systems. In: *Symp. on Principles of Database Systems (PODS)*, 1998, S. 34–43

[CHK+03] Cammert, Michael ; Heinz, Christoph ; Krämer, Jürgen ; Schneider, Martin ; Seeger, Bernhard: A Status Report on XXL – a Software Infrastructure for Efficient Query Processing. In: *IEEE Data Engineering Bulletin* 26 (2003), Nr. 2, S. 12–18

[CHK+06] Cammert, Michael ; Heinz, Christoph ; Krämer, Jürgen ; Riemenschneider, Tobias ; Schwarzkopf, Maxim ; Seeger, Bernhard ; Zeiss, Alexander: Stream Processing in Production-to-Business Software. In: *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, 2006, S. 168–169

[CHK+07] Cammert, Michael ; Heinz, Christoph ; Krämer, Jürgen ; Seeger, Bernhard ; Vaupel, Sonny ; Wolske, Udo: Flexible Multi-Threaded Scheduling for Continuous Queries over Data Streams. In: *First Int. Workshop on Scalable Stream Processing Systems (SSPS)*, 2007. – (Co-located with ICDE)

[CHKS03] Cammert, Michael ; Heinz, Christoph ; Krämer, Jürgen ; Seeger, Bernhard: Datenströme im Kontext des Verkehrsmanagements. In: *Mobilität und Informationssysteme, Workshop des GI-Arbeitskreises „Mobile Datenbanken und Informationssysteme“*, 2003

[CHKS04] Cammert, Michael ; Heinz, Christoph ; Krämer, Jürgen ; Seeger, Bernhard: Anfrageverarbeitung auf Datenströmen. In: *Datenbank Spektrum* 11 (2004), S. 5–13

[CHKS05] Cammert, Michael ; Heinz, Christoph ; Krämer, Jürgen ; Seeger, Bernhard: Sortierbasierte Joins über Datenströmen. In: *Proc. of the Conf. on Database Systems for Business, Technology, and the Web (BTW)*, 2005, S. 365–384

[CJSS03a] Cranor, Charles D. ; Johnson, Theodore ; Spatscheck, Oliver ; Shkapenyuk, Vladislav: Gigascope: A Stream Database for Network Applications. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2003, S. 647–651

[CJSS03b] Cranor, Charles D. ; Johnson, Theodore ; Spatscheck, Oliver ; Shkapenyuk, Vladislav: The Gigascope Stream Database. In: *IEEE Data Eng. Bull.* 26 (2003), Nr. 1, S. 27–32

[CKS07] Cammert, Michael ; Krämer, Jürgen ; Seeger, Bernhard: Dynamic Metadata Management for Scalable Stream Processing Systems. In: *First Int. Workshop on Scalable Stream Processing Systems (SSPS)*, 2007. – (Co-located with ICDE)

[CKSV06] Cammert, Michael ; Krämer, Jürgen ; Seeger, Bernhard ; Vaupel, Sonny: An Approach to Adaptive Memory Management in Data Stream Systems. In: *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, 2006, S. 137–139

[CKSV08] Cammert, Michael ; Krämer, Jürgen ; Seeger, Bernhard ; Vaupel, Sonny: A Cost-Based Approach to Adaptive Resource Management in Data Stream Systems. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 20 (2008), Nr. 2, S. 230–245

[Cod70] Codd, E. F.: A Relational Model of Data for Large Shared Data Banks. In: *Commun. ACM* 13 (1970), Nr. 6, S. 377–387

[Cod85a] Codd, Edgar F.: Does Your DBMS Run by the Rules? In: *Computer World* (1985), October

[Cod85b] Codd, Edgar F.: Is Your DBMS Really Relational? In: *Computer World* (1985), October

[CT85] Clifford, James ; Tansel, Abdullah U.: On An Algebra For Historical Relational Databases: Two Views. In: *SIGMOD Conference*, 1985, S. 247–265

[Des04] Deshpande, Amol: An initial study of overheads of eddies. In: *SIGMOD Record* 33 (2004), Nr. 1, S. 44–49

[DGK82] Dayal, Umeshwar ; Goodman, Nathan ; Katz, Randy H.: An Extended Relational Algebra with Control Over Duplicate Elimination. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1982. – ISBN 0–89791–070–2, S. 117–123

[DGP+07] Demers, Alan J. ; Gehrke, Johannes ; Panda, Biswanath ; Riedewald, Mirek ; Sharma, Varun ; White, Walker M.: Cayuga: A General Purpose Event Monitoring System. In: *Proc. of the Conf. on Innovative Data Systems Research (CIDR)*, 2007, S. 412–422

[DGR03] Das, Abhinandan ; Gehrke, Johannes ; Riedewald, Mirek: Approximate Join Processing over Data Streams. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2003, S. 40–51

[Din08] Ding, Luping: *Metadata-Aware Query Processing over Data Streams*, Worcester Polytechnic Institute, Diss., 2008

[DMRH04] Ding, Luping ; Mehta, Nishant ; Rundensteiner, Elke A. ; Heineman, George T.: Joining Punctuated Streams. In: *Proc. of the Int. Conf. on Extending Data Base Technology (EDBT)*, 2004, S. 587–604

[DR04] Ding, Luping ; Rundensteiner, Elke A.: Evaluating window joins over punctuated streams. In: *CIKM*, 2004, S. 98–107

[DRH03] Ding, Luping ; Rundensteiner, Elke A. ; Heineman, George T.: MJoin: a metadata-aware stream join operator. In: *DEBS*, 2003

[DS93] Dyreson, Curtis E. ; Snodgrass, Richard T.: Timestamp semantics and representation. In: *Inf. Syst.* 18 (1993), Nr. 3, S. 143–166

[DSTW02] Dittrich, Jens-Peter ; Seeger, Bernhard ; Taylor, David S. ; Widmayer, Peter: Progressive Merge Join: A Generic and Non-blocking Sort-based Join Algorithm. In: *Proc. of the VLDB Endowment (PVLDB)*, 2002, S. 299–310

[DSTW03] Dittrich, Jens-Peter ; Seeger, Bernhard ; Taylor, David S. ; Widmayer, Peter: On producing join results early. In: *Symp. on Principles of Database Systems (PODS)*, 2003, S. 134–142

[Ede80] Edelsbrunner, Herbert: Dynamic data structures for orthogonal intersection queries / Institute for Information Processing, Technical University of Graz, Graz. 1980. – Technical report

[EEVK14] Elseidy, Mohammed ; Elguindy, Abdallah ; Vitorovic, Aleksandar ; Koch, Christoph: Scalable and Adaptive Online Joins. In: *PVLDB* 7 (2014), Nr. 6, S. 441–452

[EFP06] Eurviriyanukul, Kwanchai ; Fernandes, Alvaro A. A. ; Paton, Norman W.: A Foundation for the Replacement of Pipelined Physical Join Operators in Adaptive Query Processing. In: *EDBT Workshops*, 2006, S. 589–600

[EPFL10] Eurviriyanukul, Kwanchai ; Paton, Norman W. ; Fernandes, Alvaro A. A. ; Lynden, Steven J.: Adaptive join processing in pipelined plans. In: *EDBT*, 2010, S. 183–194

[FH07] Farag, Fatima ; Hammad, Moustafa A.: Adaptive Execution of Stream Window Joins in a Limited Memory Environment. In: *IDEAS*, 2007, S. 12–20

[Gad86] Gadia, Shashi K.: Weak Temporal Relations. In: *Symp. on Principles of Database Systems (PODS)*, 1986, S. 70–77

[GBY09] Gedik, Bugra ; Bordawekar, Rajesh ; Yu, Philip S.: CellJoin: a parallel stream join operator for the cell processor. In: *VLDB J.* 18 (2009), Nr. 2, S. 501–519

[GC06] Gomes, Joseph S. ; Choi, Hyeong-Ah: Finding Optimal Join Tree forMulti-Join Stream Queries in a Production System. In: *ICDCS Workshops*, 2006, S. 27

[GC08] Gomes, Joseph S. ; Choi, Hyeong-Ah: Adaptive optimization of join trees for multi-join queries over sensor streams. In: *Information Fusion* 9 (2008), Nr. 3, S. 412–424

[GELA10] Ghanem, Thanaa M. ; Elmagarmid, Ahmed K. ; Larson, Per-Åke ; Aref, Walid G.: Supporting views in data stream management systems. In: *ACM Trans. Database Syst.* 35 (2010), Nr. 1

[GG98] Gaede, Volker ; Günther, Oliver: Multidimensional access methods. In: *ACM Computing Surveys* 30 (1998), Nr. 2, S. 170–231

[GHJV95] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1995. – ISBN 0201633612

[GHM+07] Ghanem, Thanaa M. ; Hammad, Moustafa A. ; Mokbel, Mohamed F. ; Aref, Walid G. ; Elmagarmid, Ahmed K.: Incremental Evaluation of Sliding-Window Queries over Data Streams. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 19 (2007), Nr. 1, S. 57–72

[GJSS05] Gao, Dengfeng ; Jensen, Christian S. ; Snodgrass, Richard T. ; Soo, Michael D.: Join Operations in Temporal Databases. In: *VLDB Journal* 14 (2005), Nr. 1, S. 2–29

[GÖ03a] Golab, Lukasz ; Öszu, M. Tamer: Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams. In: *Proc. of the VLDB Endowment (PVLDB)*, 2003, S. 500–511

[GÖ03b] Golab, Lukasz ; Özsu, M. Tamer: Issues in Data Stream Management. In: *SIGMOD Record* 32 (2003), Nr. 2, S. 5–14

[GÖ05] Golab, Lukasz ; Özsu, M. Tamer: Update-Pattern-Aware Modeling and Processing of Continuous Queries. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2005, S. 658–669

[Gra06] Graefe, Goetz: Implementing sorting in database systems. In: *ACM Comput. Surv.* 38 (2006), Nr. 3

[GS91] Gunadhi, Himawan ; Segev, Arie: Query Processing Algorithms for Temporal Intersection Joins. In: *ICDE*, 1991, S. 336–344

[GUW00] Garcia-Molina, Hector ; Ullman, Jeffrey D. ; Widom, Jennifer: *Database System Implementation*. Prentice Hall, 2000

[GWYL05] Gedik, Bugra ; Wu, Kun-Lung ; Yu, Philip S. ; Liu, Ling: Adaptive Load Shedding for Windowed Stream Joins. In: *Proc. of the Int. Conf. on Information and Knowledge Management (CIKM)*, 2005, S. 171–178

[GWYL07] Gedik, Bugra ; Wu, Kun-Lung ; Yu, Philip S. ; Liu, Ling: GrubJoin: An Adaptive, Multi-Way, Windowed Stream Join with Time Correlation-Aware CPU Load Shedding. In: *IEEE Trans. Knowl. Data Eng.* 19 (2007), Nr. 10, S. 1363–1380

[GYB07] Gedik, Bugra ; Yu, Philip S. ; Bordawekar, Rajesh: Executing Stream Joins on the Cell Processor. In: *VLDB*, 2007, S. 363–374

[HAE03] Hammad, Moustafa A. ; Aref, Walid G. ; Elmagarmid, Ahmed K.: Stream Window Join: Tracking Moving Objects in Sensor-Network Databases. In: *Proc. of the Int. Conf. on Scientific and Statistical Database Management (SSDBM)*, 2003, S. 75–84

[Hei07] Heinz, Christoph: *Density Estimation over Data Streams*, Philipps-Universität Marburg, Diss., 2007

[HFS12] Hossbach, Bastian ; Freisleben, Bernd ; Seeger, Bernhard: Reaktives Cloud Monitoring mit Complex Event Processing. In: *Datenbank-Spektrum* 12 (2012), Nr. 1, S. 33–42

[HH99] Haas, Peter J. ; Hellerstein, Joseph M.: Ripple Joins for Online Aggregation. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1999, S. 287–298

[HKRS07] Heinz, Christoph ; Krämer, Jürgen ; Riemenschneider, Tobias ; Seeger, Bernhard: Auf dem Weg zur allwissenden Fabrik: Vertikale Integration auf Basis kontinuierlicher Datenverarbeitung. In: *INFORMATIK 2007: Informatik trifft Logistik. Band 2. Beiträge der 37. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 24.-27. September 2007 in Bremen*, 2007, S. 339–344

[HKRS08] Heinz, Christoph ; Krämer, Jürgen ; Riemenschneider, Tobias ; Seeger, Bernhard: Toward Simulation-Based Optimization in Data Stream Management Systems. In: *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, 2008, S. 1580–1583

[HKSZ05] Heinz, Christoph ; Krämer, Jürgen ; Seeger, Bernhard ; Zeiss, Alexander: Datenstromverarbeitung in Production Intelligence Software. In: *Workshop der GI-Fachgruppe „Datenbanken“ zum Thema „Business Intelligence“*, 2005

[HMA+04] Hammad, Moustafa A. ; Mokbel, Mohamed F. ; Ali, Mohamed H. ; Aref, Walid G. ; Catlin, Ann C. ; Elmagarmid, Ahmed K. ; Eltabakh, Mohamed Y. ; Elfeky, Mohamed G. ; Ghanem, Thanaa M. ; Gwadera, R. ; Ilyas, Ihab F. ;

Marzouk, Mirette S. ; Xiong, Xiaopeng: Nile: A Query Processing Engine for Data Streams. In: *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, 2004, S. 851

[Hoa62] Hoare, C. A. R.: Quicksort. In: *Comput. J.* 5 (1962), Nr. 1, S. 10–15

[HS05] Heinz, Christoph ; Seeger, Bernhard: Wavelet Density Estimators over Data Streams. In: *Proc. of the ACM Symposium on Applied Computing (SAC)*, 2005, S. 578–579

[HS06a] Heinz, Christoph ; Seeger, Bernhard: Exploring Data Streams with Non-parametric Estimators. In: *18th International Conference on Scientific and Statistical Database Management, SSDBM 2006, 3-5 July 2006, Vienna, Austria, Proceedings*, 2006, S. 261–264

[HS06b] Heinz, Christoph ; Seeger, Bernhard: Resource-aware Kernel Density Estimators over Streaming Data. In: *Proc. of the Int. Conf. on Information and Knowledge Management (CIKM)*, 2006, S. 870–871

[HS06c] Heinz, Christoph ; Seeger, Bernhard: Stream Mining via Density Estimators: A concrete Application. In: *Proc. of the Int. Conf. on Management of Data (COMAD)*, 2006

[HS06d] Heinz, Christoph ; Seeger, Bernhard: Towards Kernel Density Estimation over Streaming Data. In: *Proceedings of the 13th International Conference on Management of Data, December 14-16, 2006, Delhi, India*, 2006, S. 80–91

[HS07] Heinz, Christoph ; Seeger, Bernhard: Adaptive Wavelet Density Estimators over Data Streams. In: *19th International Conference on Scientific and Statistical Database Management, SSDBM 2007, 9-11 July 2007, Banff, Canada, Proceedings*, 2007, S. 35

[IFF+99] Ives, Zachary G. ; Florescu, Daniela ; Friedman, Marc ; Levy, Alon ; Weld, Daniel S.: An Adaptive Query Execution System for Data Integration. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1999, S. 299–310

[JC04] Jiang, Qingchun ; Chakravarthy, Sharma: Scheduling Strategies for Processing Continuous Queries over Streams. In: *Proc. of the British National Conf. on Databases (BNCOD)*, 2004, S. 16–30

[JDA+05] Jermaine, Chris ; Dobra, Alin ; Arumugam, Subramanian ; Joshi, Shantanu ; Pol, Abhijit: A Disk-Based Join With Probabilistic Guarantees. In: *SIGMOD Conference*, 2005, S. 563–574

[JDA+06] Jermaine, Chris ; Dobra, Alin ; Arumugam, Subramanian ; Joshi, Shantanu ; Pol, Abhijit: The Sort-Merge-Shrink join. In: *ACM Trans. Database Syst.* 31 (2006), Nr. 4, S. 1382–1416

[JM80] Jones, S. ; Mason, P. J.: Handling the Time Dimension in a Data Base. In: *ICOD*, 1980, S. 65–83

[JMSS05] Johnson, Theodore ; Muthukrishnan, S. ; Shkapenyuk, Vladislav ; Spatscheck, Oliver: A Heartbeat Mechanism and Its Application in Gigascope. In: *Proc. of the VLDB Endowment (PVLDB)*, 2005, S. 1079–1088

[JSS94] Jensen, Christian S. ; Soo, Michael D. ; Snodgrass, Richard T.: Unifying Temporal Data Models via a Conceptual Model. In: *Inf. Syst.* 19 (1994), Nr. 7, S. 513–547

[KCC+03] Krishnamurthy, Sailesh ; Chandrasekaran, Sirish ; Cooper, Owen ; Deshpande, Amol ; Franklin, Michael J. ; Hellerstein, Joseph M. ; Hong, Wei ; Madden, Samuel ; Reiss, Frederick ; Shah, Mehul A.: TelegraphCQ: An Architectural Status Report. In: *IEEE Data Eng. Bull.* 26 (2003), Nr. 1, S. 11–18

[KD98] Kabra, Navin ; DeWitt, David J.: Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1998, S. 106–117

[KDH+07] Klein, Anja ; Do, Hong H. ; Hackenbroich, Gregor ; Karnstedt, Marcel ; Lehner, Wolfgang: Representing Data Quality for Streaming and Static Data. In: *ICDE Workshops*, 2007, S. 3–10

[KKKK02] Keidl, Markus ; Kreutz, Alexander ; Kemper, Alfons ; Kossmann, Donald: A Publish & Subscribe Architecture for Distributed Metadata Management. In: *ICDE*, 2002, S. 309–

[Knu73] Knuth, Donald E.: *The Art of Computer Programming*. Bd. III: Sorting and Searching. Addison-Wesley, 1973

[KNV03] Kang, J. ; Naughton, J. ; Viglas, S.: Evaluating Window Joins over Unbounded Streams. In: *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, 2003, S. 341–352

[Krä07] Krämer, Jürgen: *Continuous Queries over Data Streams – Semantics and Implementation*, Philipps-Universität Marburg, Diss., 2007

[KS04] Krämer, Jürgen ; Seeger, Bernhard: PIPES - A Public Infrastructure for Processing and Exploring Streams. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2004, S. 925–926

[KS05] Krämer, Jürgen ; Seeger, Bernhard: A Temporal Foundation for Continuous Queries over Data Streams. In: *Proc. of the Int. Conf. on Management of Data (COMAD)*, 2005, S. 70–82

[KS09] Krämer, Jürgen ; Seeger, Bernhard: Semantics and implementation of continuous sliding window queries over data streams. In: *ACM Trans. Database Syst.* 34 (2009), Nr. 1

[KSKR05] Kuntschke, Richard ; Stegmaier, Bernhard ; Kemper, Alfons ; Reiser, Angelika: StreamGlobe: Processing and Sharing Data Streams in Grid-Based P2P Infrastructures. In: *Proc. of the VLDB Endowment (PVLDB)*, 2005, S. 1259–1262

[KSPL06] Krämer, Jürgen ; Seeger, Bernhard ; Penzel, Thomas ; Lenz, Richard: $PIPES_{med}$: Ein flexibles Werkzeug zur Verarbeitung kontinuierlicher Datenströme in der Medizin. In: *51. Jahrestagung der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie (GMDS)*, 2006

[KYC+06] Krämer, Jürgen ; Yang, Yin ; Cammert, Michael ; Seeger, Bernhard ; Papadias, Dimitris: Dynamic Plan Migration for Snapshot-Equivalent Continuous Queries in Data Stream Systems. In: *Proc. of the Int. Conf. on Extending Data Base Technology (EDBT) Workshops*, 2006, S. 497–516

[Las61] Lasser, Daniel J.: Topological ordering of a list of randomly-numbered elements of a network. In: *Commun. ACM* 4 (1961), Nr. 4, S. 167–168

[Law05] Lawrence, Ramon: Early Hash Join: A Configurable Algorithm for the Efficient and Early Production of Join Results. In: *VLDB*, 2005, S. 841–852

[LCT+06] Li, Hua-Gang ; Chen, Songting ; Tatemura, Jun'ichi ; Agrawal, Divyakant ; Candan, K. S. ; Hsiung, Wang-Pin: Safety Guarantee of Continuous Join Queries over Punctuated Data Streams. In: *VLDB*, 2006, S. 19–30

[LG98] Larson, Per-Åke ; Graefe, Goetz: Memory Management During Run Generation in External Sorting. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1998, S. 472–483

[LGS02] Li, Wei ; Gao, Dengfeng ; Snodgrass, Richard T.: Skew handling techniques in sort-merge join. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2002. – ISBN 1–58113–497–5, S. 169–180

[LKM10] Levandoski, Justin K. ; Khalefa, Mohamed E. ; Mokbel, Mohamed F.: On Producing High and Early Result Throughput in Multi-join Query Plans. In: *IEEE Transactions on Knowledge and Data Engineering* 99 (2010), Nr. PrePrints. – ISSN 1041–4347

[LM93] Leung, T. Y. C. ; Muntz, Richard R.: Stream Processing: Temporal Query Processing and Optimization. In: *Temporal Databases*. Benjamin-Cummings Publishing Co., Inc., 1993, S. 329–355

[LMT+05] Li, Jin ; Maier, David ; Tufte, Kristin ; Papadimos, Vassilis ; Tucker, Peter A.: Semantics and Evaluation Techniques for Window Aggregates in Data Streams. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2005, S. 311–322

[LOT94] Lu, Hongjun ; Ooi, Beng C. ; Tan, Kian-Lee: On Spatially Partitioned Temporal Join. In: *VLDB*, 1994, S. 546–557

[LR96] Lo, Ming-Ling ; Ravishankar, Chinya V.: Spatial Hash-Joins. In: *SIGMOD Conference*, 1996, S. 247–258

[LSM+07] Li, Quanzhong ; Shao, Minglong ; Markl, Volker ; Beyer, Kevin S. ; Colby, Latha S. ; Lohman, Guy M.: Adaptively Reordering Joins during Query Execution. In: *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, 2007, S. 26–35

[LTS+08] Li, Jin ; Tufte, Kristin ; Shkapenyuk, Vladislav ; Papadimos, Vassilis ; Johnson, Theodore ; Maier, David: Out-of-order processing: a new architecture for high-performance stream systems. In: *PVLDB* 1 (2008), Nr. 1, S. 274–288

[LVP06a] Liu, Ying ; Vijayakumar, Nithya N. ; Plale, Beth: Stream processing in data-driven computational science. In: *GRID*, 2006, S. 160–167

[LVP06b] Liu, Ying ; Vijayakumar, Nithya N. ; Plale, Beth: Stream processing in data-driven computational science. In: *GRID*, 2006, S. 160–167

[MJIG10] Mihaylov, Svilen R. ; Jacob, Marie ; Ives, Zachary G. ; Guha, Sudipto: Dynamic Join Optimization in Multi-Hop Wireless Sensor Networks. In: *PVLDB* 3 (2010), Nr. 1, S. 1279–1290

[MK09] Mishra, Chaitanya ; Koudas, Nick: Join Reordering by Join Simulation. In: *ICDE*, 2009, S. 493–504

[MLA04] Mokbel, Mohamed F. ; Lu, Ming ; Aref, Walid G.: Hash-Merge Join: A Non-blocking Join Algorithm for Producing Fast and Early Join Results. In: *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, 2004, S. 251–263

[MPDSLP10] Martinez-Palau, Xavier ; Dominguez-Sal, David ; Larriba-Pey, Josep-Lluis: Two-way Replacement Selection. In: *PVLDB* 3 (2010), Nr. 1, S. 871–881

[MRS+04] Markl, Volker ; Raman, Vijayshankar ; Simmen, David E. ; Lohman, Guy M. ; Pirahesh, Hamid: Robust Query Processing through Progressive Optimization. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2004, S. 659–670

[MRW11] Mukherji, Abhishek ; Rundensteiner, Elke A. ; Ward, Matthew O.: Achieving High Freshness and Optimal Throughput in CPU-Limited Execution of Multi-join Continuous Queries. In: *BNCOD*, 2011, S. 48–65

[MWA+03] Motwani, Rajeev ; Widom, Jennifer ; Arasu, Arvind ; Babcock, Brian ; Babu, Shivnath ; Datar, Mayur ; Manku, Gurmeet ; Olsten, Chris ; Rosenstein, Justin ; Varma, Rohit: Query Processing, Resource Management, and Approximation in a Data Stream Management System. In: *Proc. of the Conf. on Innovative Data Systems Research (CIDR)*, 2003

[MWL06] Mok, Aloysius K. ; Woo, Honguk ; Lee, Chan-Gun: Probabilistic Timing Join over Uncertain Event Streams. In: *RTCSA*, 2006, S. 17–26

[NDM+01] Naughton, Jeffrey F. ; DeWitt, David J. ; Maier, David ; Aboulnaga, Ashraf ; Chen, Jianjun ; Galanis, Leonidas ; Kang, Jaewoo ; Krishnamurthy, Rajasekar ; Luo, Qiong ; Prakash, Naveen ; Ramamurthy, Ravishankar ; Shanmugasundaram, Jayavel ; Tian, Feng ; Tufte, Kristin ; Viglas, Stratis ; Wang, Yuan ; Zhang, Chun ; Jackson, Bruce ; Gupta, Anurag ; Chen, Rushan: The Niagara Internet Query System. In: *IEEE Data Engineering Bulletin* 24 (2001), Nr. 2, S. 27–33

[NP82] Nievergelt, J. ; Preparata, F. P.: Plane-Sweep Algorithms for Intersecting Geometric Figures. In: *Communications of the ACM* 25 (1982), Nr. 10, S. 739–747

[NP85] Negri, Mauro ; Pelagatti, Giuseppe: Join During Merge: An Improved Sort Based Algorithm. In: *Inf. Process. Lett.* 21 (1985), Nr. 1, S. 11–16

[OBS99] Olson, Michael A. ; Bostic, Keith ; Seltzer, Margo I.: Berkeley DB. In: *USENIX Annual Technical Conference, FREENIX Track*, 1999, S. 183–191

[PAPV08] Palma, Wenceslao ; Akbarinia, Reza ; Pacitti, Esther ; Valduriez, Patrick: Efficient Processing of Continuous Join Queries Using Distributed Hash Tables. In: *Euro-Par*, 2008, S. 632–641

[PAPV09a] Palma, Wenceslao ; Akbarinia, Reza ; Pacitti, Esther ; Valduriez, Patrick: DHTJoin: processing continuous join queries using DHT networks. In: *Distributed and Parallel Databases* 26 (2009), Nr. 2-3, S. 291–317

[PAPV09b] Palma, Wenceslao ; Akbarinia, Reza ; Pacitti, Esther ; Valduriez, Patrick: Distributed processing of continuous join queries using DHT networks. In: *EDBT/ICDT Workshops*, 2009, S. 34–41

[PCR01] Page, Kevin R. ; Cruickshank, Don ; Roure, David D.: Its about time: link streams as continuous metadata. In: *Hypertext*, 2001, S. 93–102

[PD96] Patel, Jignesh M. ; DeWitt, David J.: Partition Based Spatial-Merge Join. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1996, S. 259–270

[PLS+06] Pietzuch, Peter R. ; Ledlie, Jonathan ; Shneidman, Jeffrey ; Roussopoulos, Mema ; Welsh, Matt ; Seltzer, Margo I.: Network-Aware Operator Placement for Stream-Processing Systems. In: *ICDE*, 2006, S. 49

[PS03] Plale, Beth ; Schwan, Karsten: Dynamic Querying of Streaming Data with the dQUOB System. In: *IEEE Trans. Parallel Distrib. Syst.* 14 (2003), Nr. 4, S. 422–432

[RDH03] Raman, Vijayshankar ; Deshpande, Amol ; Hellerstein, Joseph M.: Using State Modules for Adaptive Query Processing. In: *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, 2003, S. 353

[RDS+04] Rundensteiner, Elke A. ; Ding, Luping ; Sutherland, Timothy M. ; Zhu, Yali ; Pielech, Bradford ; Mehta, Nishant: CAPE: Continuous Query Engine with Heterogeneous-Grained Adaptivity. In: *Proc. of the VLDB Endowment (PVLDB)*, 2004, S. 1353–1356

[Rie08] Riemenschneider, Tobias: *Optimierung kontinuierlicher Anfragen auf Basis statistischer Metadaten*, Philipps-Universität Marburg, Diss., 2008

[RRH99] Raman, Vijayshankar ; Raman, Bhaskaran ; Hellerstein, Joseph M.: Online Dynamic Reordering for Interactive Data Processing. In: Atkinson, Malcolm P. (Hrsg.) ; Orlowska, Maria E. (Hrsg.) ; Valduriez, Patrick (Hrsg.) ; Zdonik, Stanley B. (Hrsg.) ; Brodie, Michael L. (Hrsg.): *Proc. of the VLDB Endowment (PVLDB)*, 1999, S. 709–720

[RTG14] Roy, Pratanu ; Teubner, Jens ; Gemulla, Rainer: Low-Latency Handshake Join. In: *PVLDB* 7 (2014), Nr. 9, S. 709–720

[SAC+79] Selinger, Patricia G. ; Astrahan, Morton M. ; Chamberlin, Donald D. ; Lorie, Raymond A. ; Price, Thomas G.: Access Path Selection in a Relational Database Management System. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1979, S. 23–34

[SC75] Smith, John M. ; Chang, Philip Yen-Tang: Optimizing the performance of a relational algebra database interface. In: *Communications of the ACM* 18 (1975), October, S. 568–579. – ISSN 0001–0782

[SFL05] Schmidt, Sven ; Fiedler, Marc ; Lehner, Wolfgang: Source-aware Join Strategies of Sensor Data Streams. In: *SSDBM*, 2005, S. 123–132

[SG89] Segev, Arie ; Gunadhi, Himawan: Event-Join Optimization in Temporal Relational Databases. In: *Proc. of the VLDB Endowment (PVLDB)*, 1989, S. 205–215

[SHCF03] Shah, Mehul A. ; Hellerstein, Joseph M. ; Chandrasekaran, Sirish ; Franklin, Michael J.: Flux: An Adaptive Partitioning Operator for Continuous Query Systems. In: *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, 2003, S. 25–36

[SK04] Stegmaier, Bernhard ; Kuntschke, Richard: StreamGlobe: Adaptive Anfragebearbeitung und Optimierung auf Datenströmen. In: *GI Jahrestagung (1)*, 2004, S. 367–372

[SKK04] Stegmaier, Bernhard ; Kuntschke, Richard ; Kemper, Alfons: StreamGlobe: adaptive query processing and optimization in streaming P2P environments. In: *DMSN*, 2004, S. 88–97

[SLJR05] Sutherland, Timothy M. ; Liu, Bin ; Jbantova, Mariana ; Rundensteiner, Elke A.: D-CAPE: Distributed and Self-tuned Continuous Query Processing. In: *Proc. of the Int. Conf. on Information and Knowledge Management (CIKM)*, 2005, S. 217–218

[SLMK01] Stillger, Michael ; Lohman, Guy M. ; Markl, Volker ; Kandil, Mokhtar: LEO - DB2's LEarning Optimizer. In: *Proc. of the VLDB Endowment (PVLDB)*, 2001, S. 19–28

[SMFH01] Shah, Mehul A. ; Madden, Samuel ; Franklin, Michael J. ; Hellerstein, Joseph M.: Java Support for Data-Intensive Systems: Experiences Building the Telegraph Dataflow System. In: *SIGMOD Record* 30 (2001), Nr. 4, S. 103–114

[Sno87] Snodgrass, Richard T.: The Temporal Query Language TQuel. In: *ACM Trans. Database Syst.* 12 (1987), Nr. 2, S. 247–298

[SQR03] *SQR – A Stream Query Repository*. http://www.db.stanford.edu/stream/sqr, 2003. – Joint Effort of Several Data Stream Research Groups

[SSJ94] Soo, Michael D. ; Snodgrass, Richard T. ; Jensen, Christian S.: Efficient Evaluation of the Valid-Time Natural Join. In: *ICDE*, 1994, S. 282–292

[SW04a] Srivastava, Utkarsh ; Widom, Jennifer: Flexible Time Management in Data Stream Systems. In: *Symp. on Principles of Database Systems (PODS)*, 2004, S. 263–274

[SW04b] Srivastava, Utkarsh ; Widom, Jennifer: Memory-Limited Execution of Windowed Stream Joins. In: *Proc. of the VLDB Endowment (PVLDB)*, 2004, S. 324–335

[TCG+93] Tansel, Abdullah U. (Hrsg.) ; Clifford, James (Hrsg.) ; Gadia, Shashi K. (Hrsg.) ; Jajodia, Sushil (Hrsg.) ; Segev, Arie (Hrsg.) ; Snodgrass, Richard T. (Hrsg.): *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings, 1993. – ISBN 0–8053–2413–5

[TCZ+03] Tatbul, Nesime ; Cetintemel, Ugur ; Zdonik, Stanley B. ; Cherniack, Mitch ; Stonebraker, Michael: Load Shedding in a Data Stream Manager. In: *Proc. of the VLDB Endowment (PVLDB)*, 2003, S. 309–320

[TM02] Tucker, Peter A. ; Maier, David: Exploiting Punctuation Semantics in Data Streams. In: *ICDE*, 2002, S. 279

[TM11] Teubner, Jens ; Müller, René: How soccer players would do stream joins. In: *SIGMOD Conference*, 2011, S. 625–636

[TMSF03] Tucker, Peter A. ; Maier, David ; Sheard, Tim ; Fegaras, Leonidas: Exploiting Punctuation Semantics in Continuous Data Streams. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 15 (2003), Nr. 3, S. 555–568

[TMZ08] Thakkar, Hetal ; Mozafari, Barzan ; Zaniolo, Carlo: Designing an inductive data stream management system: the stream mill experience. In: *SSPS*, 2008, S. 79–88

[Tuc05] Tucker, Peter A.: *Punctuated Data Streams*, Whitworth University, Diss., 2005

[TYP+05] Tao, Yufei ; Yiu, Man L. ; Papadias, Dimitris ; Hadjieleftheriou, Marios ; Mamoulis, Nikos: RPJ: Producing Fast Join Results on Streams through Rate-based Optimization. In: *SIGMOD Conference*, 2005, S. 371–382

[UF00] Urhan, Tolga ; Franklin, Michael J.: XJoin: A Reactively-Scheduled Pipelined Join Operator. In: *IEEE Data Engineering Bulletin* 23 (2000), Nr. 2, S. 27–33

[UF01] Urhan, Tolga ; Franklin, Michael J.: Dynamic Pipeline Scheduling for Improving Interactive Query Performance. In: *Proc. of the VLDB Endowment (PVLDB)*, 2001, S. 501–510

[VLP06] Vijayakumar, Nithya N. ; Liu, Ying ; Plale, Beth: Calder Query Grid Service: Insights and Experimental Evaluation. In: *CCGRID*, 2006, S. 539–543

[VN02] Viglas, Stratis D. ; Naughton, Jeffrey F.: Rate-based Query Optimization for Streaming Information Sources. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2002, S. 37–48

[VNB03] Viglas, Stratis D. ; Naughton, Jeffrey F. ; Burger, Josef: Maximizing the Output Rate of Multi-Join Queries over Streaming Information Sources. In: *Proc. of the VLDB Endowment (PVLDB)*, 2003, S. 285–296

[WA90] Wilschut, A. N. ; Apers, P. M. G.: Pipelining in query execution. In: *Proceedings of the International Conference on Databases, Parallel Architectures and Their Applications (PARBASE 1990), Miami Beach, FL, USA*, 1990, S. 562–562

[WA91] Wilschut, Annita N. ; Apers, Peter M. G.: Dataflow Query Execution in a Parallel Main-Memory Environment. In: *Proceedings of the First International Conference on Parallel and Distributed Information Systems (PDIS 1991)*, IEEE Computer Society, 1991, S. 68–77

[WR09] Wang, Song ; Rundensteiner, Elke A.: Scalable stream join processing with expensive predicates: workload distribution and adaptation by time-slicing. In: *EDBT*, 2009, S. 299–310

[WTZ07] Wu, Ji ; Tan, Kian-Lee ; Zhou, Yongluan: Window-Oblivious Join: A Data-Driven Memory Management Scheme for Stream Join. In: *SSDBM*, 2007, S. 21

[XY07] Xie, Junyi ; Yang, Jun: A Survey of Join Processing in Data Streams. In: *Data Streams - Models and Algorithms*. 2007, S. 209–236

[YKPS07] Yang, Yin ; Krämer, Jürgen ; Papadias, Dimitris ; Seeger, Bernhard: HybMig: A Hybrid Approach to Dynamic Plan Migration for Continuous Queries.

In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 19 (2007), Nr. 3, S. 398–411

[ZSC+03] Zdonik, Stanley B. ; Stonebraker, Michael ; Cherniack, Mitch ; Çetintemel, Ugur ; Balazinska, Magdalena ; Balakrishnan, Hari: The Aurora and Medusa Projects. In: *IEEE Data Eng. Bull.* 26 (2003), Nr. 1, S. 3–10

[ZTZ06] Zhou, Xin ; Thakkar, Hetal ; Zaniolo, Carlo: Unifying the Processing of XML Streams and Relational Data Streams. In: *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, 2006, S. 50

Lebenslauf

Kontaktdaten Michael Cammert
Sonnenstraße 1
35435 Wettenberg
Email: cammert@mathematik.uni-marburg.de

Persönliche Daten

Geburtstag: 19.03.1975
Geburtsort: Gießen
Staatsangehörigkeit: Deutsch
Familienstand: Ledig

Ausbildung

1995 – 2002	*Studium der Informatik* Philipps-Universität Marburg
1994	*Abitur* Herderschule Gießen

Beruflicher Werdegang

seit 2013	*Software Engineer Specialist* Software AG, Marburg
2009 – 2012	*Softwareentwickler* RTM Realtime Monitoring GmbH, Marburg
2003 – 2008	*Wissenschaftlicher Mitarbeiter* Fachbereich Mathematik und Informatik, Philipps-Universität Marburg
2003	*Wissenschaftliche Hilfskraft mit Abschluß* Philipps-Universität Marburg (10 Monate)
1999	*Praktikum* media[netCom], Marburg (6 Wochen)
1998 – 2001	*Studentische Hilfskraft* Philipps-Universität Marburg (22 Monate)

Wehrdienst

1994 – 1995	*Grundwehrdienst*

Wettenberg, 22. Oktober 2014