Modeling and Analyzing Key Performance Factors of Shared Memory MapReduce

Devesh Tiwari and Yan Solihin
Electrical and Computer Engineering
North Carolina State University
Improving performance is already hard
Exploiting multithreading is even harder!
Parallel programming runtime systems are potentially part of the solution e.g. MapReduce runtime system
MapReduce Runtime Systems

- Deterministic execution
- Increased programmer productivity
- Automatic concurrency management
- Wide commercial and open-source adoption
e.g. Google, Facebook, Amazon, and Yahoo etc.

Shared memory MapReduce runtime systems

Phoenix (Stanford) and Metis (MIT)
Parallel programming runtime systems **increase programming productivity** but what about performance?
Motivation

Computation expressible in key-value pairs is expected to perform well under MapReduce, e.g. word count?
Motivation

Computation expressible in key-value pairs is expected to perform well under MapReduce, e.g. word count?

- **Input A** (all identical words)
- **Input B** (all unique words)
- **Input C** (all mixed words)

Execution Time Normalized Word Count (WC) Input A

- WC (1 Thread) = 67x
- WC (8 Threads) = 44x
Goals of This Study

- How does input content affect MapReduce performance?
- What factors affect Map and Reduce phase performance?
- What conditions create a performance bottleneck?
- How to choose a runtime system and its parameters (e.g. number of buckets, map and reduce threads)?
Goals of This Study

- How does input content affect MapReduce performance?
- An analytical model for capturing key performance factors of shared memory MapReduce
- How to choose a runtime system and its parameters (e.g. number of buckets, map and reduce threads)?
Motivation and Goals

Related Work

Shared Memory MapReduce

Analytical Performance Model

Conclusions
## Scope of This Study

### MapReduce Runtime Systems

<table>
<thead>
<tr>
<th>Cluster/ Disk Based</th>
<th>Shared Memory Multicore Runtime System</th>
<th>GPU FPGA IBM Cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop Hive</td>
<td>Phoenix Metis</td>
<td>Mars MapCG</td>
</tr>
</tbody>
</table>
Scope of This Study

MapReduce Runtime Systems

<table>
<thead>
<tr>
<th>Cluster/Disk Based</th>
<th>Shared Memory Multicore Runtime System</th>
<th>GPU FPGA IBM Cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop Hive</td>
<td>Phoenix Metis</td>
<td>Mars MapCG</td>
</tr>
</tbody>
</table>

Key performance factors of shared memory MapReduce are **not** well understood – the focus of this work
Outline

- Motivation and Goals
- Related Work
- Shared Memory MapReduce
- Analytical Performance Model
- Conclusions
Shared Memory MapReduce

Map Worker Threads

Map Tasks

Input
Hash Buckets for storing <key, value> pairs
Shared Memory MapReduce
Shared Memory MapReduce

<USA, 3>
<India, 1>
<China, 2>

<Japan, 7>
<France, 2>
Intermediate Buffer

Shared Memory MapReduce
Shared Memory MapReduce

Intermediate Buffer
Shared Memory MapReduce

Reduce Worker Threads
Shared Memory MapReduce

Reduce Worker Threads
Outline

- Motivation and Goals
- Related Work
- Shared Memory MapReduce
  - Analytical Performance Model
    - Map Phase Model
    - Reduce Phase Model
- Conclusions
Analytical Performance Model

- Markov process model
  - Accounts for order of operations

- Algorithmic complexity analysis
  - Accounts for asymptotic cost of algorithms

- Architecture dependent parameters
  - Accounts for relative weights of data structure/algorithm components
Analytical Performance Model: Outline

- Map Phase
  - Performance Model
  - Model Driven Study
  - Model Validation
  - Experimental Results
Uniform Key Ordering (UKO) ~ best case

Map Phase Performance Model
Map Phase Performance Model

Uniform Key Ordering (UKO) ~ best case

Skewed Key Ordering (SKO) ~ worst case
Map Phase Performance Model

Uniform Key Ordering (UKO) ~ best case

Skewed Key Ordering (SKO) ~ worst case

Map Phase Time = Map Comp Time + Map Output Time
Map Phase Performance Model

Uniform Key Ordering (UKO) ~ best case

Skewed Key Ordering (SKO) ~ worst case

Map Output Time = Key Search Time + Key Insert Time + Value Insert Time + Key Hashing Time
Map Phase Performance Model

Uniform Key Ordering (UKO) ~ best case

Skewed Key Ordering (SKO) ~ worst case

Map Output Time = Key Search Time + Key Insert Time + Value Insert Time + Key Hashing Time

Each component has associated weight and algorithmic complexity
<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>Total Number of Intermediate Pairs</td>
</tr>
<tr>
<td>$D$</td>
<td>Number of Distinct Keys</td>
</tr>
<tr>
<td>$B$</td>
<td>Number of Hash Buckets</td>
</tr>
<tr>
<td>$N_{\text{map}}$</td>
<td>Number of Map Threads</td>
</tr>
<tr>
<td>$N_{\text{redt}}$</td>
<td>Number of Reduce Threads</td>
</tr>
<tr>
<td>$w_1, w_2, \ldots, w_5$</td>
<td>Parameters (weights) for Map Phase</td>
</tr>
<tr>
<td>$w_6, w_7, w_8$</td>
<td>Parameters (weights) for Reduce Phase</td>
</tr>
</tbody>
</table>
Uniform Key Ordering (UKO)

\[ T_{mo}^{UKO} = w_1 \left( \frac{P}{D} \log D! - \log D \right) + w_2 \frac{D(D+1)}{2} + (w_3 + w_4)P \]

Skewed Key Ordering (SKO)

\[ T_{mo}^{SKO} = w_1 \left( \log D! + (P - D - 1) \log D \right) + w_2 \frac{D(D+1)}{2} + (w_3 + w_4)P \]

Key Searching

Key Insertion

Value Insertion

+ Key Hashing
Analytical Performance Model: Outline

- Map Phase
  - Performance Model
  - Model Driven Study
  - Model Validation
  - Experimental Results
Map Phase Model Driven Study

![Graph showing the effect of skewed key ordering versus uniform key ordering on the normalized Tmo (Phoenix) with respect to the number of keys (D).]
Map Output time depends on keys/pairs ratio (D/P), and also on key occurrence ordering.
Phoenix vs Metis Hash Bucket Structure

Phoenix: Sorted Array
Key Search: $O(\log(N))$  Key Insert: $O(N)$

Metis: B+ Tree
Key Search: $O(\log(N))$  Key Insert: $O(\log(N))$
Empirically derived weight factors determine which runtime is better.
Algorithmic complexity analysis alone leads to an incorrect conclusion.
Analytical Performance Model: Outline

- Map Phase
  - Performance Model
  - Model Driven Study
  - Model Validation
  - Experimental Results
Map Phase Model Validation Insights

Key Search Weight

Key Insert Weight
Compared to B+ Tree, sorted array has faster single key searching and insertion
Map Phase Model Validation Insights

Value Insert Weight

Key Hashing Weight
Map Phase Model Validation Insights

Value Insert Weight

Key Hashing Weight

Key hashing time > Value insertion time
Analytical Performance Model: Outline

- Map Phase
  - Performance Model
  - Model Driven Study
  - Model Validation
  - Experimental Results
Map Phase Experimental Results

Phenix

- Skewed Key Ordering
- Uniform Key Ordering

Metis

- Skewed Key Ordering
- Uniform Key Ordering

Number of Keys: 25K, 50K, 75K, 100K, 125K, 150K

Map-output Time (Phoenix) vs. Number of Keys

Map-output Time (Metis) vs. Number of Keys
Map Phase Experimental Results

Key occurrence order affects Map-output time significantly
Difference between SKO and UKO is higher for Metis as explained by analytical model.

Map Phase Experimental Results

Phoenix

Metis
Model + actual weight factors explain performance of Phoenix vs. Metis
Analytical Performance Model : Outline

- Reduce Phase
  - Performance Model
  - Model Driven Study
  - Model Validation
  - Model Validation
  - Experimental Results
Reduce Phase Model Insights

\[ T_{reduce} = w_6 \frac{P}{N_{redt}} + w_7 B \]

- Value Addition
- Reduce Task Queue Overhead
Reduce Phase Model Insights

\[ T_{reduce} = w_6 \frac{P}{N_{redt}} + w_7B \]

- Value Addition
- Reduce Task Queue Overhead

Effects of increasing number of buckets (B)

- Decreases Map time complexity from \(O(P \log D)\) to \(O(P)\)
- But increases Reduce time from \(O(P)\) to \(O(D^2)\)
Increasing number of hash buckets increases the reduce phase time
Number of reduce threads needs to be chosen carefully
Reduce Phase Model Validation Insights

Value Addition Weight

Reduce Task Queue Weight
Reduce Phase Model Validation Insights

Value Addition Weight
Reduce Task Queue Weight

Reduce task queue overhead > Value addition
Real experiments confirm the model observations

Increase in number of hash increases the reduce time
Optimal number of reduce threads may be non-intuitive
Reduce phase time may dominate execution time.
Outline

- Motivation and Goals
- Related Work
- Shared Memory MapReduce
- Analytical Performance Model
- Conclusions
Conclusions

✓ Performance model to identify and quantify key performance factors of shared memory MapReduce

✓ It captures and explains interesting performance phenomena and non-intuitive design trade-offs.

✓ Useful as a diagnosis and tuning tool for programmers and performance tuners (application classification framework in the paper).
Questions
Devesh Tiwari
Email: devesh.dtiwari@ncsu.edu
Homepage: http://www4.ncsu.edu/~dtiwari2

Thanks!
Back Up Slides
“Hello World!” of MapReduce

Map Phase

...... C B D A B A D B C ......

Reduce Phase

Key Shuffle Stage
SKO vs UKO
Map Output time depends on key ordering
Application Classification Framework

Number of Distinct Keys

Frequency of Keys

Image Type

LR

WC (All Identical)

HG

Cluster Size

KM

WC (Mixed)

SM

MM

WC (All Unique)

Application Classification Framework