Re-optimizing Data Parallel Computing

Sameer Agarwal
Srikanth Kandula, Nicolas Bruno, Ming-Chuan Wu,
Ion Stoica, Jingren Zhou
A Data Parallel Job...

can be a collection of maps,
A Data Parallel Job...

can be a collection of maps, reduces
A Data Parallel Job...

can be a collection of maps, reduces, joins
A Data Parallel Job...

... and other framework dependent operations
A Data Parallel Job...
A Data Parallel Job...

Output

Join

Join

Reduce

Map

f1

Reduce

Map

f2

Reduce

Map

f3

Map

Reduce
A Data Parallel Job...

Output

Join

Reduce

Map

f1

Reduce

Map

f2

Reduce

Map

Reduce

Map

f3

A MapReduce Phase
A Data Parallel Job...

Runs on Large Clusters

Batched Jobs (Dryad/Hadoop)
Key Metrics

Job Completion Time
Key Metrics

Cluster Utilization

Job Completion Time

Reduce Map Reduce Map Reduce Map

Join

f1 f2 f3

Key Metrics

Job Completion Time

Cluster Utilization

Reduce Map Reduce Map Reduce Map

Join

f1 f2 f3
Imbalance

Different Work is being done at Every Stage (Data + Complexity)

Jobs Finish Sooner
Data Skews

Data is not independent and uniformly distributed

Jobs Finish Sooner
These problems are real!

Imbalance ~ Avg. Task time per Stage

- Most stages take < 10 seconds
- Top 4% (1%) take > 100 (1000) seconds
These problems are real!

$$\text{Skew} = \frac{\text{Maximum Partition Size}}{\text{Average Partition Size}}$$

Most stages have skew < 2
But, 5% have > 10
These problems are real!

Others in paper...
Optimizing the Job Execution

To optimize the completion time and cluster utilization, ideally you need to control:

– Parallelism
– Partition Sizes
– Operator Sequence
– Operator Implementation

The *Execution Plan* lets you control these knobs.
Who generates execution plans?

SCOPE

Dryad
# High Level Abstractions

<table>
<thead>
<tr>
<th>Hadoop</th>
<th>Hive</th>
<th>Pig/Scope</th>
<th>Database</th>
</tr>
</thead>
</table>

- Hadoop
- Hive
- Pig/Scope
- Database
# High Level Abstractions

<table>
<thead>
<tr>
<th></th>
<th>Hadoop</th>
<th>Hive</th>
<th>Pig/Scope</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallelism</strong></td>
<td>Statically Configurable</td>
<td>Statically Configurable</td>
<td>Statically Configurable</td>
<td>Static</td>
</tr>
</tbody>
</table>
## High Level Abstractions

<table>
<thead>
<tr>
<th></th>
<th>Hadoop</th>
<th>Hive</th>
<th>Pig/Scope</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallelism</strong></td>
<td>Statically Configurable</td>
<td>Statically Configurable</td>
<td>Statically Configurable</td>
<td>Static</td>
</tr>
<tr>
<td><strong>Data Partition</strong></td>
<td>Statically Configurable</td>
<td>Rule Based (Data Size)</td>
<td>Cost Based (Fixed Cost)</td>
<td>Cost Based*</td>
</tr>
</tbody>
</table>

*Cost Based* indicates a cost-based approach to data partitioning, with variations in data size being considered.
# High Level Abstractions

<table>
<thead>
<tr>
<th></th>
<th>Hadoop</th>
<th>Hive</th>
<th>Pig/Scope</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallelism</strong></td>
<td>Statically Configurable</td>
<td>Statically Configurable</td>
<td>Statically Configurable</td>
<td>Static</td>
</tr>
<tr>
<td><strong>Data Partition</strong></td>
<td>Statically Configurable</td>
<td>Rule Based (Data Size)</td>
<td>Cost Based (Fixed Cost)</td>
<td>Cost Based*</td>
</tr>
<tr>
<td><strong>Operator Implementation</strong></td>
<td>N/A</td>
<td>Rule Based (Data Size)</td>
<td>Cost Based (Fixed Cost)</td>
<td>Cost Based*</td>
</tr>
</tbody>
</table>
# High Level Abstractions

<table>
<thead>
<tr>
<th></th>
<th>Hadoop</th>
<th>Hive</th>
<th>Pig/Scope</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallelism</strong></td>
<td>Statically Configurable</td>
<td>Statically Configurable</td>
<td>Statically Configurable</td>
<td>Static</td>
</tr>
<tr>
<td><strong>Data Partition</strong></td>
<td>Statically Configurable</td>
<td>Rule Based (Data Size)</td>
<td>Cost Based (Fixed Cost)</td>
<td>Cost Based*</td>
</tr>
<tr>
<td><strong>Operator Implementation</strong></td>
<td>N/A</td>
<td>Rule Based (Data Size)</td>
<td>Cost Based (Fixed Cost)</td>
<td>Cost Based*</td>
</tr>
<tr>
<td><strong>Operator Sequence</strong></td>
<td>N/A</td>
<td>Rule Based (Data Size)</td>
<td>Cost Based (Fixed Cost)</td>
<td>Cost Based*</td>
</tr>
</tbody>
</table>
# High Level Abstractions

<table>
<thead>
<tr>
<th></th>
<th>Hadoop</th>
<th>Hive</th>
<th>Pig/Scope</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallelism</strong></td>
<td>Statically Configurable</td>
<td>Statically Configurable</td>
<td>Statically Configurable</td>
<td>Static</td>
</tr>
<tr>
<td><strong>Data Partition</strong></td>
<td>Statically Configurable</td>
<td>Rule Based (Data Size)</td>
<td>Cost Based (Fixed Cost)</td>
<td>Cost Based*</td>
</tr>
<tr>
<td><strong>Operator Implementation</strong></td>
<td>N/A</td>
<td>Rule Based (Data Size)</td>
<td>Cost Based (Fixed Cost)</td>
<td>Cost Based*</td>
</tr>
<tr>
<td><strong>Operator Sequence</strong></td>
<td>N/A</td>
<td>Rule Based (Data Size)</td>
<td>Cost Based (Fixed Cost)</td>
<td>Cost Based*</td>
</tr>
</tbody>
</table>
High Level Abstractions

<table>
<thead>
<tr>
<th></th>
<th>Hadoop</th>
<th>Hive</th>
<th>Pig/Scope</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data partition</td>
<td>Statically Configurable</td>
<td>Rule Based (Data Size)</td>
<td>Cost Based (Fixed Cost)</td>
<td>Cost Based*</td>
</tr>
<tr>
<td>Operator Implementation</td>
<td>N/A</td>
<td>Rule Based (Data Size)</td>
<td>Cost Based (Fixed Cost)</td>
<td>Cost Based*</td>
</tr>
<tr>
<td>Operator Sequence</td>
<td>N/A</td>
<td>Rule Based (Data Size)</td>
<td>Cost Based (Fixed Cost)</td>
<td>Cost Based*</td>
</tr>
</tbody>
</table>

Want automatic control of all four knobs
- Based on the data and computation
- Robust for User Defined Functions (UDFs)
RoPE: Re-optimizer for Parallel Executions

- Degree of Parallelism at every operation
- Data Partitions for each operation
- Implementations for each operation
- Sequence of operations
Cost-based query optimizer
+ information about code and data

Job → Re-Optimizer → Plan → Execution

Measured Properties
Better Execution Plans For

1. Recurring Jobs
   Same “code” runs hourly on new data
   Data properties are stable

2. “Similar” Jobs
   e.g., jobs with identical parts

3. Future parts of this Job
   e.g., after a barrier
Information collected

Data Properties
- Cardinality
- Avg. Row Length
- # of Distinct Values
- Heavy Hitters

Code Properties
- CPU Cycles
- Peak Memory

Leading Statistics
- Hash Histograms
- Data Samples

Balance
Efficiently collecting this information is Challenging
Collecting Information

Option 1: Measure Input Properties, Propagate over operations
Collecting Information

Option 1: Measure Input Properties, Propagate over operations

- Estimation error increases exponentially with #operations [Rio]
- Hard to reason about user defined functions

Low Overhead

Bad Accuracy
Collecting Information

Option 2: Store all intermediate data, analyze offline
Collecting Information

Option 2: Store all intermediate data, analyze offline

- Lots of Intermediate Data
- Job latencies and resource requirements went up ~10x
RoPE’s Distributed Stats Collection

- Composable
- Light (single pass, sub-linear state)
Composing statistics with Resource constraints

- Trivial for some ... e.g., cardinality
- Doable for some ... e.g., heavy hitters, #distinct values
- Open problem for others ... e.g., equi-width histograms
Related Work

Implementing Execution Plans Well

• Placing Tasks to maximize locality (Quincy, Delay Scheduling)
• Dealing with Outliers (LATE, Mantri)
• Orchestrating Network Transfers (Orchestra, Camdoop, SUDO)

RoPE finds better execution plans
Related Work

Re-optimization in Databases

• Single Machine/Short Queries (Kabra/Dewitt)
• Creating Robust plans given uncertainty in properties (Rio)
• Parametric Query optimization

RoPE reasons about Parallel Plans and deals with User-Defined Functions
Does it work?
Evaluation

• We evaluated RoPE on a large production cluster
• Our workload suite consisted of hundreds of production jobs from a wide range of users during March 2012.
• **Baseline:** Production Scope (State-of-Art-QO)
• **Metrics:** Completion Time & Cluster Utilization
Highlights

- **2x** improvements across **job latency** for production jobs at the **75th percentile** while using **1.5x fewer resources**
  - Better Execution Plans
  - Reduction in terabytes of Intermediate data and cross-rack shuffles
  - Better Parallelism

- **2-5%** overhead incurred by our distributed statistics collection framework
Job Completion Time

Reduction in Job Latency (%)

Job Size

< 2 min  |  2-10 min  |  10-30 min  |  30 min - 1 hr  |  > 1hr
Job Completion Time

<table>
<thead>
<tr>
<th>Job Size</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 2 min</td>
<td></td>
</tr>
<tr>
<td>2-10 min</td>
<td></td>
</tr>
<tr>
<td>10-30 min</td>
<td></td>
</tr>
<tr>
<td>30 min - 1 hr</td>
<td></td>
</tr>
<tr>
<td>&gt; 1 hr</td>
<td></td>
</tr>
</tbody>
</table>

Reduction in Job Latency (%)

> 1hr
Cluster Utilization

Reduction in Cluster Resources (%)

Job Size

< 2 min  2-10 min  10-30 min  30 min - 1 hr  > 1 hr
Cluster Utilization

Reduction in Cluster Resources (%)

Job Size

< 2 min 2-10 min 10-30 min 30 min - 1 hr > 1 hr
RoPE improves execution plans for data-parallel jobs

• Measurements identify novel problems and opportunities
• To leverage, built RoPE, a re-optimizer, that learns/uses code- and data- properties
• Running live in Bing Production Clusters since December 2011
Thank you