The Myria Big Data Management and Analytics System and Cloud Service

Jingjing Wang, Tobin Baker, Magdalena Balazinska, Daniel Halperin, Brandon Haynes, Bill Howe, Dylan Hutchison, Shrainik Jain, Ryan Maas, Parmita Mehta, Dominik Moritz, Brandon Myers, Jennifer Ortiz, Dan Suciu, Andrew Whitaker, Shengliang Xu

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF WASHINGTON
http://myria.cs.washington.edu
Acknowledgments

The Myria Team!

Our science collaborators!!

• Andrew Connolly, Tom Quinn, Sarah Loebman, Ariel Rokem, Ginger Armbrust, Yejin Choi

Our sponsors!!!

• National Science Foundation, Moore & Sloan Foundations, Washington Research Foundation, eScience Institute, ISTC Big Data, Petrobras, EMC, Amazon, and Facebook
Goals of the Myria stack
• Advance state-of-the-art in big data systems
• Focus on efficiency and productivity
• Test on real applications and support real users

Deliverables:
• Built a new big data mgmt & analytics system
• Deployed and operate Myria as a service
• Source code and demo service: http://myria.cs.washington.edu
Myria has been developed and is operated by
• Database Group in the Computer Science & Engineering Department at UW
• UW eScience Institute

Co-PIs: Dan Suciu and Bill Howe
Myria Demo
Myria Cloud Service

Myria  Big Data as a Service

Myria is a distributed, shared-nothing Big Data management system and Cloud service from the University of Washington. We derive requirements from real users and complex workflows, especially in science.

Try the free tier Myria service

Deploy your own Myria cluster

Service available through project website
Analysis in the Browser with Myria

Declarative-imperative analysis with MyriaL and Python
Myria Operates Directly on Data in S3

For efficient processing, caches query results internally in cluster.

```rust
t1 = load("https://s3-us-west-2.amazonaws.com/uwdb/sampleData/TwitterK.csv",
csv(schema(a:int, b:int),skip=0));
store(t1, TwitterK, [a, b]);
```
MyriaL is Imperative+Declarative with Iterations

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

```
E = scan(TwitterK);
V = select distinct E.$0 from E;
CC = [from V emit V.$0 as node_id, V.$0 as component_id];
do
   new_CC = [from E, CC where E.$0 = CC.$0 emit E.$1, CC.$1] + CC;
   new_CC = [from new_CC emit new_CC.$0, MIN(new_CC.$1)];
   delta = diff(CC, new_CC);
   CC = new_CC;
while [from delta emit count(*) > 0];
   comp = [from CC emit CC.$1 as id, count(CC.$0) as cnt];
store(comp, TwitterCC);
```
Myria Provides Details of Query Execution

Physical Query Plan:

Fragment 1
- Split
- SymmetricHashJoin($1 = $2); $0,$1,$3
- ShuffleConsumer
- SplitProducer

Fragment 2
- Scan(public:adhoc:TwitterK)
- ShuffleProducer(h($1))

Fragment 3
- Scan(public:adhoc:TwitterK)
- ShuffleProducer(h($0))

Overview / Operators inside fragment 1

Query time contribution collapse/expand

Detailed execution
Myria Service includes Jupyter Notebook

Jupyter notebook available directly with Myria service
Myria Supports Python User-Defined Functions

```python
# define python function
def denoise(dt):
    from dipy.denoise import nlmeans
    from dipy.denoise.noise_estimate import estimate_sigma
    image = dt[0]
    mask = dt[1]
    sigma = estimate_sigma(image)
    denoised_data = nlmeans.nlmeans(image, sigma=sigma, mask=mask)
    return denoised_data

# register python functions
connection.create_function("denoise", inspect.getsource(denoise), inSchema, outType, py=denoise)
```

```python
# this query takes 2.40 mins
query = MyriaQuery.submit(
    """T1=scan(public:blob_operator:binarydata);
    imgs = [from T1 emit PYUDF(denoise, T1.images, T1.mask) As denoised];
    store(imgs, Denoised_imgs);""
)
print query.status
```

Python UDFs enable running legacy code and complex analytics beyond SQL/MyriaL
Users Can Deploy Own Service

```sh
pip install myria-cluster

myria-cluster create [OPTIONS] CLUSTER_NAME

myria-cluster stop/start/destroy [...]```
Example Myria Applications

Natural Language Processing

SYNTACTIC N-GRAM (NID: 12345, FREQ: 50)

nsubj  ROOT  advmod  num  pobj

I  brought  away  one  unicorn

PRP  VBD  RB  CD  NN

Picture from Leila Zilles

Neuroscience

Data from the Human Connectome project

Astronomy

Oceanography

Bibliometrics

MyMergerTree Screenshot

1. RED fluorescence

2. Nanoplankton

3. Ultraplankton

4. Picoplankton

5. Prochlorococcus

Data from the Human Connectome project
Myria Internals
Myria Polystore Stack

- Browser
- Python/Jupyter
- Specialized Services

RACO

Query Translation, Optimization, and Orchestration

MyriaX

Parallel, Iterative, and Elastic Query Execution

- MPI
- SciDB
- Graphs
- NoSQL

Magdalena Balazinska - University of Washington
Myria’s Data Model and Query Interface

• Relational Algebra Compiler (RACO)
  – Myria’s query optimizer and federator

• RACO core: relational algebra extended with
  – Iterations for multi-pass algorithms
  – Flatmap to explode non-1NF attribute values into many tuples
  – Stateful apply for windowed and neighborhood functions

• Query language: MyriaL (Imperative+Declarative)
  – Each statement is declarative (SQL, comprehensions, function calls)
  – Statements are combined with imperative constructs
    • Variable assignment
    • Iteration

• Python UDFs/UDAs
  – Minimize barriers to adoption and run legacy code

• Python API
  – Fluent API with Python lambda functions
Polystore Optimization

• Rule-based opt. with three types of rules
  – Optimize logical Myria algebra plans
  – Translate logical plans into back-end specific physical plans
  – Optimize back-end specific physical plans

• To add a new back-end, developer must specify
  – Tree representation of query language
  – Rules that translate Myria algebra into this representation
  – Administrative functions including one to submit queries

• Data model independence
  – Myria hides the existence of various back-ends
  – Users write MyriaL scripts assuming relational model
  – Back-ends include select array, graph, and key-value systems
Federated plans require fast data movement

Source DBMS

Worker₁

Workerₙ

Target DBMS

Worker₁

Workerₘ

Worker Directory

source.ₖ → target.ₖ

source.ₙ → target.ₘ

User or Opt.

[1] t = scan(data)
x = distances(t, t)
export(x, 'db://Target')

[2] x = import('db://Source')
u = cluster(x)

[3]

[4]

Federated Query Execution
Data Movement with PipeGen

PipeGen: Data Pipe Generator for Hybrid Analytics
Brandon Haynes, Alvin Cheung, and Magdalena Balazinska. SOCC 2016.

PipeGen:
- Data Pipe Generator for Hybrid Analytics
- Pipegen-Enabled DBMS

DBMS Bytecode
- Unit Tests
- DBMS bytecode

PipeGen
- Data Pipe Generator
- IORedirect: I/O Redirector
- Instrument Unit Tests
- Identify File Open Expressions
- Inject Conditional Redirection

FormOpt: Format Optimizer
- Instrument Unit Tests
- Data Flow Analysis
- Type Substitution

PipeVerify: Verification
- DBMS with optimized data pipe
- Augmented Types
- Data Pipe Type
PipeGen’s Performance

16-node cluster with 16 workers/tasks
Transfer $10^9$ tuples with 4 ints and 3 doubles
Myria Polystore Stack

Browser

Python/Jupyter

Specialized Services

MyMergerTree

RACO

Query Translation, Optimization, and Orchestration

MyriaX

Parallel, Iterative, and Elastic Query Execution

MPI

SciDB

Graphs

NoSQL

Magdalena Balazinska - University of Washington
MyriaX Engine and Cloud Deployment

JSON query plans & API calls

REST Interface
Coordinator
YARN Container
Amazon EC2 Instance
Worker
YARN Container
Amazon EC2 Instance
Worker
YARN Container
Amazon EC2 Instance
Worker
YARN Container
Amazon EC2 Instance

RDBMS
RDBMS
RDBMS

HDFS

Amazon EBS Volumes and/or Local Storage
Amazon S3

Magdalena Balazinska - University of Washington
MyriaX Overview

• **Data storage**
  – Read data from S3, HDFS, local files
  – Parse CSV, TSV, and various scientific file formats
  – Store data in local relational DBMS instances
    • Fast storage with physical tuning (indexing, hash-partitioning)

• **Query execution**
  – Fundamentally a parallel DBMS
    • Fast, pipelined query execution
  – But scheduling more flexible to support elasticity
  – Novel features: Multiway joins and iterations

• **Resource management**
  – Executes on top of the YARN resource manager
Efficient Iterative Processing

Asynchronous and Fault-Tolerant Recursive Datalog Evaluation in Shared-Nothing Engines

• User specifies query declaratively
  – Subset of Datalog with aggregation

• Generate efficient, shared-nothing query plan
  – Small extensions to existing shared-nothing systems

• Plan amenable to runtime optimizations
  – Synchronous vs asynchronous
  – Different processing priorities

• Optimizations significantly affect performance
Myria’s Optimized Iterations Example

Asynchronous and Fault-Tolerant Recursive Datalog Evaluation in Shared-Nothing Engines

Declarative Query

E = scan(jwang:cc:graph);
V = select distinct E.$0 from E;
do
  CC := [$0, MIN($1)] <-
    [from V emit V.$0 as x, V.$0 as y] +
    [from E, CC where E.$0 = CC.$0 emit E.$1, CC.$1];
until convergence;
store(CC, CC);

// Can have multiple relations
// with recursive dep.

Compiled to a Distributed Query Plan

Scan(Edges) -> IDBController(CC) -> Join

Magdalena Balazinska - University of Washington
Performance Comparison with Spark

- Declarative Query (subset of Datalog with agg.)
- Shared-Nothing Query Plan In-Memory Processing
- Synchronous
- Asynchronous
- Prioritize New Data
- Prioritize Base Data

# of Workers

- Spark (GraphX)
- Myria, Sync
- Myria, Async

Connected Components – Twitter subgraph 221 million edges and 5 million vertices
Myria Polystore Stack

Browser

Python/Jupyter

Specialized Services

MyMergerTree

MyriaX

Parallel, Iterative, and Elastic Query Execution

RACO

Query Translation, Optimization, and Orchestration

MPI

SciDB

Graphs

NoSQL
Cloud Operation in Myria

Or point to data in Amazon S3
Myria’s Personalized Service Level Agreements

Changing the Face of Database Cloud Services with Personalized Service Level Agreements
Jennifer Ortiz, Victor T. Almeida, and Magdalena Balazinska. CIDR 2015

<table>
<thead>
<tr>
<th>Tier #1</th>
<th>Query Template</th>
<th>Runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SELECT (17 attributes) FROM (lineitem) WHERE (1% of data selected)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>SELECT (9 attributes) FROM (part) WHERE (100% of data selected)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SELECT (9 attributes) FROM (customer) WHERE (100% of data selected)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SELECT (17 attributes) FROM (date) WHERE (100% of data selected)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SELECT (35 attributes) FROM (3 TABLES) WHERE (100% of data selected)</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>SELECT (48 attributes) FROM (5 TABLES) WHERE (10% of data selected)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SELECT (60 attributes) FROM (5 TABLES) WHERE (1% of data selected)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SELECT (49 attributes) FROM (5 TABLES) WHERE (100% of data selected)</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>SELECT (60 attributes) FROM (5 TABLES) WHERE (10% of data selected)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SELECT (60 attributes) FROM (5 TABLES) WHERE (100% of data selected)</td>
<td>600</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tier #2</th>
<th>Query Template</th>
<th>Runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SELECT (26 attributes) FROM (2 TABLES) WHERE (100% of data selected)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>SELECT (60 attributes) FROM (5 TABLES) WHERE (10% of data selected)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SELECT (60 attributes) FROM (5 TABLES) WHERE (100% of data selected)</td>
<td>300</td>
</tr>
</tbody>
</table>

Myria’s SLA generation

Workload Compression into PSLA

Schema → Workload Generation → Runtime Prediction → Query Clustering → Template Generation → Cross-Tier Pruning → PSLA

Magdalena Balazinska - University of Washington
Myria’s PerfEnforce Subsystem

**PerfEnforce Demonstration: Data Analytics with Performance Guarantees**

Jennifer Ortiz, Brendan Lee, and Magdalena Balazinska. **SIGMOD 2016**.

Query Session

Write a Query...

```sql
1. SELECT *
2. FROM "public:adhoc:lineitem" AS L
3. WHERE l_linenumber = 7;
```

[See SLA Runtime] [Execute]

Query Information

- **Expected Runtime (from SLA):** 11.756 seconds
- **status:** SUCCESS
- **seconds elapsed:** 1.168460994
- **Cluster is using 12 workers**

Previous Queries Log

- **Query:** SELECT * FROM "public:adhoc:lineitem" AS L WHERE l_linenumber = 7;
- **Actual Runtime:** 1.168460994
- **Expected Runtime:** 11.756
- **Cluster Size Ran:** 12
Myria’s PerfEnforce Subsystem

PerfEnforce Demonstration: Data Analytics with Performance Guarantees
Jennifer Ortiz, Brendan Lee, and Magdalena Balazinska. SIGMOD 2016.

Query Session

Write a Query...

```
1. SELECT *
2. FROM "public:adhoc:lineitem" AS L
3. WHERE l_linenumber = 7;
```

How can the cloud provider guarantee these runtimes?

Cluster size changes during query session
Myria’s Innovations Summary

Efficient Processing & Complex Analytics with MyriaX
- Efficient Multi-Join
- Iterative Queries
- Data Summaries
- Image Processing

Myria Polystore
- Federated Analytics
- Automatic Data Pipes
- Perf. Debugging

Myria Cloud Operation
- Cloud PSLAs
- Performance Guarantees
- Elastic Memory
Conclusion

• Highly expressive
  – MyriaL (RA+iterations) & Python
• Polystore with hybrid analytics
• High performance on variety of queries
• Available as a service
  – Focus on low barrier to entry
  – And turning users into self-sufficient experts
  – Also focus on the service provider: Operate Myria
• Source code and more info (includes videos)
  http://myria.cs.washington.edu/