ARTICLE IN PRESS

J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Goal-based composition of scalable hybrid analytics for heterogeneous architectures

P. Coetzee [a,*], S.A. Jarvis [a,b]

[a] Department of Computer Science, University of Warwick, United Kingdom
[b] The Alan Turing Institute, The British Library, London, United Kingdom

HIGHLIGHTS

- A new abstract model of assembly and execution for arbitrary analytics, centred around a semantically rich type system.
- Goal-based planning of hybrid analytic applications using this abstract model, requiring little programming ability from the user.
- Automatic code generation across scalable compute architectures, integrating heterogeneous on- and off-line runtime environments.
- Validation of the planning approach through its application to four case studies in telecommunications and image analysis, including an exploration of the performance and scalability of the planning engine for each of these case studies.
- A demonstration of comparable performance with equivalent hand-written alternatives in both on- and off-line runtime environments.

ARTICLE INFO

Article history:
Received 15 June 2015
Received in revised form
12 August 2016
Accepted 18 November 2016
Available online xxxx

Keywords:
Data science
Hybrid analytics
Analytic planning
Streaming analysis
Hadoop
Data intensive computing
Heterogeneous compute

ABSTRACT

Crafting scalable analytics in order to extract actionable business intelligence is a challenging endeavour, requiring multiple layers of expertise and experience. Often, this expertise is irreconcilably split between an organisation's engineers and subject matter domain experts. Previous approaches to this problem have relied on technically adept users with tool-specific training.

Such an approach has a number of challenges: *Expertise* — There are few data-analytic subject domain experts with in-depth technical knowledge of compute architectures; *Performance* — Analysts do not generally make full use of the performance and scalability capabilities of the underlying architectures; *Heterogeneity* — calculating the most performant and scalable mix of real-time (on-line) and batch (off-line) analytics in a problem domain is difficult; *Tools* — Supporting frameworks will often direct several tasks, including, composition, planning, code generation, validation, performance tuning and analysis, but do not typically provide end-to-end solutions embedding all of these activities.

In this paper, we present a novel semi-automated approach to the composition, planning, code generation and performance tuning of scalable hybrid analytics, using a semantically rich type system which requires little programming expertise from the user. This approach is the first of its kind to permit domain experts with little or no technical expertise to assemble complex and scalable analytics, for hybrid on- and off-line analytic environments, with no additional requirement for low-level engineering support.

This paper describes (i) an abstract model of analytic assembly and execution, (ii) goal-based planning and (iii) code generation for hybrid on- and off-line analytics. An implementation, through a system which we call Mendeleev, is used to (iv) demonstrate the applicability of this technique through a series of case studies, where a single interface is used to create analytics that can be run simultaneously over on- and off-line environments. Finally, we (v) analyse the performance of the planner, and (vi) show that the performance of Mendeleev's generated code is comparable with that of hand-written analytics.

1. Introduction

Large organisations rely on the craft of systems engineers and domain scientists to create specialist analytics which provide actionable business intelligence. In many cases their knowledge is complementary; the engineer has knowledge of concurrency,

* Corresponding author.
E-mail addresses: p.l.coetzee@warwick.ac.uk (P. Coetzee), s.a.jarvis@warwick.ac.uk (S.A. Jarvis).

http://dx.doi.org/10.1016/j.jpdc.2016.11.009

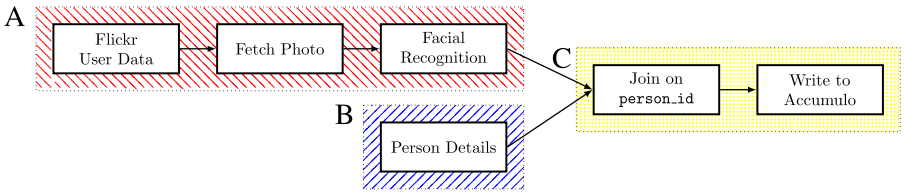


Fig. 1. A sample analytic, reading profile pictures from Flickr and using facial recognition to populate an Accumulo table.

parallel architectures and engineering scalable systems, and the domain expert understands detailed semantics of their data and appropriate queries on that data.

Identifying individuals with both sets of knowledge is challenging, particularly in a growing market, so typically organisations are left with two options: 1. They make use of traditional development models, in which engineers elucidate requirements from stakeholders, develop a solution to meet those requirements, and then seek approval from the stakeholders; or 2. Engineers empower domain experts by offering high-level abstract interfaces to their execution environments, thus concealing the difficulty (and often the potential for high performance and scalability) of developing a hand-tuned analytic.

Consider the Flickr[1] analytic depicted in Fig. 1. Each component of the analysis is represented by a box, with arrows indicating the flow of data from one component to another. There are many runtime environments in which the components of this analytic could be deployed, depending on the wider system context. If user data is being crawled, for example, a streaming (on-line) analytic engine such as Apache Storm [2] or IBM InfoSphere Streams [27] might be employed for subset **A**, while person data in subset **B** might reside in an HDFS (Hadoop Distributed File System) [32] data store. Each of these runtime environments specify their own programming model, optimisation constraints and engineering best practices. This complexity is increased when constructing a hybrid analytic which makes use of data from multiple runtimes: should subset **C** of this Flickr analytic be executed in an on- or off-line runtime environment, and which configuration would be most performant and scalable?

The divide between engineering expertise and domain knowledge has led researchers to consider approaches which make best use of available skills, without the drawbacks inherent in traditional models of cooperation. This paper presents a new approach to this problem, in providing a framework in which domain experts can compose and deploy efficient and scalable hybrid analytics without prior engineering knowledge.

The research described in this paper directly targets the challenges of delivering on-demand results to novel analytics, in the face of ever increasing complexity and heterogeneity of both large networked data sources and the systems used to analyse these data at scale. As the range of software models and hardware platforms increases apace, new models for creating fast data analytics must target not only the engineers with experience of these systems, but specialists with domain knowledge to craft the right analytics, fast enough to deliver results on time. Traditional languages are not sufficient for this: automated composition presents the best opportunity to enable non-technical domain specialists to interact with the analytic platforms crafted by expert engineers.

Specific contributions of this research are:

1. A new abstract model of assembly and execution for arbitrary analytics, centred around a semantically rich type system (Section 4);
2. Goal-based planning of on- and off-line hybrid analytic applications using this abstract model of assembly (1), which requires little programming ability or prior knowledge of available analytic components by the user (Section 5);
3. Automatic code generation for the planned analytic across scalable compute architectures, integrating heterogeneous on- and off-line runtime environments (Section 6);
4. Validation of the type-based planning approach through its application to five case studies taken from the domains of telecommunications, image analysis, and financial technology (Section 7);
5. An exploration of the performance and scalability of the planning engine for each of the case studies in telecommunications and image analysis (Section 8.1);
6. An exploration of the performance and scalability of the resulting analytics in both on- and off-line runtime environments, demonstrating comparable performance with equivalent hand-written alternatives (Section 8.2).

The remainder of this paper is structured as follows: Section 2 describes related work; Section 3 outlines the high-level approach adopted in this research and the implications of design choices; Sections 4 and 5 detail our approach to modelling analytics and planning their execution respectively; Section 6 describes the process of efficient code generation; Section 7 illustrates the application of this approach through four case studies; Finally, Sections 8 and 9 provide a performance evaluation of this framework and conclude the paper.

2. Related work

A variety of approaches to analytic planning exist and have been reported in related literature. Research in this area is often in the context of web-based mashups, however some of the requirements behind such systems are relevant to the research found here. Yu et al. [40] provide a rich overview of a number of different approaches, including Yahoo! Pipes [26], one of the first in a number of recent dataflow-based visual programming paradigms for mashups and analytics. Such solutions require sufficient technical knowledge from their users so that they can navigate, select and compose components of a processing pipeline. Knowledge of a supporting programming language is not required, which removes the challenge of learning programming syntax, but this does not obviate the need for a detailed understanding of the available components, their semantics and their use.

Pipes has inspired a number of extensions and improvements, such as Damia [1], PopFly [19] and Marmite [38]. The work of Daniel et al. [10] aims to simplify the use of tools like Pipes by providing recommendations to a non-expert on how to compose their workflows. Others, such as Google's (discontinued) Mashup Editor [12] take a more technical approach, requiring an in-depth knowledge of XML, JavaScript, and related technologies, but in so doing permit a greater degree of flexibility. Finding domain experts with sufficient expertise in these areas remains challenging.

Some vendors offer alternative solutions for authoring analytics that do not employ complete programming languages. SQL provides one such vehicle for this; Apache Spark SQL [41,39] and

[1] http://www.flickr.com/, a photo sharing website.

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009

Cloudera Impala [16] both offer an SQL-style interface onto NoSQL data stores. The work of Jain et al. [15] aims to standardise the use of SQL for streaming analysis, but its techniques have not been applied to both on- and off-line analytics. Furthermore, other than through the introduction of User Defined Functions, there exist entire classes of analytics that cannot be represented in SQL [18]. Approaches which assemble general-purpose code into complex analytics do not suffer these limitations.

Whitehouse et al. [36] propose a semantic approach to composing queries over streams of sensor data, employing a declarative mechanism to drive a backward-chaining reasoner and solving for possible plans at execution time. Sirin et al. [33] introduce the use of OWL-S [20] for query component descriptions in the SHOP2 [21] planner (a hierarchical task network planner). OWL-S extends the purely syntactic composition of services afforded by WSDL by adding a semantic model of the inputs and outputs to a web service. Another common approach, taken by Pistore et al. in BPEL4WS [25], uses transition systems as a basis for planning. A recurring theme in these approaches is that of composing queries by satisfying the preconditions for executing composable components. The runtime composition approach is flexible, but has implications for performance at scale.

There has been considerable work in the area of web service composition for bioinformatics; BioMOBY [37] specifies a software interface to which services must adhere, then permits a user to perform discovery of a single service based on their available inputs and desired outputs; it does not manage the planning and composition of an entire workflow. Taverna [23] offers a traditional "search" interface (making use of full-text and tag-based search) to locate web services which a user can manually compose in the Taverna interface. This form of manual search and assembly requires considerable user expertise, and an understanding of the art of the possible, which a general-purpose analytic planner does not.

Research in Software Engineering has examined analogous problems to this. Stolee et al. [34] examined the use of semantic models of source code as an indexing strategy to help identify blocks of code that will pass a set of test cases, presenting the user with a collection of existing candidate solutions to their problem. Such semantic searches have additionally been trialled in web service composition [3,9]. However, the complexity of the semantic model and inherent uncertainty in retrieval accuracy, make assembly of multiple blocks of code somewhat risky—there is a considerable probability that the retrieved code samples are not compatible.

These web-services-based systems typically involve considerable user training (whether in the composition interface or in the formal specification of their query language), and at their core aim to answer single questions through service-oriented protocols such as WSDL and SOAP. Often, large-scale data analytic workflows aim instead to analyse significant amounts of data in parallel—an execution model which is closer to that found in high-performance computing simulations than in web mashups. In addition to the complexity of WSDL and SOAP definitions, the services offered must often be written specifically for use with such a system: their implementation depends directly on, *e.g.*, a SOAP implementation. There are many existing libraries of components in the data analytics space which cannot be reasonably re-written to enable integration with a composition system: instead, it is desirable for such a system to interface with the existing APIs of the target runtime directly.

One noteworthy solution to this problem is that taken by IBM's research prototype, MARIO [29], which builds on SPPL, the Streaming Processing Planning Language [30,31]. IBM characterises MARIO as offering *wishful search*, which a user drives by entering a set of goal tags. The MARIO planning engine then aims to construct a sequence of analytical components that will satisfy those goals. These tags correspond to those applied to flows of components within engineer-defined code templates. In practice, due to the tight coupling between the engineer-created *tagsonomy* and the actions available to the end user (components are often manually tagged as compatible), it is rare for MARIO to create a novel or unforeseen solution to a problem.

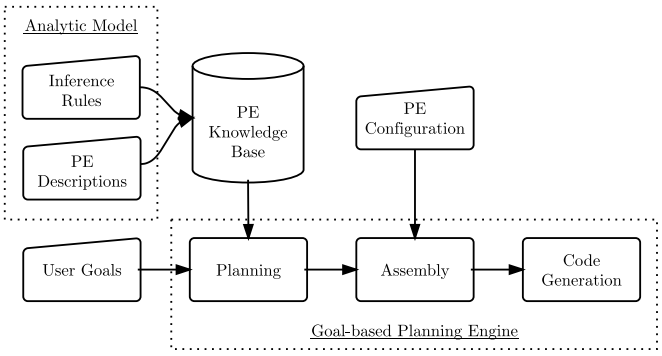


Fig. 2. Steps in composing an analytic.

The research presented in this paper builds on the wishful-search concept behind MARIO. At the same time, it allows the discovery and composition of novel complex analytics, while using a higher-level granular model of analytic behaviour, utilising existing techniques from AI planning. This model is in many ways similar to those proposed in OWL-S, but the way in which the model is applied differs considerably. It is the first approach to target execution of automatically generated hybrid analytics in heterogeneous compute environments. It is the only automated planning engine of its type to offer domain scientists such wide-ranging application for on- and off-line analytic planning, composition and efficient and scalable code-generation, without also requiring significant engineering support.

3. High-Level overview

To compose an analytic from a user's goals, our approach employs the components outlined in Fig. 2. An abstract Analytic Model (detailed in Section 4) is used to create a knowledge-base of processing elements (PEs). This knowledge-base encodes information about the types available in the planning system, the PEs which produce and consume these data types, and a collection of pre- and post-conditions attached to these PEs. It is important to note that the creation of this knowledge-base is beyond the scope of this research: it is assumed that engineers in organisations with a need for an analytic planning system are willing to undertake the manual annotation of the PEs they make available to their users.

This knowledge-base provides a semantically precise description of the information encoded in the data both required and produced by the available PEs. It is the contention of this research that this metadata is sufficient to facilitate the automated composition and deployment of complex analytics across multiple runtime platforms in a heterogeneous data-intensive compute environment.

In order to do this, the system collects goals from the user as a second input to the planning process. There are three types of goals that the user may offer to constrain the planning process (see Section 5):

- The output types that the analytic must produce;
- The datasource with which the analytic must begin;
- Post-conditions, including those concerned with the state of the runtime environment in which the analytic executed.

For example, to create the sample analytic described in Fig. 1, the user might specify:

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009

- Types: `person_id`, `person_name`, `postal_address`, `email_address`
- Source: `FlickrUserData`
- PE Used: `AccumuloSink`

These constraints are provided to a planning process (Section 5), which uses a bidirectional search strategy to traverse the graph of possible PE connections. It aims to satisfy the given constraints using a minimal number of PEs, producing a set of possible analytics which can be presented to the user. A user-friendly rendering of the analytic can be provided along with textual descriptions of the PEs in the analytic to help the user select which version to deploy. Any unbound configuration options are then supplied by the user to the assembly process (*e.g.*, which Accumulo table to write to, or tunable parameters for the facial recognition), which makes the abstract plan concrete and resolves any ambiguities. Finally, code generation (Section 6) is invoked on the concrete plan to create an executable analytic.

3.1. Methodology

The approach described in this paper is applicable to a number of runtime models and analytic frameworks. We have implemented and tested it using real analytics in a system we call Mendeleev, named after the scientist responsible for composing and organising the periodic table as we know it today. We use a library of real PEs and customer problems to test the scalability of the code generation, and a synthetically generated representative PE library to test the scalability of the planning approach. This, coupled with a qualitative investigation of the use of the planner to generate solutions to these customer problems, forms the basis of the rigorous evaluation in Sections 7 and 8.

3.2. Impact of design choices

One of the key assumptions made in this research is a workflow-style execution model. This model pervades the literature on fast Big Data analytics [2,27,41,11,14,6,24,13]: while it places a limitation on the range of frameworks that can be used (particularly outside of the real of scalable data analysis), it enables high performance execution across the most common data analytic platforms.

The use of an RDF model to encode the PE knowledge base slightly increases the set of skills required by engineers to annotate their PEs. However, as discussed in Section 5, the strong semantics behind an RDF ontology enable the use of both system- and engineer-defined inference rules (along with special predicates, as in Section 5.2) to enrich the knowledge base, ultimately reducing the effort required to describe all aspects of the PEs.

Finally, in order to make best use of the applicability of the message-passing model, no further assumptions are made during the planning process as to the suite of runtime frameworks which are available or in use. These frameworks are encoded in two places only: the customer-specific model of library PEs, and in a set of pluggable code generation modules. This prevents the planning process from using runtime-specific knowledge (which must be encoded in inference rules or special predicates), but makes it simple to add further runtime frameworks to the Mendeleev implementation.

4. Modelling analytics

As our first contribution, we introduce a novel abstraction by which the planning and the concrete implementation of an analytic can be logically separated. This abstraction is based on our previous model [7], extending it for use in hybrid analytic planning. There are two components to this model: a semantically rich type system, and a set of analytic components which reference these types. This research models an analytic as a set of parallel-composed communicating sequential processes [28], called Processing Elements (PEs). These pass tuples of data (consisting of a set of named, strongly typed elements) from one PE to the next. When a PE receives a tuple, it causes a computation to occur, and zero or more tuples are emitted on its output based on the results of that computation. Nothing in the model is specific to the planning process—it is an abstract representation of the concrete implementation of a collection of composable components.

The model is encoded in an RDF [17] graph describing the available types and PEs.[2] Types may exhibit polymorphic inheritance, as in a typical second-order type system. This inheritance is indicated using the `mlv:parent` relationship, and may form an inheritance graph provided each type cycle declares only one `mlv:nativeCode` per runtime; that is, the name of the type in the target language that is represented by this concept (for example, a Java class or SPL primitive type). For example, a buffer of bytes might represent more than one type of information (*e.g.*, a PDF file or an image), even though the data underlying it is the same type, as in Listing 1.

Listing 1: RDF graph for a simple type hierarchy

```
# The "raw" ByteBuffer parent type
type:byteBuffer rdf:type mlv:type ;
 mlv:nativeCode  [ rdfs:label "java.nio.ByteBuffer" ;
      mlv:runtime mlv:crucible ] ;
 mlv:nativeCode  [ rdfs:label "list<uint8>" ; mlv:runtime
       mlv:streams ] .
# An image encoded in a ByteBuffer
type:image rdf:type mlv:type ;
 mlv:parent type:byteBuffer .
# A PDF file encoded in a ByteBuffer
type:pdfFile rdf:type mlv:type ;
 mlv:parent type:byteBuffer .
```

In addition to this basic polymorphism, a type may contain an *unbound variable* with an optional type constraint (akin to a generic type in Java [5], or a template in C++ [35]). This is used in PEs which transform an input type to an output without precise knowledge of the information encoded in the data. For example (see Listing 2), a PE for fetching data over HTTP might take an input of `type:URL` parameterised with `<?T mlv:parent type:byteBuffer>`, and output data with the same type as the variable `?T`, an as-yet unbound subtype of `type:byteBuffer`. A priori, `?T` is known to be a `type:byteBuffer`; during planning it may be bound to a more specific type (*e.g.*, `type:image` in the Flickr analytic described above).

Listing 2: Modelling unbound type variables in RDF

```
# Declaration of a generic type
type:URL rdf:type mlv:genericType ;
 mlv:nativeCode [ rdfs:label "java.net.URL" ; mlv:runtime
       mlv:crucible , mlv:accumulo ] ;
 mlv:nativeCode [ rdfs:label "rstring" ; mlv:runtime
       mlv:streams ] .
# PE input declaration for url<?T>
#    (bnode _:urlType represents variable)
_:sampleInput rdf:type [
```

[2] RDF types are given in this paper using W3C CURIE [4] syntax. The following RDF namespaces are used:

```
rdf http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs http://www.w3.org/2000/01/rdf-schema#
mlv http://go.warwick.ac.uk/crucible/mendeleev/ns#
type http://go.warwick.ac.uk/crucible/mendeleev/types#
```

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009

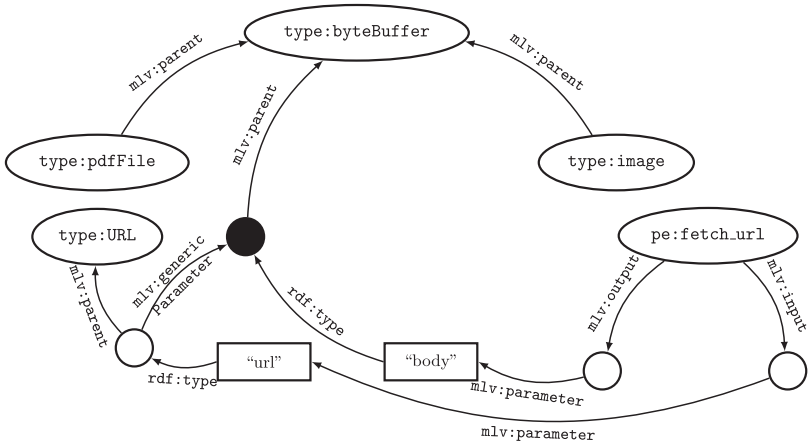


Fig. 3. Graph visualisation of the RDF description of a portion of the example model. `_:urlType` bnode represented by ●.

```
 mlv:parent type:URL ;
 mlv:genericParameter _:urlType
] .
# Variable for the type parameter to URL
_:urlType rdf:type type:byteBuffer .
# PE output parameter using the variable
_:sampleParameter rdf:type _:urlType .
```

A visualisation of the RDF graph resulting from this type hierarchy (along with a subset of the PE model described in Listing 3) can be seen in Fig. 3. The unbound variable `_:urlType` is highlighted as a filled black circle in this figure.

As suggested by the types used above, the engineers who describe their PEs are encouraged to do so using the most specific types possible. For example, the more precise semantics of `type:image` are to be preferred to `type:byteBuffer`, even though both result in the same `mlv:nativeCode`.

4.1. PE formalism

We consider a PE χ_n to have a set of declared input types μ_n, and a set of declared output types ν_n. For a data source, $\mu_n = \emptyset$ (it produces data without any inputs being present), while for a sink $\nu_n = \emptyset$ (it receives inputs of data, but produces no output). Tuple data generally accumulates as it passes through each PE, treating it as an enrichment process on the data it receives. No specific knowledge about the processing performed is encoded in the model. More formally, a PE χ_n has an accumulated output type (denoted as τ_n) based on the type of the tuple received on its input, τ_{n-1}. Thus, to determine τ_n for a given PE, the entire enrichment chain must be known:

$$\tau_n = \nu_0 \cup \nu_1 \cup \cdots \cup \nu_{n-1} \cup \nu_n. \tag{1}$$

Or, inductively:

$$\tau_n = \tau_{n-1} \cup \nu_n. \tag{2}$$

This model can be extended to include PEs (*e.g.*, complex aggregations) that clear the accumulated data in a tuple declaration before emitting their outputs; this extension is considered in greater detail as part of the planning process in Section 5.2.

One important extension to this model is in support of operators which require inputs on more than one port, such as join operators (discussed in further detail in Section 5). These receive two or more discrete sets of input types, and by default emit the union of their accumulated inputs. Thus, for an operator χ_n with inputs χ_i and χ_j, τ_n is given as follows:

$$\tau_n = \tau_i \cup \tau_j. \tag{3}$$

PE connectivity utilises a form of subsumption compatible with the type model described above. A type u can be said to be subsumed by a type v ($u \lhd v$) if one of the following cases hold true:

$$u \lhd v \Leftarrow \begin{cases} u\ \texttt{mlv:parent}\ v \\ u\ \texttt{mlv:parent}\ t, t \lhd v \end{cases} \tag{4}$$

$$u\langle t \rangle \lhd v\langle s \rangle \Leftarrow u \lhd v \wedge t \lhd s. \tag{5}$$

A PE χ_x is considered *fully compatible* with χ_y, and is thus able to emit tuples to PE χ_y, if the following holds true:

$$\forall t \in \mu_y, \exists u \in \tau_x \mid u \lhd t. \tag{6}$$

In the RDF model, each PE definition includes the native type name associated with the PE, as well as the set of (typed) configuration parameters, and input and output ports. Additionally, the model may include user-friendly labels and descriptions for each of these definitions. Unlike other planning engines (particularly HTN style planners such as MARIO), which require the engineer to additionally implement prototype code templates, this RDF model is the only integration that is required between a PE and the MENDELEEV system. For example, a more complete version of the HTTP fetching PE described above is shown in Listing 3.

Listing 3: Modelling an SPL (IBM's Streams Processing Language) HTTP Fetch PE in RDF

```
pe:fetch_url rdf:type mlv:spl_pe ;
 mlv:nativeCode "lib.web::FetchURL" ;
 mlv:input [
  mlv:parameter [  # url is a URL<?T>
   rdfs:label "url" ;
   rdf:type [
    mlv:parent type:URL ;
    mlv:genericParameter _:fetch_type
   ]
  ]
 ] ;  # End input declaration
 mlv:output [
  rdfs:label "HttpOut" ;
  mlv:parameter [  # httpHeaders is a header_list
   rdfs:label "httpHeaders" ;
   rdf:type type:header_list
  ] ;
  mlv:parameter [  # body is a ?T
   rdfs:label "body" ;
   rdf:type _:fetch_type
  ]
 ] .  # End output declaration
# byteBuffer is the parent type of ?T
_:fetch_type rdf:type type:byteBuffer .
```

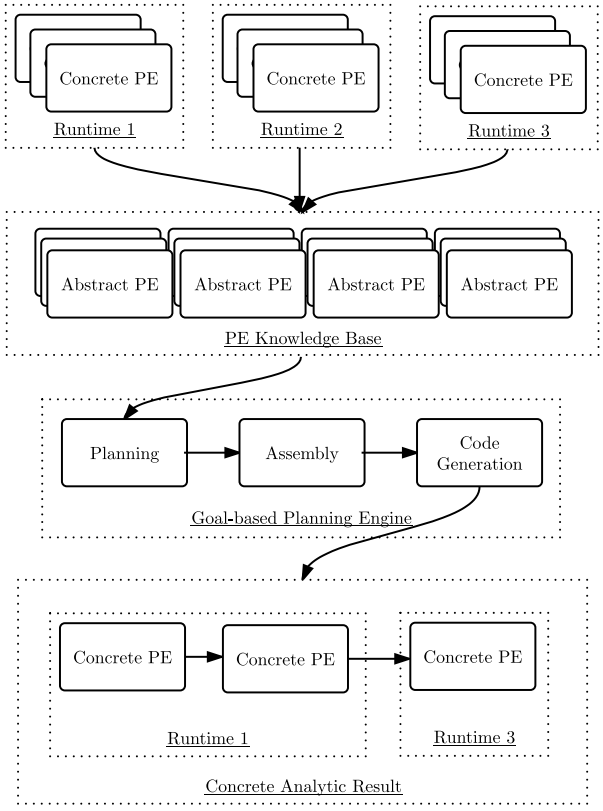


Fig. 4. Using the PE Model abstraction to separate planning and concrete PE implementations.

4.2. PE model abstraction

This model abstracts the concrete implementation of an analytic away from runtime framework-specific details. This is vital to enable hybrid planning, as all runtime frameworks may be treated equally: as seen in Section 5, PEs from any framework may be assembled in a workflow. The PEs represented by this abstraction are later made concrete by the code generation process (Section 6), which translates from the execution model assumed in the PE model to runtime primitives, invoking the user-defined components the model describes, as shown in Fig. 4.

5. Goal-based planning

Our second contribution is a goal-based planner based around the semantically rich type system described above. The goal of this planner is to explore the graph of possible connections between PEs using heuristics to direct the search, accumulating types in the τ set until the user-supplied constraints have all been satisfied, or the planner determines that no solution exists.

5.1. Type closure

Given the RDF model of the PE knowledge-base, a suite of forward inference rules are pre-computed before any planning may occur. These rules are applied using a forward chaining reasoner (the FuXi [22] RETE-UL algorithm), and compute three key types of closure. First, RDFS reasoning is applied to the types in the knowledge base (primarily to compute the closure over second-order types). Next, unbound type variables are compared, to compute potential subsumption. Finally, candidate PE matches are inferred based on rules derived from the *full compatibility* rules described in Section 4, Eq. (6). A PE χ_x is considered *partially compatible* with χ_y, and is thus a candidate for sending tuples to PE χ_y, if one of the following holds true:

$$\exists t \in \mu_y, u \in \nu_x \mid u \lhd t \tag{7}$$

$$\exists t \in \mu_y, u\langle v\rangle \in \mu_x, v \in \nu_x \mid v \lhd t \tag{8}$$

$$\exists s\langle t\rangle \in \mu_y, u\langle v\rangle \in \nu_x \mid u \lhd s, v \lhd t. \tag{9}$$

For example, consider `pe:fetch_url` described above; it requires a URL parameterised with any `type:byteBuffer`. Consider also `pe:exif`, which requires a `type:image` on its input (where `type:image` $\lhd$ `type:byteBuffer`), and outputs a number of Exif[3]-related fields:

$$\mu_{\text{fetch\_url}} = \{\texttt{type} : \texttt{url}\langle ?T \lhd \texttt{type} : \texttt{byteBuffer}\rangle\} \tag{10}$$

$$\nu_{\text{fetch\_url}} = \{?T\} \tag{11}$$

$$\mu_{\text{exif}} = \{\texttt{type} : \texttt{image}\} \tag{12}$$

$$\nu_{\text{exif}} = \{\texttt{type} : \texttt{camera}, \texttt{type} : \texttt{lat}, \texttt{type} : \texttt{lon}, \texttt{type} : \texttt{fstop}, \ldots\}. \tag{13}$$

Through Eq. (8) above, the `?T` output by `pe:fetch_url` can potentially be used to satisfy the input to `pe:exif`. In this case, `pe:fetch_url` is considered partially compatible with `pe:exif`, and is marked as a candidate connection when `?T` is bound to `type:image`.

5.2. Conditions

Once the type closure is computed, a further suite of rules annotates each PE in the knowledge-base with a set of pre- and post-conditions, derived from the input and output specification. A pre-condition is automatically applied specifying the runtime environment for each PE: this is derived from the `rdf:type` specified for the PE. These inferred conditions can be manually augmented in the RDF PE model with two further types of condition.

The first of these condition types are used to alter the behaviour of the inference or the search process. For example, a `mlv:clearPreConditions` statement is used when modelling PEs which do not automatically pass on the data received on their inputs. Such PEs may include aggregation operations (grouping *etc.*), windowing operators, or others which do not have 1–1 cardinality and either filter or group records. Another special condition, `mlv:clearRuntime` is implemented to remove post-conditions from the τ set which specify the current runtime environment. For example, Listing 4 models a PE which aggregates input data into a Gaussian Mixture Model using an Expectation Maximisation algorithm.

Listing 4: RDF Model for a Gaussian Mixture Model implemented on Apache Spark

```
pe:gmm2d a mlv:spark_pe ;
 rdfs:label "Apply␣EM␣to␣generate␣a␣2D␣Mixture␣of␣
     Gaussians␣modelling␣the␣input" ;
 mlv:nativeCode "mendeleev.pe.GMM2D" ;
 mlv:input [
  mlv:parameter [ rdfs:label "x" ; rdf:type type:double ]
      ;
  mlv:parameter [ rdfs:label "y" ; rdf:type type:double ]
 ] ; # Clear existing pre - conditions
 mlv:postCondition [ mlv:clearPreConditions pe:gmm2d ] ;
 mlv:output [  # Emit a collection of weighted 2D
     Gaussians
  rdfs:label "Gaussians" ;
  mlv:parameter [ rdfs:label "weight" ; rdf:type type:
      gmm_weight ] ;
```

[3] Exchangeable image file format; image file metadata.

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009

```
  mlv:parameter [ rdfs:label "x" ;    rdf:type type:
      gaussian_x ] ;
  mlv:parameter [ rdfs:label "y" ;    rdf:type type:
      gaussian_y ] ;
  mlv:parameter [ rdfs:label "theta" ;  rdf:type type:
      gaussian_rotation ] ;
  mlv:parameter [ rdfs:label "A" ;    rdf:type type:
      gaussian_magnitude ]
] .
```

The second type of manual condition is one which has no special meaning to the planner, but makes an assertion about the state of the analytic. For example, these are employed (in conjunction with the `mlv:clearRuntime` condition above) to manage the transition between runtimes. Listing 5 gives an example of how synthetic runtimes are used (in this case, `mlv:accumulo_to_streams`) to constrain the planner, so that an Export node from one runtime is followed immediately by an Import node for the next. These provide the necessary hooks for the code generators (discussed in Section 6) to create suitable code for managing the inter-runtime transport of data.

Listing 5: RDF Model for an Import and Export transport from the Accumulo Iterator paradigm into IBM InfoSphere Streams

```
pe:accumulo_to_streams_export a mlv:accumulo_pe ;
 rdfs:label "Export␣Accumulo–>Streams" ;
 mlv:nativeCode "mendeleev.pe.StreamsExportIterator" ;
 mlv:input [ rdfs:label "Data" ] ;
 # Clear existing runtime; reset to the synthetic mlv:
     accumulo_to_streams runtime
 mlv:postCondition [ mlv:clearRuntime pe:
     accumulo_to_streams_export ] ;
 mlv:postCondition [ mlv:runtime mlv:accumulo_to_streams
     ] .

pe:accumulo_to_streams_import a mlv:spl_import_pe ;
 rdfs:label "Import␣Accumulo–>Streams" ;
 mlv:nativeCode "mendeleev.pe::AccumuloImport" ;
 mlv:output [ rdfs:label "Data"] ;
 # Require the synthetic mlv:accumulo_to_streams runtime
 mlv:preCondition [ mlv:runtime
      mlv:accumulo_to_streams ] ;
 # Replace the synthetic mlv:accumulo_to_streams runtime
     with mlv:streams
 mlv:postCondition [ mlv:clearRuntime pe:
     accumulo_to_streams_import ] ;
 mlv:postCondition [ mlv:runtime mlv:streams ] .
```

In practice, this inference closure is calculated offline and the resultant graph is stored for interactive use. These manually specified conditions may also be used to describe particular transformations performed on the data (*e.g.*, rotation of an image) which are not otherwise captured by the semantics of the output tuple. As with the runtime conditions, these post-conditions may be used internally in the planning process to satisfy the pre-conditions of other PEs, or they may be explicitly requested by the user (as with the "PE Used" post-condition introduced in Section 3).

5.3. Search & assembly

The search through the graph of partially compatible PEs is outlined in Algorithm 1. This algorithm finds a set of pathways through the graph of candidate PE connections which will generate the required set of post-conditions, while fulfilling the pre-condition requirements of each PE. In order to minimise the search-space explosion, the search is performed bi-directionally, with an empirically selected heuristic expanding the search space backwards for every three levels of forward search. Similarly, if a source or a sink constraint is specified, it is used to optimise the search process. The algorithm has six stages:

Algorithm 1 Bidirectional Planning, searching for a given set of target conditions (ϕ), source PE (σ), accumulated conditions (τ), and backwards search set (β)

```
1:  procedure SOLVE(φ, σ, τ, β)
    ▷ Every 3 levels of forward search, advance backwards
2:      if search_level % 3 == 0 then
3:          β ← β ∪ providers_of(φ)
4:      end if
5:      if σ not given then
6:          results ← ∅
7:          for all source s in model do
8:              results ← solve(φ, s, τ, β)
9:          end for
10:         return results
11:     end if
12:     results ← ∅
    ▷ Check σ for secondary inputs
13:     for all input i in inputs(σ) do
14:         if i not satisfied by τ then
15:             results ← results ∪ solve(preConditions(i), σ, ∅, ∅)
16:         end if
17:     end for
    ▷ Update τ with postConditions of σ, and check for completion
18:     τ ← τ ∪ postConditions(σ)
19:     if τ satisfies φ then
20:         return [σ]
21:     end if
    ▷ Depth-first search of PEs in β
22:     forward ← consumers_of(τ)
23:     candidates ← dfs_search(forward ∩ β, φ, σ, τ)
24:     for all candidate in candidates do
25:         results ← results ∪ [σ, candidate]
26:     end for
    ▷ Depth-first search of remaining candidates
27:     if results == ∅ then
28:         results ← dfs_search(forward − β, φ, σ, τ)
29:     end if
30:     return results
31: end procedure
```

Algorithm 2 Type Pruning

```
1: procedure PRUNE_TYPES(pe, φ)
   ▷ Remove types from τ_pe that are not in the φ set
2:     τ_pe ← τ_pe ∩ φ
   ▷ Add types to the φ set that are required by this PE
3:     φ ← φ ∪ μ_pe
   ▷ Recurse to all publishers of data to this PE
4:     for all σ in publishers(pe) do
5:         prune_types(σ, φ)
6:     end for
7: end procedure
```

L2-4: Every 3 levels of forward search, expand the set of backward search candidates by one more step;
L5-11: If the call to SOLVE does not provide a bound on the source, launch a solver to generate results for all sources in the model;
L13-17: If the current PE has more than one input, launch a new SOLVE to satisfy the pre-conditions of each input;
L18-21: Update the sets of accumulated conditions (τ), and test to see if all required post-conditions are satisfied; if so, this branch of the search terminates;
L22-26: Attempt to search the next level (recursively), using only the set of backwards candidates;

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009

L27-29: If the above step did not yield any new paths, repeat the search with PEs not in the set of backwards candidates.

A simple heuristic ranking may be applied to this set of candidate pathways *e.g.*, based on the number of PEs in the path (if two paths accumulate the same post-conditions, it can be considered that their results are similar, and thus the shorter, "simpler" path should be preferred). It is not sufficient to automatically select and assemble one of the available paths arbitrarily.

Although users of the MENDELEEV system are considered non-technical in the sense of not being expert software engineers, they are still experts in their field. As such, they can be expected to understand the semantics of the operations in the knowledge-base, and to comprehend the meaning of an assembly of these operations, even if they do not know the full set of available operators or rules for their composition *a priori*. As a result, a visual inspection of the available plans (presented graphically in the MENDELEEV UI) is generally sufficient for the user to select the plan which best represents their intent.

Once the user selects an execution plan from the generated options, it must be assembled into a concrete plan. This process involves binding keys from each tuple to the required output types. For example, if a tuple of Flickr user data contained two `type:url<type:image>` parameters, a profile background and a user avatar, and it was passed to the aforementioned `pe:fetch_url`, the assembly process must bind one of these parameters on its input. In practice, no reliable heuristic is available for this, and user configuration is required. For a domain expert this should not present a difficulty, as they can be expected to understand the nature of the fields in their data.

This planning and assembly process generates an acyclic graph of PEs as its output, with a single goal-state node and one or more source nodes. It can also be considered a tree, rooted on the goal node. The goal node will have a τ of the union of all PE post-conditions up to that point in the analytic—however, many of the types specified in the post-conditions may not be needed in order to correctly complete the computation. During assembly, a second pass is therefore taken backwards across the topology (in a breadth-first traversal from the goal node) using the type pruning algorithm outlined in Algorithm 2 to prevent the topology from passing unnecessary data forwards. This helps to control the otherwise unlimited expansion of tuple width, improving the space, time and message passing complexity of the resultant analytic.

For example, in the sample analytic shown in Fig. 1, a number of fields are emitted by the `FlickrUser-Data` crawl, such as `realName`, `location`, `mbox_sha1sum`, `profileUrl`, and `buddyIconUrl`. The only field type from this set which is required later in the analytic is a `type:URL`, so the type pruning algorithm in Algorithm 2 discards all fields other than the field the user has selected to satisfy this constraint (in this case, `buddyIconUrl`) from the output. Similarly, HTTP status information output by the `FetchURL` PE are discarded, along with any outputs from the `PersonDetails` database which are not requested by the user. This process significantly reduces the amount of data passed by MENDELEEV-generated PEs, thereby improving the performance of the resulting analytic.

6. Code generation

Once the concrete execution plan is assembled, it is passed to a pluggable code generator; the third contribution of the research documented in this paper. MENDELEEV's planner produces a concrete plan, which the code generator must turn into native code for execution on the correct mix of on- and off-line runtimes. To achieve this, it may either generate native code for each runtime directly, or use an intermediate representation to manage the differences in runtime models. MENDELEEV offers the user both capabilities.

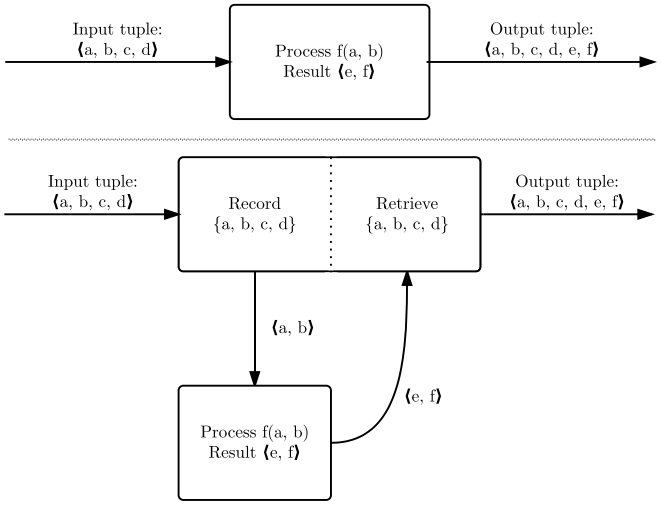


Fig. 5. Top: MENDELEEV message passing model for a process `f`. Bottom: Wrapper-based model of field copying semantics.

6.1. DSL code generation

MENDELEEV has been designed to generate code using the CRUCIBLE [8] domain specific language (DSL) as an intermediate representation. CRUCIBLE offers a DSL and a suite of runtime environments, adhering to a common runtime model, that provide consistent execution semantics for an analytic across on- and off-line runtimes. A performance evaluation of CRUCIBLE [8] showed consistent scalability and applicability of CRUCIBLE topologies in a standalone environment, in batch mode on Apache Spark, and in streaming mode on IBM's InfoSphere Streams. CRUCIBLE's DSL is source-to-source compiled through a series of code generation modules to optimised native code for each runtime environment on which it can be executed.

There is one key difference between the MENDELEEV and CRUCIBLE execution models: whereas MENDELEEV assumes that all keys in the input tuple are passed through on the output, CRUCIBLE does not perform this pass-through automatically. It is possible to implement these semantics in CRUCIBLE, however. Fig. 5 illustrates how this might be achieved in the basic CRUCIBLE execution model. MENDELEEV's conceptual model (top) shows a PE `f(a, b)` which generates the tuple `<e, f>` as its results, passing through the full input tuple along with those results. At the bottom of Fig. 5, an implementation of the MENDELEEV tuple field copying semantics in the basic CRUCIBLE model shows how each functional PE is wrapped in one which stores the input tuple fields, and appends them to the output of each tuple from that functional PE.

While this theoretical approach produces correct results, the extra message passing it involves would slow topologies down considerably. Instead, MENDELEEV generates a synthetic parent PE in Java for each PE in the CRUCIBLE topology, overriding a small portion of the base CRUCIBLE runtime on a per-PE basis with generated code. This parent is responsible for intercepting received and emitted tuples, recording the inputs in local state, and appending the relevant outputs of that PE's pruned accumulated type on tuple output. To use the example of `pe:fetch_url` in the Flickr analytic above, this synthetic parent might record the `type:profile_image` URL on its input, and append it to the output tuple. At this time it will also prune out unused tuple fields from the output as per Algorithm 2.

6.2. Native code generation

When an analytic does not require the flexibility or features of the CRUCIBLE DSL (or PEs are only available in their native

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009

Table 1
MENDELEEV Import/Export implementations.

Source runtime	Export behaviour	Import behaviour	Destination runtime
Accumulo	No-Op	Scanner; SPL type conversion	Streams
Accumulo	JSON Serialisation	Scanner; JSON parse	Meteor.js
Streams	Convert SPL types to Java; Kryo Serialise; write to Accumulo table	No-Op	Accumulo
Streams	JSON Serialisation; TCP socket server	TCP socket client; JSON parse	Meteor.js
Meteor.js	TCP socket client	TCP socket server; JSON parse; SPL type conversion	Streams
CRUCIBLE	TCP socket server	TCP socket client; SPL type conversion	Streams
CRUCIBLE	Kryo Serialise; write to Accumulo table	No-Op	Accumulo

form, not a CRUCIBLE library), direct native code generation may be a more performant option. This code generation option relies on the accuracy of both the input and output specifications, and the manually entered pre- and post-conditions of PEs to generate the correct code. Four native code generators are currently implemented in MENDELEEV: two for Accumulo, one for IBM InfoSphere Streams SPL, and one for the Meteor.js reactive web presentation framework.

Accumulo makes use of separate code generators: one for the base Accumulo table (consisting of heterogeneous rows of Key–Value pairs), and one for an Iterator stack which may be applied on top of this. These generators simply create a pair of Java classes which configure an Accumulo connection, set up the requisite Iterator stack, and return a Scanner of rows to the calling site. This Iterator stack will be executed on the server-side as the Scanner is consumed on whichever client uses the data (be it an Accumulo writer, or another runtime).

MENDELEEV includes a set of inter-runtime transports, in order to facilitate the motion of data from one analytic to another. The modelling of these, discussed in Section 5.2, offers the ability for each transport pairing to have a distinct implementation designed for optimal performance (such as sinking JSON to Meteor.js, but writing Key–Value Mutations to an Accumulo table). The implemented transports, and their Export/Import mechanisms, are outlined in Table 1.

6.3. Integrating complex analytics

Some users of a system such as MENDELEEV require complex carefully engineered analytics (*e.g.*, to build up state about a set of identifiers, or for performance-tuned machine learning algorithms). In the interests of efficient system utilisation, it is often desirable to run these types of analytic as a central job to which other analytics may subscribe. Several patterns can be used to expose this behaviour transparently to a user in MENDELEEV (illustrated in Fig. 6).

First, it is possible to simply write all results from the "Complex Analytic Job" to a persistent store, as a results cache (shown below the complex job in Fig. 6). This approach results in treating the output as an offline data source for each new MENDELEEV analytic. It is also possible to achieve a streaming equivalent by simply exporting results on a TCP Socket Server. This approach has a relatively low implementation overhead, but depending on the use cases for the complex analytic may result in more complex MENDELEEV plans (*e.g.*, due to a frequent need to join this data with other sources). An alternative for analytics which use the complex job as a source of enrichment is that of an RPC-style model (shown to the right of the job in Fig. 6). This is suitable for large stateful analytics, although it requires the maintenance of an RPC query server and associated infrastructure for distributed configuration. This infrastructure is outside the scope of MENDELEEV; systems such as Apache ZooKeeper have been found to fulfil this requirement in practice.

As well as complex analytic jobs, some organisations will have complex sources of data (*e.g.*, a large relational database with many views). The complexity of extracting data from such a system need not be reflected by equivalent complexity in MENDELEEV; each potential view can be considered a different source PE in the knowledge base, with its own tuple type information entered accordingly. Note that MENDELEEV has no restriction on the number of times a single target PE type may appear in a knowledge base (*e.g.*, with different configuration parameters to turn on or off features of that PE). This is a useful design pattern; one can engineer a single general-purpose accessor PE which is configured in the knowledge base to represent many different data sources. As the PE knowledge base is RDF-based, it is additionally possible to extend the set of inference rules to include generators for permutations and combinations of different PE parameters, rather than entering them by hand.

7. Case studies

To better understand the process of composing analytics in MENDELEEV, we present a series of case studies and an evaluation of this technique; this represents the fourth contribution of this research paper. These analytics have been generated with MENDELEEV, using a small library of general-purpose PEs. Fig. 7 illustrates the generated analytics for each case study below; each figure shows the PEs in an analytic (as boxes), the tuple subscriptions between those PEs (arrows indicate the direction of flow), and the runtime for each subset of PEs (shaded outer boxes). Note that, for brevity, explicit Import/Export nodes have been omitted from these representations.

7.1. Case study: Flickr FFT workflow

The user wishes to compute and store the Fourier transform of images from Flickr, and store those results in HDFS for use later in their workflow. Engineers have exposed a crawl operator which emits Flickr photo metadata to the MENDELEEV system. The user selects the following bounds from the user interface:

1. PE Used: HDFS
2. Types: `image`, `fft2d`.[4]

With each refinement of a bound, the MENDELEEV UI plans a new set of plausible analytics to answer that query. It is interesting to note here, that the query does not explicitly require data from Flickr; any data sources in the knowledge base which can be used to return an `image` may be offered to complete this query. In this instance, MENDELEEV produces a single result: the analytic shown in Fig. 7(a).

7.2. Case study: Flickr facial recognition

A different analyst has an interest in annotating Flickr images with the email addresses of the people in them using a facial recognition system, sending their results to an Accumulo table

[4] The output of a Fourier transform on 2-D input data.

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009

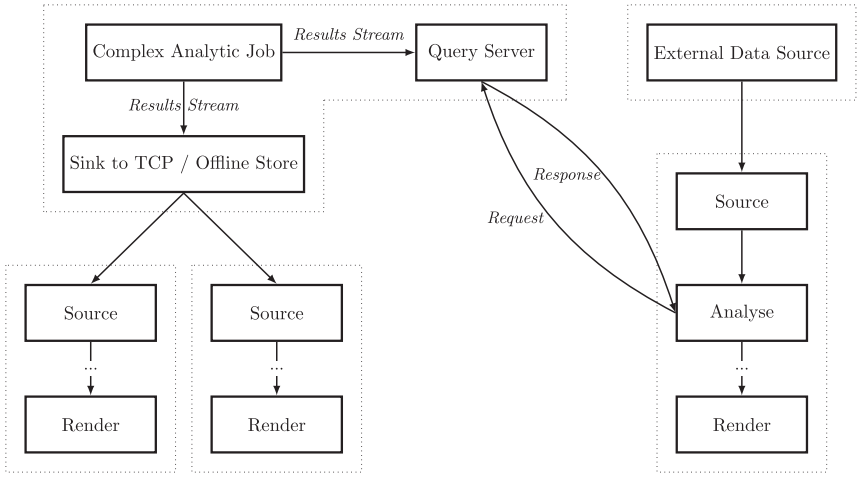
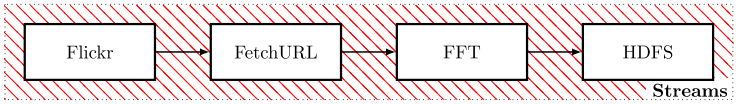
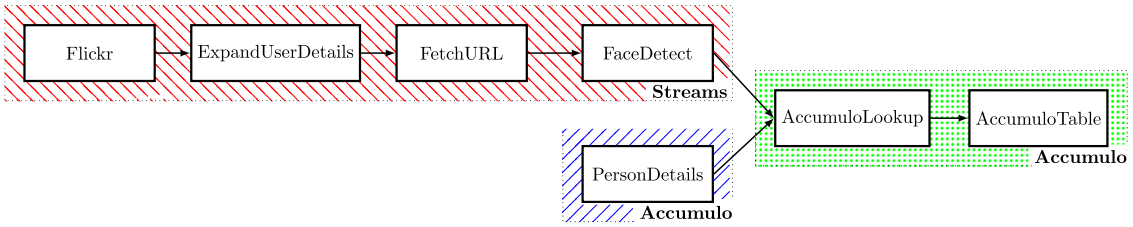


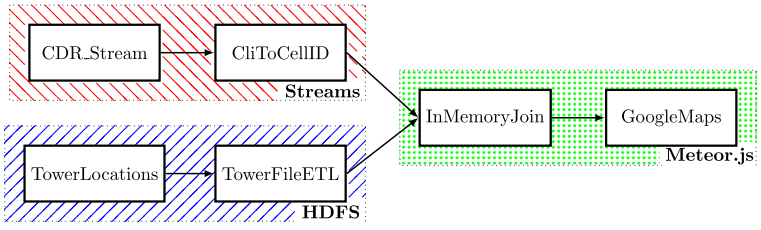
Fig. 6. Deployment scenarios for complex analytics.



(a) Flickr Photo FFTs.

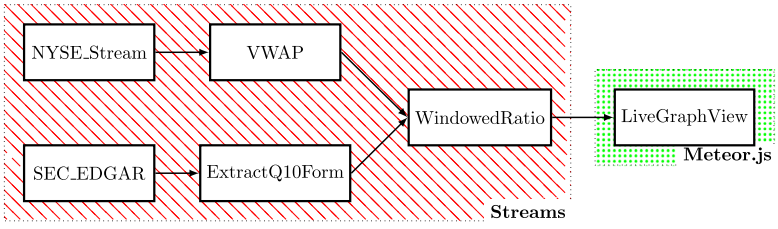


(b) Profile picture face detection.



(c) CDR call locations.

(d) Customer endpoint clustering.



(e) VWAP-earnings ratio calculation.

Fig. 7. Planned analytics for Flickr image, telecommunications data, and financial data analysis.

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009

(as described in the original example in Fig. 1). They configure MENDELEEV to search as follows:

1. PE Used: AccumuloTable
2. Types: `person`, `emailaddress`.

The user is presented with a number of analytics, but on closer inspection none of these use Flickr as a datasource. They refine their query interactively to bind the source to "Flickr". This returns a small number of candidate analytics and the user selects the version which crawls Flickr for new results using the Streams runtime (shown in Fig. 7(b)), writing results to an Accumulo table. This data is used to lookup PersonDetails from an Accumulo table in a compaction-time Accumulo Iterator. During the assembly stage, there are two image URLs to choose between; the Flickr photo and the user's profile picture. They configure the FetchURL PE to use the latter and complete their assembly.

7.3. Case study: Telecommunications call events

An analyst for a mobile telecommunications company wishes to display a live map of call events for a video wall in their Network Operations Centre. They configure the following query, which results in the analytic in Fig. 7(c):

1. PE Used: GoogleMaps
2. Types: `msisdn`,[5] `tower_latitude`, `tower_longitude`.

7.4. Case study: Telecommunications IP endpoints

A further analyst, with an interest in IP traffic and routing, wishes to determine hotspots with which their customers communicate, for both network layout purposes and to check the telecommunications company has the right peering agreements in place. They configure a query:

1. PE Used: BoundingBoxFilter, GoogleMaps
2. Types: `ipaddress`, `cluster_latitude`, `cluster_longitude`.

Their resulting analytic is shown in Fig. 7(d). However, their analytic is not fully assembled until the GeoIP PE has its `ipaddress` parameter bound to the source or destination IP. As the analyst is interested in determining the locations their connections terminate, they select the destination IP, and complete the analytic assembly. Note here that plans were additionally generated for deployment against streaming IP Netflow data, as well as this historical database of events. This is an ideal use case for a CRUCIBLE-based solution: the generated code can then simply be deployed to either their streaming or their offline platform, and the CRUCIBLE framework will select the relevant instance of the datasource.

7.5. Case study: Financial P/E analysis

An analyst at a financial institution wishes to express a common financial analytic: computing the volume-weighted average price (VWAP) for a live stock ticker, and comparing it to earnings information (available from the SEC). They configure the following query:

1. Types: `ratio<vwap, company_earnings>`
2. PE Used: LiveGraphView.

Their resulting analytic is shown in Fig. 7(e). As part of the assembly process, they configure the stock tickers they are interested in: these values are compiled directly into the resulting SPL and JavaScript.

[5] A unique telecoms subscriber identifier.

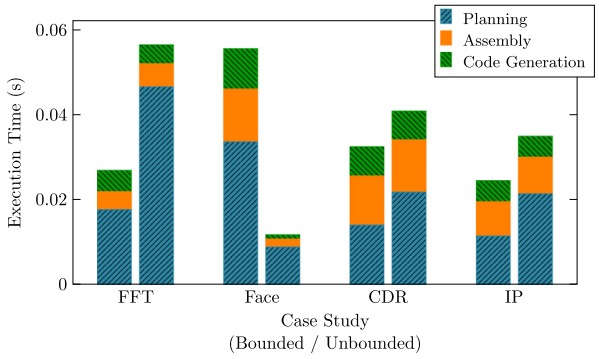


Fig. 8. Benchmark results for the MENDELEEV planner when applied to the case studies.

8. Performance evaluation

In order to better understand the performance characteristics of the MENDELEEV implementation, and thus demonstrate its viability for real-world use, two key aspects of performance are examined: (i) the time taken for the planning and assembly process and (ii) the runtime performance of its resulting output.

8.1. Planner performance

Our fifth contribution examines the performance of the planning process using the first four case studies discussed above. Each case study has been benchmarked as a bounded query (with a fixed source) and as an unbounded query (no source specified, forcing the planner to attempt to infer possible sources). The performance of the planner against a test knowledge-base of 20 PEs can be seen in Fig. 8. This test knowledge-base describes the real PEs used in the case studies described in Section 7. On average, each PE is described with 11 RDF statements, and there are 75 types described in the model. To highlight the accessibility of this approach, these planner experiments are performed on a typical workstation class machine — containing a 4-core Intel Core i7 CPU with 8 GB of RAM.

The backwards search optimisations used in the planning algorithm prevent many of the unbounded queries from taking significantly longer than their bounded equivalents. The two notable exceptions to this are in the FFT query (which does not list any grounded types in its goal to inform the choice of source), and the Face Detection query, which fails to generate a correct solution altogether in its unbounded form (but fails quickly). The bounded Face Detection query is the longest-running assembly and generation process, due to the complexity of the resulting analytic; both in terms of the number of tuple fields to be processed in the pruning analysis, and the number of PEs in the resulting analytic.

In order to better understand how the bidirectional search in the planning phase scales as the knowledge-base expands, a further set of planner benchmarks are presented for knowledge bases of varying size over both the bounded (b) and unbounded (u) query variants above. These knowledge-bases are synthetically generated, using around 100 different types from each of the domains used in the case studies above. The PEs in this expanded knowledge base are all "reachable" in the graph search, and as such could be expected to have an impact on planning time. Table 2 outlines the frequency with which unique combinations of input types and output types are repeated in the generated knowledge-base's PEs.

The results in Fig. 9 show that in scaling the size of the knowledge-base from 20 to 50 PEs there is a noticeable performance impact. However, due to the bidirectional optimisation in the search, beyond this scale there is little negative impact on the

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009

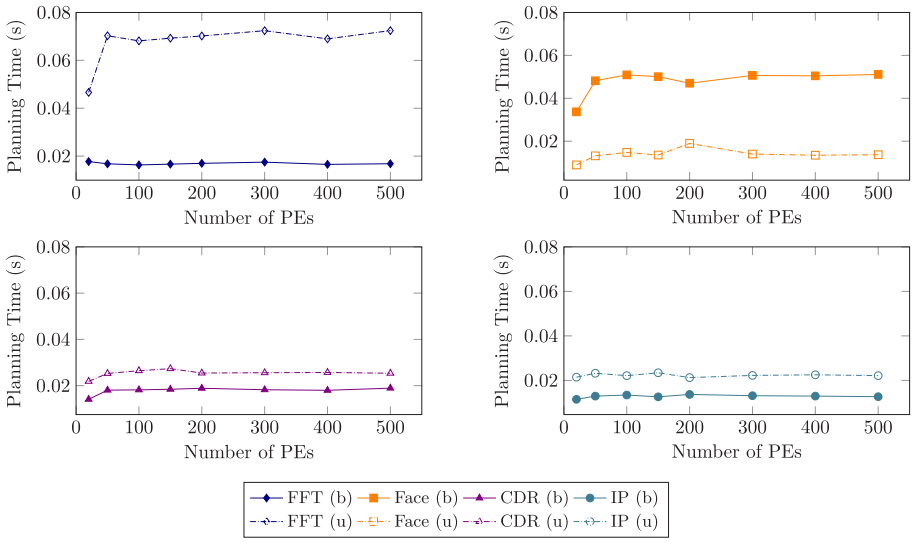


Fig. 9. Scaling of the MENDELEEV planner with knowledge-base size for both (b)ounded and (u)nbounded case studies.

Table 2
Distribution of PE input and output types in the 500 PE stress test knowledge-base.

Input		Output	
Repetition count	Percentage of KB	Repetition count	Percentage of KB
1	79.4%	1	83.6%
2	5.6%	2	2.4%
3	1.8%	3	1.8%
5	1.0%	4	1.6%
61	12.2%	53	10.6%

Table 3
Number of plans considered and returned in the 500 PE stress test knowledge-base.

Query	Plans considered	Plans returned	Planning time (s)
FFT (b)	53	9	0.017
FFT (u)	126	14	0.072
Face (b)	16	4	0.051
Face (u)	1	1	0.013
CDR (b)	40	31	0.019
CDR (u)	40	31	0.025
IP (b)	8	3	0.012
IP (u)	9	3	0.022

search time. At no point does the planning take longer than 0.08 s in the case studies tested, regardless of knowledge base size. More complete information about the number of plans considered in the search, and the number found and returned, can be seen in Table 3.

8.2. Runtime performance

It is valuable to compare the performance of MENDELEEV's generated code to hand-written analytics in both the CRUCIBLE DSL and in native code. For this, hand-written native and CRUCIBLE code for each runtime is compared to MENDELEEV, using a shared library of basic Java operations to implement two variants of the "IP Communications Endpoints" case study described above (Fig. 7(d)). In the first set of experiments, CRUCIBLE is used as the target for comparison, comparing the performance of MENDELEEV-generated CRUCIBLE code to both hand-written CRUCIBLE and native implementations. For these experiments, the full un-filtered dataset is explored. The second set of experiments compare the performance of the native code generation to hand-written native code for the bounding-box filtered analytic as described in Fig. 7(d).

These analytics were all executed against 194 offline packet capture files, corresponding to 100Gb of raw capture data (5.8 GB of packet headers). Results were collected on a test cluster consisting of three Hadoop Data Nodes/Accumulo Tablet Servers, one NameNode/Accumulo Master, and two Streams nodes. Each node hosts two 3.0 GHz Intel Xeon 5160 CPUs, 8 GB RAM and 2 × 1GbE interfaces.

8.2.1. *Unfiltered* CRUCIBLE *analysis*

Five equivalent variants of the unfiltered analytic were created: (i) MENDELEEV-generated CRUCIBLE; (ii) hand-written CRUCIBLE; (iii) a multi-threaded Java analytic; (iv) a Spark topology written in Java; and (v) an SPL topology, with associated Java primitive operators. The upper half of Table 4 shows the performance and scalability (makespan time for a given input size and latency per tuple) of the analytic on each runtime type in turn; Standalone, Apache Spark (HDFS mode) and on IBM InfoSphere Streams. These data are additionally presented in Fig. 10.

These benchmark results show that MENDELEEV's auto-generated code consistently outperforms the hand-written CRUCIBLE topology by as much as 1.4×, without any engineering expertise from the user. Due to the additional compile-time knowledge that MENDELEEV infers about the required values in the input and output tuples, it is able to prune values out of the tuples it passes resulting in more efficient message passing than in the base CRUCIBLE runtime's implementation.

An equivalent analytic, hand-written and hand-tuned for each runtime, outperforms MENDELEEV by a maximum of 1.3× in these experiments. Furthermore, the latency on a per-tuple basis remains low, with a variance of between 10^{-3} and 10^{-5}. The relative speedup of MENDELEEV to CRUCIBLE and a manually written topology on each runtime environment is detailed in Table 5.

8.2.2. *Filtered native analysis*

This final set of experiments examines the performance of the MENDELEEV-generated native code, across all three supported

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009

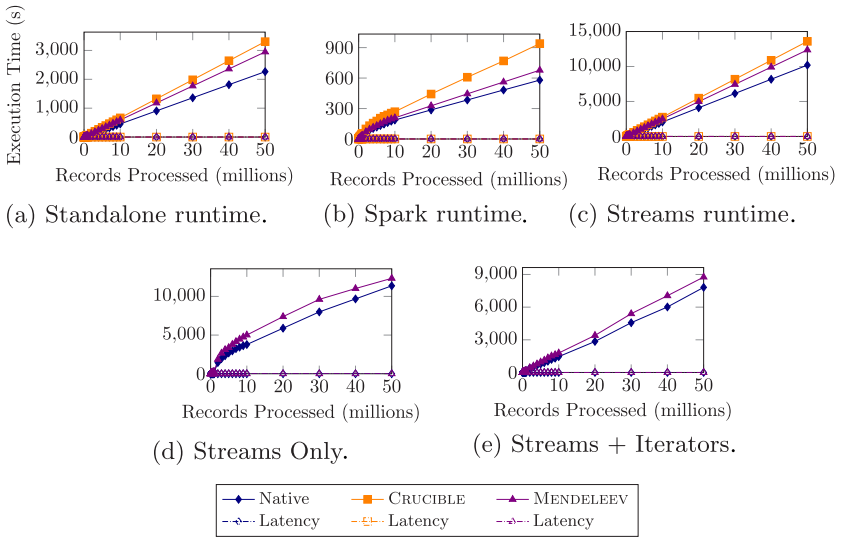


Fig. 10. Benchmarking results for each runtime mode and code type. NB: Charts (d) and (e) have no Crucible implementation.

Table 4
Benchmarking results (makespan wall time and per-tuple latency) for each runtime mode and code type.

	Code type	Records processed (millions)											
		5		10		20		30		40		50	
		Time	Latency	Time	Latency	Time	Latency	Time	Latency	Time	Latency	Time	Latency
Standalone runtime	Auto-generated DSL	296.73	0.13	591.69	0.11	1179.93	0.13	1770.09	0.11	2359.72	0.10	2948.60	0.12
	Hand-written DSL	333.53	0.16	664.23	0.16	1324.30	0.17	1983.13	0.16	2644.04	0.16	3305.23	0.16
	Hand-written Java	227.52	0.40	453.88	0.38	906.48	0.40	1360.02	0.39	1813.83	0.40	2265.44	0.38
Spark runtime	Auto-generated DSL	131.69	0.14	208.44	0.14	326.59	0.16	444.34	0.14	561.01	0.14	677.45	0.13
	Hand-written DSL	177.22	1.52	268.73	0.24	442.72	0.29	608.24	0.29	768.83	0.24	939.39	0.40
	Hand-written Spark	117.75	1.24	186.86	1.56	286.40	1.58	384.19	1.38	482.51	1.93	579.88	1.54
Streams runtime	Auto-generated DSL	1274.68	1.03	2509.74	1.09	4977.64	1.06	7443.37	1.08	9906.07	1.04	12369.67	1.00
	Hand-written DSL	1401.68	1.20	2762.88	1.18	5476.11	1.20	8181.20	1.15	10886.18	1.15	13595.48	1.14
	Hand-written SPL	1041.24	1.00	2063.17	0.98	4103.68	0.97	6143.75	1.00	8173.90	1.01	10195.93	1.01
Streams only	Auto-generated	3382.75	0.25	5007.45	0.12	7385.10	0.98	9616.30	0.19	11018.85	0.18	12338.60	0.25
	Hand-written	2691.5	0.13	3776.9	0.10	5887.65	0.12	7995.05	0.17	9677.60	0.15	11369.15	0.20
Streams + Iterators	Auto-generated	957.75	0.12	1788.00	0.08	3404.00	0.10	5395.80	0.28	7046.48	0.59	8761.28	0.13
	Hand-written	774.45	0.10	1455.70	0.07	2846.55	0.09	4569.50	0.27	6014.65	0.34	7814.35	0.11

Table 5
Relative speedup of Mendeleev to Crucible and hand-written code over Mendeleev.

Environment	Mendeleev vs Crucible	Manual vs Mendeleev
Standalone	1.12×	1.30×
Spark	1.39×	1.15×
Streams	1.10×	1.22×

Table 6
Relative speedup of hand-implemented native runtimes over Mendeleev.

Environment	Native vs Mendeleev
Streams Only	1.12×
Streams + Iterators	1.09×

runtimes simultaneously. Four variants of the filtered analytic are used: (i) Mendeleev-generated SPL, pulling all data out of Accumulo and processing it entirely in InfoSphere Streams; (ii) An equivalent Streams-only hand-written analytic; (iii) Mendeleev-generated SPL with Accumulo Iterators to perform the GeoIP and BoundingBoxFilter steps; and (iv) An equivalent hand-written Streams with Accumulo Iterators implementation. The first two variants perform the entire work of the analytic in Streams, while the latter two implementations push the GeoIP and bounding box filtering work into the Accumulo Iterator, and perform the clustering calculations in Streams.

The performance gap between the auto-generated and hand-written code is smaller here than when Crucible is used; on average, Mendeleev's code is only 1.1× slower than the equivalent hand-written implementation. The full results for both makespan and per-tuple latency are shown in the latter half of Table 4 and Fig. 10. These results are summarised in the relative speedup of Mendeleev to these hand-written implementations in Table 6.

In addition to assessing the performance of Mendeleev, these results also highlight the value of a hybrid approach to analytic execution: the hybrid Streams-Iterator approach is at least 1.5× faster than a pure streaming solution. This performance increase is not as a result of Accumulo Iterators being inherently faster than Streams, but rather through the reduction in data passed over the network, and the extra parallelism enabled by Accumulo's Iterator execution model.

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009

9. Conclusions & further work

In this paper we document: (i) A new abstract model for the assembly and execution of hybrid analytics, based on a semantically rich type system; (ii) A novel approach to goal-based planning using this model, which requires little engineering expertise from the user; (iii) A mechanism for performant, scalable code generation for these analytics, integrating data across heterogeneous on- and off-line platforms; (iv) An implementation through a system which we call MENDELEEV and (v) demonstration of the applicability of this technique through a series of case studies, where a single interface is used to create analytics that can be run simultaneously over on- and off-line environments; (vi) Performance benchmarking that shows that MENDELEEV-generated analytics offer runtime performance comparable with hand-written code.

Crafting scalable analytics in order to extract actionable business intelligence is challenging. It requires both domain-level and technical expertise, experience of tuning and scaling, and supporting tools for analytic composition, planning, code-generation and effective deployment. Few frameworks exist that provide end-to-end solutions that address these challenges.

The research presented in this paper builds on the wishful-search concept behind MARIO, yet at the same time allows the discovery and composition of novel analytics. We believe it is the first approach to target the execution of automatically generated hybrid analytics in heterogeneous compute environments. The performance penalty over hand-written and tuned analytics has been shown to be a maximum of 1.3× in our experiments, which we believe is an acceptable cost for an automated framework of this type.

There are a number of avenues to be explored in future work. One promising area of research is in the automated learning of analytic design patterns. As a MENDELEEV instance is deployed over an extended period of time, analysis of usage patterns may permit the system to recommend to the user analysis for a given data source, or to alter rankings based on those analytics users typically deploy for a given query.

It would additionally be valuable to investigate an analytic design approach with a shorter gap between generating and validating an analytic, by demonstrating an example set of results a user can expect to receive from candidate analytics before the assembly is completed. This would necessitate some engineering around the automated compilation and deployment of analytics in an interactive timeframe, but could significantly aid a user's understanding of available plans.

Finally, there are currently a limited set of pre- and post-conditions used to influence the planning process. We propose modelling more advanced primitives (*e.g.*, reductions or filters) using these conditions, and exploring their impact on both the usability and the expressivity of MENDELEEV. In a related vein, while it will likely never be possible to fully remove human intervention from the process of generating these annotations, one interesting avenue for further research would be the application of machine learning classification techniques to the automated extraction of the PE knowledge base. One of the key challenges with this would be the automated creation of a coherent ontology representing the domain in which the PEs function.

Acknowledgments

This research was funded by a UK Engineering and Physical Sciences Research Council (EPSRC) (EP/K503204/1) Industrial CASE Studentship, entitled "Platforms for Deploying Scalable Parallel Analytic Jobs over High Frequency Data Streams". Compute platforms are provided through an EPSRC Capital Equipment Grant, "Provision of a Portfolio of Massively Parallel, Data-intensive Analytics Platforms". The Alan Turing Institute is a joint venture between the universities of Cambridge, Edinburgh, Oxford, Warwick, University College London and the Engineering and Physical Sciences Research Council. The Institute will promote the development and use of advanced mathematics, computer science, algorithms and big data for human benefit.

References

[1] M. Altinel, P. Brown, S. Cline, R. Kartha, E. Louie, V. Markl, L. Mau, Y.-H. Ng, D. Simmen, A. Singh, Damia: A data mashup fabric for intranet applications, in: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB Endowment, San Jose, CA, USA, 2007, pp. 1370–1373.
[2] Apache Software Foundation, Apache Storm, http://storm.apache.org/, accessed 2015.
[3] R. Bergmann, Y. Gil, Retrieval of semantic workflows with knowledge intensive similarity measures, in: Case-Based Reasoning Research and Development, Springer, New York, NY, USA, 2011, pp. 17–31.
[4] M. Birbeck, S. McCarron, CURIE Syntax 1.0: A syntax for expressing Compact URIs, W3C Working Group Note.
[5] G. Bracha, Generics in the Java programming language, Sun Microsystems, java.sun.com.
[6] Cascading Project, Cascading—Platform for Big Data, URL: http://www.cascading.org/, accessed 2015.
[7] P. Coetzee, S. Jarvis, Goal-based analytic composition for on- and off-line execution at scale, in: Proceedings of IEEE Trustcom/BigDataSE/ISPA, 2015, vol. 2, IEEE, Helsinki, Finland, 2015, pp. 56–65.
[8] P. Coetzee, M. Leeke, S. Jarvis, Towards unified secure on-and off-line analytics at scale, Parallel Comput. 40 (10) (2014) 738–753.
[9] I. Constantinescu, B. Faltings, W. Binde, Large scale, type-compatible service composition, in: Proceedings of the IEEE International Conference on Web Services, IEEE, San Diego, CA, USA, 2004, pp. 506–513.
[10] F. Daniel, C. Rodríguez, S. Roy Chowdhury, H.R. Motahari Nezhad, F. Casati, Discovery and reuse of composition knowledge for assisted mashup development, in: Proceedings of the 21st International Conference on World Wide Web, ACM, Lyon, France, 2012, pp. 493–494.
[11] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[12] Google, Inc., Google Mashup Editor, URL: https://developers.google.com/mashup-editor/, accessed 2015.
[13] M. Hausenblas, J. Nadeau, Apache drill: Interactive Ad-Hoc analysis at scale, Big Data 1 (2) (2013) 100–104.
[14] M. Islam, A.K. Huang, M. Battisha, M. Chiang, S. Srinivasan, C. Peters, A. Neumann, A. Abdelnur, Oozie: Towards a scalable workflow management system for Hadoop, in: Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, ACM, Scottsdale, AZ, USA, 2012, p. 4.
[15] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts, S. Zdonik, Towards a streaming SQL standard, Proc. VLDB Endowment 1 (2) (2008) 1379–1390.
[16] M. Kornacker, J. Erickson, Cloudera Impala: Real-time queries in Apache Hadoop, 2012. URL: http://blog.cloudera.com/blog/2012/10/cloudera-impala-real-time-queries-in-apache-hadoop-for-real/.
[17] O. Lassila, R. Swick, et al. Resource Description Framework (RDF) model and syntax specification, W3C Recommendation.
[18] Y.-N. Law, H. Wang, C. Zaniolo, Query languages and data models for database sequences and data streams, in: Proceedings of the 30th International Conference on Very Large Data Bases, VLDB Endowment, San Jose, CA, USA, 2004, pp. 492–503.
[19] T. Loton, Introduction to Microsoft Popfly, No Programming Required, Lotontech Limited, 2008.
[20] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, et al., OWL-S: Semantic markup for web services, W3C member submission 22 (2004) 2007–04.
[21] D.S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, F. Yaman, SHOP2: An HTN planning system, J. Artificial Intelligence Res. 20 (1) (2003) 379–404.
[22] C. Ogbuji, et al. FuXi 1.4: A Python-based, bi-directional logical reasoning system for the semantic web, URL: https://code.google.com/p/fuxi/, accessed 2015.
[23] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, et al., Taverna: A tool for the composition and enactment of bioinformatics workflows, Bioinformatics 20 (17) (2004) 3045–3054.
[24] C. Olston, B. Reed, U. Srivastava, R. Kumar, A. Tomkins, Pig Latin: A Not-So-Foreign language for data processing, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, Vancouver, BC, Canada, 2008, pp. 1099–1110.
[25] M. Pistore, P. Traverso, P. Bertoli, A. Marconi, Automated synthesis of composite BPEL4WS web services, in: Proceedings of the IEEE International Conference on Web Services, IEEE, Orlando, FL, USA, 2005, pp. 293–301.
[26] M. Pruett, Yahoo! Pipes, first ed., O'Reilly, Sebastopol, CA, USA, 2007.
[27] R. Rea, K. Mamidipaka, IBM InfoSphere Streams: Enabling complex analytics with ultra-low latencies on data in motion, IBM White Paper.

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009

[28] C.A.R. Hoare, Communicating sequential processes, Commun. ACM 21 (8) (1978) 666–677.
[29] A.V. Riabov, E. Boillet, M.D. Feblowitz, Z. Liu, A. Ranganathan, Wishful search: Interactive composition of data mashups, in: Proceedings of the 17th International Conference on World Wide Web, ACM, Beijing, China, 2008, pp. 775–784.
[30] A. Riabov, Z. Liu, Planning for stream processing systems, Proc. AAAI Natl. Artif. Intell. 20 (3) (2005) 1205.
[31] A. Riabov, Z. Liu, Scalable planning for distributed stream processing systems, in: Proceedings of the International Conference on Automated Planning and Scheduling, AAAI Press, Pittsburgh, PA, USA, 2006, pp. 31–41.
[32] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, in: Proceedings of the 26th Symposium on Mass Storage Systems and Technologies, MSST, IEEE, Lake Tahoe, NV, USA, 2010, pp. 1–10.
[33] E. Sirin, B. Parsia, Planning for semantic web services, in: Semantic Web Services Workshop at 3rd International Semantic Web Conference, Springer, Hiroshima, Japan, 2004, pp. 33–40.
[34] K.T. Stolee, S. Elbaum, et al., Solving the search for source code, ACM Trans. Softw. Eng. Methodol. 23 (3) (2014) 26.
[35] T. Veldhuizen, Expression templates, C++ Report 7 (5) (1995) 26–31.
[36] K. Whitehouse, F. Zhao, J. Liu, Semantic streams: A framework for composable semantic interpretation of sensor data, in: Wireless Sensor Networks, Springer, Banff, Alberta, Canada, 2006, pp. 5–20.
[37] M.D. Wilkinson, M. Links, BioMOBY: An open source biological web services proposal, Brief. Bioinform. 3 (4) (2002) 331–341.
[38] J. Wong, Marmite: Towards end-user programming for the web, in: IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE, Coeur d'Alene, ID, USA, 2007, pp. 270–271.
[39] R. Xin, J. Rosen, et al., Shark: SQL and rich analytics at scale, in: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 2013, pp. 13–24.
[40] J. Yu, B. Benatallah, F. Casati, F. Daniel, Understanding mashup development, Internet Computing, IEEE 12 (5) (2008) 44–52.
[41] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets, in: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, vol. 10, USENIX Association, Boston, MA, USA, 2010, p. 10.

Peter Coetzee is a Ph.D. student in the Performance Computing and Visualisation Group at the University of Warwick. His research focuses on tools to enable domain experts to explore high-frequency data streams using appropriate analysis and visualisation tools, both in near-real-time as well as over historical data. His research also examines issues around deployment of these user-defined analyses; selection of suitable architectures, and acceleration techniques to achieve maximum throughput from high-volume data sources. Peter graduated from Imperial College London in 2010 with a MEng. (Hons) in Computing, gaining a few years of industrial experience in an R&D department for secure communications systems before starting his Ph.D.

Stephen Jarvis is Professor of High Performance Computing at the University of Warwick and co-organiser for one of the UK's High End Scientific Computing Training Centres. He has authored more than 130 refereed publications (including three books) and has been a member of more than 50 programme committees for IEEE/ACM international conferences and workshops since 2003, including: IPDPS, HPDC, CCGrid, SC, DSN, ICPP. He is a former member of the University of Oxford Computing Laboratory, and in 2009 was awarded a prestigious Royal Society Industry Fellowship in support of his industry-focused work on HPC.

Please cite this article in press as: P. Coetzee, S.A. Jarvis, Goal-based composition of scalable hybrid analytics for heterogeneous architectures, J. Parallel Distrib. Comput. (2016), http://dx.doi.org/10.1016/j.jpdc.2016.11.009