Stubby: A Transformation-based Optimizer for MapReduce Workflows

Harold Lim, Herodotos Herodotou, Shivnath Babu
Duke University
MapReduce Workflow
MapReduce Workflow
MapReduce Workflow
Automatic MapReduce Workflow Optimizer
Automatic MapReduce Workflow Optimizer

7 Jobs to 2 Jobs!
Automatic MapReduce Workflow Optimizer

- **Stubby** [ˈstʌbI] adj - short and broad
Automatic MapReduce Workflow Optimizer

- **Stubby** [ˈstʌbɪ] *adj* - short and broad
Automatic MapReduce Workflow Optimizer

- **Stubby** ['stʌbɪ] adj - short and broad
Automatic MapReduce Workflow Optimizer

- **Stubby** ['stʌbi] adj - short and broad
Outline

- MapReduce
- Workflow Optimization
- Challenges
Outline

Transformations

Many Interfaces

MapReduce Workflow Optimization Challenges

Large Plan Space

Interactions

Annotations

Information Spectrum
Outline

Transformations

Many Interfaces

MapReduce Workflow Optimization Challenges

Large Plan Space

Interactions

Information Spectrum

Annotations

Baseline
Stubby

Speedup

IR SN LA WG BA BR PJ US
MapReduce Ecosystem
MapReduce Ecosystem

Pig          Hive          Jaql          FlumeJava

MapReduce Workflow

MapReduce Execution Engine
MapReduce Ecosystem

MapReduce Workflow

MapReduce Execution Engine

Many Interfaces

Optimization Challenges
MapReduce Ecosystem

MapReduce Workflow

MapReduce Execution Engine

Many Interfaces

Optimization Challenges
MapReduce Ecosystem

- Pig
- Hive
- Jaql
- FlumeJava

MapReduce Workflow

Optimized MapReduce Workflow

MapReduce Execution Engine

Many Interfaces

Optimization Challenges
MapReduce Ecosystem

- Schema
  - Pig
  - Hive

- Filters
  - Jaql

- Partitions
  - FlumeJava

- Dataset

MapReduce Workflow

Optimized MapReduce Workflow

MapReduce Execution Engine

Many Interfaces

Optimization Challenges
MapReduce Ecosystem

- Schema
- Filters
- Partitions
- Dataset

Optimized MapReduce Workflow

Transformation-based

MapReduce Workflow

Optimized MapReduce Workflow

MapReduce Execution Engine

Many Interfaces

Information Spectrum

Optimization Challenges
MapReduce Ecosystem

Annotations, MapReduce Workflow

Optimized MapReduce Workflow

MapReduce Execution Engine

Many Interfaces

Information Spectrum

Optimization Challenges
MapReduce Ecosystem

- Pig
- Hive
- Jaql
- FlumeJava

Annotations, MapReduce Workflow

- Stubby
  Transformation-based

Optimized MapReduce Workflow

MapReduce Execution Engine

- Many Interfaces
- Information Spectrum
- Optimization Challenges
MapReduce Ecosystem

Annotations, MapReduce Workflow

Optimized MapReduce Workflow

MapReduce Execution Engine

Many Interfaces

Information Spectrum

Optimization Challenges

Large Plan Space
MapReduce Ecosystem

Pig, Hive, Jaql, FlumeJava

Annotations, MapReduce Workflow

Stubby

Transformation-based

Optimized MapReduce Workflow

MapReduce Execution Engine

Many Interfaces

Information Spectrum

Optimization Challenges

Large Plan Space
Design Principles of Stubby
Design Principles of Stubby

• Optional *Annotations* convey information to Stubby

• *Transformations* allow for easy extension and customization of functionality

• Identification of *non-interacting subspaces* to deal with large plan space
Design Principles of Stubby

• Optional *Annotations* convey information to Stubby

• *Transformations* allow for easy extension and customization of functionality

• Identification of *non-interacting subspaces* to deal with large plan space
Design Principles of Stubby

• Optional *Annotations* convey information to Stubby

• *Transformations* allow for easy extension and customization of functionality

• Identification of *non-interacting subspaces* to deal with large plan space
Next

Transformations

Many Interfaces

MapReduce Workflow Optimization Challenges

Annotations

Information Spectrum

Large Plan Space

Interactions
Next

Transformations

Many Interfaces

MapReduce Workflow Optimization Challenges

Annotations

Information Spectrum

Large Plan Space

Interactions
Annotations

• Mechanism for higher levels to communicate information needed for workflow optimization
• 3 Types of Annotations

dataset = {schema=<C,O,I,N,SH>,
          partition=<hash(C)>}

K_1 = {C}
filter = {C<100}
K_2 = {O}
K_3 = {O}
map_cost = {50}
reduce_cost = {20}
Annotations

- Mechanism for higher levels to communicate information needed for workflow optimization
- 3 Types of Annotations

Dataset Annotations

\[
\text{dataset} = \{\text{schema}=<C,O,I,N,SH>, \text{partition}=<\text{hash}(C)>\}
\]

- \(K_1 = \{C\}\)
- \(\text{filter} = \{C<100\}\)
- \(K_2 = \{O\}\)
- \(K_3 = \{O\}\)
- \(\text{map\_cost} = \{50\}\)
- \(\text{reduce\_cost} = \{20\}\)
Annotations

- Mechanism for higher levels to communicate information needed for workflow optimization
- 3 Types of Annotations

\[
\begin{align*}
K_1 &= \{C\} \\
\text{filter} &= \{C < 100\} \\
K_2 &= \{O\} \\
K_3 &= \{O\} \\
\text{map_cost} &= 50 \\
\text{reduce_cost} &= 20
\end{align*}
\]

Dataset:
- \{schema=<C,O,I,N,SH>, partition=<hash(C)>\}

Schema and Filter Annotations
Annotations

• Mechanism for higher levels to communicate information needed for workflow optimization
• 3 Types ofAnnotations

\[
K_1 = \{C\}, \quad \text{filter} = \{C < 100\} \\
K_2 = \{O\} \\
K_3 = \{O\} \\
\text{map\_cost} = \{50\}, \quad \text{reduce\_cost} = \{20\}
\]

\[\text{dataset} = \{\text{schema} = \langle C, O, I, N, S, H \rangle, \text{partition} = \langle \text{hash}(C) \rangle\}\]
Who Creates the Annotations?
Who Creates the Annotations?

• Interfaces have all the information. Just propagate it
  • E.g., PigLatin statement: A = LOAD 'data' AS (A,B,C);
• Modified Pig to automatically generate dataset, schema, & filter annotations
  • Only ~570 lines of code! (Pig is ~80000 lines of code)
• Profile Annotations generated using Starfish [Herodotou VLDB 2011]
• Stubby considers optimizations based on what is given
Who Creates the Annotations?

- Interfaces have all the information. Just propagate it
  - E.g., PigLatin statement: `A = LOAD 'data' AS (A,B,C);`
- Modified Pig to automatically generate dataset, schema, & filter annotations
  - Only ~570 lines of code! (Pig is ~80000 lines of code)
- Profile Annotations generated using Starfish [Herodotou VLDB 2011]
- Stubby considers optimizations based on what is given
Who Creates the Annotations?

• Interfaces have all the information. Just propagate it
  • E.g., PigLatin statement: A = LOAD ‘data’ AS (A,B,C);
• Modified Pig to automatically generate dataset, schema, & filter annotations
  • Only ~570 lines of code! (Pig is ~80000 lines of code)
• Profile Annotations generated using Starfish [Herodotou VLDB 2011]
• Stubby considers optimizations based on what is given
Who Creates the Annotations?

- Interfaces have all the information. Just propagate it
  - E.g., PigLatin statement: $A = \text{LOAD} \ 'data' \ \text{AS} \ (A,B,C)$;
- Modified Pig to automatically generate dataset, schema, & filter annotations
  - Only $\sim 570$ lines of code! (Pig is $\sim 80000$ lines of code)
- Profile Annotations generated using Starfish [Herodotou VLDB 2011]
- Stubby considers optimizations based on what is given
Next

Transformations

Many Interfaces

MapReduce Workflow Optimization Challenges

Information Spectrum

Annotations

Large Plan Space

Interactions
Transformations

Many Interfaces

MapReduce Workflow Optimization Challenges

Annotations

Information Spectrum

Large Plan Space

Interactions
Transformations

- Transformations + Annotations allow Stubby to support different interfaces by being *External* to any interface
Transformations

Transformations + Annotations allow Stubby to support different interfaces by being *External* to any interface.
Transformations

• Transformations + Annotations allow Stubby to support different interfaces by being *External* to any interface.
Transformations

- Transformations + Annotations allow Stubby to support different interfaces by being *External* to any interface.
Transformations

- Transformations + Annotations allow Stubby to support different interfaces by being *External* to any interface

- Annotations ensure only *valid* transformations are considered
Transformations

- Transformations + Annotations allow Stubby to support different interfaces by being *External* to any interface.

- Annotations ensure only *valid* transformations are considered.
- Transformations can be combined (whole >> sum of parts!)
- Stubby considers 5 types of transformations (more to come)
Transformations

- Transformations + Annotations allow Stubby to support different interfaces by being *External* to any interface.

- Annotations ensure only *valid* transformations are considered.
- Transformations can be combined (whole >> sum of parts!)
- Stubby considers 5 types of transformations (more to come)
Intra-Job Vertical Packing

• Transforms a MapReduce job into a Map-only job
Intra-Job Vertical Packing

- Transforms a MapReduce job into a Map-only job
Intra-Job Vertical Packing

- Transforms a MapReduce job into a Map-only job

![Diagram showing the transformation process]

**Part 1:**
- Initial MapReduce job
- 
  - \(M\) to \(R\) to \(M\)
  - Hash \((O, Z)\)
  - Sort \((O, Z)\)
  - \(J.K_2 = \{O, Z\}\)

**Part 2:**
- Map-only job
- 
  - \(M\) to \(R\) to \(M\)
  - Hash \((O)\)
  - Sort \((O, Z)\)

**Transformation:**
- arrows indicating the flow of data and operations from the initial MapReduce job to the Map-only job.
Intra-Job Vertical Packing

- Transforms a MapReduce job into a Map-only job
Intra-Job Vertical Packing

- Transforms a MapReduce job into a Map-only job
- Group/Partition requirements of both jobs is now enforced at the same time
Intra-job Vertical Packing (2)

- Can have positive / negative effect on performance -> Need cost-based approach
Intra-job Vertical Packing (2)

- Can have positive / negative effect on performance -> Need cost-based approach

- Forces dependencies of configurations (e.g., parallelism)
- Resource contention (more functions in a task)
Intra-job Vertical Packing (2)

- Can have positive / negative effect on performance -> Need cost-based approach

+ Eliminates inter-task data transfer
+ Eliminates sorting overhead
+ Eliminates writing output to disk

- Forces dependencies of configurations (e.g., parallelism)
- Resource contention (more functions in a task)
Inter-job Vertical Packing

- Merges a map-only job with another job
Inter-job Vertical Packing

- Merges a map-only job with another job
Inter-job Vertical Packing

- Merges a map-only job with another job
Inter-job Vertical Packing

- Merges a map-only job with another job
Inter-job Vertical Packing

• Merges a map-only job with another job

• If combine intra-job + inter-job -> 2 MapReduce jobs to 1 MapReduce job
Inter-job Vertical Packing

- Merges a map-only job with another job

Again, not always a good thing
+ Eliminates writing to disk
- Forces dependencies
Horizontal Packing

• Combine concurrent running jobs into a single job
Horizontal Packing

• Combine concurrent running jobs into a single job
Horizontal Packing

• Combine concurrent running jobs into a single job
Horizontal Packing

• Combine concurrent running jobs into a single job
Horizontal Packing

- Combine concurrent running jobs into a single job
  - + Read dataset once
  - + Share overhead of launching jobs
  - - Extra overhead of sorting/partitioning combined map output
  - - Share limited memory resources per task (can spill more)
Partition Function

- Change how map outputs are partitioned and sorted
Partition Function

- Change how map outputs are partitioned and sorted

\[ \text{hash}(O) \]

\[ \text{filter} = \{0 \leq O < 100\} \]
Partition Function

- Change how map outputs are partitioned and sorted

![Diagram showing partition function with hash(O) and a filter={0<=O<100} transformation]
Partition Function

• Change how map outputs are partitioned and sorted

\[ \text{hash}(O) \]

\[ \text{range}(O) \text{ \textit{split-points}}(100, 200, \ldots) \]

\[ \text{filter} = \{0 \leq O < 100\} \]
Partition Function

- Change how map outputs are partitioned and sorted

\[
\text{hash}(O) \quad \text{range}(O) \quad \text{split-points}(100, 200, \ldots)
\]

- Enables partition pruning
- Enables vertical packing transformation
Configuration Transformation

• Changes the configuration of a MapReduce job
Configuration Transformation

• Changes the configuration of a MapReduce job

Memory Buffer 512MB

2 Reduce Tasks
Configuration Transformation

- Changes the configuration of a MapReduce job

- Memory Buffer 512MB vs. Memory Buffer 128MB

- Transformation

- 2 Reduce Tasks vs. 4 Reduce Tasks
Configuration Transformation

- Changes the configuration of a MapReduce job

- Many configurations that affect performance (e.g., sort buffer, compression, combiner, reduce tasks, etc)

- Impact of configuration depends on other transformations (interaction)
Configuration Transformation

- Changes the configuration of a MapReduce job

- Many configurations that affect performance (e.g., sort buffer, compression, combiner, reduce tasks, etc)

- Impact of configuration depends on other transformations (interaction)
Next

Transformations

Many Interfaces

MapReduce Workflow Optimization Challenges

Annotations

Information Spectrum

Large Plan Space

Interactions
Next

- Transformations
  - Many Interfaces
  - MapReduce Workflow Optimization Challenges

- Annotations
  - Information Spectrum

- Interactions
  - Large Plan Space
Optimization Process
Optimization Process

Optimization unit localizes interactions among plan space choices
Optimization Process
Optimization Process
Optimization Process

Dynamically generated because previous optimization unit transforms workflow.

*Top-Down* because producer jobs affect the input datasets of consumer jobs.
Optimization Process

D0₁  D0₂
M1  M2
R1  R2
D1  D2

M3  R3  M4
M5  M6
R5  R6
M7  R7

D7  D6

J1-2

U⁽⁴⁾

J3-7

(4)
Divide and Conquer
Divide and Conquer

• Divide workflow into *Optimization Units* to have smaller plan spaces
• Issue: *Interactions* among plan space choices
• Insight: Based on *Dataset* and *Resource* dependencies
Divide and Conquer

• Divide workflow into *Optimization Units* to have smaller plan spaces

• Issue: *Interactions* among plan space choices

• Insight: Based on *Dataset* and *Resource* dependencies
Divide and Conquer

- Divide workflow into *Optimization Units* to have smaller plan spaces
- Issue: *Interactions* among plan space choices
- Insight: Based on *Dataset* and *Resource* dependencies
Divide and Conquer

- Divide workflow into *Optimization Units* to have smaller plan spaces
- Issue: *Interactions* among plan space choices
- Insight: Based on *Dataset* and *Resource* dependencies

- Divide into producer-consumer relationships
  - Transformations on producer jobs, affect transformations on consumer jobs
    - E.g, partition function on J5 -> vertical packing on J7, compressing D5 forces J7 to decompress
Divide and Conquer

• Divide workflow into *Optimization Units* to have smaller plan spaces
• Issue: *Interactions* among plan space choices
• Insight: Based on *Dataset* and *Resource* dependencies

- Divide into producer-consumer relationships
  - Transformations on producer jobs, affect transformations on consumer jobs
    - E.g, partition function on J5 -> vertical packing on J7, compressing D5 forces J7 to decompress
Divide and Conquer

- Divide workflow into *Optimization Units* to have smaller plan spaces
- Issue: *Interactions* among plan space choices
- Insight: Based on *Dataset* and *Resource* dependencies

- Concurrent jobs use the same cluster resources
  - E.g., affect configuration and horizontal packing transformations
Divide and Conquer

- Divide workflow into *Optimization Units* to have smaller plan spaces
- Issue: *Interactions* among plan space choices
- Insight: Based on *Dataset* and *Resource* dependencies
Within an Optimization Unit

- Enumerate all valid combinations of *packing* transformations
Within an Optimization Unit

- Enumerate all valid combinations of packing transformations
Within an Optimization Unit

- Enumerate all valid combinations of packing transformations
Within an Optimization Unit

- Enumerate all valid combinations of \textit{packing} transformations
- Use Starfish’s What-If Engine [Herodotou VLDB 2011] for costing
- Use Recursive Random Search [Ye SIGMETRICS 03] to find \textit{configurations} with best cost for each combination $p_i$
Within an Optimization Unit

- Enumerate all valid combinations of packing transformations
- Use Starfish’s What-If Engine [Herodotou VLDB 2011] for costing
- Use Recursive Random Search [Ye SIGMETRICS 03] to find configurations with best cost for each combination $p_i$
Within an Optimization Unit

- Enumerate all valid combinations of *packing* transformations
- Use Starfish’s What-If Engine [Herodotou VLDB 2011] for costing
- Use Recursive Random Search [Ye SIGMETRICS 03] to find *configurations* with best cost for each combination $p_i$
Within an Optimization Unit

• Enumerate all valid combinations of *packing* transformations
• Use Starfish’s What-If Engine [Herodotou VLDB 2011] for costing
• Use Recursive Random Search [Ye SIGMETRICS 03] to find *configurations* with best cost for each combination $p_i$
• Pick combination with lowest cost
Transformations

Many Interfaces

MapReduce Workflow Optimization Challenges

Large Plan Space

Annotations

Information Spectrum

Speedup

Baseline

Stubby

IR  SN  LA  WG  BA  BR  PJ  US
Next

Transformations

Many Interfaces

Information Spectrum

Annotations

MapReduce Workflow Optimization Challenges

Large Plan Space

Interactions

<table>
<thead>
<tr>
<th>Speedup</th>
<th>IR</th>
<th>SN</th>
<th>LA</th>
<th>WG</th>
<th>BA</th>
<th>BR</th>
<th>PJ</th>
<th>US</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Stubby</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Implementation
Implementation

• Minimal code changes to Apache Pig
  • ~570 lines to generate annotations
  • ~65 lines to import/export workflows
  • ~800 lines for runtime support of optimized workflows (e.g., wrapper MapReduce classes to run multiple functions in map/reduce tasks)

• Similar effort expected for Stubby to support other interfaces

Out of 80000 lines of Pig source code!
Implementation

• Minimal code changes to Apache Pig
  • ~570 lines to generate annotations
  • ~65 lines to import/export workflows
  • ~800 lines for runtime support of optimized workflows (e.g., wrapper MapReduce classes to run multiple functions in map/reduce tasks)

• Similar effort expected for Stubby to support other interfaces

Out of 80000 lines of Pig source code!
Experimental Evaluation

- 51 Amazon EC2 m1.large nodes
- Representative MapReduce workflows from several application domains (ranges from 2 to 7 jobs)
- *Baseline* – Enabled all rule-based optimization supported in Pig and manually-tuned configurations using rules-of-thumb

<table>
<thead>
<tr>
<th>Abbr</th>
<th>Workflow</th>
<th>Dataset Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>Information Retrieval</td>
<td>264GB</td>
</tr>
<tr>
<td>SN</td>
<td>Social Network Analysis</td>
<td>267 GB</td>
</tr>
<tr>
<td>LA</td>
<td>Log Analysis</td>
<td>500 GB</td>
</tr>
<tr>
<td>WG</td>
<td>Web Graph Analysis</td>
<td>255 GB</td>
</tr>
<tr>
<td>BA</td>
<td>Business Analytics Query</td>
<td>550 GB</td>
</tr>
<tr>
<td>BR</td>
<td>Business Report Generation</td>
<td>530 GB</td>
</tr>
<tr>
<td>PJ</td>
<td>Post-processing Jobs</td>
<td>10 GB</td>
</tr>
<tr>
<td>US</td>
<td>User-defined Logical Splits</td>
<td>530 GB</td>
</tr>
</tbody>
</table>
Performance Improvements

• Different workflows present different transformation opportunities
• 2X to 4.5X speedup over Baseline

![Graph showing performance improvements across different workflows](Image)

- Baseline
- Stubby
- Vertical
- Horizontal

<table>
<thead>
<tr>
<th>Workflow</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>2.1</td>
</tr>
<tr>
<td>SN</td>
<td>3.4</td>
</tr>
<tr>
<td>LA</td>
<td>2.3</td>
</tr>
<tr>
<td>WG</td>
<td>2.7</td>
</tr>
<tr>
<td>BA</td>
<td>4.5</td>
</tr>
<tr>
<td>BR</td>
<td>1.8</td>
</tr>
<tr>
<td>PJ</td>
<td>2.2</td>
</tr>
<tr>
<td>US</td>
<td>1.9</td>
</tr>
</tbody>
</table>
Comparison against State-of-the-Art

- Starfish [Herodotou VLDB 2011] – Cost-based selection of configuration parameters
- MRShare [Nykiel VLDB 2010] – Cost-based horizontal packing transformation
Optimization Efficiency

- Average case: < 2 minutes optimization time, 3% overhead
- Worst case: 5 minutes optimization time, 10.5% overhead

![Graph showing optimization time and overhead for different models](image-url)
Related Work

- Optimizing data-parallel workflows
  - Rule-based: FlumeJava [PLDI 2010], YSmart [ICDCS 2011], Manimal [VLDB 2011], Jaql [VLDB 2011]
  - Cost-based: MRShare [VLDB 2010], Starfish [VLDB 2011]
- Other transformations
  - Multi-way joins: Wu et al. [SOCC 2011]
  - Transformation-based optimizer for SCOPE system: Zhou et al. [ICDE 2010]
  - Fault-tolerance: FTOpt [SIGMOD 2011]
- Computation of multiple aggregates over the same or similar sets of grouping attributes: Chatziantoniou et al. [VLDB 1996]
- ETL workflows: Simitsis et al. [ICDE 2005]
Conclusions

• Extensible transformation-based optimizer
• Annotations as medium for information
• Identify non-interacting subspaces
• Speedups of up to 4.5X over the baseline
• http://www.cs.duke.edu/starfish