Outline

• **Background and Motivation**
• Design and Implementation of Hadoop Acceleration
• Performance Evaluation
• More Scalable Algorithm and Performance
• Conclusions and Future Work
Cloud Computing for Emerging Data Crisis

- 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 the cloud
MapReduce and Hadoop

- MapReduce paper by J. Dean et al. *MapReduce: Simplified Data Processing on Large Clusters*. OSDI 2004
- Hadoop: Open-source implementation of MapReduce with apache license.
- Exposes simple `map` and `reduce` interfaces for users.
- Hadoop is written in Java, but supports interfaces for other languages.
  - C++, Python, etc
- Designed for commodity machines rather than relying on hardware.
High-Level Overview of Hadoop Framework

- Applications
- Retained User Interface
- Submitter
- JobTracker
- HDFS
- Task Tracker/Runner
- Task Tracker/Runner
- Data Input
- Result Output
- Shuffle
- Intermediate Data
- Map
- Reduce
Data Movement in Hadoop MapReduce

- **Split**: Data is split into smaller chunks.
- **Map Task**: Each Map Task processes a chunk of data.
- **Shuffle**: Data is shuffled and moved to memory (MEM) or distributed file system (DFS).
- **Reduce Task**: Reduce Tasks merge and aggregate the shuffled data.

**Job Tracker**:
- Assigns Map Tasks to available machines.
- Assigns Reduce Tasks to available machines.
Issues in Hadoop Framework

- There are several critical issues in the existing Hadoop framework:
  - The serialization between Hadoop shuffle/merge and reduce Phases
  - Repetitive merges and disk access
  - Lack of support high-speed interconnects
Issues in Hadoop Framework cont’d

1: Serialization between shuffle/merge and reduce phases

Start → First MOF → Serialization → Time

- shuffle
- merge
- map
- reduce
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
No Support for High Speed Interconnects RDMA

- Data Shuffling through HTTP-Get operations
- Built entirely on Java Sockets
  - Lots of overheads going through the following
    - HTTP library
    - Java Socket library
    - JVM
- No support of RDMA available from InfiniBand and RoCE (RDMA over Converged Ethernet)
Hadoop-A (Unstructured Data Accelerator)

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

- **shuffle**
- **merge**
- **map**
- **reduce**
- **header**
- **PQ setup**

Time scale:
- **start**
- **First MOF**
- **Last MOF**
Network-Levitated Merge Algorithm

(a) Fetching Header

(b) Priority Queue Setup

(c) Concurrent Fetching & Merging

(d) Towards Completion
RDMA-Accelerated Data Shuffling

Hadoop

HTTP Servlet
JVM Sockets
TCP/IP Sockets
Ethernet

HTTP Get
JVM Sockets
TCP/IP Sockets

Hadoop Acceleration

No JVM
No JVM

RDMA Server
Verbs
Interconnects (InfiniBand/RoCE, etc)

RDMA Client
Verbs

Java
C++
Evaluation Results: Overall Performance

- a) UDA speeds up the total execution time significantly by more than 47%
- b) Both MapTasks and ReduceTasks are improved

![Graph of Map Progress of TeraSort](image1)

![Graph of Reduce Progress of TeraSort](image2)
Breakdown of ReduceTask Execution Time (sec)

- Results from 24-node tests.
- Significantly reduced the execution time of ReduceTask
- 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></td>
<td>1150.01 (65.0%)</td>
<td>599.97 (35.0%)</td>
</tr>
<tr>
<td>Hadoop-IPoIB</td>
<td></td>
<td>1148.26 (65.9%)</td>
<td>597.09 (34.1%)</td>
</tr>
<tr>
<td>Hadoop-A</td>
<td>460.01 (47.4%)</td>
<td></td>
<td>509.81 (52.6%)</td>
</tr>
</tbody>
</table>
Improved Data Processing Scalability with UDA

- a) Significant performance improvement with UDA
- b) Compared to Hadoop, UDA improves the performance by 50% and 40%-43% with weak scaling and strong scaling.

![Graphs showing execution time comparisons between Hadoop and UDA](image-url)
Reduced CPU Utilization with UDA

- a) UDA has less CPU utilization and save more resources.
- b) Compared to Hadoop-IPoIB and Hadoop-GigE, cumulative reduction of CPU utilization is 36.2% and 44.2%.
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
- Experimental results demonstrate that Hadoop-A doubles the data processing throughput of Hadoop and reduces the CPU utilization by more that 35%
Sponsors of our research