Exploiting HPC Technologies for Accelerating Big Data Processing and Storage

Talk in the 5194 class

by

Xiaoyi Lu
The Ohio State University
E-mail: luxi@cse.ohio-state.edu
http://www.cse.ohio-state.edu/~luxi
Introduction to Big Data Analytics and Trends

• Big Data has changed the way people understand and harness the power of data, both in the business and research domains

• Big Data has become one of the most important elements in business analytics

• Big Data and High Performance Computing (HPC) are converging to meet large scale data processing challenges

• Running High Performance Data Analysis (HPDA) workloads in the cloud is gaining popularity
  • According to the latest OpenStack survey, 27% of cloud deployments are running HPDA workloads
4V Characteristics of Big Data

- Commonly accepted 3V’s of Big Data
  - Volume, Velocity, Variety

- 4/5V’s of Big Data – 3V + *Veracity, *Value

Courtesy: http://api.ning.com/files/tRHkwQN7s-Xz5cxylXG004GLGfjdoPd6bVFVBBwvgu*F5MwDDUCiHHdmBW-JTEz0cfjJGurJucBMfNiUaNdL3jcZT81PfNWfN9/dv1.jpg
• From 2005 to 2020, the digital universe will grow by a factor of 300, from 130 exabytes to 40,000 exabytes.

• By 2020, a third of the data in the digital universe (more than 13,000 exabytes) will have Big Data Value, but only if it is tagged and analyzed.

Big Velocity – How Much Data Is Generated Every Minute on the Internet?

The global Internet population grew 7.5% from 2016 and now represents 3.7 Billion People.

Data Management and Processing on Modern Clusters

- Substantial impact on designing and utilizing data management and processing systems in multiple tiers
  - Front-end data accessing and serving (Online)
    - Memcached + DB (e.g. MySQL), HBase
  - Back-end data analytics (Offline)
    - HDFS, MapReduce, Spark
Not Only in Internet Services - Big Data in Scientific Domains

- Scientific Data Management, Analysis, and Visualization
- Applications examples
  - Climate modeling
  - Combustion
  - Fusion
  - Astrophysics
  - Bioinformatics
- Data Intensive Tasks
  - Runs large-scale simulations on supercomputers
  - Dump data on parallel storage systems
  - Collect experimental / observational data
  - Move experimental / observational data to analysis sites
  - Visual analytics – help understand data visually
Presentation Outline

• Overview
  – MapReduce and RDD Programming Models
  – Apache Hadoop, Spark, Memcached, gRPC, and TensorFlow
  – Modern Interconnects and Protocols

• Challenges in Accelerating Hadoop, Spark, Memcached, gRPC, and TensorFlow

• Acceleration Case Studies and In-Depth Performance Evaluation

• The High-Performance Big Data (HiBD) Project and Associated Releases

• Conclusion and Q&A
WordCount Execution in MapReduce

- The overall execution process of WordCount in MapReduce

```
Input: Deer Bear River Car Car River Deer Car Bear

Splitting: Deer Bear River, Car Car River, Deer Car Bear

Mapping: Deer, 1 Bear, 1 River, 1 Car, 1

Shuffling: Bear, 1 Bear, 1 Car, 1 Car, 1 Deer, 1 River, 1

Reducing: Bear, 2 Car, 3 Deer, 2 River, 2

Final result: Bear, 2 Car, 3 Deer, 2 River, 2
```
A Hadoop MapReduce Example - WordCount

public class WordCount {
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    }
    public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterator<IntWritable> values, Context context)
                throws IOException, InterruptedException {
            int sum = 0;
            while (values.hasNext()) {
                sum += values.next().get();
            }
            context.write(key, new IntWritable(sum));
        }
    }
}
Data Sharing Problems in MapReduce

Slow due to replication, serialization, and disk IO

10-100× faster than network and disk
RDD Programming Model in Spark

• Key idea: *Resilient Distributed Datasets* (**RDDs**)
  – Immutable distributed collections of objects that can be cached in memory across cluster nodes
  – Created by transforming data in stable storage using data flow operators (map, filter, groupBy, ...)
  – Manipulated through various parallel operators
  – Automatically rebuilt on failure
    • rebuilt if a partition is lost

• Interface
  – Clean language-integrated API in Scala (Python & Java)
  – Can be used *interactively* from Scala console
## RDD Operations

<table>
<thead>
<tr>
<th>Transformations (define a new RDD)</th>
<th>Actions (return a result to driver)</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>reduce</td>
</tr>
<tr>
<td>filter</td>
<td>collect</td>
</tr>
<tr>
<td>sample</td>
<td>count</td>
</tr>
<tr>
<td>union</td>
<td>first</td>
</tr>
<tr>
<td>groupByKey</td>
<td>Take</td>
</tr>
<tr>
<td>reduceByKey</td>
<td>countByKey</td>
</tr>
<tr>
<td>sortByKey</td>
<td>saveAsTextFile</td>
</tr>
<tr>
<td>join</td>
<td>saveAsSequenceFile</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**More Information:**
- [https://spark.apache.org/docs/latest/programming-guide.html#transformations](https://spark.apache.org/docs/latest/programming-guide.html#transformations)
- [https://spark.apache.org/docs/latest/programming-guide.html#actions](https://spark.apache.org/docs/latest/programming-guide.html#actions)
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('t')(2))
cachedMsgs = messages.cache()
cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```

Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)

Courtesy: https://spark.apache.org/
Lineage-based Fault Tolerance

- RDDs maintain **lineage** information that can be used to reconstruct lost partitions
- Example

```
cachedMsgs = textFile(...).filter(_.contains("error"))
  .map(_.split('\t')(2))
  .cache()
```
RDD Example: Word Count in Spark!

```scala
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)

counts.saveAsTextFile("hdfs://...")
```

**Productive**

**High-Performance**

**Scalable**

**Fault-Tolerant**
Presentation Outline

- Overview
  - MapReduce and RDD Programming Models
  - Apache Hadoop, Spark, Memcached, gRPC, and TensorFlow
- Challenges in Accelerating Hadoop, Spark, Memcached, gRPC, and TensorFlow
- Acceleration Case Studies and In-Depth Performance Evaluation
- The High-Performance Big Data (HiBD) Project and Associated Releases
- Conclusion and Q&A
Overview of Hadoop, Spark, Memcached, gRPC, and TensorFlow

- Overview of Apache Hadoop and Spark Architecture and its Components
  - MapReduce
  - HDFS
  - Spark
  - HBase

- Overview of Web 2.0 Architecture and Memcached

- Overview of gRPC and TensorFlow Architecture
Overview of Apache Hadoop Architecture

- Open-source implementation of Google MapReduce, GFS, and BigTable for Big Data Analytics
  - Hadoop Common Utilities (RPC, etc.), HDFS, MapReduce, YARN

- `http://hadoop.apache.org`

**Hadoop 1.x**

- MapReduce (Cluster Resource Management & Data Processing)
- Hadoop Distributed File System (HDFS)
- Hadoop Common/Core (RPC, ..)

**Hadoop 2.x**

- MapReduce (Data Processing)
- YARN (Cluster Resource Management & Job Scheduling)
- Hadoop Distributed File System (HDFS)
- Hadoop Common/Core (RPC, ..)
MapReduce on Hadoop 2.x -- YARN Architecture

- **Resource Manager**: coordinates the allocation of compute resources
- **Node Manager**: in charge of resource containers, monitoring resource usage, and reporting to Resource Manager
- **Application Master**: in charge of the life cycle of an application, like a MapReduce job. It negotiates with Resource Manager of cluster resources and keeps track of task progress and status

Courtesy: http://www.cyanny.com/2013/12/05/hadoop-mapreduce-2-yarn/
Data Movement in Hadoop MapReduce

**Disk Operations**

- Map and Reduce Tasks carry out the total job execution
  - Map tasks read from HDFS, operate on it, and write the intermediate data to local disk
  - Reduce tasks get these data by shuffle from TaskTrackers, operate on it and write to HDFS

- Communication in shuffle phase uses HTTP over Java Sockets
Hadoop Distributed File System (HDFS)

- Primary storage of Hadoop; highly reliable and fault-tolerant
- Adopted by many reputed organizations
  - eg: Facebook, Yahoo!
- NameNode: stores the file system namespace
- DataNode: stores data blocks
- Developed in Java for platform-independence and portability
- Uses sockets for communication!
New Features of Apache Hadoop 3.x Architecture

- **HDFS**
  - Erasure Coding
  - Support for more than 2 NameNodes
  - Intra-datanode balancer
- **YARN**
  - Built-in support for Long Running Services
  - Better resource isolation (isolation supports for disk and network) and Docker
  - Scheduling enhancement (enhance container scheduling throughput by 6x)
  - Re-architecture for YARN Timeline Service - ATS v2

- **MapReduce**
  - Task-level native optimization (up to 30% faster for shuffle-intensive jobs)
Spark Architecture Overview

- An in-memory data-processing framework
  - Iterative machine learning jobs
  - Interactive data analytics
  - Scala based Implementation
  - Standalone, YARN, Mesos

- A unified engine to support Batch, Streaming, SQL, Graph, ML/DL workloads

- Scalable and communication intensive
  - Wide dependencies between Resilient Distributed Datasets (RDDs)
  - MapReduce-like shuffle operations to repartition RDDs
  - Sockets based communication

http://spark.apache.org
HBase Architecture Overview

- Apache Hadoop Database (http://hbase.apache.org/)
  - Semi-structured database, which is highly scalable
  - Integral part of many datacenter applications
    - eg: Facebook Social Inbox
  - Developed in Java for platform-independence and portability
  - Uses sockets for communication!
Architecture Overview of Memcached

- Three-layer architecture of Web 2.0
  - Web Servers, Memcached Servers, Database Servers
- Memcached is a core component of Web 2.0 architecture
- Distributed Caching Layer
  - Allows to aggregate spare memory from multiple nodes
  - General purpose
- Typically used to cache database queries, results of API calls
- Scalable model, but typical usage very network intensive
Architecture Overview of gRPC

Key Features:
- Simple service definition
- Works across languages and platforms
  - C++, Java, Python, Android Java etc
  - Linux, Mac, Windows.
- Start quickly and scale
- Bi-directional streaming and integrated authentication
- Used by Google (several of Google’s cloud products and Google externally facing APIs, TensorFlow), NetFlix, Docker, Cisco, Juniper Networks etc.
- Uses sockets for communication!

Source: http://www.grpc.io/

Large-scale distributed systems composed of micro services
Key Features:
• Widely used for Deep Learning
• Open source software library for numerical computation using data flow graphs
• Graph edges represent the multidimensional data arrays
• Nodes in the graph represent mathematical operations
• Flexible architecture allows to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API
• Used by Google, Airbnb, DropBox, Snapchat, Twitter
• Communication and Computation intensive

Source: https://www.tensorflow.org/
Worker services communicate among each other using RDMA-gRPC
Presentation Outline

• Overview
  – MapReduce and RDD Programming Models
  – Apache Hadoop, Spark, Memcached, gRPC, and TensorFlow

• Challenges in Accelerating Hadoop, Spark, Memcached, gRPC, and TensorFlow

• Acceleration Case Studies and In-Depth Performance Evaluation

• The High-Performance Big Data (HiBD) Project and Associated Releases

• Conclusion and Q&A
Increasing Usage of HPC, Big Data and Deep Learning

Convergence of HPC, Big Data, and Deep Learning!!!
How Can HPC Clusters with High-Performance Interconnect and Storage Architectures Benefit Big Data and Deep Learning Applications?

- Can the bottlenecks be alleviated with new designs by taking advantage of HPC technologies?
- What are the major bottlenecks in current Big Data processing and Deep Learning middleware (e.g. Hadoop, Spark)?
- Can RDMA-enabled high-performance interconnects benefit Big Data processing and Deep Learning?
- How much performance benefits can be achieved through enhanced designs?
- Can HPC Clusters with high-performance storage systems (e.g. SSD, parallel file systems) benefit Big Data and Deep Learning applications?
- How to design benchmarks for evaluating the performance of Big Data and Deep Learning middleware on HPC clusters?

Bring HPC, Big Data processing, and Deep Learning into a “convergent trajectory”!
Can We Run Big Data and Deep Learning Jobs on Existing HPC Infrastructure?
Can We Run Big Data and Deep Learning Jobs on Existing HPC Infrastructure?
Can We Run Big Data and Deep Learning Jobs on Existing HPC Infrastructure?
Can We Run Big Data and Deep Learning Jobs on Existing HPC Infrastructure?
Designing Communication and I/O Libraries for Big Data Systems: Challenges

Big Data Middleware
(HDFS, MapReduce, HBase, Spark, gRPC/TensorFlow, and Memcached)

Programming Models
(Sockets)

Networking Technologies
(InfiniBand, 1/10/40/100 GigE and Intelligent NICs)

Commodity Computing System Architectures
(Multi- and Many-core architectures and accelerators)

Storage Technologies
(HDD, SSD, NVM, and NVMe-SSD)

Applications

Benchmarks

Communication and I/O Library

Point-to-Point Communication

Threaded Models and Synchronization

Virtualization (SR-IOV)

I/O and File Systems

QoS & Fault Tolerance

Performance Tuning

Upper level Changes?
Presentation Outline

• Overview
  – MapReduce and RDD Programming Models
  – Apache Hadoop, Spark, Memcached, gRPC, and TensorFlow

• Challenges in Accelerating Hadoop, Spark, Memcached, gRPC, and TensorFlow

• Acceleration Case Studies and In-Depth Performance Evaluation

• The High-Performance Big Data (HiBD) Project and Associated Releases

• Conclusion and Q&A
Designing Communication and I/O Libraries for Big Data Systems: Challenges

Applications

Big Data Middleware
(HDFS, MapReduce, HBase, Spark, gRPC/TensorFlow, and Memcached)

Benchmarks

Programming Models
(Sockets)

RDMA Protocols

Communication and I/O Library

Point-to-Point Communication

Threaded Models and Synchronization

Virtualization (SR-IOV)

I/O and File Systems

QoS & Fault Tolerance

Performance Tuning

Networking Technologies
(InfiniBand, 1/10/40/100 GigE and Intelligent NICs)

Commodity Computing System Architectures
(Multi- and Many-core architectures and accelerators)

Storage Technologies
(HDD, SSD, NVM, and NVMe-SSD)
Basic Acceleration Case Studies and In-Depth Performance Evaluation

- High-Performance Designs with RDMA, In-memory, SSD, Parallel Filesystems
  - HDFS
  - MapReduce
  - Spark
  - Hadoop RPC and HBase
  - Memcached
  - gRPC and TensorFlow
  - Kafka
Design Overview of HDFS with RDMA

- **Design Features**
  - RDMA-based HDFS write
  - RDMA-based HDFS replication
  - Parallel replication support
  - On-demand connection setup
  - InfiniBand/RoCE support

- **Applications**
- **HDFS**
- **Write**
  - Java Socket Interface
  - Java Native Interface (JNI)
  - OSU Design
  - Verbs
  - RDMA Capable Networks (IB, iWARP, RoCE ..)

- **Others**
- **1/10/40/100 GigE, IPoIB Network**

---


Enhanced HDFS with In-Memory and Heterogeneous Storage

Design Features
- Three modes
  - Default (HHH)
  - In-Memory (HHH-M)
  - Lustre-Integrated (HHH-L)
- Policies to efficiently utilize the heterogeneous storage devices
  - RAM, SSD, HDD, Lustre
- Eviction/Promotion based on data usage pattern
- Hybrid Replication
- Lustre-Integrated mode:
  - Lustre-based fault-tolerance

For 200GB TeraGen on 32 nodes

- Spark-TeraGen: HHH has 2.4x improvement over Tachyon; 2.3x over HDFS-IPoIB (QDR)
- Spark-TeraSort: HHH has 25.2% improvement over Tachyon; 17% over HDFS-IPoIB (QDR)

Basic Acceleration Case Studies and In-Depth Performance Evaluation

- High-Performance Designs with RDMA, In-memory, SSD, Parallel Filesystems
  - HDFS
  - MapReduce
  - Spark
  - Hadoop RPC and HBase
  - Memcached
  - gRPC and TensorFlow
  - Kafka
Design Overview of MapReduce with RDMA

- Design Features
  - RDMA-based shuffle
  - Prefetching and caching map output
  - Efficient Shuffle Algorithms
  - In-memory merge
  - On-demand Shuffle Adjustment
  - Advanced overlapping
    - map, shuffle, and merge
    - shuffle, merge, and reduce
  - On-demand connection setup
  - InfiniBand/RoCE support

M. W. Rahman, N. S. Islam, X. Lu, J. Jose, H. Subramon, H. Wang, and D. K. Panda, High-Performance RDMA-based Design of Hadoop MapReduce over InfiniBand, HPDIC Workshop, held in conjunction with IPDPS, May 2013

• 50% improvement in Self Join over IPoIB (QDR) for 80 GB data size
• 49% improvement in Sequence Count over IPoIB (QDR) for 30 GB data size
Optimize Hadoop YARN MapReduce over Parallel File Systems

- HPC Cluster Deployment
  - Hybrid topological solution of Beowulf architecture with separate I/O nodes
  - Lean compute nodes with light OS; more memory space; small local storage
  - Sub-cluster of dedicated I/O nodes with parallel file systems, such as Lustre

- MapReduce over Lustre
  - Local disk is used as the intermediate data directory
  - Lustre is used as the intermediate data directory
Design Overview of Shuffle Strategies for MapReduce over Lustre

- **Design Features**
  - Two shuffle approaches
    - Lustre read based shuffle
    - RDMA based shuffle
  - Hybrid shuffle algorithm to take benefit from both shuffle approaches
  - Dynamically adapts to the better shuffle approach for each shuffle request based on profiling values for each Lustre read operation
  - In-memory merge and overlapping of different phases are kept similar to RDMA-enhanced MapReduce design

Case Study - Performance Improvement of MapReduce over Lustre on SDSC-Gordon

- Lustre is used as the intermediate data directory

- For 80GB Sort in 8 nodes
  - 34% improvement over IPoIB (QDR)

- For 120GB TeraSort in 16 nodes
  - 25% improvement over IPoIB (QDR)
Basic Acceleration Case Studies and In-Depth Performance Evaluation

- High-Performance Designs with RDMA, In-memory, SSD, Parallel Filesystems
  - HDFS
  - MapReduce
  - Spark
  - Hadoop RPC and HBase
  - Memcached
  - gRPC and TensorFlow
  - Kafka
**Design Overview of Spark with RDMA**

- **Spark Core**
  - Design Features
    - RDMA based shuffle plugin
    - SEDA-based architecture
    - Dynamic connection management and sharing
    - Non-blocking data transfer
    - Off-JVM-heap buffer management
    - InfiniBand/RoCE support

- **Apache Spark Benchmarks/Applications/Libraries/Frameworks**

- **Shuffle Manager (Sort, Hash, Tungsten-Sort)**

- **Block Transfer Service (Netty, NIO, RDMA-Plugin)**
  - Netty Server
  - NIO Server
  - RDMA Server
  - Netty Client
  - NIO Client
  - RDMA Client

- **Java Socket Interface**

- **Java Native Interface (JNI)**

- **Native RDMA-based Comm. Engine**

- **RDMA Capable Networks** (IB, iWARP, RoCE ..)

- **1/10/40/100 GigE, IPoIB Network**

- **Design Features**
  - Enables high performance RDMA communication, while supporting traditional socket interface
  - JNI Layer bridges Scala based Spark with communication library written in native code

---


Performance Evaluation on SDSC Comet – HiBench PageRank

32 Worker Nodes, 768 cores, PageRank Total Time

- InfiniBand FDR, SSD, 32/64 Worker Nodes, 768/1536 Cores, (768/1536M 768/1536R)
- RDMA-based design for Spark 1.5.1
- RDMA vs. IPoIB with 768/1536 concurrent tasks, single SSD per node.
  - 32 nodes/768 cores: Total time reduced by 37% over IPoIB (56Gbps)
  - 64 nodes/1536 cores: Total time reduced by 43% over IPoIB (56Gbps)
Basic Acceleration Case Studies and In-Depth Performance Evaluation

• High-Performance Designs with RDMA, In-memory, SSD, Parallel Filesystems
  – HDFS
  – MapReduce
  – Spark
  – Hadoop RPC and HBase
  – Memcached
  – gRPC and TensorFlow
  – Kafka
Performance Benefits for Hadoop RPC and HBase

- Hadoop RPC Throughput on Chameleon-Cloud-FDR
  - up to 2.6x performance speedup over IPoIB for throughput
- HBase YCSB Workload A (read: write=50:50) on SDSC-Comet-FDR
  - Native designs always perform better than the IPoIB-UD transport
  - up to 2.4x performance speedup over IPoIB for throughput


Accelerating Hybrid Memcached with RDMA, Non-blocking Extensions and SSDs

- RDMA-Accelerated Communication for Memcached Get/Set
- Hybrid ‘RAM+SSD’ slab management for higher data retention
- Non-blocking API extensions
  - `memcached_(iset/iget/bset/bget/test/wait)`
  - Achieve near in-memory speeds while hiding bottlenecks of network and SSD I/O
  - Ability to exploit communication/computation overlap
  - Optional buffer re-use guarantees
- Adaptive slab manager with different I/O schemes for higher throughput.

Data does not fit in memory: Non-blocking Memcached Set/Get API Extensions can achieve
- >16x latency improvement vs. blocking API over RDMA-Hybrid/RDMA-Mem w/ penalty
- >2.5x throughput improvement vs. blocking API over default/optimized RDMA-Hybrid

Data fits in memory: Non-blocking Extensions perform similar to RDMA-Mem/RDMA-Hybrid and >3.6x improvement over IPoIB-Mem
Performance Benefits for RDMA-gRPC with Micro-Benchmark

• **gRPC-RDMA Latency on SDSC-Comet-FDR**
  - **Up to 2.7x** performance speedup over IPoIB for Latency for small messages
  - **Up to 2.8x** performance speedup over IPoIB for Latency for medium messages
  - **Up to 2.5x** performance speedup over IPoIB for Latency for large messages


Network Based Computing Laboratory
5194 Class at OSU
Performance Benefit for TensorFlow - Resnet50

- TensorFlow Resnet50 performance evaluation on an IB EDR cluster
  - Up to 35% performance speedup over IPoIB for 4 nodes.
  - Up to 41% performance speedup over IPoIB for 8 nodes.
Performance Benefit for TensorFlow - Inception3

TensorFlow Inception3 performance evaluation on an IB EDR cluster
- Up to 27% performance speedup over IPoIB for 4 nodes
- Up to 36% performance speedup over IPoIB for 8 nodes.
RDMA-Kafka: High-Performance Message Broker for Streaming Workloads

- Experiments run on OSU-RI2 cluster
- 2.4GHz 28 cores, InfiniBand EDR, 512 GB RAM, 400GB SSD
  - Up to 98% improvement in latency compared to IPoIB
  - Up to 7x increase in throughput over IPoIB

Presentation Outline

• Overview
  – MapReduce and RDD Programming Models
  – Apache Hadoop, Spark, Memcached, gRPC, and TensorFlow
• Challenges in Accelerating Hadoop, Spark, Memcached, gRPC, and TensorFlow
• Acceleration Case Studies and In-Depth Performance Evaluation
• The High-Performance Big Data (HiBD) Project and Associated Releases
• Conclusion and Q&A
The High-Performance Big Data (HiBD) Project

- RDMA for Apache Spark
- RDMA for Apache Hadoop 2.x (RDMA-Hadoop-2.x)
  - Plugins for Apache, Hortonworks (HDP) and Cloudera (CDH) Hadoop distributions
- RDMA for Apache HBase
- RDMA for Memcached (RDMA-Memcached)
- RDMA for Apache Hadoop 1.x (RDMA-Hadoop)
- OSU HiBD-Benchmarks (OHB)
  - HDFS, Memcached, HBase, and Spark Micro-benchmarks
- http://hibd.cse.ohio-state.edu

Available for InfiniBand and RoCE
Also run on Ethernet
Available for x86 and OpenPOWER
Support for Singularity and Docker

- Users Base: 290 organizations from 34 countries
- More than 27,850 downloads from the project site
HiBD Release Timeline and Downloads

Number of Downloads

Timeline

RDMA-Hadoop 1.x 0.9.0
RDMA-Hadoop 1.x 0.9.8
RDMA-Hadoop 1.x 0.9.9
RDMA-Memcached 0.9.1 & OHB-0.7.1
RDMA-Hadoop 2.x 0.9.1
RDMA-Hadoop 2.x 0.9.6
RDMA-Hadoop 2.x 0.9.7
RDMA-Memcached 0.9.4
RDMA-Spark 0.9.4
RDMA-Hadoop 2.x 1.0.0
RDMA-Memcached 0.9.6 & OHB-0.9.3
RDMA-Spark 0.9.5
RDMA-Hadoop 2.x 1.3.0
RDMA-Spark 0.9.4

Network Based Computing Laboratory
5194 Class at OSU
RDMA for Apache Hadoop 2.x Distribution

• High-Performance Design of Hadoop over RDMA-enabled Interconnects
  – High performance RDMA-enhanced design with native InfiniBand and RoCE support at the verbs-level for HDFS, MapReduce, and RPC components
  – Enhanced HDFS with in-memory and heterogeneous storage
  – High performance design of MapReduce over Lustre
  – Memcached-based burst buffer for MapReduce over Lustre-integrated HDFS (HHH-L-BB mode)
  – Plugin-based architecture supporting RDMA-based designs for Apache Hadoop, CDH and HDP
  – Support for OpenPOWER, Singularity, and Docker

• Current release: 1.3.5
  – Based on Apache Hadoop 2.8.0
  – Compliant with Apache Hadoop 2.8.0, HDP 2.5.0.3 and CDH 5.8.2 APIs and applications
  – Tested with
    • Mellanox InfiniBand adapters (DDR, QDR, FDR, and EDR)
    • RoCE support with Mellanox adapters
    • Various multi-core platforms (x86, POWER)
    • Different file systems with disks and SSDs and Lustre

http://hibd.cse.ohio-state.edu
• **HHH**: Heterogeneous storage devices with hybrid replication schemes are supported in this mode of operation to have better fault-tolerance as well as performance. This mode is enabled by default in the package.

• **HHH-M**: A high-performance in-memory based setup has been introduced in this package that can be utilized to perform all I/O operations in-memory and obtain as much performance benefit as possible.

• **HHH-L**: With parallel file systems integrated, HHH-L mode can take advantage of the Lustre available in the cluster.

• **HHH-L-BB**: This mode deploys a Memcached-based burst buffer system to reduce the bandwidth bottleneck of shared file system access. The burst buffer design is hosted by Memcached servers, each of which has a local SSD.

• **MapReduce over Lustre, with/without local disks**: Besides, HDFS based solutions, this package also provides support to run MapReduce jobs on top of Lustre alone. Here, two different modes are introduced: with local disks and without local disks.

• **Running with Slurm and PBS**: Supports deploying RDMA for Apache Hadoop 2.x with Slurm and PBS in different running modes (HHH, HHH-M, HHH-L, and MapReduce over Lustre).
RDMA for Apache Spark Distribution

- High-Performance Design of Spark over RDMA-enabled Interconnects
  - High performance RDMA-enhanced design with native InfiniBand and RoCE support at the verbs-level for Spark
  - RDMA-based data shuffle and SEDA-based shuffle architecture
  - Non-blocking and chunk-based data transfer
  - Off-JVM-heap buffer management
  - Support for OpenPOWER
  - Easily configurable for different protocols (native InfiniBand, RoCE, and IPoIB)

- Current release: 0.9.5
  - Based on Apache Spark 2.1.0
  - Tested with
    - Mellanox InfiniBand adapters (DDR, QDR, FDR, and EDR)
    - RoCE support with Mellanox adapters
    - Various multi-core platforms (x86, POWER)
    - RAM disks, SSDs, and HDD
  - [http://hibd.cse.ohio-state.edu](http://hibd.cse.ohio-state.edu)
HiBD Packages on SDSC Comet and Chameleon Cloud

- RDMA for Apache Hadoop 2.x and RDMA for Apache Spark are installed and available on SDSC Comet.
  - Examples for various modes of usage are available in:
    - RDMA for Apache Hadoop 2.x: /share/apps/examples/HADOOP
    - RDMA for Apache Spark: /share/apps/examples/SPARK/
  - Please email help@xsede.org (reference Comet as the machine, and SDSC as the site) if you have any further questions about usage and configuration.

- RDMA for Apache Hadoop is also available on Chameleon Cloud as an appliance
  - https://www.chameleoncloud.org/appliances/17/

M. Tatineni, X. Lu, D. J. Choi, A. Majumdar, and D. K. Panda, Experiences and Benefits of Running RDMA Hadoop and Spark on SDSC Comet, XSEDE’16, July 2016
Using HiBD Packages for Big Data Processing on Existing HPC Infrastructure

- Hadoop Job with HiBD
  - HHH (-M, -L, -BB-L)
  - RDMA-MapReduce (over Lustre)
  - HBase, Hive, Pig, etc.
Using HiBD Packages for Big Data Processing on Existing HPC Infrastructure

Spark Job with HiBD
- RDMA-Spark
- Integration with HHH
- Spark SQL, MLlib, etc.
RDMA for Apache HBase Distribution

- High-Performance Design of HBase over RDMA-enabled Interconnects
  - High performance RDMA-enhanced design with native InfiniBand and RoCE support at the verbs-level for HBase
  - Compliant with Apache HBase 1.1.2 APIs and applications
  - On-demand connection setup
  - Easily configurable for different protocols (native InfiniBand, RoCE, and IPoIB)

- Current release: 0.9.1
  - Based on Apache HBase 1.1.2
  - Tested with
    - Mellanox InfiniBand adapters (DDR, QDR, FDR, and EDR)
    - RoCE support with Mellanox adapters
    - Various multi-core platforms
  - [http://hibd.cse.ohio-state.edu](http://hibd.cse.ohio-state.edu)
RDMA for Memcached Distribution

- High-Performance Design of Memcached over RDMA-enabled Interconnects
  - High performance RDMA-enhanced design with native InfiniBand and RoCE support at the verbs-level for Memcached and libMemcached components
  - High performance design of SSD-Assisted Hybrid Memory
  - Non-Blocking Libmemcached Set/Get API extensions
  - Support for burst-buffer mode in Lustre-integrated design of HDFS in RDMA for Apache Hadoop-2.x
  - Easily configurable for native InfiniBand, RoCE and the traditional sockets-based support (Ethernet and InfiniBand with IPoIB)

- Current release: 0.9.6
  - Based on Memcached 1.5.3 and libMemcached 1.0.18
  - Compliant with libMemcached APIs and applications
  - Tested with
    - Mellanox InfiniBand adapters (DDR, QDR, FDR, and EDR)
    - RoCE support with Mellanox adapters
    - Various multi-core platforms
    - SSD
  - [http://hibd.cse.ohio-state.edu](http://hibd.cse.ohio-state.edu)
OSU HiBD Micro-Benchmark (OHB) Suite – HDFS, Memcached, HBase, and Spark

- Micro-benchmarks for Hadoop Distributed File System (HDFS)
  - Sequential Write Latency (SWL) Benchmark, Sequential Read Latency (SRL) Benchmark, Random Read Latency (RRL) Benchmark, Sequential Write Throughput (SWT) Benchmark, Sequential Read Throughput (SRT) Benchmark
  - Support benchmarking of
    - Apache Hadoop 1.x and 2.x HDFS, Hortonworks Data Platform (HDP) HDFS, Cloudera Distribution of Hadoop (CDH) HDFS

- Micro-benchmarks for Memcached
  - Get Benchmark, Set Benchmark, and Mixed Get/Set Benchmark, Non-Blocking API Latency Benchmark, Hybrid Memory Latency Benchmark
  - Yahoo! Cloud Serving Benchmark (YCSB) Extension for RDMA-Memcached

- Micro-benchmarks for HBase
  - Get Latency Benchmark, Put Latency Benchmark

- Micro-benchmarks for Spark
  - GroupBy, SortBy

- Current release: 0.9.3

- http://hibd.cse.ohio-state.edu
Performance Numbers of RDMA for Apache Hadoop 2.x – RandomWriter & TeraGen in OSU-RI2 (EDR)

Cluster with 8 Nodes with a total of 64 maps

- RandomWriter
  - $3x$ improvement over IPoIB for 80-160 GB file size

- TeraGen
  - $4x$ improvement over IPoIB for 80-240 GB file size
Performance Numbers of RDMA for Apache Hadoop 2.x – Sort & TeraSort in OSU-RI2 (EDR)

- **Sort**
  - 61% improvement over IPoIB for 80-160 GB data

- **TeraSort**
  - 18% improvement over IPoIB for 80-240 GB data
Performance Evaluation on SDSC-Comet-IB-FDR – with RDMA-HDFS

**8 Worker Nodes, HiBench Sort Total Time**

- InfiniBand FDR, SSD, YARN-Client Mode, 192 Cores
- Benefit of RDMA-Spark and the capability of combining with other advanced technologies, such as RDMA-HDFS
- A combined version of ‘RDMA-Spark+RDMA-HDFS’ can achieve the best performance
- RDMA vs. IPoIB
  - HiBench Sort: Total time reduced by up to 38% over IPoIB through RDMA-Spark, up to 82% through RDMA-Spark+RDMA-HDFS
  - HiBench TeraSort: Total time reduced by up to 17% over IPoIB through RDMA-Spark, up to 29% through RDMA-Spark+RDMA-HDFS
Presentation Outline

• Overview
  – MapReduce and RDD Programming Models
  – Apache Hadoop, Spark, Memcached, gRPC, and TensorFlow
• Challenges in Accelerating Hadoop, Spark, Memcached, gRPC, and TensorFlow
• Acceleration Case Studies and In-Depth Performance Evaluation
• The High-Performance Big Data (HiBD) Project and Associated Releases
• Conclusion and Q&A
Concluding Remarks

- Presented an overview of MapReduce and RDD programming models
- Presented an overview of Big Data, Hadoop, Spark, Memcached, gRPC, and TensorFlow
- Provided an overview of Networking Technologies
- Discussed challenges in accelerating Hadoop, Spark, Memcached, gRPC, and TensorFlow
- Presented basic and advanced designs to take advantage of InfiniBand/RDMA for HDFS, MapReduce, HBase, Spark, Memcached, gRPC, and TensorFlow on HPC clusters and clouds
- Results are promising for Big Data processing and the associated Deep Learning tools
- Many other open issues need to be solved
- Will enable Big Data and Deep Learning communities to take advantage of modern HPC technologies to carry out their analytics in a fast and scalable manner
Thank You!

luxi@cse.ohio-state.edu

http://www.cse.ohio-state.edu/~panda

http://www.cse.ohio-state.edu/~luxi

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

The High-Performance Big Data Project
http://hibd.cse.ohio-state.edu/