Sven Schmidt, Thomas Legler, Sebastian Schär, Wolfgang Lehner

Robust Real-time Query Processing with QStream

Erstveröffentlichung in / First published in:

ICMI05: Seventh International Conference on Multimodal Interfaces 2005. Trondheim, 30. August – 2. September 2005, S. 1299–1301. ACM. ISBN 978-1-59593-154-2.

Link: https://dl.acm.org/doi/10.5555/1083592.1083756

ICMI05: Seventh International Conference on Multimodal Interfaces 2005", S. 1299–1301, ISBN 978-1-59593-154-2
https://dl.acm.org/doi/10.5555/1083592.1083756

Robust Real-time Query Processing with QStream

Sven Schmidt Thomas Legler Sebastian Schär Wolfgang Lehner

Dresden University of Technology
Database Technology Group
dbgroup@mail.inf.tu-dresden.de

Abstract

Processing data streams with Quality-of-Service (QoS) guarantees is an emerging area in existing streaming applications. Although it is possible to negotiate the result quality and to reserve the required processing resources in advance, it remains a challenge to adapt the DSMS to data stream characteristics which are not known in advance or are difficult to obtain. Within this paper we present the second generation of our QStream DSMS which addresses the above challenge by using a real-time capable operating system environment for resource reservation and by applying an adaptation mechanism if the data stream characteristics change spontaneously.

1 Introduction

Currently, many existing Data Stream Management Systems (DSMS), such as [2, 3, 5], aim at an efficient, flexible, and sometimes also distributed evaluation of standing queries against streaming data. Often, it is very valuable for the user if the DSMS assures a certain Quality-of-Service (QoS) for query processing. For example, such QoS requirements may include upper bounds for the *tuple processing delay* or a guaranteed *result data rate*. In [5], we presented an approach of running a DSMS on top of a real-time- capable operating system to meet the QoS requirements mentioned above. Our work was based on the calculation and reservation of necessary resources in advance. However, the main assumptions of our real-time DSMS approach were that the data-dependent parameters (e.g. data rate, data distribution) are known a-priori and, moreover, are constant on average. Only a cumulatively limited jitter of processing time or data amount was allowed and thus, had influence on the resource calculation and reservation process. It is obvious that these assumptions do not hold for many application scenarios, and thus, an extension towards a more flexible management regarding the data-dependent parameters is required. With this second QStream prototype, we establish a so-called *adaptation framework* on the basis of the real-time-capable and QoS-guaranteeing system core. The benefits are twofold: on the one hand, we are able to handle arbitrary input data streams, whether or not the stream characteristics are known a-priori. On the other hand, we enable the user to adjust the *robustness* of the DSMS by granting more or less resources. As a result, processing of arbitrary input streams may be achieved in real-time, but will be interrupted from time to time to adapt the system to changes in the streaming environment.

2 Adaptation Framework

The general way of resource reservation based on QoS requirements is illustrated in figure 1. First of all, the QoS requirements are negotiated. Then, if the calculated resources are available, they are reserved and guaranteed during runtime. With the resource guarantee, the quality requirements are guaranteed as well.

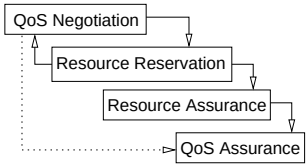

Figure 1: QoS negotiation process

For our work, we make use of a resource calculation model which is extensively described in [1]. We implement a standing query as a network of independently running basic operators. For optimum resource usage, and in order to enable QoS guarantees, we ad-

ICMI05: Seventh International Conference on Multimodal Interfaces 2005", S. 1299–1301, ISBN 978-1-59593-154-2
https://dl.acm.org/doi/10.5555/1083592.1083756

just the 'speed' of this operator network based on the data throughput requirements. The data throughput is determined firstly, by the individual period length of an operator's *run* (called frequency) and secondly, by the amount of data which is processed during one run. For successful operation, a FIFO buffer is established between each pair of operators.

DSMS Steady State

Once the resource-determining parameters (like input data rate, selectivity of filter and join operators, etc.) are known or estimated, we schedule the DSMS operators to work in real-time at the presumed network speed, thereby achieving the desired data throughput as well as the maximum tuple delay. As this scheduling is based on average parameters, these parameters (mainly the data rate, which we will focus on) may jitter around their average. It is up to the user to limit the amount of jitter while calculating necessary resources. Toleration of more jitter leads to higher intermediate buffer sizes, a longer tuple delay and higher resource requirements. We suppose the DSMS to run in a *steady state* as long as tolerated jitter is sufficient for the fluctuating data stream characteristics. Only while being in this steady state, the DSMS runs in real-time and all time-dependent QoS requirements (tuple delay, data throughput) can be met.

Unsteady DSMS - Indication

From time to time, certain situations require a rescheduling of the DSMS operators. Such situations include, for example, changing input data rates as well as varying filter predicate and join selectivities. An indicator for a transition into an unsteady state can be found when the continuous data flow through network operators and FIFO buffers cannot be held up any longer due to blocking buffer accesses: operators repeatedly try either to write data into full buffers or to read data from empty buffers. Thereby, not every (small) increase or decrease of the data rate causes a buffer fault: whether our DSMS is able to overcome fluctuations of the input data rate, and thus, stays in a steady state depends mainly on the tolerated jitter included in the calculation.

Unsteady DSMS - Adaptation

If the DSMS becomes unsteady, the initially negotiated QoS requirements cannot be assured any longer. More precisely, this means that once a buffer read fault or a buffer write fault has occurred, the tuple delay guarantees (and thus the data throughput) are considered broken and, hence, we have to adapt the running DSMS as soon as possible. Thereby, the DSMS's processing speed has to be adjusted to the new situation.

Our general goal of the adaptation is to bring the system back into a steady state. This comprises three subgoals:

- Depending on the direction of the adaptation (faster or slower network), we either have to free some of the initially occupied resources or, if possible, we have to implement further allocations.
- If the new amount of required resources is available, we will be able to reschedule the DSMS to work at optimum speed regarding the data stream parameters.
- Based on the new DSMS speed, we are able to negotiate new QoS requirements with the DSMS user, and as long as the DSMS stays steady, we are able to give QoS guarantees.

Within QStream, we define the *robustness* of a DSMS by the duration of its steady states. Figure 2 illustrates the influence of data stream characteristics and generous resource allocation (jitter tolerance) on the robustness.

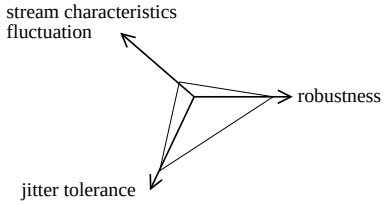

Figure 2: Robustness-determining factors

If the stream characteristics do not change significantly, and if a high amount of jitter was tolerated when calculating and reserving the required resources, the system will become very robust to any change of its real-time behavior, and thus, the time spans of the steady states are maximal.

Generally, it's up to the users how much resources they are willing to spend for covering jitter within the stream characteristics. The more resources they spend, the more robust the DSMS will run.

3 Prototype

Our QStream prototype is implemented on top of the Realtime Application Interface (RTAI, [4]). The overall architecture is illustrated in figure 3.

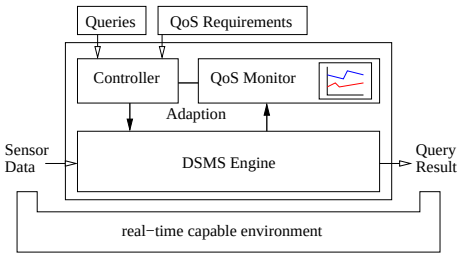

Figure 3: Architecture of the QStream system

ICMI05: Seventh International Conference on Multimodal Interfaces 2005", S. 1299–1301, ISBN 978-1-59593-154-2
https://dl.acm.org/doi/10.5555/1083592.1083756

The system's core for evaluating standing queries against incoming data streams consists of an operator network running stepwise at a fixed speed to achieve the desired data throughput. Within the operator network, FIFO buffers are implemented as shared memory areas to hand over the tuples. A monitor component is a central part of QStream and may be connected to each operator or to each point of the data flow, where it can gather low-level information, such as operator runtimes and data throughput statistics, as well as high-level information, like filter predicate selectivities or even histograms of the streaming data. Depending on the amount and on the level of detail, this information forms the basis for planning the resources of the DSMS. Every time the system is being transferred from one steady state into another one, the resources for the future state have to be determined based on statistics and current runtime information.

The following diagrams show how QStream adapts to new data characteristics:

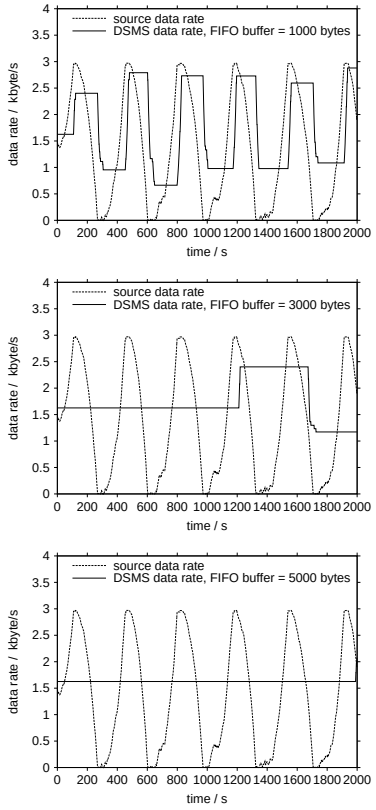

Figure 4: Data rate adaptation example

In figure 4, the adaptation scenarios for three different values of jitter tolerance (which directly correlates with the size of the buffer) are illustrated. The larger the tolerated jitter, the less often the system has to adapt its processing speed, and the longer the steady states last. The latter comes along with long-lasting and stable QoS guarantees.

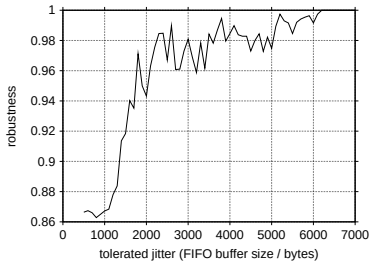

Figure 5: Robustness measure

In figure 5, we define the robustness as the inverse of the number of adaptations per time unit and mark it depending on the tolerated jitter. The general observation is that a larger tolerated jitter, and thus a higher resource consumption, leads to a higher robustness of our prototype.

With the QStream demonstration, we will show how data stream processing can be achieved in real-time even if the data stream characteristics change over time. As a quantitative measure, the *robustness* enables us to reason about how steady a DSMS is and how stable QoS guarantees can be met.

References

[1] H. Berthold, S. Schmidt, W. Lehner, and C.-J. Hamann. Integrated resource management for data stream systems. In *Proc. of the Annual ACM Symposium on Applied Computing (SAC'05, March 13-17, Santa Fe (NM), USA)*, pages 555–562, 2005.

[2] Don Carney, Ugur Çetintemel, Alex Rasin, Stan Zdonik, Mitch Cherniack, and Michael Stonebraker. Operator scheduling in a data stream manager. In *Proc. of 29th International Conference on Very Large Databases (VLDB'03, September 9-12, Berlin, Germany)*, pages 838–849, 2003.

[3] Charles D. Cranor, Theodore Johnson, Oliver Spatscheck, and Vladislav Shkapenyuk. Gigascope: A stream database for network applications. In *Proc. of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD 2003, June 9-12, San Diego (CA), USA)*, pages 647–651, 2003.

[4] Lorenzo Dozio and Paolo Mantegazza. Real time distributed control systems using rtai. In *Proc. of the 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2003, May 14-16, Hakodate, Hokkaido, Japan)*, pages 11–18, 2003.

[5] S. Schmidt, H. Berthold, and W. Lehner. Qstream: Deterministic querying of data streams. In *Proc. of International Conference on Very Large Data Bases (VLDB'04, August 30 - September 3, Toronto, Canada)*, pages 1365–1368, 2004.