Introduction to Hadoop on Gordon and Comet

Mahidhar Tatineni

UCLA, Nov 12, 2015

Overview

- Hadoop framework extensively used for scalable distributed processing of large datasets. Hadoop is built to process data in orders of several hundred gigabytes to several terabytes (and even petabytes at the extreme end).

- Data sizes are much bigger than the capacities (both disk and memory) of individual nodes. Under Hadoop Distributed Filesystem (HDFS), data is split into chunks which are managed by different nodes.

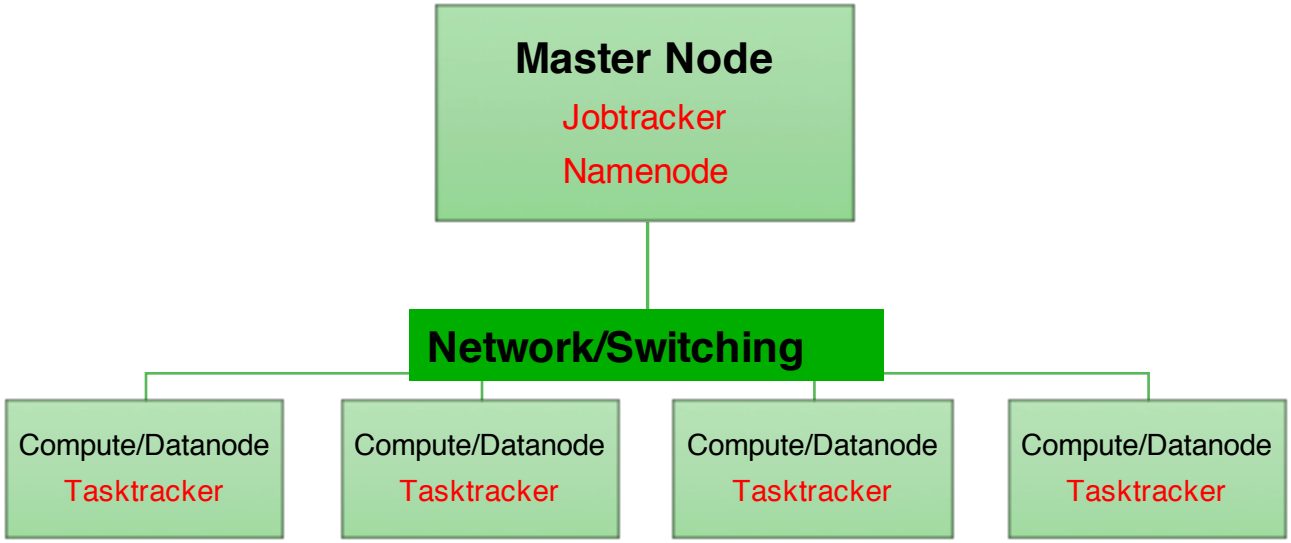
- Data chunks are replicated across several machines to provide redundancy in case of an individual node failure.

- Processing must conform to "Map-Reduce" programming model. Processes are scheduled close to location of data chunks being accessed.

Hadoop: Application Areas

- **Hadoop is widely used in data intensive analysis. Some application areas include:**
  - Log aggregation and processing
  - Video and Image analysis
  - Data mining, Machine learning
  - Indexing
  - Recommendation systems
- **Data intensive scientific applications can make use of the Hadoop MapReduce framework. Application areas include:**
  - Bioinformatics and computational biology
  - Astronomical image processing
  - Natural Language Processing
  - Geospatial data processing
- **Some Example Projects**
  - Genetic algorithms, particle swarm optimization, ant colony optimization
  - Big data for business analytics (class)
  - Hadoop for remote sensing analysis
- **Extensive list online at:**
  - http://wiki.apache.org/hadoop/PoweredBy

Hadoop Architecture



Map/Reduce Framework

- Software to enable distributed computation.
- Jobtracker schedules and manages map/reduce tasks.
- Tasktracker does the execution of tasks on the nodes.

HDFS – Distributed Filesystem

- Metadata handled by the Namenode.
- Files are split up and stored on datanodes (typically local disk).
- Scalable and fault tolerance.
- Replication is done asynchronously.

Simple Example – From Apache Site*

- **Simple wordcount example.**
- **Code details:**

Functions defined

- Wordcount map class : reads file and isolates each word
- Reduce class : counts words and sums up

Call sequence

- Mapper class
- Combiner class (Reduce locally)
- Reduce class
- Output

*http://hadoop.apache.org/docs/r0.18.3/mapred_tutorial.html

Simple Example – From Apache Site*

- Simple wordcount example. Two input files.
- File 1 contains: Hello World Bye World
- File 2 contains: Hello Hadoop Goodbye Hadoop
- Assuming we use two map tasks (one for each file).

- Step 1: Map read/parse tasks are complete. Result:

TASK 1	TASK 2
<Hello, 1>	< Hello, 1>
<World, 1>	< Hadoop, 1>
<Bye, 1>	< Goodbye, 1>
<World, 1>	<Hadoop, 1>

*http://hadoop.apache.org/docs/r0.18.3/mapred_tutorial.html

Simple Example (Contd)

- Step 2 : Combine on each node, sorted:

<Bye, 1>
<Hello, 1>
<World, 2>

< Goodbye, 1>
< Hadoop, 2>
<Hello, 1>

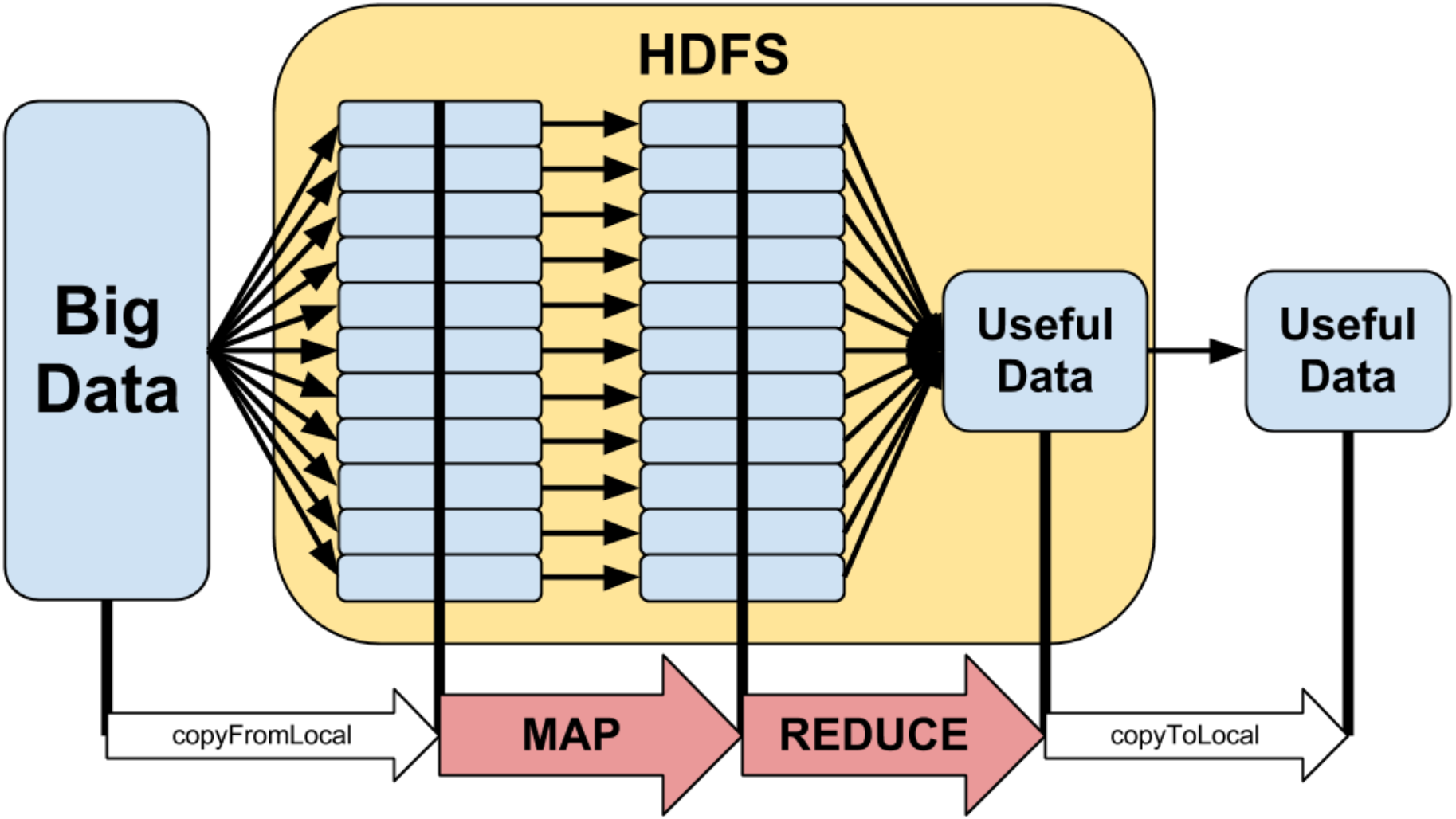
- Step 3 : Global reduce:

<Bye, 1>
<Goodbye, 1>
<Hadoop, 2>
<Hello, 2>
<World, 2>

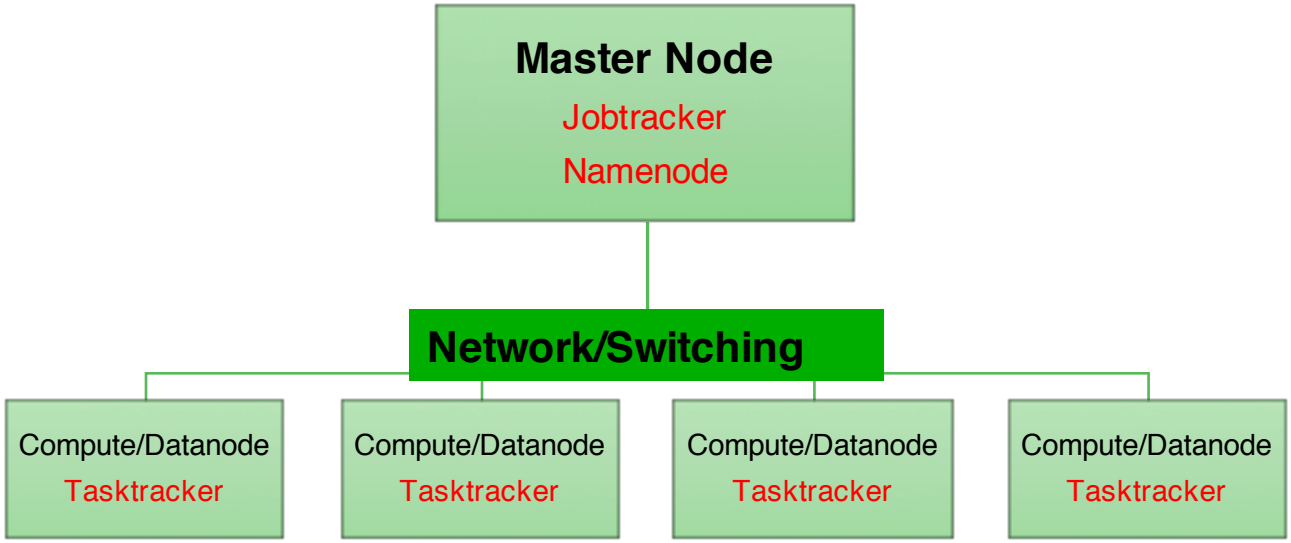
Map/Reduce Execution Process

- **Components**
  - Input / Map () / Shuffle / Sort / Reduce () / Output
- **Jobtracker determines number of splits (configurable).**
- **Jobtracker selects compute nodes for tasks based on network proximity to data sources.**
- **Tasktracker on each compute node manages the tasks assigned and reports back to jobtracker when task is complete.**
- **As map tasks complete jobtracker notifies selected task trackers for reduce phase.**
- **Job is completed once reduce phase is complete.**

Hadoop Workflow



Hadoop Architecture



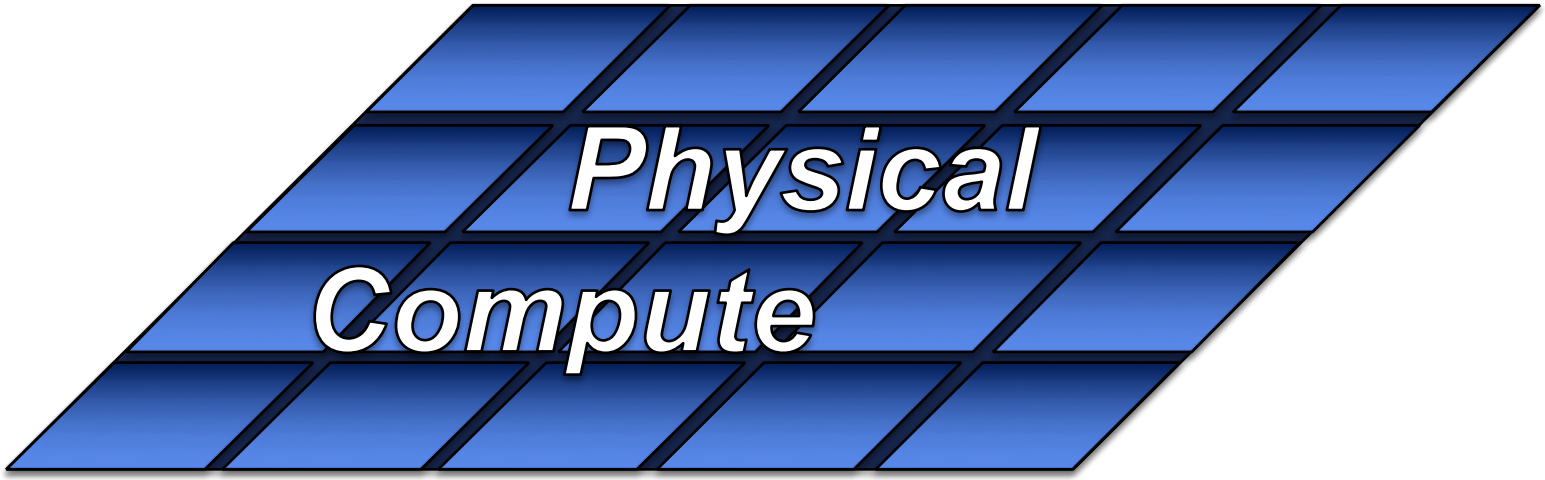
Map/Reduce Framework

- Software to enable distributed computation.
- Jobtracker schedules and manages map/reduce tasks.
- Tasktracker does the execution of tasks on the nodes.

HDFS – Distributed Filesystem

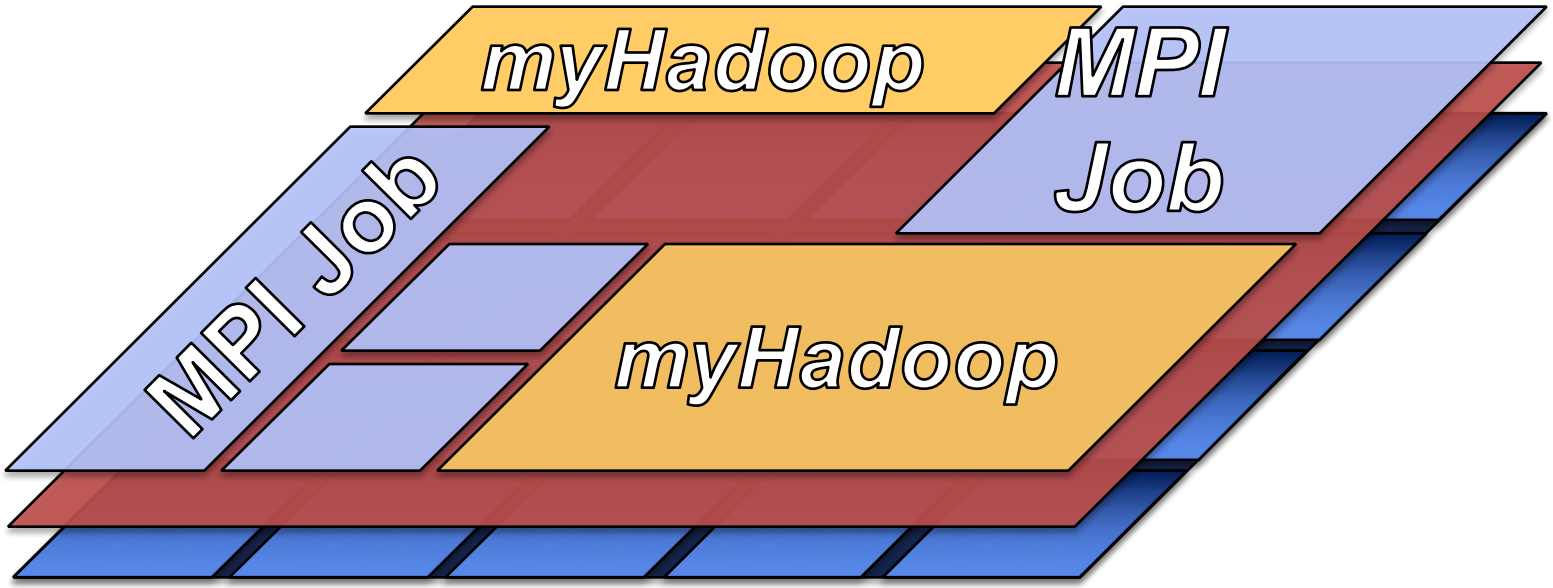
- Metadata handled by the Namenode.
- Files are split up and stored on datanodes (typically local disk).
- Scalable and fault tolerance.
- Replication is done asynchronously.

Add Data Analysis to Existing Compute Infrastructure

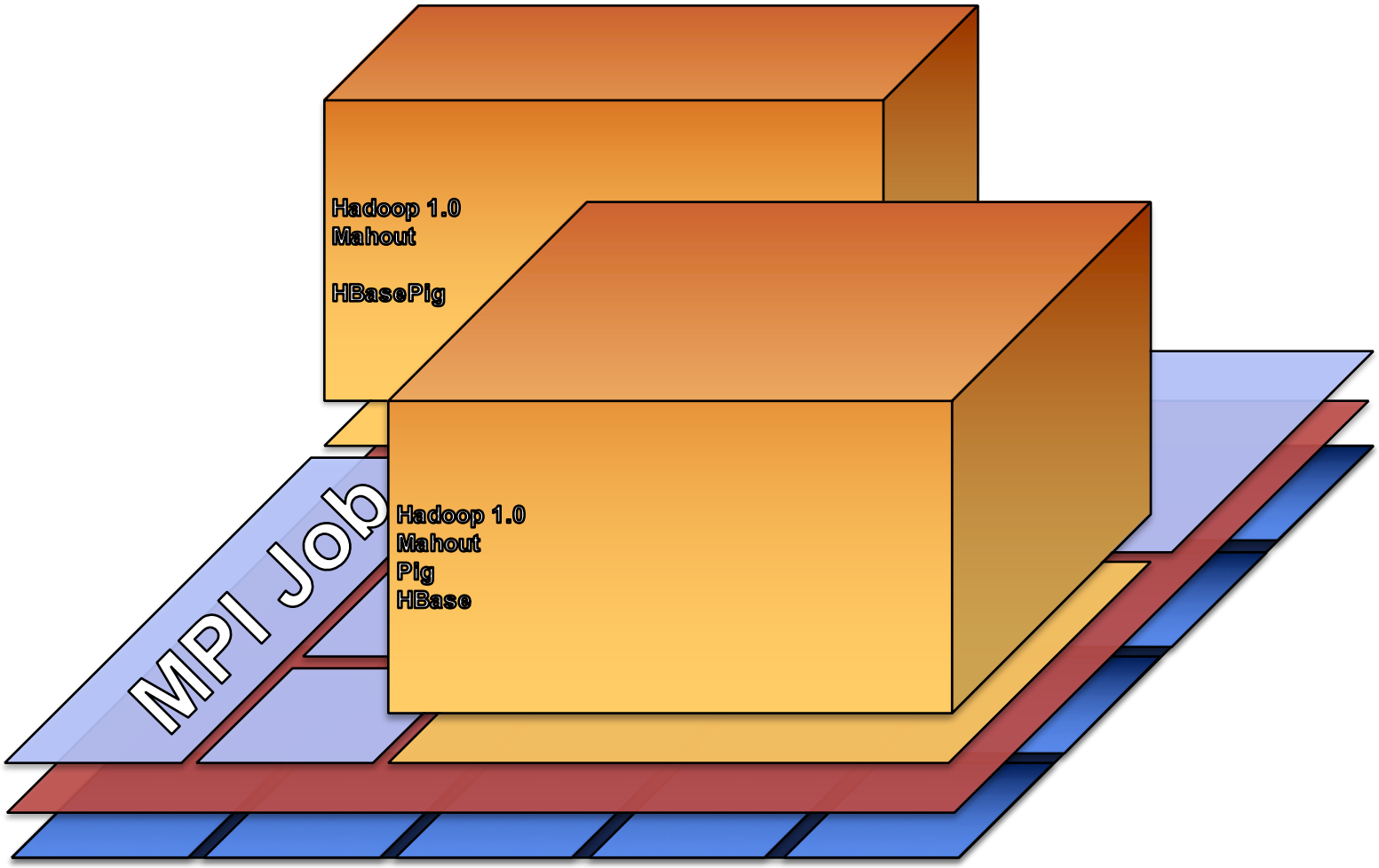


Add Data Analysis to Existing Compute Infrastructure

Add Data Analysis to Existing Compute Infrastructure



Add Data Analysis to Existing Compute Infrastructure



myHadoop – 3-step Cluster

1. Set a few environment variables

```
# sets HADOOP_HOME, JAVA_HOME, and PATH
$ module load hadoop
$ export HADOOP_CONF_DIR=$HOME/mycluster.conf
```

2. Run myhadoop-configure.sh to set up Hadoop

```
$ myhadoop-configure.sh -s /scratch/$USER/$PBS_JOBID
```

3. Start cluster with Hadoop's `start-all.sh`

```
$ start-all.sh
```

Advanced Features - Useability

- **System-wide default configurations**
  - `myhadoop-0.30/conf/myhadoop.conf`
  - `MH_SCRATCH_DIR` – specify location of node-local storage for all users
  - `MH_IPOIB_TRANSFORM` – specify regex to transform node hostnames into IP over InfiniBand hostnames
- **Users can remain totally ignorant of scratch disks and InfiniBand**
- **Literally define `HADOOP_CONF_DIR` and run `myhadoop-configure.sh` with no parameters – myHadoop figures out everything else**

Advanced Features - Useability

- **Parallel filesystem support**
  - HDFS on Lustre via myHadoop persistent mode (`-p`)
  - Direct Lustre support (IDH)
  - No performance loss at smaller scales for HDFS on Lustre
- **Resource managers supported in unified framework:**
  - Torque – Tested on SDSC Gordon
  - SLURM – Tested on SDSC Comet, TACC Stampede
  - Grid Engine
  - Can support LSF, PBSpro, Condor easily (need testbeds)

hadoop-env.sh

- **Establishes environment variables for all Hadoop components**
- **Essentials:**
  - `HADOOP_LOG_DIR` – location of Hadoop logs
  - `HADOOP_PID_DIR` – location of Hadoop PID files
  - `JAVA_HOME` – location of Java that Hadoop should use
- **Other common additions**
  - `LD_LIBRARY_PATH` - for mappers/reducers
  - `HADOOP_CLASSPATH` - for mappers/reducers
  - `_JAVA_OPTIONS` - to hack global Java options

Interactive Hadoop Cluster on Gordon

```
### Request two nodes for four hours on Gordon
$ qsub -I -l nodes=2:ppn=16:native:flash,walltime=4:00:00 -q normal
### Configure $HADOOP_CONF_DIR on Gordon
$ myhadoop-configure.sh
### Hadoop control script to start all nodes
$ start-all.sh
### Verify HDFS is online
$ hadoop dfsadmin -report
### Copy file to HDFS
$ hadoop dfs -put somelocalfile hdfsdir/
### View file information on HDFS
$ hadoop fsck -block hdfsdir/somelocalfile
### Run a map/reduce job
$ hadoop jar somejarfile.jar -option1 -option2
### View job info after it completes
$ hadoop job -history hdfsdir/outputdir
### Shut down all Hadoop nodes
$ stop-all.sh
### Copy logfiles back from nodes
$ myhadoop-cleanup.sh
```

Interactive Hadoop Cluster on Comet

```
### Request two nodes for 20 minutes on Comet
$ /share/apps/compute/interactive/qsubi.bash -p compute --nodes=2 --ntasks-per-node=24 -t 00:20:00 --export=ALL
### Configure $HADOOP_CONF_DIR on Gordon
$ myhadoop-configure.sh
### Hadoop control script to start all nodes
$ start-all.sh
### Verify HDFS is online
$ hadoop dfsadmin -report
### Copy file to HDFS
$ hadoop dfs -put somelocalfile hdfsdir/
### View file information on HDFS
$ hadoop fsck -block hdfsdir/somelocalfile
### Run a map/reduce job
$ hadoop jar somejarfile.jar -option1 -option2
### View job info after it completes
$ hadoop job -history hdfsdir/outputdir
### Shut down all Hadoop nodes
$ stop-all.sh
### Copy logfiles back from nodes
$ myhadoop-cleanup.sh
```

Anagram Example – Comet Submit Script

```
#!/bin/bash
#SBATCH --job-name="Anagram"
#SBATCH --output="Anagram.%j.%N.out"
#SBATCH –partition=compute
#SBATCH –nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:00:00
export WRKDIR=`pwd`
myhadoop-configure.sh
start-all.sh
hadoop dfs -mkdir input
hadoop dfs -copyFromLocal $WRKDIR/SINGLE.TXT input/
hadoop jar $WRKDIR/AnagramJob.jar input/SINGLE.TXT output
hadoop dfs -copyToLocal output/part* $PBS_O_WORKDIR
stop-all.sh
myhadoop-cleanup.sh
```

Anagram Example

- **Source:**

  https://code.google.com/p/hadoop-map-reduce-examples/wiki/Anagram_Example

- **Uses Map-Reduce approach to process a file with a list of words, and identify all the anagrams in the file**

- **Code is written in Java. Example has already been compiled and the resulting jar file is in the example directory.**

Anagram – Map Class (Detail)

- String word = value.toString();

Convert the word to a string

- char[] wordChars = word.toCharArray();

Assign to character array

- Arrays.sort(wordChars);

Sort the array of characters

- String sortedWord = new String(wordChars);

Create new string with sorted characters

- sortedText.set(sortedWord);
  orginalText.set(word);
  outputCollector.collect(sortedText, orginalText);

Prepare and output the sorted text string (serves as the key), and the original test string.

Anagram – Map Class (Detail)

- Consider file with list of words : alpha, hills, shill, truck
- Alphabetically sorted words : aahlp, hills, hills, ckrtu
- Hence after the Map Step is done, the following key pairs would be generated:

(aahlp,alpha)
(hills,hills)
(hills,shill)
(ckrtu,truck)

Anagram Example – Reducer (Detail)

- while(anagramValues.hasNext())

```
{
    Text anagam = anagramValues.next();
    output = output + anagam.toString() + "~";
}
```

Iterate over all the values for a key. hasNext() is a Java method that allows you to do this.

We are also creating an output string which has all the words separated with ~.

- StringTokenizer outputTokenizer = new StringTokenizer(output,"~”);

StringTokenizer class allows you to store this delimited string and has functions that allow you to count the number of tokens.

- if(outputTokenizer.countTokens()>=2)

```
{
    output = output.replace("~", ",");
    outputKey.set(anagramKey.toString());
    outputValue.set(output);
    results.collect(outputKey, outputValue);
}
```

We output the anagram key and the word lists if the number of tokens is >=2 (i.e. we have an anagram pair).

Anagram Reducer Class (Detail)

- For our example set, the input to the Reducers is:

  (aahlp,alpha)
  (hills,hills)
  (hills,shill)
  (ckrtu,truck)

- The only key with #tokens >=2 is <hills>.

- Hence, the Reducer output will be:

  hills hills, shill,

Anagram Example – Sample Output

```
cat part-00000
…
aabcdelmnu          manducable,ambulanced,
aabcdeorrsst        broadcasters,rebroadcasts,
aabcdeorrst         rebroadcast,broadcaster,
aabcdkrsw           drawbacks,backwards,
aabcdkrw            drawback,backward,
aabceeehlnsst       teachableness,cheatableness,
aabceeelnnrsstu     uncreatableness,untraceableness,
aabceeelrrt         recreatable,retraceable,
aabceehlt           cheatable,teachable,
aabceellr           lacerable,clearable,
aabceelnrtu         uncreatable,untraceable,
aabceelorrrstv      vertebrosacral,sacrovertebral,
…
…
```

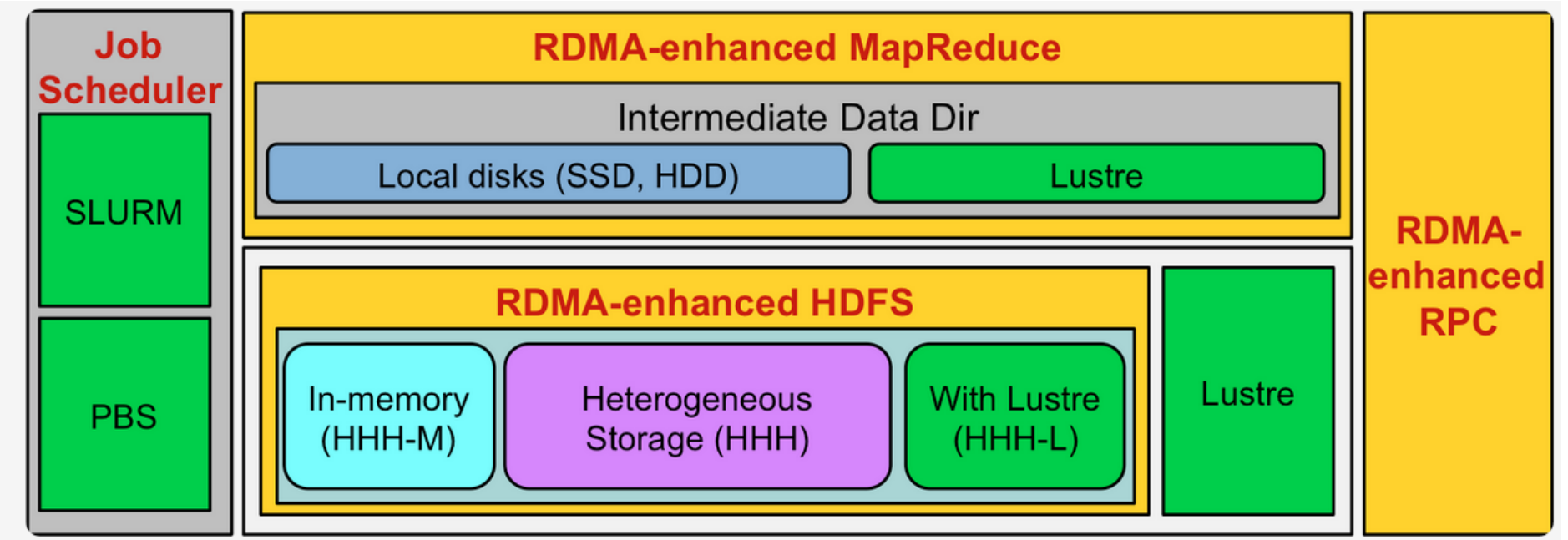
Hadoop-RDMA

Network-Based Computing Lab (Ohio State University)

- **HDFS, MapReduce, and RPC over native InfiniBand and RDMA over Converged Ethernet (RoCE).**
- **Based on Apache Hadoop.**
- **Can use myHadoop to partially set up configuration. Need to add to some of the configuration files.**
- **Version RDMA-Apache-Hadoop-2.x 0.9.7 available on Comet:**
  - **/share/apps/compute/hadoop**
- **More details :**
  - http://hibd.cse.ohio-state.edu/

RDMA-Hadoop

- Exploit performance on modern clusters with RDMA-enabled interconnects for Big Data applications.
- Hybrid design with in-memory and heterogeneous storage (HDD, SSDs, Lustre).
- Keep compliance with standard distributions from Apache.

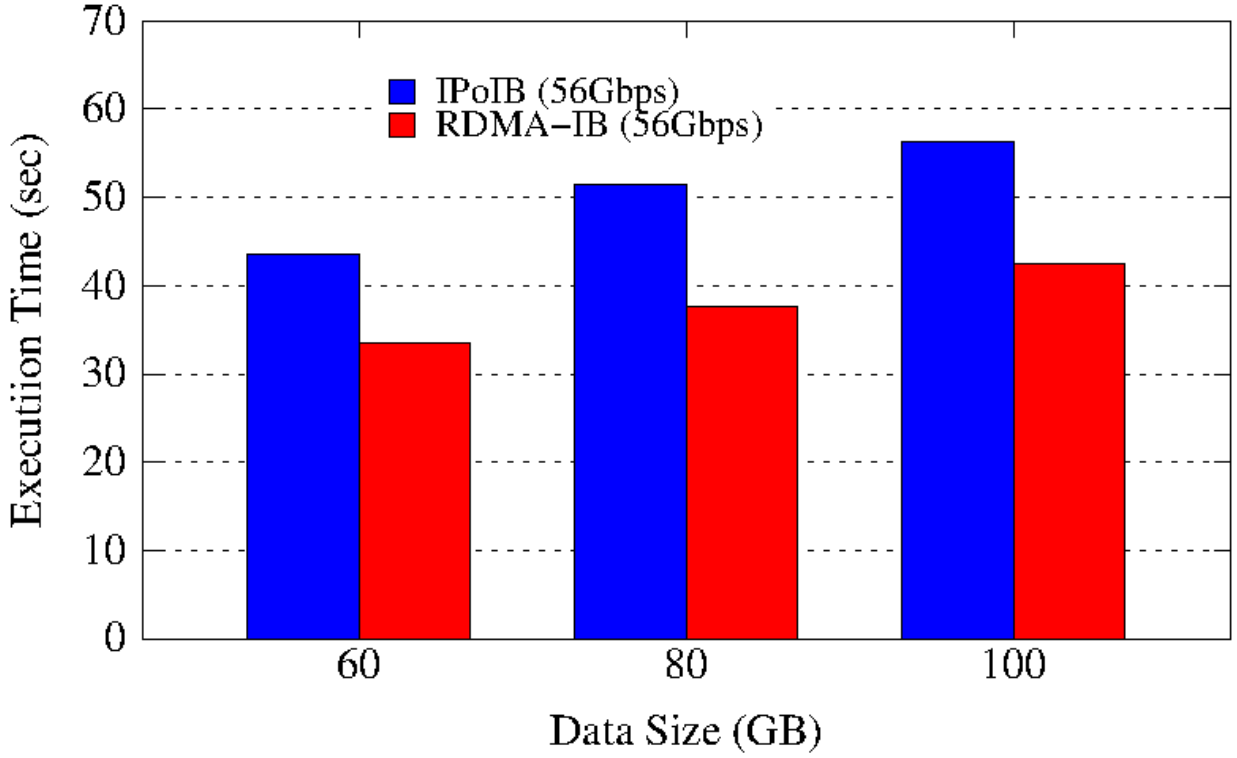


TestDFSIO

- Test to verify HDFS read/write performance
- Generates its own input via `-write` test
- For example to make 8 files, each 1024 MB large:

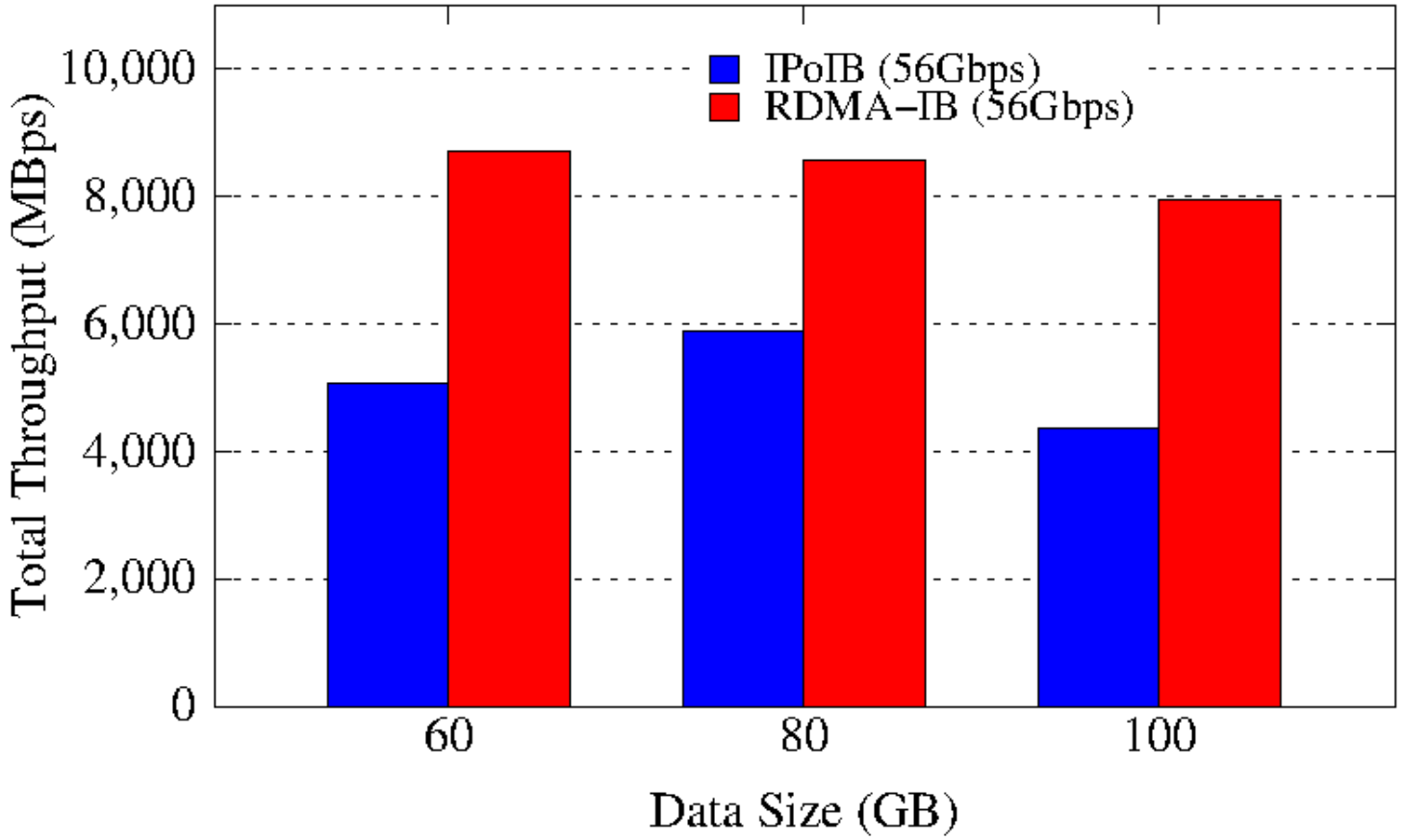
```
$ hadoop jar $HADOOP_HOME/hadoop-test-1.2.1.jar TestDFSIO
    -write -nrFiles 8 -fileSize 1024
14/03/16 16:17:20 INFO fs.TestDFSIO: nrFiles = 8
14/03/16 16:17:20 INFO fs.TestDFSIO: fileSize (MB) = 1024
14/03/16 16:17:20 INFO fs.TestDFSIO: bufferSize = 1000000
14/03/16 16:17:20 INFO fs.TestDFSIO: creating control file: 1024 mega bytes, 8 files
14/03/16 16:17:20 INFO fs.TestDFSIO: created control files for: 8 files
...
14/03/16 16:19:04 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
14/03/16 16:19:04 INFO fs.TestDFSIO:            Date & time: Sun Mar 16 16:19:04 PDT 2014
14/03/16 16:19:04 INFO fs.TestDFSIO:        Number of files: 8
14/03/16 16:19:04 INFO fs.TestDFSIO: Total MBytes processed: 8192
14/03/16 16:19:04 INFO fs.TestDFSIO:     Throughput mb/sec: 39.83815748521631
14/03/16 16:19:04 INFO fs.TestDFSIO: Average IO rate mb/sec: 139.90382385253906
14/03/16 16:19:04 INFO fs.TestDFSIO:  IO rate std deviation: 102.63743717054572
14/03/16 16:19:04 INFO fs.TestDFSIO:    Test exec time sec: 103.64
```

TestDFSIO: Latency* HHH-M mode on Comet



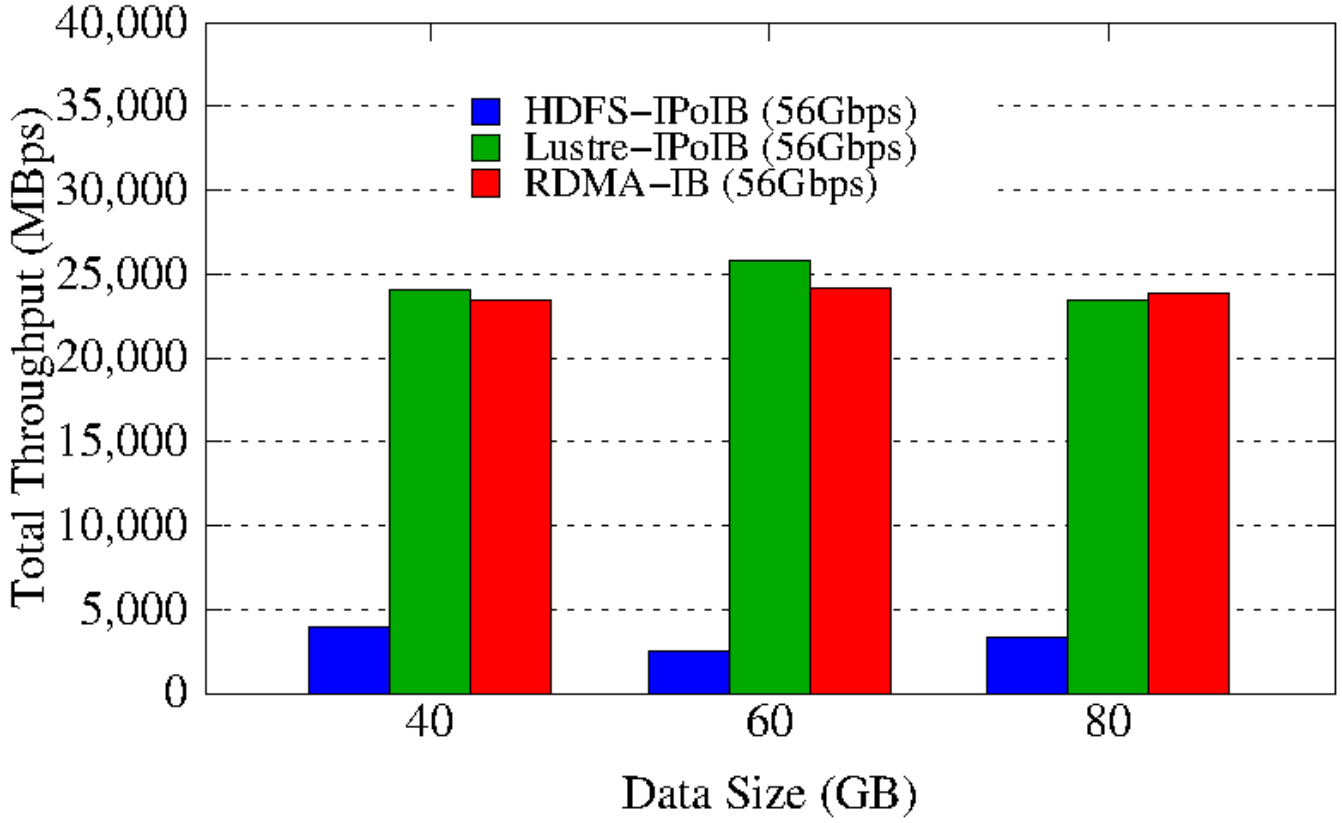
*Reference results from Dr. D.K. Panda's HiBD group at OSU
http://hibd.cse.ohio-state.edu/performance/inmemory/

TestDFSIO: Throughput* HHH-M mode on Comet



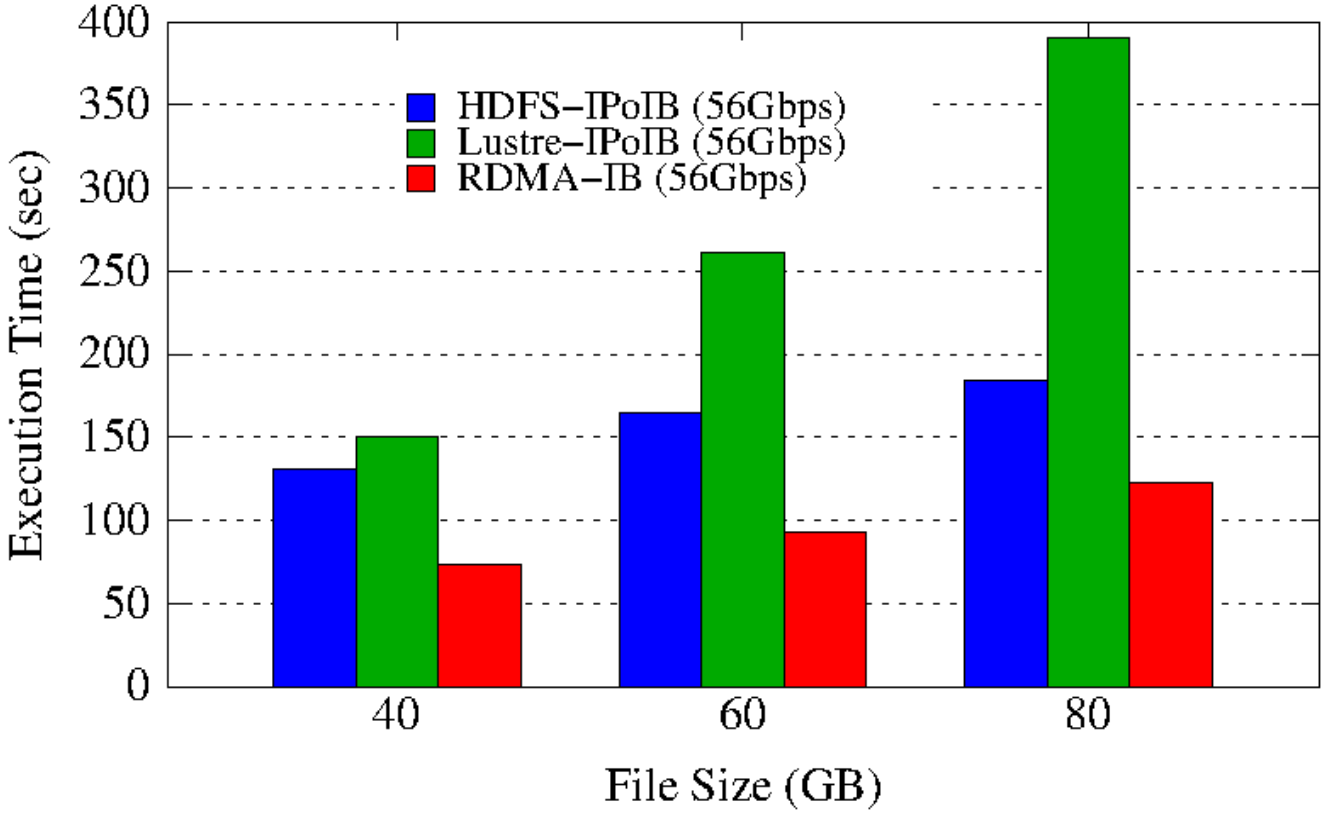
*Reference results from Dr. D.K. Panda's HiBD group at OSU
http://hibd.cse.ohio-state.edu/performance/inmemory/

TestDFSIO: Throughput* HHH-L mode on Comet



***Reference results from Dr. D.K. Panda's HiBD group at OSU**
http://hibd.cse.ohio-state.edu/performance/withlustre/

Sort*
HHH-L mode on Comet



***Reference results from Dr. D.K. Panda's HiBD group at OSU**
http://hibd.cse.ohio-state.edu/performance/withlustre/

Hands On: Anagram using HHH-M mode

```
#!/bin/bash
#SBATCH --job-name="rdmahadoopanagram"
#SBATCH --output="rdmahadoopanagram.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=3
#SBATCH --ntasks-per-node=24
#SBATCH -t 00:15:00

#Script request 3 nodes - one used for namenode, 2 for data nodes/processing

#Set modulepath and load RDMA Hadoop Module
export
MODULEPATH=/share/apps/compute/modulefiles/applications:$MODULEPATH
module load rdma-hadoop/2x-0.9.7
```

Hands On: Anagram using HHH-M mode

```
#Get the host list
export SLURM_NODEFILE=`generate_pbs_nodefile`
cat $SLURM_NODEFILE | sort -u > hosts.hadoop.list

#Use SLURM integrated configuration/startup script
hibd_install_configure_start.sh -s -n ./hosts.hadoop.list -i $SLURM_JOBID -h $HA
DOOP_HOME -j $JAVA_HOME -m hhh-m -r /dev/shm -d /scratch/$USER/$SLURM_JOBID -t /
scratch/$USER/$SLURM_JOBID/hadoop_local

#Commands to run ANAGRAM example
$HADOOP_HOME/bin/hdfs --config $HOME/conf_$SLURM_JOBID dfs -mkdir -p /user/$USER
/input
$HADOOP_HOME/bin/hdfs --config $HOME/conf_$SLURM_JOBID dfs -put SINGLE.TXT /user
/$USER/input/SINGLE.TXT
$HADOOP_HOME/bin/hadoop --config $HOME/conf_$SLURM_JOBID jar AnagramJob.jar /use
r/$USER/input/SINGLE.TXT /user/$USER/output
$HADOOP_HOME/bin/hdfs --config $HOME/conf_$SLURM_JOBID dfs -get /user/$USER/outp
ut/part* $SLURM_WORKING_DIR

#Clean up
hibd_stop_cleanup.sh -d -h $HADOOP_HOME -m hhh-m -r /dev/shm
```

Hands On: Anagram using HHH-L mode

```
#!/bin/bash
#SBATCH --job-name="rdmahadoopanagram"
#SBATCH --output="rdmahadoopanagram.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=3
#SBATCH --ntasks-per-node=24
#SBATCH -t 00:15:00

#Script request 3 nodes - one used for namenode, 2 for data nodes/processing

#Set modulepath and load RDMA Hadoop Module
export
MODULEPATH=/share/apps/compute/modulefiles/applications:$MODULEPATH
module load rdma-hadoop/2x-0.9.7
```

Hands On: Anagram using HHH-L mode

```
#Get the host list
export SLURM_NODEFILE=`generate_pbs_nodefile`
cat $SLURM_NODEFILE | sort -u > hosts.hadoop.list

#Setup Lustre location for HDFS storage and set stripe.
export HDATADIR="/oasis/scratch/comet/$USER/temp_project/HDATA"
if [ ! -d "$HDATADIR" ]; then
        mkdir -p $HDATADIR
fi
lfs setstripe --stripe-size 64m $HDATADIR


#Use SLURM integrated configuration/startup script
hibd_install_configure_start.sh -s -n ./hosts.hadoop.list -i $SLURM_JOBID -h $HADOOP_HOME -
j $JAVA_HOME -m hhh-l -l $HDATADIR -r /dev/shm -d /scratch/$USER/$SLURM_JOBID -t /scratch/$
USER/$SLURM_JOBID/hadoop_local

#Commands to run ANAGRAM example
$HADOOP_HOME/bin/hdfs --config $HOME/conf_$SLURM_JOBID dfs -mkdir -p /user/$USER/input
$HADOOP_HOME/bin/hdfs --config $HOME/conf_$SLURM_JOBID dfs -put SINGLE.TXT /user/$USER/inpu
t/SINGLE.TXT
$HADOOP_HOME/bin/hadoop --config $HOME/conf_$SLURM_JOBID jar AnagramJob.jar /user/$USER/inp
ut/SINGLE.TXT /user/$USER/output
$HADOOP_HOME/bin/hdfs --config $HOME/conf_$SLURM_JOBID dfs -get /user/$USER/output/part* $S
LURM_WORKING_DIR

#Clean up
hibd_stop_cleanup.sh -d -h $HADOOP_HOME -m hhh-l -l $HDATADIR -r /dev/shm
```

Thanks!
Questions: Email mahidhar@sdsc.edu