— Turning a Zoo into a Circus —
Let’s Start with 2 Facts
Nature Inspired
Go fast?
Endurance?
Go to the forest?
FACT 1: One Size Doesn’t Fit All
FACT 2: Zoo of Systems
What is RHEEM?

A System Tamer
Where in the Analytics Stack?

- **Applications**
  - Hive
  - Crunch
  - MLlib
  - Mahout
  - Pig
  - Giraph

- **Processing Platforms**
  - Hadoop
  - MR
  - DBMS
  - Storm
  - Spark
  - Flink

- **Storage Engines**
  - HDFS
  - S3
  - Local FS
Where in the Analytics Stack?

- **Applications**
  - Hive
  - Crunch
  - MLlib
  - Mahout
  - Pig
  - Giraph

- **Cross-Platform System**
  - Hadoop
  - MR
  - DBMS
  - Storm
  - Spark
  - Flink
  - Giraph
  - MLlib

- **Storage Engines**
  - HDFS
  - S3
  - Local FS
What is RHEEM for?
Data Model
Three-Layer Abstraction

Three-Layer Optimization

Platform Agnostic

Application Agnostic

Always Evolving
This Tutorial

Getting Ready
- How to get Rheem
- How to setup Rheem

Hands on Rheem
- Word count
- IND discovery
- Pagerank

Demo
- ML app (ML4all)
- Extending operators

Rheem cost functions
- Execution logs
- Regression on the logs
- Cost functions
This Tutorial

Getting Ready
- How to get Rheem
- How to setup Rheem

Hands on Rheem
- Word count
- IND discovery
- Pagerank

Demo
- ML app (ML4all)
- Extending operators

Rheem cost functions
- Execution logs
- Regression on the logs
- Cost functions
Get Rheem

- **Rheem web page**
  http://da.qcri.org/rheem/

- **Rheem repository**
  https://github.com/daqcri/rheem
  $ git clone https://github.com/daqcri/rheem.git

- **Examples**
  https://github.com/sekruse/rheem-examples
  $ git clone \ 
  https://github.com/sekruse/rheem-examples.git

- **Useful apps**
  IntelliJ IDEA/eclipse, Git, Maven
  IPython/Jupyter with jupyter-scala kernel

- **Data**
  realworld://flash.disk/
Time to Play

Getting Ready
- How to get Rheem
- How to setup Rheem

Hands on Rheem
- Word count
- IND discovery
- Pagerank

Demo
- ML app (ML4all)
- Extending operators

Rheem cost functions
- Execution logs
- Regression on the logs
- Cost functions
This sentence contains “twice” twice.

Wordcount

Flatmap

Map

Reduce by key

- (this, 1)
- (sentence, 1)
- (contains, 1)
- (twice, 1)
- (twice, 1)

- (this, 1)
- (sentence, 1)
- (contains, 1)
- (twice, 2)
Wordcount on Rheem

![Graph showing wordcount execution time: estimated vs. actual](image)
Blueprint for Rheem Apps

1. Declare Rheem dependency
1. Declare Dependencies

- Available in Maven Central
- optimizer, execution: **rheem-core**
- Java and Scala API: **rheem-api**
- modules: **rheem-basic, rheem-java, rheem-spark,**
  **rheem-sqlite3, rheem-postgres, rheem-graphchi**

```xml
<dependency>
    <groupId>org.qcri.rheem</groupId>
    <artifactId>rheem-***</artifactId>
    <version>0.2.0</version>
</dependency>
```
2. Obtain a Configuration

```yaml
rheem.basic.tempdir = hdfs://namenode/tmp/
rheem.java.cpu.mhz = 2200
rheem.java.hdfs.ms-per-mb = 2.7

spark.master = spark://sparkmaster:7077/
rheem.spark.cpu.mhz = 2000
rheem.spark.cpu.cores = 4
rheem.spark.hdfs.ms-per-mb = 0.3
rheem.spark.network.ms-per-mb = 8.6
rheem.spark.init.ms = 9000

rheem.postgres.jdbc.url = jdbc:postgres:my-db
rheem.postgres.cpu.mhz = ...
```
2. Obtain a Configuration

- Configuration defines cost functions, advanced features, app properties

- `val configuration = new Configuration()`
  - Explicitly specify a configuration file
    ```java
    java -Drheem.configuration=url://to/my/rheem.properties ...
    ```
  - Put a `rheem.properties` file on your classpath
  - If none applies, there are fallback values
3. Register Plugins

- **new** `RheemContext(configuration).withPlugin(...)`
- **Plugins provide execution platforms and/or operators**
- **Available plugins:**
  - Java.basicPlugin(): Java implementations basic operators (as found in API)
  - Spark.basicPlugin(): Spark plugins
  - GraphChi.plugin(): GraphChi plugin
  - Postgres.plugin(), Sqlite3.plugin(): database plugins
  - Java.graphPlugin(), Spark.graphPlugin(), `RheemBasic.graphPlugin()`: support for graph operators
  - DynamicPlugin.loadYaml(...): user-configurable plugins
4. Build a Rheem Plan (I)

Start with `PlanBuilder` and chain operations.

```scala
val planBuilder =
  new PlanBuilder(rheemContext)
  .withJobName(s"WordCount ($url)")
  .withUdfJarsOf(this.getClass)

val wordCounts = planBuilder
  .readTextFile(url)
  .flatMap(…, selectivity = …)
  .filter(…, selectivity = …)
  .map(…)
  .reduceByKey(…)
  .collect()```

Text file source
Flat map
Filter
Map
Reduce by key
Collection sink
Rheem supports loops, which is important for machine learning algorithms:

```scala
val myResult = ...
  .repeat(n, i => i.map(...))
...
  .collect()
```

Data flows can be joined in a flexible manner:

```scala
val sizeDataset = myDataset.count
val result = myDataset
  .map(...)
  .withBroadcast(sizeDataset, "size")
  .collect()
```
API is just syntactic sugar - use your own operators!

```java
planBuilder.load(new CustomSource())
  .flatMap(…)
  .customOperator(new CustomOperator())
  .map(…)
  .reduceByKey(…)
  .customOperator(new CustomSink())
```
5. Trigger Execution

- Design choice: creating a sink triggers execution
  - Rheem allows for multiple sinks → stay tuned

- Available sinks
  - ....collect(): fetch dataset as JVM-based collection
  - ....writeAsTextFile(...): format & write dataset to a text file

- Note: Only when execution is triggered, Rheem start its optimization, let alone execution.
6. Let Rheem do the Rest
6. Let Rheem do the Rest

- Inflate
- Estimate
- Enumerate
- Execute

Text file source

Flat map

Filter

Flat map

Java flat map

Spark flat map

Filter
6. Let Rheem do the Rest

Inflate → Estimate → Enumerate → Execute

Text file source → 10M..12M → Flat map → 100..240 s → Java flat map → 100M..240M → Spark flat map → 10..24 s → Filter
6. Let Rheem do the Rest

- Inflate
  - Java text file src.
    - 120..280 s
- Estimate
  - Java collect
    - 8..10 s
  - Spark collect
    - 10..12 s
- Enumerate
  - Spark collection src.
    - 10..12 s
- Execute
  - Spark text file src.
    - 12..28 s
  - Spark flat map
    - 10..24 s
6. Let Rheem do the Rest

- Inflate
- Estimate
- Enumerate
- Execute

Spark text file src.
Spark flat map
Spark map
Spark filter
Spark collect
Java reduce by key
Java collection sink
Task: Wordcount’

• Reduce number of Rheem operators.

• What are possible implications of doing so?
  • w.r.t. performance?
  • w.r.t. optimization hints?
  • w.r.t. maintainability?
  • w.r.t. Rheem’s optimization pipeline?
Detect column pairs in a database, such that all values of the one are included in the other.

### Customer

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Deng</td>
</tr>
<tr>
<td>1</td>
<td>Lavel</td>
</tr>
<tr>
<td>2</td>
<td>Doe</td>
</tr>
<tr>
<td>3</td>
<td>Miller</td>
</tr>
</tbody>
</table>

### Address

<table>
<thead>
<tr>
<th>c_id</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12 Key Str</td>
</tr>
<tr>
<td>1</td>
<td>883 Data Dr</td>
</tr>
<tr>
<td>3</td>
<td>78 Base Pkw</td>
</tr>
</tbody>
</table>

\[ c_{id} \subseteq id \]
IND Discovery

Customer

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Project

Flatmap

Address

<table>
<thead>
<tr>
<th>c_id</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Flatmap

Union

SINDY

Collection Sink
RDF PageRank

1. Parse
   - Text File Source
     - Map
     - Flatmap
     - Zip with ID
     - Join
     - PageRank
     - Join
     - Join
     - Collection Sink

2. Introduce IDs
3. Resolve IDs
Task: Tune PageRank

- Entities URLs all conform to the pattern
  \(<\text{http://dbpedia.org/resource/...}>\)
  \(\rightarrow\) eliminate that redundancy

- \(<\text{http://dbpedia.org/resource/Category:...}>\)
  are no real entities
  \(\rightarrow\) remove them

- boost entities having a lot of outgoing links
  \(\rightarrow\) make graph undirected
Advanced App

Getting Ready
• How to get Rheem
• How to setup Rheem

Hands on Rheem
• Word count
• IND discovery
• Pagerank

Demo
• ML app (ML4all)
• Extending operators

Rheem cost functions
• Execution logs
• Regression on the logs
• Cost functions
ML4all: ML on top of Rheem
Gradient Descent

initialize \( w^0 \)

while !converged {
    \[
    \text{grad} = \sum_i \text{gradient}(f_i(w)), \text{ for all } i \text{ in } D
    \]
    \[
    w^{k+1} := w^k + \alpha_k \times \frac{1}{n} \times \text{grad}
    \]
    converged := \|w^{k+1} - w^{k+1}\| < 0.001
    k := k+1
}
initialize $w^0$  

while !converged {
    $j := \text{sample from } D$
    \[ \text{grad} = \text{gradient}(f_j(w)) \]
    \[ w^{k+1} := w^k + \alpha_k \times \text{grad} \]
    converged := $||w^{k+1} - w^{k+1}|| < 0.001$
    $k := k+1$
}
SGD Plan in Rheem

1. **Data Points**
2. **Map**
3. **Sample**
4. **DoWhileLoop**
5. **GlobalReduce**
6. **Map**
7. **Map**
8. **Map**
9. **Map**
10. **Map**
11. **Map**

- **Weights Initialization**
- **Gradient Computation**
- **Weights Update**
- **Convergence Computation**
- **Convergence Value**
- **New Weights**
SGD Plan in Rheem

data points

weights

weights initialization

source

DoWhileLoop

Map

Sample

GlobalReduce

Map

Map

Map

Map

gradient computation

weights update

convergence computation

new weights

convergence value
SGD Plan in Rheem

- **Source**: data points
- **Map**: weights initialization
- **Sample**: weights
- **DoWhileLoop**: new weights, convergence value
- **GlobalReduce**: new weights
- **Map**: convergence computation
- **Map**: weights update
- **Map**: gradient computation
- **Collect**: final weights
- **Sink**: final weights
Extending Rheem with new operators

- Create core *Rheem (platform-agnostic) operator*
  - Consider adding a cardinality estimator
- Create *execution (platform-specific) operator*
  - Consider adding a cost function
- Create *mappings* from the Rheem operator to the execution operator
Bootstrapping Rheem

Getting Ready
- How to get Rheem
- How to setup Rheem

Hands on Rheem
- Word count
- IND discovery
- Pagerank

Demo
- ML app (ML4all)
- Extending operators

Rheem cost functions
- Execution logs
- Regression on the logs
- Cost functions
Rheem Workflow

Inflate → Estimate → Enumerate → Execute → Log stats

- Estimate Cardinalities
- Estimate Execution Costs
Lazy Execution

- Spark text file src.
- Spark flat map
- Spark map
- Spark filter
- Spark collect
- Java reduce by key
- Java collection sink

Execution time: 23,338 ms

Execution time: 821 ms
Resolving Cost Functions

It's an optimization problem!
Find a parameterization $x$ for the operator cost functions that minimizes some loss function between measured block execution times and estimated block execution times under $x$. 
Applying Genetic Optimization

Collecting execution data

- Rheem must keep track properly of when is what executed
  → operators are self-descriptive in that respect
- Get cardinality estimates as accurate as possible
  → Rheem monitors cardinalities and updates all estimates
- Gather sufficient, variant execution data

Complement execution data

- Provide models for the cost functions
- Account for heavy-weight UDFs
Take Away

• A Cross-Platform System
• Focus on your App and let Rheem do the rest

Stay Tuned!

http://daqcri.org/rheem/
https://github.com/daqcri/rheem

Still To Come…

• Learning Cost Functions
• In-Memory Data Processing
• More Data Processing Platforms
• Cross-Platform Fault-Tolerance