Hadoop Optimizations for BigData Analytics

Weikuan Yu
Auburn University

Auburn University
Mellanox®
TECHNOLOGIES
IBM®
Outline

• Background
• Network Levitated Merge
• JVM-Bypass Shuffling
• Fast Completion Scheduler
Emerging Demand for BigData Analytics

• Big demand from many organizations in various domains
  – Scalable computing power without worrying about system maintenance.
  – Ubiquitously accessible computing and storage resources.
  – Low cost, highly reliable, trusted computing infrastructure.

• Commercial companies are gearing up resources for BigData
MapReduce

- A simple data processing model to process big data
- Designed for commodity off-the-shelf hardware components.
- Strong merits for big data analytics
  - **Scalability**: increase throughput by increasing # of nodes
  - **Fault-tolerance** (quick and low cost recovery of the failures of tasks)
- Hadoop, An open-source implementation of MapReduce:
  - Widely deployed by many big data companies: AOL, Baidu, EBay, Facebook, IBM, NY Times, Yahoo! ...
High-Level Overview of Hadoop

- HDFS and the MapReduce Framework
- Data processing with MapTasks and ReduceTasks
- Three main steps of data movement.
  - Intermediate data shuffling in the MapReduce is time-consuming
Outline

• Background
• Network Levitated Merge
• JVM-Bypass Shuffling
• Fast Completion Scheduler
Motivation for Network-Levitated Merge

1: Serialization between shuffle/merge and reduce phases
Repetitive Merges and Disk Access

- Hadoop data spilling controlled through parameters
  - To limit the number of outstanding files
  - An example with io.sort.factor=3
Hadoop Acceleration (Hadoop-A)

- Pipelined shuffle, merge and reduce
- Network-levitated data merge
Pipelined Data Shuffle, Merge and Reduce

- **Shuffle**
- **Merge**
- **Map**
- **Reduce**
- **Header**
- **PQ Setup**

Timeline:
- **Start**
- **First MOF**
- **Last MOF**

Time
Network-Levitated Merge Algorithm

(a) Fetching Header

(b) Priority Queue Setup

(c) Concurrent Fetching & Merging

(d) Towards Completion
Job Progression with Network-levitated Merge

• a) Hadoop-A speeds up the execution time by more than 47%
• b) Both MapTasks and ReduceTasks are improved

![Graph showing progression of TeraSort](image)

a) Map Progress of TeraSort

b) Reduce Progress of TeraSort
Breakdown of ReduceTask Execution Time (sec)

- Significantly reduced the execution time of ReduceTasks
- Most came from reduced shuffle/merge time
  - An improvement of 2.5 times
- Also improved the time to reduce data
  - An improvement of 15%

<table>
<thead>
<tr>
<th>Category</th>
<th>PQ-Setup</th>
<th>Shuffle/Merge</th>
<th>Reduce or Merge/Reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop-GigE</td>
<td>1150.01  (65.0%)</td>
<td>599.97 (35.0%)</td>
<td></td>
</tr>
<tr>
<td>Hadoop-IPoIB</td>
<td>1148.26  (65.9%)</td>
<td>597.09 (34.1%)</td>
<td></td>
</tr>
<tr>
<td>Hadoop-A</td>
<td>460.01   (47.4%)</td>
<td>509.81 (52.6%)</td>
<td></td>
</tr>
</tbody>
</table>
Outline

• Background
• Network Levitated Merge
• JVM-Bypass Shuffling
• Fast Completion Scheduler
JVM-Dependent Intermediate Data Shuffling

Heavily relies on Java

TCP/IP-Only

HDFS
JVM-Bypass Shuffling (JBS)

- JBS removes JVM from the critical path of intermediate data shuffling
- JBS is a portable library supporting both TCP/IP and RDMA protocols
Benefits of JBS: 1/10 Gigabit Ethernets

- JBS is effective for intermediate data of different sizes
  - Using Terasort benchmark, size of intermediate data = size of input data
- JBS reduces the execution time by **20.9%** on average in 1GigE, **19.3%** on average in 10GigE

![Graph](a): 1 Gigabit Ethernet

![Graph](b): 10 Gigabit Ethernet
Benefits of JBS: InfiniBand Cluster

- JBS on IPoIB outperforms Hadoop on IPoIB and SDP by 14.1%, 14.8%, respectively.
- Hadoop performs similarly when using IPoIB or SDP.
Outline

• Background
• Network Levitated Merge
• JVM-Bypass Shuffling
• Fast Completion Scheduler
Hadoop Fair Scheduler

- Scheduler assigns tasks to the TaskTrackers
- Tasks occupy slots until completion or failure

![Diagram showing task scheduling and job arrival (shuffle and reduce)]
Fair Completion Scheduler

• Prioritize ReduceTasks based on the shortest remaining map phase

• When remaining map phases are equal, prioritize ReduceTasks of jobs with least remaining reduce data

• Track the slowdown of preempted ReduceTasks
  – Prevent large jobs from being preempted for too long
Average ReduceTask Waiting Time

• ReduceTasks in small jobs are significantly speedup
Conclusions

• Examined the design and architecture of Hadoop MapReduce framework and reveal critical issues faced by the existing implementation

• Designed and implemented Hadoop-A as an extensible acceleration framework which addresses all these issues

• Provided JVM-Bypass Shuffling to avoid JVM overhead, meanwhile we enable it to be a portable library that can run on both TCP/IP and RDMA protocols.

• Designed and Implemented Fast Completion Scheduler for fast job completion and job fairness.
Sponsors of our research