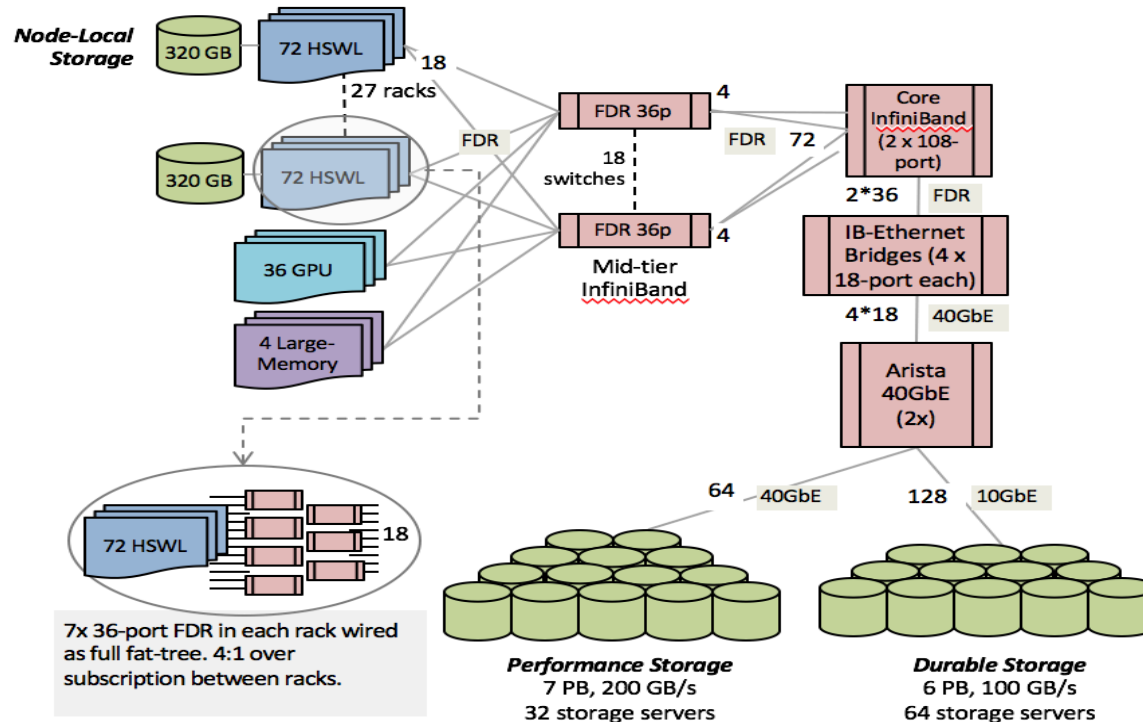


Running and Programming with Spark on Comet

Mahidhar Tatineni
UCSB
April 26, 2018



Comet Architecture



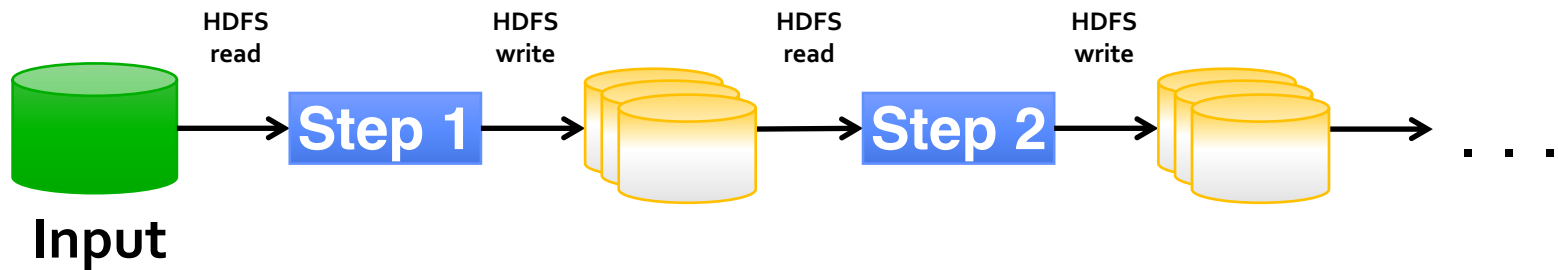
- 24 cores, 128GB RAM per node
- Parallel Lustre filesystem (peak of 100GB/s)
- FDR InfiniBand network
- 320 GB SSD space per node

Spark Overview

- **A distributed data analysis and computing framework**
- **Designed for performance**
 - Uses RAM as much as possible
 - Lazy execution - can be smart about the computations based on the knowledge of steps needed for result
 - Resilient to node failures
- **Easy to use**
 - Python, Scala, Java interfaces
 - Interactive shells
 - Many distributed primitives
- **Many paradigms available**
 - SparkSQL
 - Streaming
 - Mlib
 - Graphx

Efficiency of using RAM for Data Sharing

Hadoop

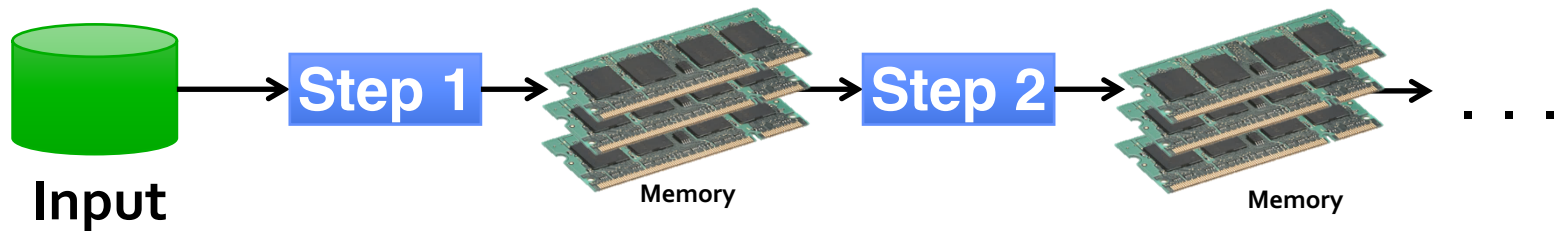


Slow due to block replication, serialization, and disk IO



Spark Uses In-Memory

Spark



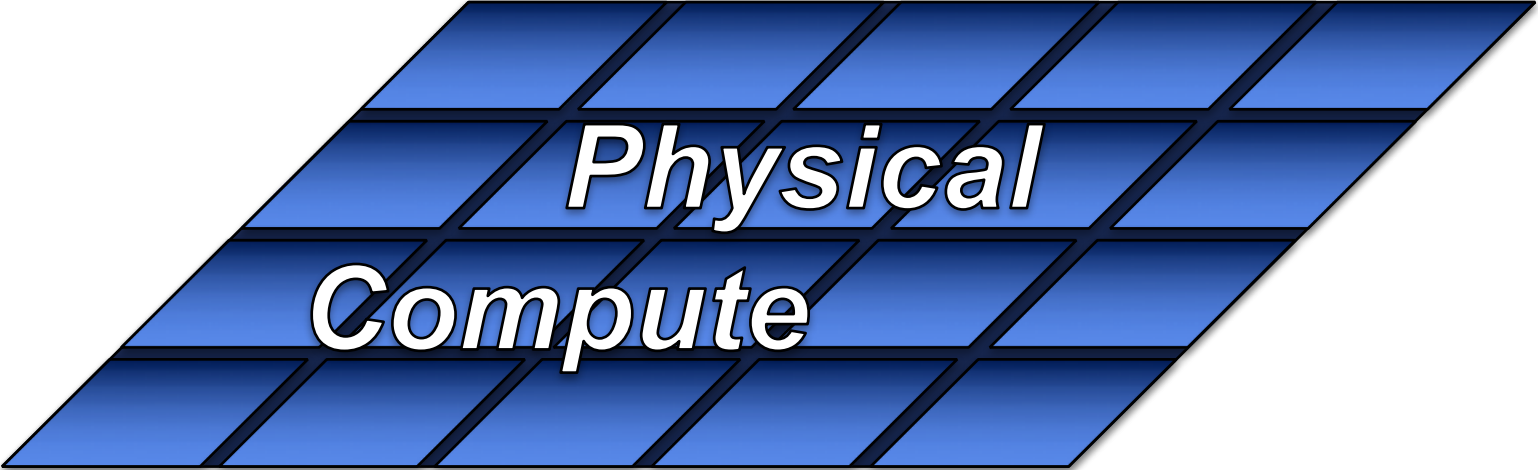
Keeping required data in memory can be 10-100× faster than going to disk/network + allows for optimizations with RDD approach

Figure credit: Xiaoyi Lu, “How to Accelerate Your Big Data Applications with Hadoop and Spark?”, PEARC17

Benefits of RDD Model

- Consistency (helped by immutability)
- Inexpensive fault tolerance (log lineage rather than replicating/checkpointing data)
- Locality-aware scheduling of tasks on partitions
- Model applicable to a broad variety of applications
- **Easy Programming**
 - Java, R, Python, Scala interfaces
- **High-Performance**
 - Memory speeds are much higher than disk speeds!
- **Scalable**

Add Data Analysis to Existing Compute Infrastructure



*Physical
Compute*

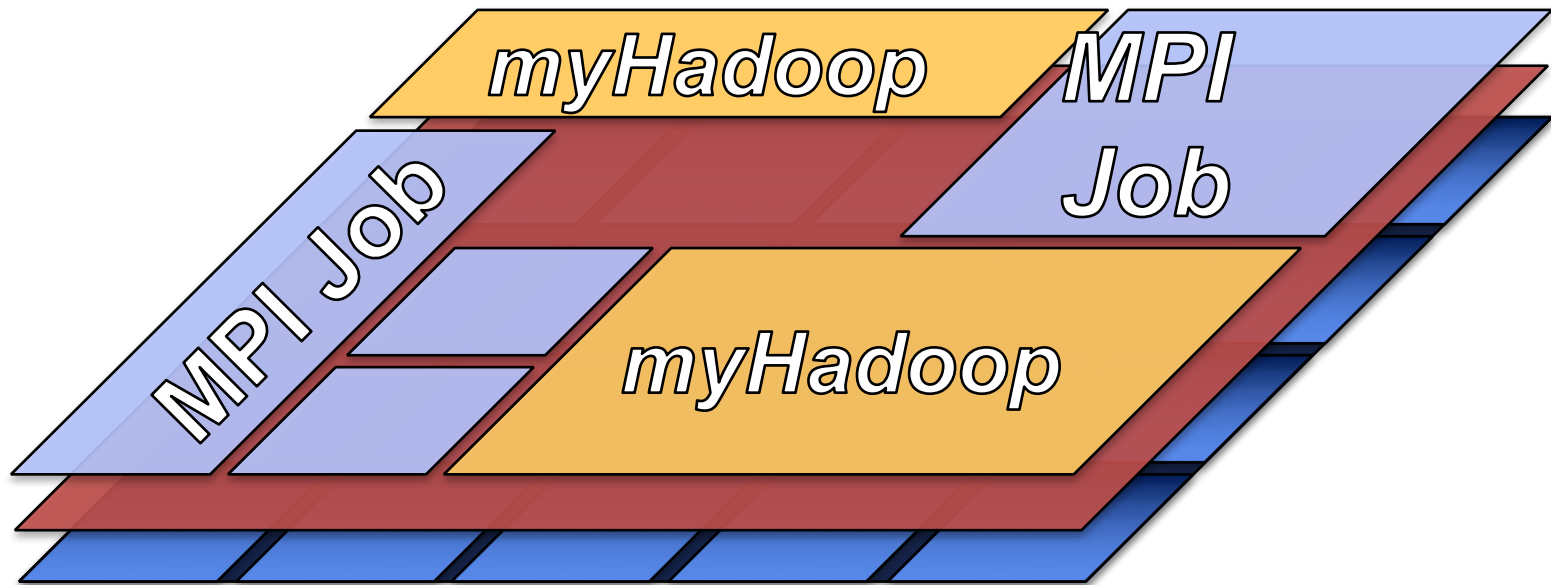
Add Data Analysis to Existing Compute Infrastructure



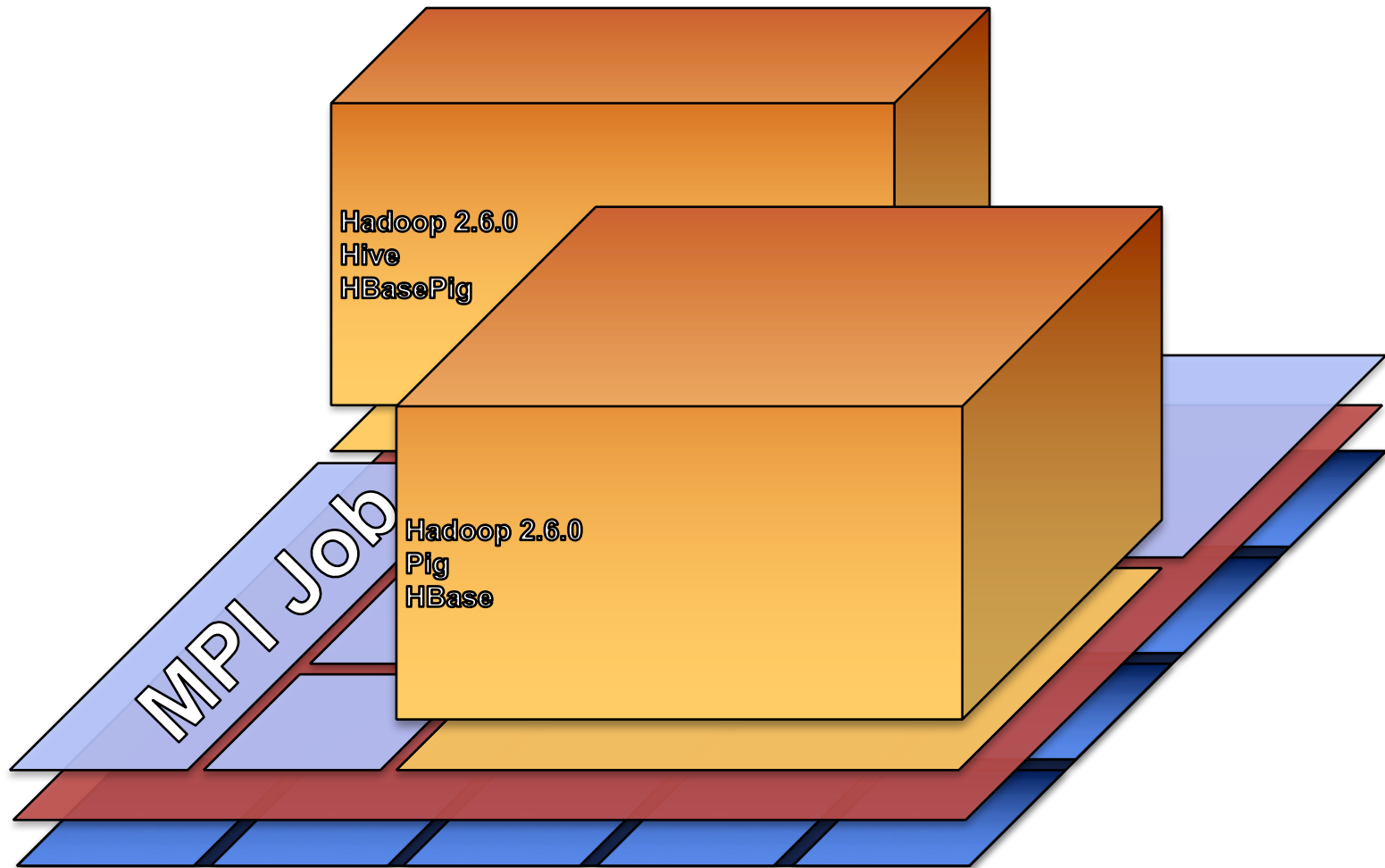
Resource Manager

(Torque, SLURM, SGE)

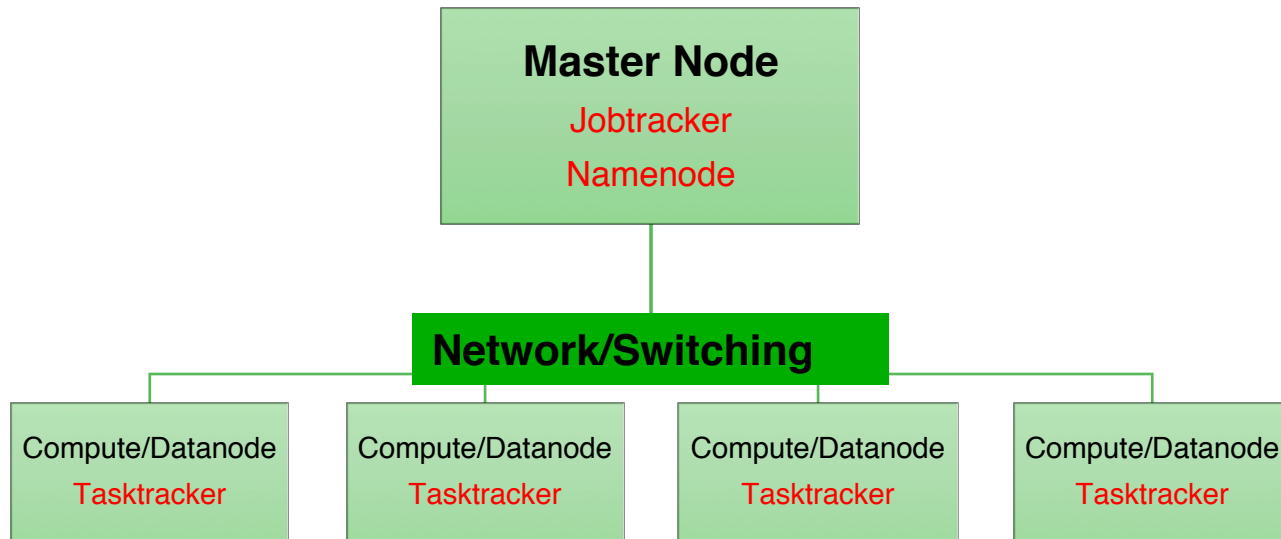
Add Data Analysis to Existing Compute Infrastructure



Add Data Analysis to Existing Compute Infrastructure



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.

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`
export HADOOP_CONF_DIR=/home/$USER/cometcluster
export WORKDIR=`pwd`
module load hadoop
myhadoop-configure.sh
export PATH=/opt/hadoop/2.6.0/sbin:$PATH
start-all.sh
hdfs dfs -mkdir -p /user/$USER/input
hdfs dfs -put $WORKDIR/SINGLE.TXT /user/$USER/input/SINGLE.TXT
hadoop jar $WORKDIR/AnagramJob.jar /user/$USER/input/SINGLE.TXT
/user/$USER/output
hdfs dfs -get /user/$USER/output/part* $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);`

`originalText.set(word);`

`outputCollector.collect(sortedText, originalText);`

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 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

- **Change to directory:**
`cd $HOME/UCSB2018/HADOOP/ANAGRAM_Hadoop2`
(remember to copy /share/apps/examples/UCSB2018)
- **Submit job:**
`sbatch anagram.script`
- **Check configuration in directory:**
`ls $HOME/cometcluster`

Anagram Example – Sample Output

cat part-00000

...

aabcdelmnu	manducable,ambulanced,
aabcdeorrsst	broadcasters,rebroadcasts,
aabcdeorrst	rebroadcast,broadcaster,
aabcdkrsw	drawbacks,backwards,
aabcdkrw	drawback,backward,
aabceeehlmsst	teachableness,cheatableness,
aabceeehlmsstu	uncreatableness,untraceableness,
aabceeehlrt	recreatable,retraceable,
aabceehl	cheatable,teachable,
aabceellr	lacerable,clearable,
aabceelnrstu	uncreatable,untraceable,
aabceelorrstv	vertebrosacral,sacrovertebral,

...

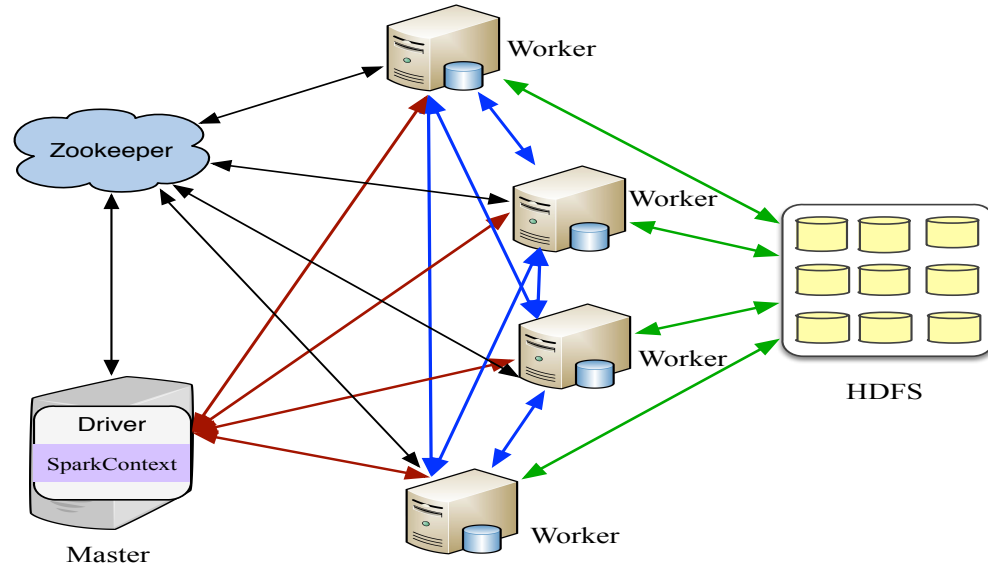
...

Spark Resilient Distributed Dataset (RDD) Approach

- **Create or Load RDD - from data in storage (HDFS, HBase, JSON, S3 etc)**
- **Can also create by transforming another RDD**
- **Transform RDDs - filtering, unique entries, sampling, union, intersection etc**
- **Spark framework will not compute these immediately - only when results are needed. This allows for optimization of the whole process (input to first result).**
- **Only perform actions to get results.**
- **Automatically rebuilt on failure - rebuilt if a partition is lost.**

Spark Architecture Overview

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

Slide credit: Xiaoyi Lu, "How to Accelerate Your Big Data Applications with Hadoop and Spark?", PEARC17

Spark Machine Learning Libraries

- Spark features large suite of ML libraries.
- Basic Statistics
 - Correlations, sampling, hypothesis testing
- Classification and regression
 - Linear models (SVMs, logistic regression, linear regression)
 - Decision trees, Naïve Bayes
- Collaborative filtering
- Clustering - e.g. k-means
- Dimensionality reduction - SVD, PCA
- Feature extraction, transformation

Example pyspark script

`$HOME/UCSB2018/SPARK/pyspark_v210`

```
#!/bin/bash
#####
#  A simple Scala based example for Spark
#  Designed to run on SDSC's Comet resource.
#  Mahidhar Tatineni, San Diego Supercomputer Center March 2016
#####
#SBATCH --job-name="pagerank-pyspark"
#SBATCH --output="pagerank-pyspark.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 00:30:00

### Environment setup for Hadoop and Spark
export MODULEPATH=/share/apps/compute/modulefiles/applications:$MODULEPATH
module load spark/2.1.0
export HADOOP_CONF_DIR=$HOME/mycluster.conf
export WORKDIR=`pwd`

### Sleep for prolog race condition
sleep 30s
myhadoop-configure.sh
```

Example pyspark script

`$HOME/UCSB2018/SPARK/pyspark_v210`

```
### Start HDFS. Starting YARN isn't necessary since Spark will be running in
### standalone mode on our cluster.
start-dfs.sh

### Load in the necessary Spark environment variables
source $HADOOP_CONF_DIR/spark/spark-env.sh

### Start the Spark masters and workers. Do NOT use the start-all.sh provided
### by Spark, as they do not correctly honor $SPARK_CONF_DIR
myspark start
hdfs dfs -mkdir -p /user/input
hdfs dfs -put $SLURM_SUBMIT_DIR/pagerank_data.txt /user/input/pagerank_data.txt
spark-submit $SLURM_SUBMIT_DIR/pagerank.py /user/input/pagerank_data.txt 10

### Shut down Spark and HDFS
myspark stop
stop-dfs.sh

### Clean up
myhadoop-cleanup.sh
```


Customizing Configuration

```
### Sleep for prolog race condition
```

```
sleep 30s
```

```
myhadoop-configure.sh
```

```
### Edit yarn-site.xml
```

```
head -n -1 $HADOOP_CONF_DIR/yarn-site.xml > yarn-site.xml
```

```
cat yarn.snippet.txt >> yarn-site.xml
```

```
cp yarn-site.xml $HADOOP_CONF_DIR/yarn-site.xml
```

```
rm yarn-site.xml
```

```
<property>
```

```
  <name>yarn.scheduler.maximum-allocation-mb</name>
```

```
  <value>12288</value>
```

```
</property>
```

```
<property>
```

```
  <name>yarn.nodemanager.resource.memory-mb</name>
```

```
  <value>12288</value>
```

```
</property>
```

```
<!--property>
```

```
  <name>yarn.nodemanager.local-cache.max-files-per-directory</name>
```

```
  <value>12288</value>
```

```
</property-->
```

```
</configuration>
```

RDMA Spark: Background and Motivation

- HPC architectures feature hardware designed for large scale processing with advanced processors, large and high performance parallel filesystems, advanced networks.
 - RDMA technology allows processes to access remote memory locations with very low CPU load on the remote process.
 - SSDs with high data reliability and performance available on several HPC machines
 - Standard big data middleware and tools like Apache Hadoop and Spark are currently not able to fully take advantage of these capabilities
- ⇒ **Motivation** to develop optimized versions of Apache Hadoop and Spark to fully realize the benefits of the architecture.
- Comet is an excellent test-bed for such tools, with advanced processors, filesystems, SSDs, and high performance networks.

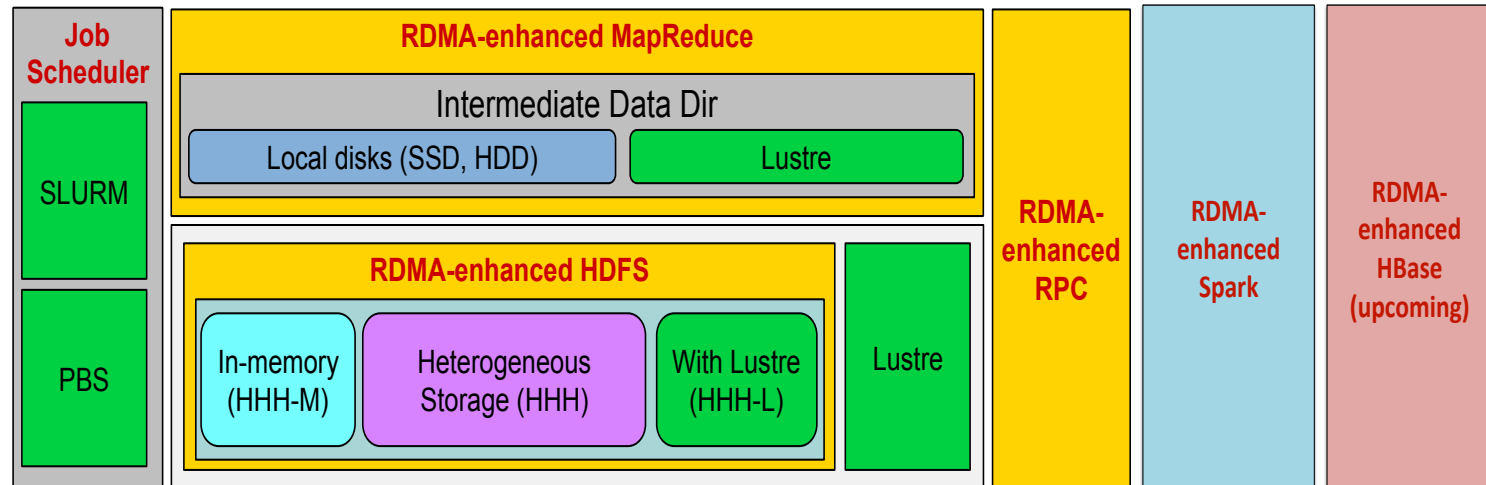
RDMA-Hadoop and RDMA-Spark

Network-Based Computing Lab (Ohio State University)

- HDFS, MapReduce, and RPC over native InfiniBand and RDMA over Converged Ethernet (RoCE).
- Based on Apache distributions of Hadoop and Spark.
- Version **RDMA-Apache-Hadoop-2.x 0.9.7** (based on **Apache Hadoop 2.6.0**) available on Comet
- Version **RDMA-Spark 0.9.1** (based on **Apache Spark 1.5.1**) is available on Comet.
- More details on the RDMA-Hadoop and RDMA-Spark projects at:
 - <http://hibd.cse.ohio-state.edu/>

RDMA-Hadoop and Spark Design

- 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.



RDMA Hadoop – Scheduler Integration

- **Directly integrated with Slurm/PBS**
- **Several running modes:**
 - **HHH:** Heterogeneous storage devices with hybrid replication schemes for fault tolerance and performance.
 - **HHH-M:** A high-performance in-memory based setup.
 - **HHH-L:** Integrated with Lustre parallel filesystem.
 - **MapReduce over Lustre, with/without local disks:** Provides support to run MapReduce jobs on top of Lustre alone.

myHadoop/mySpark Framework

- The framework dynamically populates the Hadoop configuration files.
- Spark cluster startup scripts to avoid issues due to the alternate locations of configuration files.
- The frameworks do not change the Hadoop and Spark distributions and are mainly tools to enable automated configuration and setup on HPC resources.
- **mySpark used for deployment of RDMA-Spark on Comet.**

RDMA Hadoop script

```
#!/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
```

```
hibd_install_configure_start.sh -s -n ./hosts.hadoop.list -i  
$SLURM_JOBID -h $HADOOP_HOME -j $JAVA_HOME -m  
hhh-m -r /dev/shm -d /scratch/$USER/$SLURM_JOBID -t  
/scratch/$USER/$SLURM_JOBID/hadoop_local
```

**At this point in the script we have an active
Hadoop Cluster!**

RDMA Spark script (mySpark)

```
export  
HADOOP_CONF_DIR=$HOME/mycluster.conf
```

```
myhadoop-configure.sh
```

```
start-dfs.sh
```

```
source $HADOOP_CONF_DIR/spark/spark-  
env.sh
```

```
myspark start
```

```
..
```

```
Spark Cluster active ... run regular commands
```

```
..
```

```
myspark stop
```

```
stop-dfs.sh
```

```
myhadoop-cleanup.sh
```

RDMA Hadoop: Benchmark Results

- HDFS I/O performance tested using TestDFSIO and RandomWriter.
- Hadoop performance was tested using Sort and TeraSort benchmarks.
- Performance of RDMA version compared with standard Apache version using the same hardware (Comet).
- Detailed results are available on the HiBD website

RDMA Hadoop: Benchmark Results

<i>Benchmark Name</i>	<i>Configuration</i>	<i>Max. Benefit over Apache Hadoop using IPoIB</i>
<i>TestDFSIO Throughput</i>	<i>RDMA Hadoop HHH-M, 16 Data Nodes</i>	<i>1.82x</i>
<i>RandomWriter</i>	<i>RDMA Hadoop HHH, 16 Data Nodes</i>	<i>1.32x</i>
<i>Sort</i>	<i>RDMA Hadoop HHH, 16 Data Nodes</i>	<i>1.48x</i>
<i>TeraSort</i>	<i>RDMA Hadoop HHH, 16 Data Nodes</i>	<i>1.23x</i>

RDMA Spark: Benchmark Results

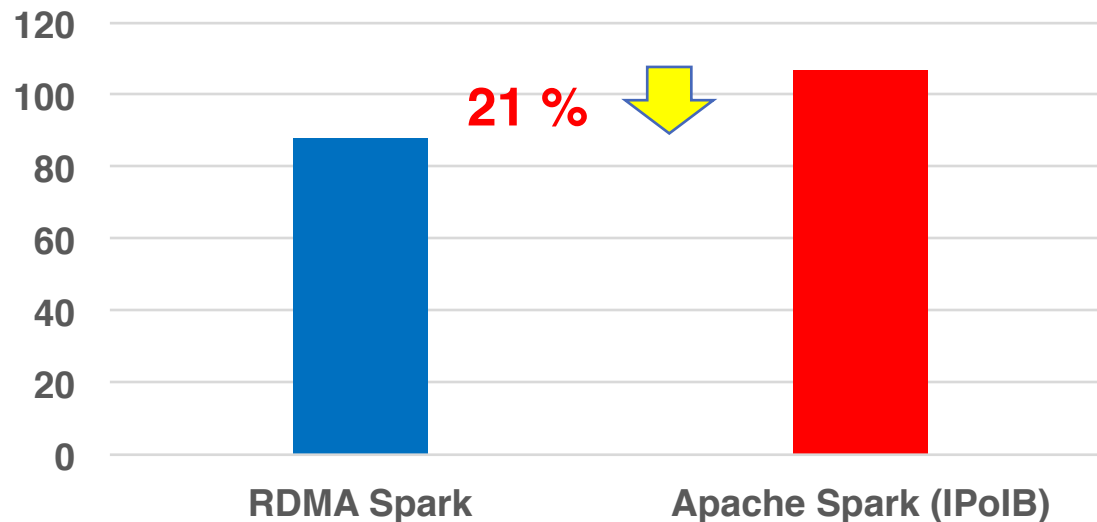
<i>Benchmark Name</i>	<i>Configuration</i>	<i>Max. Benefit over Apache Hadoop using IPoIB</i>
<i>GroupBy</i>	<i>RDMA Spark 64 nodes, 1536 maps/reduces</i>	<i>1.57x</i>
<i>SortBy</i>	<i>RDMA Spark 64 nodes, 1536 maps/reduces</i>	<i>1.8x</i>
<i>HiBench TeraSort</i>	<i>RDMA Spark 64 nodes, 1536 cores</i>	<i>1.22x</i>
<i>HiBench PageRank</i>	<i>RDMA Spark 64 nodes, 1536 cores</i>	<i>1.46x</i>

RDMA-Spark: Applications

- **Kira Toolkit¹**: Distributed astronomy image processing toolkit implemented using Apache Spark.
- Source extractor application, using a 65GB dataset from the SDSS DR2 survey that comprises 11,150 image files.
- Compare RDMA Spark performance with the standard apache implementation using IPoIB.

1. Z. Zhang, K. Barbary, F. A. Nothaft, E.R. Sparks, M.J. Franklin, D.A. Patterson, S. Perlmutter. Scientific Computing meets Big Data Technology: An Astronomy Use Case. *CoRR*, vol: *abs/1507.03325*, Aug 2015.

RDMA-Spark: Applications



**Execution times (s) for Kira SE benchmark
using 65 GB dataset, 48 cores.**

RDMA-Spark: Applications

- **Keystone ML*** : Large scale machine learning pipelines using Spark.
- Benchmark used: RandomPatchCifar pipeline is used with the CIFAR-10 dataset
- Compare RDMA Spark performance with the standard apache implementation using IPoIB.
- With 16 compute nodes an improvement of **4.7%** was seen with the RDMA version.

*<http://keystone-ml.org/>

Developed by UC Berkeley

AMPLab

RDMA Spark: Applications

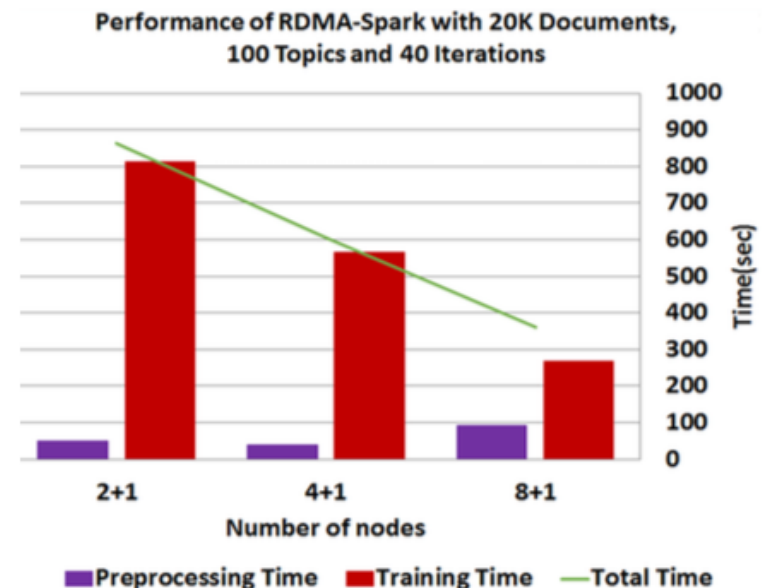
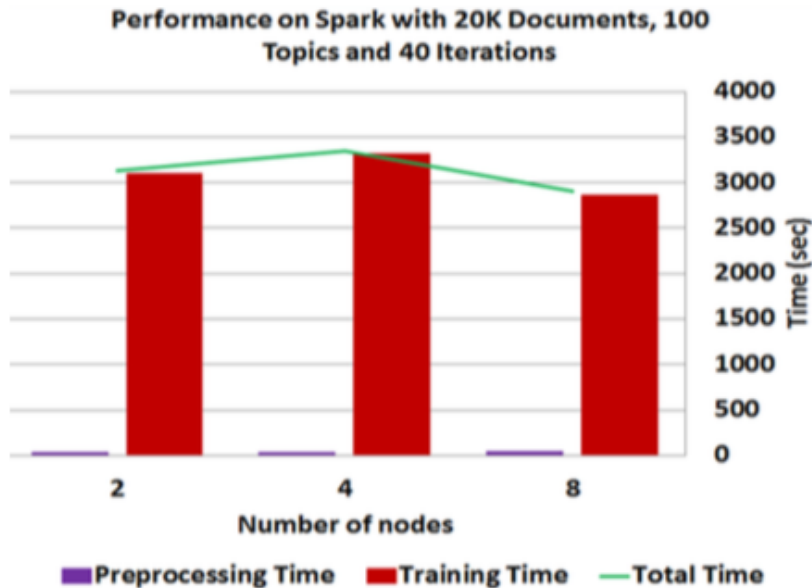
- **Application in Social Sciences: Topic modeling using big data middleware and tools*.**
- **Latent Dirichlet Allocation (LDA) for unsupervised analysis of large document collections.**
- **Computational complexity increases as the volume of data increases.**
- **RDMA Spark enabled simulation of largest test cases.**

***See XSEDE16 Poster:**

Investigating Topic Models for Big Data Analysis in Social Science Domain

Nitin Sukhija, Nicole Brown, Paul Rodriguez, Mahidhar Tatineni, and Mark Van Moer

RDMA Spark: Topic Modeling Application

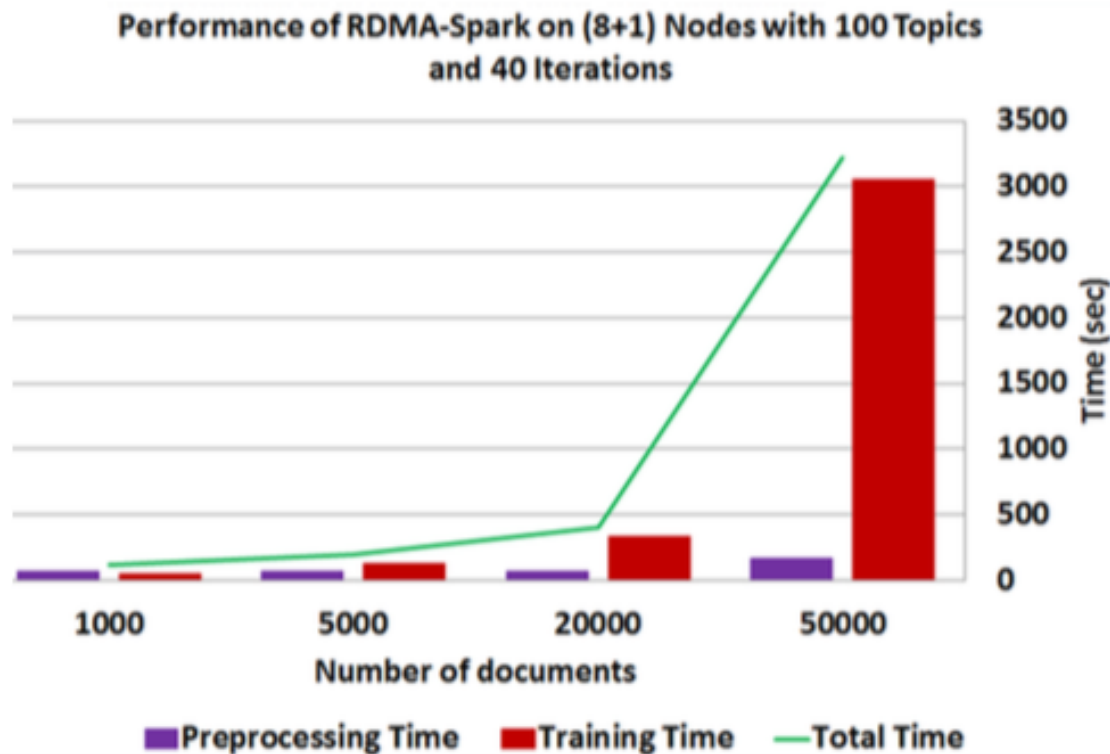


***Reference: XSEDE16 Poster:**

Investigating Topic Models for Big Data Analysis in Social Science Domain

Nitin Sukhija, Nicole Brown, Paul Rodriguez, Mahidhar Tatineni, and Mark Van Moer

RDMA Spark: Topic Modeling Application



***Reference: XSEDE16 Poster:**

Investigating Topic Models for Big Data Analysis in Social Science Domain
Nitin Sukhija, Nicole Brown, Paul Rodriguez, Mahidhar Tatineni, and Mark Van Moer

Summary

- Hadoop and Spark clusters can be dynamically started within the scheduler framework using myHadoop.
- RDMA Hadoop and Spark are designed to take advantage of advanced processor, storage, and networks available on HPC systems.
- RDMA Hadoop integrated with Slurm/PBS schedulers.
- RDMA Spark enabled via myHadoop and mySpark framework.
- Tests on Comet show performance enhancements for both standard benchmarks and user applications. **No change was required in the user applications!**

Thanks!

Questions: Email mahidhar@sdsc.edu