Demonstration of the Myria Big Data Management Service

Daniel Halperin, Victor Teixeira de Almeida, Lee Lee Choo, Shumo Chu,
Paraschos Koutris, Dominik Moritz, Jennifer Ortiz, Vaspol Ruamviboonsuk, Jingjing Wang,
Andrew Whitaker, Shengliang Xu, Magdalena Balazinska, Bill Howe, and Dan Suciu
Computer Science & Engineering and eScience Institute, University of Washington

ABSTRACT

In this demonstration, we will showcase Myria, our novel cloud service for big data management and analytics designed to improve productivity. Myria's goal is for users to simply upload their data and for the system to help them be self-sufficient data science experts on their data – *self-serve analytics*. From a web browser, Myria users can upload data, author efficient queries to process and explore the data, and debug correctness and performance issues. Myria queries are executed on a scalable, parallel cluster that uses both state-of-the-art and novel methods for distributed query processing. Our interactive demonstration will guide visitors through an exploration of several key Myria features by interfacing with the live system to analyze big datasets over the web.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—Distributed databases, Query processing, Relational databases

Keywords

Myria; data management; distributed database; query visualization

1. INTRODUCTION

Over the past year and a half, our group at the University of Washington has developed a new online service for managing big data. Our service, called Myria, now runs on 100-node Amazon EC2 deployments and processes terabytes of data from applications in astronomy, oceanography, social media, and cybersecurity, as well as standard benchmarks. More importantly, Myria is set up as a cloud service that users access directly from their browsers, dramatically reducing the "activation energy" required to be productive with big data. This demonstration will showcase the overall Myria service and system and several of its specific novel features.

Myria motivation: The analysis of massive-scale datasets has become an important capability both in industry and in the sciences and many systems have recently emerged to support it. But deep data analytics today is a high-touch business: it requires a highly specialized expert who thoroughly understands both the application domain and a growing ecosystem of complex distributed systems and advanced statistical methods, and who needs to perform repeatedly a series of data exploration steps to prepare the data for machine learning and other analyses. Myria aims to make data management and analysis a pleasant and productive experience. Our project started from a re-examination of the foundations of big data management with the ultimate goal of significantly improving *productivity* in big data management and analytics.

How can we improve productivity in the analysis of big data? We identify three key challenges that need to be overcome, which, in ensemble, are not met by any system available today, and we describe how Myria tackles these challenges head on.

- **Cloud service deployment.** For many of today's systems, users must still purchase and administer physical or virtual computers; even with a pre-configured cluster, the setup instructions on the Hadoop, Spark, or GraphLab project websites take hours to days to execute successfully the first time. For many users, the complexity and cost of startup and continual administration and maintenance are prohibitive. Some data management systems are also available as cloud services, but these services often offer obscure black-boxes that make it difficult for users to predict and debug performance or to predict and control costs.

In contrast to existing systems, Myria is designed as a cloud service from the ground up. The primary interface is a web browser that uses the same programmatic REST interface as any low-level system tools. From innovative service level agreements and resource management, to multi-language query support, to graphical query debugging, Myria reinvents the interface between users and cloud services. Myria's goal is for users to simply upload their data and for the system to help them be self-sufficient data science experts on their data – *self-serve analytics*.

- **Programmability.** SQL is perceived as ill-suited for analytics tasks, motivating a number of extensions — window functions, pivoting, UDAs, UDFs, in-database analytics packages such as MADlib, etc. But analytics remain a second-class feature in these RDBMS-based solutions, requiring significant expertise by both users and algorithm designers. Hadoop, GraphLab, Spark, and related systems require users to develop algorithms in low-level imperative languages, reducing opportunities for algebraic optimization and reuse. Language layers on top of these runtimes (Pig, Hive, Shark, etc.) limit expressiveness to that of the language.

Myria strikes a balance between these extremes: we adopt a core programming model that extends relational algebra with iteration that affords rich, iteration-aware optimization without sacrificing expressive power. Guided by prior experience in delivering database-as-a-service capabilities to scientists [3], we aim to support both "users" and "algorithm designers" with a common set of web-based interfaces, languages, and APIs that scale gracefully from simple SPJ queries to advanced application-specific analytics tasks. Like Hyracks, we emphasize the use of core parallel query processing concepts as a first-class concern, but we place less emphasis on supporting legacy code written for Hadoop or Pregel and more emphasis on empowering non-specialists, especially scientists. Myria currently supports SQL, a variant of Datalog, and a new hybrid declarative/imperative analytics language called MyriaL.

- **Efficient processing.** Finally, users want good performance, so our system must be able to optimize and scale queries. Myria puts a strong emphasis on efficient query processing, by combining state-of-the-art and novel ideas in both theory and systems.

With its foundation in relational algebra, we understand well how to optimize and parallelize many Myria queries. However, opti-

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2376-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2588555.2594530.

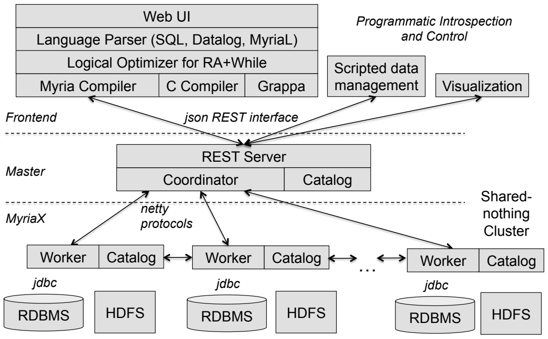


Figure 1: Myria System Architecture

mizations for queries that combine basic algebra, recursion, and aggregations are still not well understood. The Myria team is developing new techniques for large distributed query processing [2], and we are validating and leveraging them using the system itself.

For scalability and efficiency, Myria's execution layer, MyriaX adopts state-of-the-art system design principles. MyriaX uses a parallel, pipelined, possibly cyclic graph of dataflow operators with built-in support for asynchronous evaluation of recursive queries. Our research extends the system in many important directions including lightweight failure handling in iterative computations and dynamic resource allocation for elastic scalability.

In this demonstration, we will showcase Myria's capabilities both as a cloud service and a big data management system. We will do so through a coordinated set of high-level and low-level demonstrations on multiple laptops. The demonstrations will use a 64-instance Myria deployment and be driven by real queries from domain scientists in astronomy, oceanography, and social science.

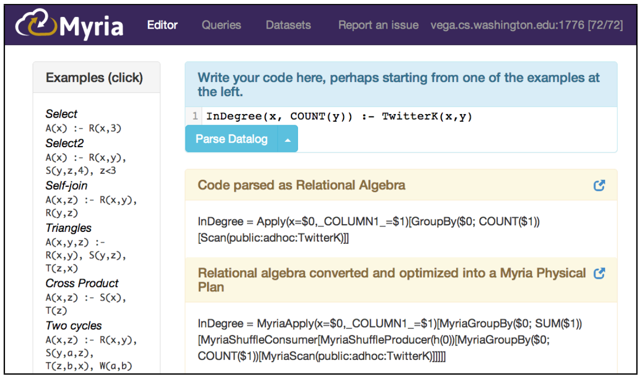
2. MYRIA OVERVIEW

In this section, we present a brief overview of Myria as context for understanding the novel features we will demonstrate, which are described in subsequent sections. Figure 1 shows the overall architecture of the Myria system.

The primary interface to Myria is a frontend website hosted on Google App Engine, with software components as shown above the dashed line in Figure 1. The screenshot in Figure 2 shows the main query editor. For our supported languages, Myria provides a set of sample queries (left) as a starting point for query authoring (right top). Myria parses the query to basic relational algebra (right middle) and produces an optimized, parallel, distributed physical plan for a shared-nothing cluster (right bottom). The user can visualize a query plan via the pop-out links, and can submit the query. Other panels let users examine datasets in the system, previously submitted and running queries, and provide other features.

The center component in Figure 1 is the master. It hosts the REST interface used by the web UI and other tools, maintains metadata about system resources and datasets (e.g., used by the query optimizer and web UI), and it mediates access to the cluster.

The bottom part of Figure 1 shows the architecture of Myria's query execution engine, MyriaX. MyriaX is a pipelined, parallel, distributed dataflow evaluation system that takes a (possibly cyclic) graph of operators and executes it efficiently on a shared-nothing cluster. This pipeline is divided into fragments, which are trees of operators that can operate on a single compute node in a single thread; fragment leaves and roots are typically system I/O operators that read, write, or transfer data. MyriaX can incorporate data from many sources, such as relational DBMS, Hadoop FS, Amazon S3, public data on the Internet, etc. Additionally, MyriaX can scale the cluster up and down dynamically as the system workload changes.



Write your code here, perhaps starting from one of the examples at the left.

Select
```
A(x) :- R(x,3)
```
Select2
```
A(x) :- R(x,y),
S(y,z,4), z<3
```
Self-join
```
A(x,z) :- R(x,y),
R(y,z)
```
Triangles
```
A(x,y,z) :-
R(x,y), S(y,z),
T(z,x)
```
Cross Product
```
A(x,z) :- S(x),
T(z)
```
Two cycles
```
A(x,z) :- R(x,y),
S(y,a,z),
T(z,b,x), W(a,b)
```

```
InDegree(x, COUNT(y)) :- TwitterK(x,y)
```

Code parsed as Relational Algebra

```
InDegree = Apply(x=$0,_COLUMN1_=$1)[GroupBy($0; COUNT($1))
[Scan(public:adhoc:TwitterK)]]
```

Relational algebra converted and optimized into a Myria Physical Plan

```
InDegree = MyriaApply(x=$0,_COLUMN1_=$1)[MyriaGroupBy($0; SUM($1))
[MyriaShuffleConsumer[MyriaShuffleProducer(h(0))[MyriaGroupBy($0;
COUNT($1))[MyriaScan(public:adhoc:TwitterK)]]]]]
```

Figure 2: Myria Browser-based Front-End

3. DEMONSTRATED FEATURES

We divide our demonstrations of Myria features into three phases: 1) Myria to its users, 2) Myria's execution engine and algorithms, and 3) Myria as a service. Combined, these features show how Myria comprises a system designed from the ground up to improve productivity for both nascent and experienced data scientists (1 and 2) in an efficient and scalable, and hence sustainable way (2 and 3).

3.1 Myria to its Customers

In this phase of the demonstration, the visitor will act as a scientist analyzing a big dataset.

3.1.1 How users access Myria

The website described in §2 is the principal way that most users will access Myria, but this is not the only way to use Myria. We designed the REST interface to Myria—the same interface used by the website—to be sufficiently flexible to support an entire ecosystem of clients that connect with Myria programmatically. Supplemental clients provide tools for: data management tasks such as uploading a folder of related datasets and automatically building a union for entire-corpus analysis; downloading data for offline analysis in domain-specific tools; and exploring data visually.

3.1.2 Supported query languages

Myria accepts queries written in SQL, Datalog, or a new Pig Latin-like language we call MyriaL. Based on our experience with SQLShare [3], we believe that science users can write data analysis tasks in SQL. We expect Datalog's declarative style to have similar appeal, especially for recursive queries. Myria's Datalog compiler has support for stratified negation and a variety of simple aggregates. Myria uses semi-naive evaluation to efficiently compute recursive results, using asynchronous computation when possible.

MyriaL is a hybrid imperative/declarative language, similar to Pig Latin [7] extended with iteration. We developed MyriaL because certain iterative programs—in which facts from one iteration are aggregated to produce the facts in the next iteration, e.g., PageRank—are cumbersome to express in Datalog and hard for the query execution engine to optimize. MyriaL lets users directly express efficient execution strategies, such as custom variants of standard semi-naive evaluation that overwrite facts rather than accumulate them, when aggregation is used (e.g., for connected components). Users can also encode optimizations that incorporate domain-specific knowledge. Each line of a MyriaL program corresponds to a declarative query comprising one or more relational algebra operations, expressed in a comprehension syntax. MyriaL iteration employs a simple do/while construct, in which the continuation condition is a singleton boolean relation. Figure 3 shows the MyriaL program for semi-naive evaluation of graph reachability.

```
Edge = SCAN(user@uw.edu:edges_table);
Reachable = [1 AS addr]; Delta = Reachable;
DO
  NewNodes = [FROM Delta, Edge WHERE Delta.addr = Edge.src
              EMIT Edge.dst AS addr];
  Delta = DIFF(DISTINCT(NewNodes), Reachable);
  Reachable = UNIONALL(Delta, Reachable);
WHILE [*COUNTALL(Delta) > 0];
```

Figure 3: Graph reachability in MyriaL.

3.1.3 What the user will see and do

For the demonstration, Myria will be pre-loaded with scientific datasets including an n-body simulation that models matter in the evolving early universe, flow cytometry measurements of ocean microorganisms, and the Twitter social network graph [5]. The user can tour the datasets in the system and learn their relationships by inspecting provenance trees generated from the Myria query log.

The user will be presented with sample queries in the three languages. The system will automatically translate user-generated SQL or Datalog queries into MyriaL to aid in comprehension.

Finally, the user will examine the standard sigma-clipping outlier removal algorithm [6] to see how it admits domain-specific algorithmic optimizations for incremental evaluation. Specifically, we will guide the user through authoring a sequence of MyriaL programs which refine the algorithmic and relational algebra algorithms for noticeable, orders-of-magnitude speedups in query processing time.

3.2 Myria Engine and Algorithms

In the second phase of the demonstration, the visitor will learn about Myria's query execution and monitoring features.

3.2.1 Query Execution in MyriaX

As described in §2, Myria's physical query plans take the form of pipelined, possibly cyclic, dataflow graphs of operators. In Myria's execution engine, MyriaX, data flow between operators takes the form of batches of tuples, aggregated to amortize per-operator-invocation overheads. MyriaX uses both standard techniques and novel algorithms for evaluation of relational algebra. Execution proceeds in parallel both within a node—multiple independent or pipelined parts of a query plan can be executed concurrently in separate threads—and across nodes, in the common case of data-level parallelism in query evaluation.

MyriaX natively supports iterative and recursive computations. For some classes of programs (e.g., reachability, connected components, or PageRank), MyriaX provides asynchronous query processing capabilities using incremental variants of physical operators with stream-oriented computation. For the general case, iterative queries are optimized and executed as a series of synchronous jobs.

3.2.2 Query Monitoring and Debugging

Myria provides extensive visual query monitoring and debugging functionalities to aid in self-serve data exploration. Myria uses modern data visualization techniques built on d3.js combined with Myria's analytics capabilities to let users interactively explore query execution profiles. Our in-browser tool provides a suite of visualizations to enable query execution profiling from various perspectives; Figure 4 shows an example Myria visualization that shows the execution of a parallel join. We found these views to be critical for performance debugging: (1) The *query plan* visualization shows a line chart of cluster utilization for each query fragment, measured as the fraction of workers executing it. In this view, a long tail identifies a query bottleneck. (2) The user can visualize *communication* between workers in a communication matrix and request more details on the communication between pairs of workers over time. (3) Finally, the *operator level* visualization (Figure 4) shows a Gantt chart of operators executing a specific query fragment across workers. Operators of the specified fragment are shown in unique colors in separate lanes for each worker. This view serves to identify or explain low-level implementation problems.

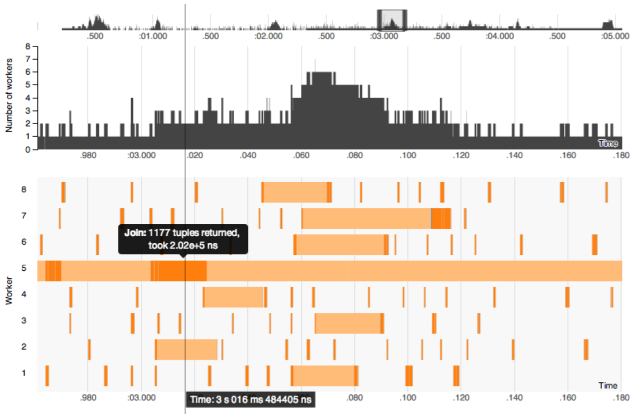


Figure 4: Visualization of 100 ms (x-axis, zoomed via the minimap on top) of a query fragment performing a join in parallel on 8 workers. An operator is visible in the bottom chart if it is computing at a point in time on the specified worker.

3.2.3 Distributed Join Algorithms

In Myria, we are pushing the bounds of algorithmic understanding of efficient distributed query execution and putting these ideas into practice in our optimizer and executer.

To illustrate these novel techniques [1, 2], consider triangles: `Triangles(x,y,z) :- R(x,y), S(y,z), T(z,x).` A typical parallel database might decompose the computation into two pipelined hash-joins:

`A(x,y,z) = R(x,y)` $\bowtie_y$ `S(y,z)` and then
`Triangles(x,y,z) = A(x,y,z)` $\bowtie_{xz}$ `T(z,x).`

However, such a query plan is highly inefficient if the intermediate result A is large, since A is shuffled and sent to the next join. For the Twitter follower graph [5], A contains 9 trillion tuples.

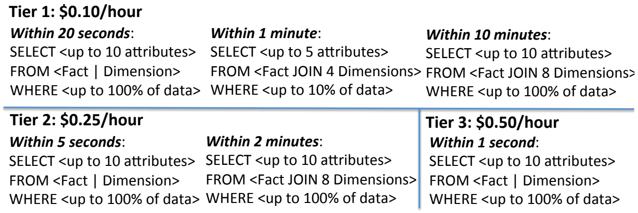
Myria can instead apply the HYPERCUBE algorithm [2], which computes the triangles using a *single* shuffling step. The algorithm assigns each worker node a virtual coordinate in a 3-dimensional hypercube, (p_x, p_y, p_z), and hashes each tuple involving x, y, or z to the corresponding space. $R(x, y)$ is sent to the xy subspace containing all nodes of the form $(h(x), h(y), *)$. Correspondingly, the tuples of S and T are sent to the yz and xz subspaces. Each node finally performs a local join to discover those triangles (x, y, z) that are hashed to its coordinates (p_x, p_y, p_z). Myria uses a simplified Leapfrog Triejoin [9] to compute triangles locally in a single multiway join step without computing the large intermediate results.

This single-step algorithm transfers only the input datasets rather than large intermediate results, at the cost of data replication: tuples are hashed to multiple workers, whereas the pipelined hash-join approach sends each tuple to only one worker. For skewed datasets, this replication has further gains, because "hot" vertices with high degree are balanced across multiple workers.

Myria optimizes for the right combination of join techniques given the input data statistics and the structure of the joins.

3.2.4 Failure Handling During Iteration

MyriaX provides intra-query failure handling: During query execution, without blocking the main data flow, each worker records the data sent to downstream workers in an in-memory buffer, spilling to disks if necessary. When a worker fails, the master starts a recovery worker as the replacement. The live workers send the buffered out-



Tier 1: $0.10/hour

Within 20 seconds:
SELECT <up to 10 attributes>
FROM <Fact | Dimension>
WHERE <up to 100% of data>

Within 1 minute:
SELECT <up to 5 attributes>
FROM <Fact JOIN 4 Dimensions>
WHERE <up to 10% of data>

Within 10 minutes:
SELECT <up to 10 attributes>
FROM <Fact JOIN 8 Dimensions>
WHERE <up to 100% of data>

Tier 2: $0.25/hour

Within 5 seconds:
SELECT <up to 10 attributes>
FROM <Fact | Dimension>
WHERE <up to 100% of data>

Within 2 minutes:
SELECT <up to 10 attributes>
FROM <Fact JOIN 8 Dimensions>
WHERE <up to 100% of data>

Tier 3: $0.50/hour

Within 1 second:
SELECT <up to 10 attributes>
FROM <Fact | Dimension>
WHERE <up to 100% of data>

Figure 5: A PSLA only shows improvements across tiers

going information to the new worker, and the new worker catches up using these data directly.

Most modern engines provide this or other failure-handling methods during query execution. The novelty in Myria lies in our study of failure-handling methods for iterative queries. Iterative applications have special properties that we can exploit: First, data often includes inherent structural redundancy. For instance, we can discover that a vertex is reachable in the presence of data loss as long as any incoming edge from any reachable neighbor is known. Second, many iterative computations incrementally build or refine their states and outputs. Together, these factors facilitate state compression and accelerate recomputation using more recent data when state is lost. Myria exploits these properties when possible.

Depending on the iterative query, Myria uses different state buffering strategies: 1) Simple buffers (default): keep data in FIFO queues. 2) Replacement buffers: In many iterative computations, new messages overwrite or update old ones—stale labels in connected components—so only updated results need be retained. 3) Prioritized buffers: By ordering messages with a query-specific objective function, MyriaX can propagate "good" tuples (e.g., weights of high-degree vertices in PageRank) earlier when recovering, leading to faster convergence.

3.2.5 What the user will see and do

The user will begin with a triangular query on a Twitter dataset representing the following-followee relation. The user will choose between two query plans for the query: (1) a traditional query plan consisting of two pipelined distributed hash joins, and (2) Myria's novel query plan consisting of one HyperCube shuffle step and a local multiway join step. Myria's visualizer will serve to compare and analyze the performance of the two query plans.

As a second step, the user will execute a connected-components query over the Twitter dataset using either synchronous or asynchronous iterative processing. The user will be invited to kill a worker during the query execution and see how the system recovers automatically under different fault-tolerance strategies. The focus will be on the performance implications of each combination of failure-handling and execution methods.

3.3 Myria as a Service

We aim to run Myria as a sustainable, publicly available resource for big data analytics. This goal leads us to a research agenda in cost-effective cluster management and better customer-provider relationships, which we will demonstrate in the third phase.

3.3.1 Personalized Service Level Agreements

We have developed a new type of relationship between customers and cloud service providers, the *personalized* service level agreement (PSLA) [8] illustrated in Figure 5. The goal of a PSLA is to switch from a resource-centric approach—in which users lease CPU, memory, disk, or network resources—and instead focus on allowing the user to think directly about query capabilities (in the form of query templates) and performance versus service price. Given a user's data, our approach automatically generates sample queries and a PSLA. The key challenge is to generate PSLAs that are both concise (few tiers with few query templates) and precise (time thresholds are close to expected query runtimes).

3.3.2 Elastic Scalability

Myria can scale its resource usage and performance as query workloads change and to satisfy its customer PSLAs. First, Myria can dynamically add or remove *compute nodes*—worker instances that process queries, but do not store data. Compute nodes can be added or removed from query processing even while queries execute. Second, data in Myria can be partitioned across *storage nodes* using consistent hashing [4]. This enables Myria to rebalance storage workloads with minimal data transfer when *storage nodes* are added or removed. Third, consistent hashing is also used for data *replication* in Myria. Replicas provide resilience to storage node loss and increase the system's overall data parallelism by providing concurrent reads and distributed writes. Our research focuses on how best to leverage these techniques when operating Myria as a service in a public cloud.

3.3.3 What the user will see and do

The user will start by exploring the N-body universe simulation dataset. N-body simulations are widely used tools in astrophyics, that help astronomers understand how gravity influences dark, star and gas particles across time. The user will be able to select any subset of tables from the dataset and change the table sizes (different simulations produce different dataset sizes). She will then observe how changing the input changes the generated PSLA. Finally, the user will be able to execute queries at the different service tiers in the PSLA and will compare the observed performance across these tiers. As the queries execute, we will also demonstrate how Myria can add and remove data and compute resources on the fly and how doing so affects performance.

4. CONCLUSION

Our Myria service aims to improve productivity for data scientists by accelerating the bottleneck steps in data management and exploration, and to make these capabilities accessible to a new class of users – *self-serve analytics*. This demonstration shows the three key aspects of Myria: Myria's interface to users, its system internals, and its as-a-service features.

Acknowledgments. This work was supported in part by the National Science Foundation grant IIS-1247469 and by the Intel Science and Technology Center for Big Data. We thank Ginger Armbrust, Sophie Clayton, Fabio Governato, Sarah Loebman, Tom Quinn, and Francois Ribalet for providing scientific data and queries.

5. REFERENCES

[1] F. N. Afrati and J. D. Ullman. Optimizing joins in a Map-Reduce environment. In *EDBT*, 2010.

[2] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. In *PODS*, pages 273–284, 2013.

[3] B. Howe et al. Collaborative science workflows in SQL. *Computing in Science & Engineering, Special Issue on Science Data Management*, 15(2), May/June 2013.

[4] D. Karger et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *STOC*, 1997.

[5] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *Intl. Conference on World Wide Web (WWW)*, 2010.

[6] M. Moyers et al. A demonstration of iterative parallel array processing in support of telescope image analysis. *PVLDB*, 6(12):1322–1325, 2013.

[7] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A not-so-foreign language for data processing. In *SIGMOD*, 2008.

[8] J. Ortiz, V. Teixeira de Almeida, and M. Balazinska. A vision for personalized service level agreements in the cloud. In *SIGMOD DanaC Workshop*, 2013.

[9] T. L. Veldhuizen. Leapfrog Triejoin: a worst-case optimal join algorithm. *CoRR*, abs/1210.0481, 2012.