

Practical Problem Solving with Hadoop and Pig

Milind Bhandarkar
(milindb@yahoo-inc.com)

Agenda

- Introduction
- Hadoop
 - Distributed File System
 - Map-Reduce
- Pig
- Q & A



Agenda: Morning (8.30 - 12.00)

- Introduction
- Motivating Examples
- Hadoop Distributed File System
- Hadoop Map-Reduce
- Q & A



Agenda: Afternoon (1.30 - 5.00)

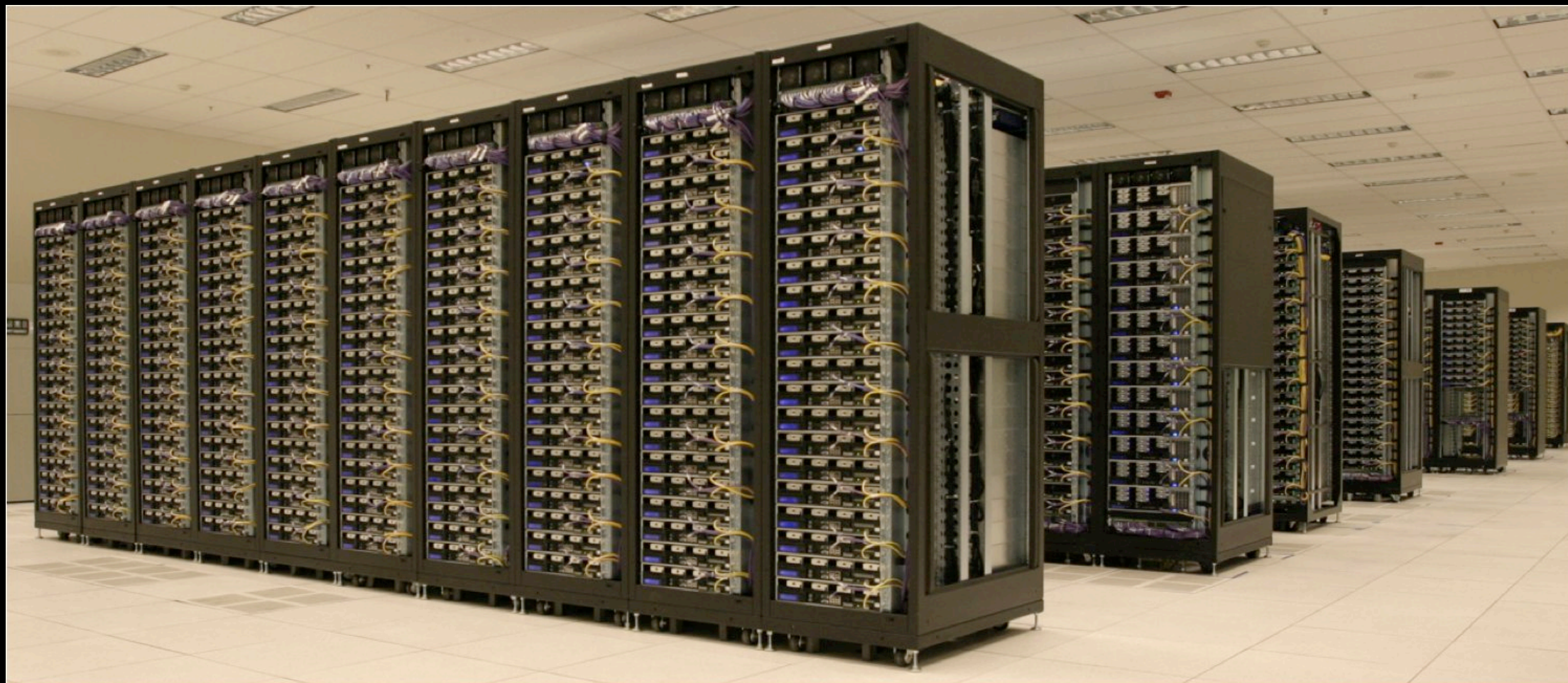
- Performance Tuning
- Hadoop Examples
- Pig
 - Pig Latin Language & Examples
 - Architecture
- Q & A



About Me

- Lead Yahoo! Grid Solutions Team since June 2005
- Contributor to Hadoop since January 2006
- Trained 1000+ Hadoop users at Yahoo! & elsewhere
- 20+ years of experience in Parallel Programming





Hadoop At Yahoo!

Hadoop At Yahoo!

(Some Statistics)

- 25,000 + machines in 10+ clusters
- Largest cluster is 3,000 machines
- 3 Petabytes of data (compressed, unreplicated)
- 700+ users
- 10,000+ jobs/week



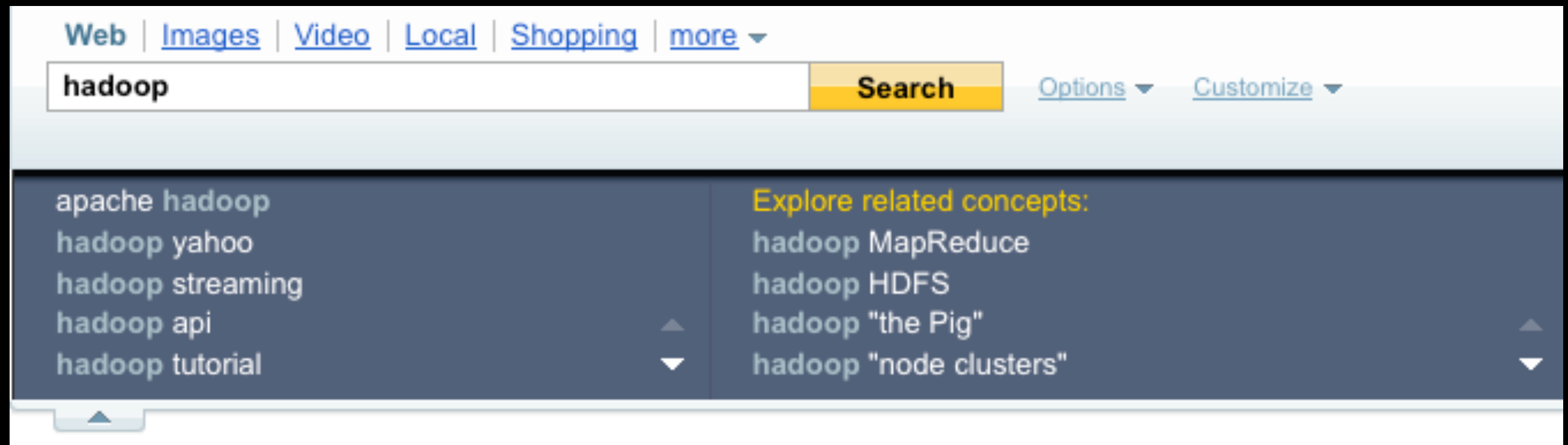
Sample Applications

- Data analysis is the inner loop of Web 2.0
 - Data \Rightarrow Information \Rightarrow Value
- Log processing: reporting, buzz
- Search index
- Machine learning: Spam filters
- Competitive intelligence



Prominent Hadoop Users

- Yahoo!
- A9.com
- EHarmony
- Facebook
- Fox Interactive Media
- IBM
- Quantcast
- Joost
- Last.fm
- Powerset
- New York Times
- Rackspace



Yahoo! Search Assist

Search Assist

- Insight: Related concepts appear close together in text corpus
- Input: Web pages
 - 1 Billion Pages, 10K bytes each
 - 10 TB of input data
- Output: List(word, List(related words))



Search Assist

```
// Input: List(URL, Text)
foreach URL in Input :
    Words = Tokenize(Text(URL));
    foreach word in Tokens :
        Insert (word, Next(word, Tokens)) in Pairs;
        Insert (word, Previous(word, Tokens)) in Pairs;
// Result: Pairs = List (word, RelatedWord)
Group Pairs by word;
// Result: List (word, List(RelatedWords))
foreach word in Pairs :
    Count RelatedWords in GroupedPairs;
// Result: List (word, List(RelatedWords, count))
foreach word in CountedPairs :
    Sort Pairs(word, *) descending by count;
    choose Top 5 Pairs;
// Result: List (word, Top5(RelatedWords))
```



People you may know

Yong Gao

Senior Engineering Manager at
Yahoo Inc.

Invite | ✕

Priyank Garg

Director Product Management,
Yahoo! Search; Jt Managing Director
at Advance Valves

Invite | ✕

Andrew Yu

Yahoo!, Co-creator of PostgreSQL

Invite | ✕

[See more »](#)

You Might Also Know



You Might Also Know

- Insight: You might also know Joe Smith if a lot of folks you know, know Joe Smith
 - if you don't know Joe Smith already
- Numbers:
 - 300 MM users
 - Average connections per user is 100



You Might Also Know

```
// Input: List(Username, List(Connections))

foreach u in UserList : // 300 MM
  foreach x in Connections(u) : // 100
    foreach y in Connections(x) : // 100
      if (y not in Connections(u)) :
        Count(u, y)++; // 3 Trillion Iterations
Sort (u,y) in descending order of Count(u,y);
Choose Top 3 y;
Store (u, {y0, y1, y2}) for serving;
```



Performance

- 101 Random accesses for each user
 - Assume 1 ms per random access
 - 100 ms per user
- 300 MM users
 - 300 days on a single machine



MapReduce Paradigm



Map & Reduce

- Primitives in Lisp (& Other functional languages) 1970s
- Google Paper 2004
 - <http://labs.google.com/papers/mapreduce.html>



Map

```
Output_List = Map (Input_List)
```

```
Square (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) =
```

```
(1, 4, 9, 16, 25, 36, 49, 64, 81, 100)
```

Reduce

`Output_Element = Reduce (Input_List)`

`Sum (1, 4, 9, 16, 25, 36, 49, 64, 81, 100) = 385`



Parallelism

- Map is inherently parallel
 - Each list element processed independently
- Reduce is inherently sequential
 - Unless processing multiple lists
- Grouping to produce multiple lists



Search Assist Map

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

```
Pairs = Tokenize_And_Pair ( Text ( Input ) )
```

```
Output = {  
(apache, hadoop) (hadoop, mapreduce) (hadoop, streaming)  
(hadoop, pig) (apache, pig) (hadoop, DFS) (streaming,  
commandline) (hadoop, java) (DFS, namenode) (datanode,  
block) (replication, default)...  
}
```



Search Assist Reduce

```
// Input: GroupedList (word, GroupedList(words))  
CountedPairs = CountOccurrences (word, RelatedWords)
```

```
Output = {  
(hadoop, apache, 7) (hadoop, DFS, 3) (hadoop, streaming,  
4) (hadoop, mapreduce, 9) ...  
}
```



Issues with Large Data

- Map Parallelism: Splitting input data
 - Shipping input data
- Reduce Parallelism:
 - Grouping related data
- Dealing with failures
 - Load imbalance





Apache Hadoop

- January 2006: Subproject of Lucene
- January 2008: Top-level Apache project
- Latest Version: 0.21
- Stable Version: 0.20.x
- Major contributors: Yahoo!, Facebook, Powerset



Apache Hadoop

- Reliable, Performant Distributed file system
- MapReduce Programming framework
- Sub-Projects: HBase, Hive, Pig, Zookeeper, Chukwa, Avro
- Related Projects: Mahout, Hama, Cascading, Scribe, Cassandra, Dumbo, Hypertable, KosmosFS



Problem: Bandwidth to Data

- Scan 100TB Datasets on 1000 node cluster
 - Remote storage @ 10MB/s = 165 mins
 - Local storage @ 50-200MB/s = 33-8 mins
- Moving computation is more efficient than moving data
 - Need visibility into data placement



Problem: Scaling Reliably

- Failure is not an option, it's a rule !
 - 1000 nodes, MTBF < 1 day
 - 4000 disks, 8000 cores, 25 switches, 1000 NICs, 2000 DIMMS (16TB RAM)
- Need fault tolerant store with reasonable availability guarantees
 - Handle hardware faults transparently



Hadoop Goals

- Scalable: Petabytes (10^{15} Bytes) of data on thousands of nodes
- Economical: Commodity components only
- Reliable
 - Engineering reliability into every application is expensive



Hadoop Distributed File System



HDFS

- Data is organized into files and directories
- Files are divided into uniform sized blocks (default 64MB) and distributed across cluster nodes
- HDFS exposes block placement so that computation can be migrated to data



HDFS

- Blocks are replicated (default 3) to handle hardware failure
- Replication for performance and fault tolerance (Rack-Aware placement)
- HDFS keeps checksums of data for corruption detection and recovery



HDFS

- Master-Worker Architecture
- Single NameNode
- Many (Thousands) DataNodes



HDFS Master (NameNode)

- Manages filesystem namespace
- File metadata (i.e. “inode”)
- Mapping inode to list of blocks + locations
- Authorization & Authentication
- Checkpoint & journal namespace changes



Namenode

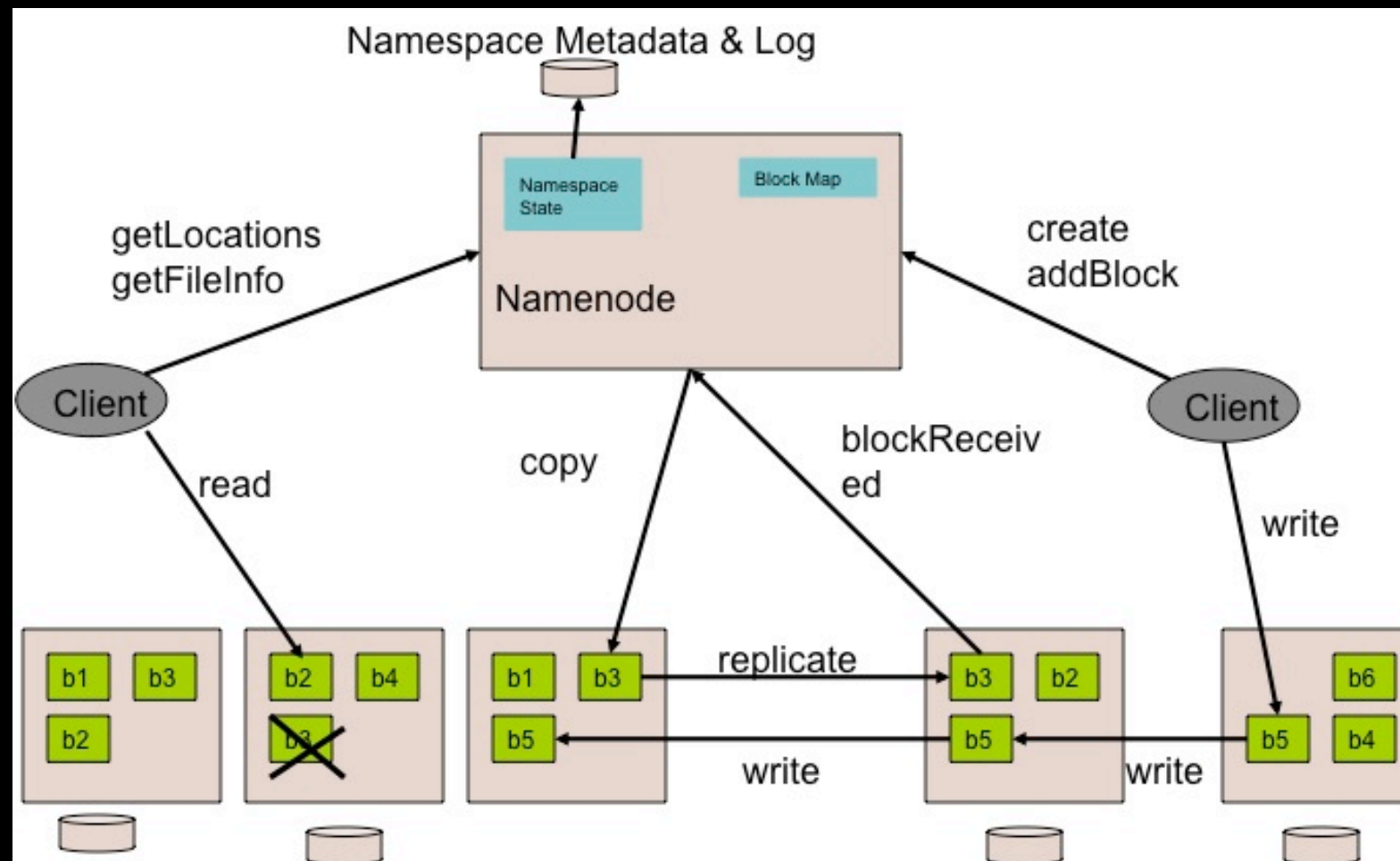
- Mapping of datanode to list of blocks
- Monitor datanode health
- Replicate missing blocks
- Keeps ALL namespace in memory
- 60M objects (File/Block) in 16GB



Datanodes

- Handle block storage on multiple volumes & block integrity
- Clients access the blocks directly from data nodes
- Periodically send heartbeats and block reports to Namenode
- Blocks are stored as underlying OS's files





HDFS Architecture

Replication

- A file's replication factor can be changed dynamically (default 3)
- Block placement is rack aware
- Block under-replication & over-replication is detected by Namenode
- Balancer application rebalances blocks to balance datanode utilization



Accessing HDFS

```
hadoop fs [-fs <local | file system URI>] [-conf <configuration file>]
  [-D <property=value>] [-ls <path>] [-lsr <path>] [-du <path>]
  [-dus <path>] [-mv <src> <dst>] [-cp <src> <dst>] [-rm <src>]
  [-rmr <src>] [-put <localsrc> ... <dst>] [-copyFromLocal <localsrc> ... <dst>]
  [-moveFromLocal <localsrc> ... <dst>] [-get [-ignoreCrc] [-crc] <src> <localdst>]
  [-getmerge <src> <localdst> [addnl]] [-cat <src>]
  [-copyToLocal [-ignoreCrc] [-crc] <src> <localdst>] [-moveToLocal <src> <localdst>]
  [-mkdir <path>] [-report] [-setrep [-R] [-w] <rep> <path/file>]
  [-touchz <path>] [-test [-ezd] <path>] [-stat [format] <path>]
  [-tail [-f] <path>] [-text <path>]
  [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
  [-chown [-R] [OWNER][:[GROUP]] PATH...]
  [-chgrp [-R] GROUP PATH...]
  [-count[-q] <path>]
  [-help [cmd]]
```



HDFS Java API

```
// Get default file system instance  
fs = Filesystem.get(new Configuration());
```

```
// Or Get file system instance from URI  
fs = Filesystem.get(URI.create(uri),  
                    new Configuration());
```

```
// Create, open, list, ...  
OutputStream out = fs.create(path, ...);
```

```
InputStream in = fs.open(path, ...);
```

```
boolean isDone = fs.delete(path, recursive);
```

```
FileStatus[] fstat = fs.listStatus(path);
```



libHDFS

```
#include "hdfs.h"
```

```
hdfsFS fs = hdfsConnectNewInstance("default", 0);
```

```
hdfsFile writeFile = hdfsOpenFile(fs, "/tmp/test.txt",  
                                O_WRONLY|O_CREAT, 0, 0, 0);
```

```
tSize num_written = hdfsWrite(fs, writeFile,  
                             (void*)buffer, sizeof(buffer));
```

```
hdfsCloseFile(fs, writeFile);
```

```
hdfsFile readFile = hdfsOpenFile(fs, "/tmp/test.txt",  
                                O_RDONLY, 0, 0, 0);
```

```
tSize num_read = hdfsRead(fs, readFile, (void*)buffer,  
                          sizeof(buffer));
```

```
hdfsCloseFile(fs, readFile);
```

```
hdfsDisconnect(fs);
```



Installing Hadoop

- Check requirements
 - Java 1.6+
 - bash (Cygwin on Windows)
- Download Hadoop release
- Change configuration
- Launch daemons



Download Hadoop

```
$ wget http://www.apache.org/dist/hadoop/core/hadoop-0.18.3/hadoop-0.18.3.tar.gz
```

```
$ tar zxvf hadoop-0.18.3.tar.gz
```

```
$ cd hadoop-0.18.3
```

```
$ ls -cF conf
```

commons-logging.properties	hadoop-site.xml
configuration.xml	log4j.properties
hadoop-default.xml	masters
hadoop-env.sh	slaves
hadoop-metrics.properties	sslinfo.xml.example



Set Environment

```
# Modify conf/hadoop-env.sh
```

```
$ export JAVA_HOME=....
```

```
$ export HADOOP_HOME=....
```

```
$ export HADOOP_SLAVES=${HADOOP_HOME}/conf/slaves
```

```
$ export HADOOP_CONF_DIR=${HADOOP_HOME}/conf
```

```
# Enable password-less ssh
```

```
# Assuming $HOME is shared across all nodes
```

```
$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
```

```
$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```



Make Directories

```
# On Namenode, create metadata storage and tmp space
```

```
$ mkdir -p /home/hadoop/dfs/name
```

```
$ mkdir -p /tmp/hadoop
```

```
# Create "slaves" file
```

```
$ cat > conf/slaves
```

```
slave00
```

```
slave01
```

```
slave02
```

```
...
```

```
^D
```

```
# Create data directories on each slave
```

```
$ bin/slaves.sh "mkdir -p /tmp/hadoop"
```

```
$ bin/slaves.sh "mkdir -p /home/hadoop/dfs/data"
```



Start Daemons

```
# Modify hadoop-site.xml with appropriate  
# fs.default.name, mapred.job.tracker, etc.
```

```
$ mv ~/myconf.xml conf/hadoop-site.xml
```

```
# On Namenode
```

```
$ bin/hadoop namenode -format
```

```
# Start all daemons
```

```
$ bin/start-all.sh
```

```
# Done !
```



NameNode 'kryptoniteblue-nn1'

Started: Mon Apr 27 04:21:47 UTC 2009
Version: 0.20.0-2701599, r
Compiled: Tue Apr 21 18:09:33 UTC 2009 by hadoopqa
Upgrades: There are no upgrades in progress.

[Browse the filesystem](#)

[Namenode Logs](#)

Check Namenode

Cluster Summary

10314198 files and directories, 12246847 blocks = 22561045 total. Heap Size is 9.57 GB / 13.57 GB (70%)

Configured Capacity	:	1.73 PB
DFS Used	:	1.53 PB
Non DFS Used	:	97.8 GB
DFS Remaining	:	201.15 TB
DFS Used%	:	88.63 %
DFS Remaining%	:	11.36 %
Live Nodes	:	1495
Dead Nodes	:	122

Cluster Summary

Contents of directory [/data](#)/wikipedia

Goto :

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
070719.english	dir				2007-09-26 18:48	rwxr-xr-x	dfsload	users

[Go back to DFS home](#)

Browse Filesystem

Contents of directory [/data/wikipedia](#)/070719.english

Goto :

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
wikipedia-070719-preprocessed.tar.bz2	file	2.12 GB	3	128 MB	1970-01-01 00:00	rw-r--r--	dfsload	users

[Go back to DFS home](#)

Browse Filesystem

Total number of blocks: 17

-3103537327602516330:	72.30.62.138:50010	74.6.129.237:50010	74.6.129.166:50010
-7369203098763751824:	72.30.62.232:50010	74.6.130.98:50010	74.6.133.137:50010
7541578718198662046:	72.30.117.212:50010	72.30.127.149:50010	74.6.133.45:50010
-3788675958108545909:	72.30.117.229:50010	72.30.117.206:50010	74.6.129.172:50010
609766051635167935:	72.30.62.164:50010	74.6.132.164:50010	74.6.132.231:50010
3700676280369084767:	74.6.128.103:50010	72.30.126.229:50010	74.6.128.157:50010
2726417406454976834:	74.6.133.30:50010	74.6.129.122:50010	74.6.134.227:50010
6159498111536046095:	72.30.62.143:50010	74.6.132.152:50010	74.6.128.103:50010
6333317394369287582:	74.6.128.162:50010	74.6.129.91:50010	74.6.128.172:50010
2686045215525902675:	72.30.127.150:50010	72.30.127.143:50010	72.30.117.214:50010
-766635134612161096:	74.6.133.137:50010	72.30.126.103:50010	74.6.132.207:50010
-1127054624816251773:	74.6.128.91:50010	72.30.116.102:50010	72.30.117.116:50010
128162120561656539:	72.30.62.162:50010	74.6.134.202:50010	74.6.130.70:50010
-492484878073149708:	74.6.132.154:50010	74.6.133.76:50010	74.6.129.52:50010
5753314516432447758:	74.6.129.42:50010	74.6.128.232:50010	74.6.132.216:50010
5651363982528296006:	72.30.117.139:50010	74.6.130.124:50010	72.30.116.142:50010
-982935715488760047:	74.6.134.145:50010	72.30.116.228:50010	74.6.130.52:50010

Browse Filesystem



Questions ?

Hadoop MapReduce



Think MR

- Record = (Key, Value)
- Key : Comparable, Serializable
- Value: Serializable
- Input, Map, Shuffle, Reduce, Output



Seems Familiar ?

```
cat /var/log/auth.log* | \  
grep "session opened" | cut -d' ' -f10 | \  
sort | \  
uniq -c > \  
~/userlist
```



Map

- Input: $(Key_1, Value_1)$
- Output: $List(Key_2, Value_2)$
- Projections, Filtering, Transformation



Shuffle

- Input: $\text{List}(\text{Key}_2, \text{Value}_2)$
- Output
 - $\text{Sort}(\text{Partition}(\text{List}(\text{Key}_2, \text{List}(\text{Value}_2))))$
- Provided by Hadoop



Reduce

- Input: $\text{List}(\text{Key}_2, \text{List}(\text{Value}_2))$
- Output: $\text{List}(\text{Key}_3, \text{Value}_3)$
- Aggregation



Example: Unigrams

- Input: Huge text corpus
 - Wikipedia Articles (40GB uncompressed)
- Output: List of words sorted in descending order of frequency



Unigrams

```
$ cat ~/wikipedia.txt | \  
sed -e 's/ /\n/g' | grep . | \  
sort | \  
uniq -c > \  
~/frequencies.txt
```

```
$ cat ~/frequencies.txt | \  
# cat | \  
sort -n -k1,1 -r | \  
# cat > \  
~/unigrams.txt
```



MR for Unigrams

```
mapper (filename, file-contents):  
    for each word in file-contents:  
        emit (word, 1)
```

```
reducer (word, values):  
    sum = 0  
    for each value in values:  
        sum = sum + value  
    emit (word, sum)
```

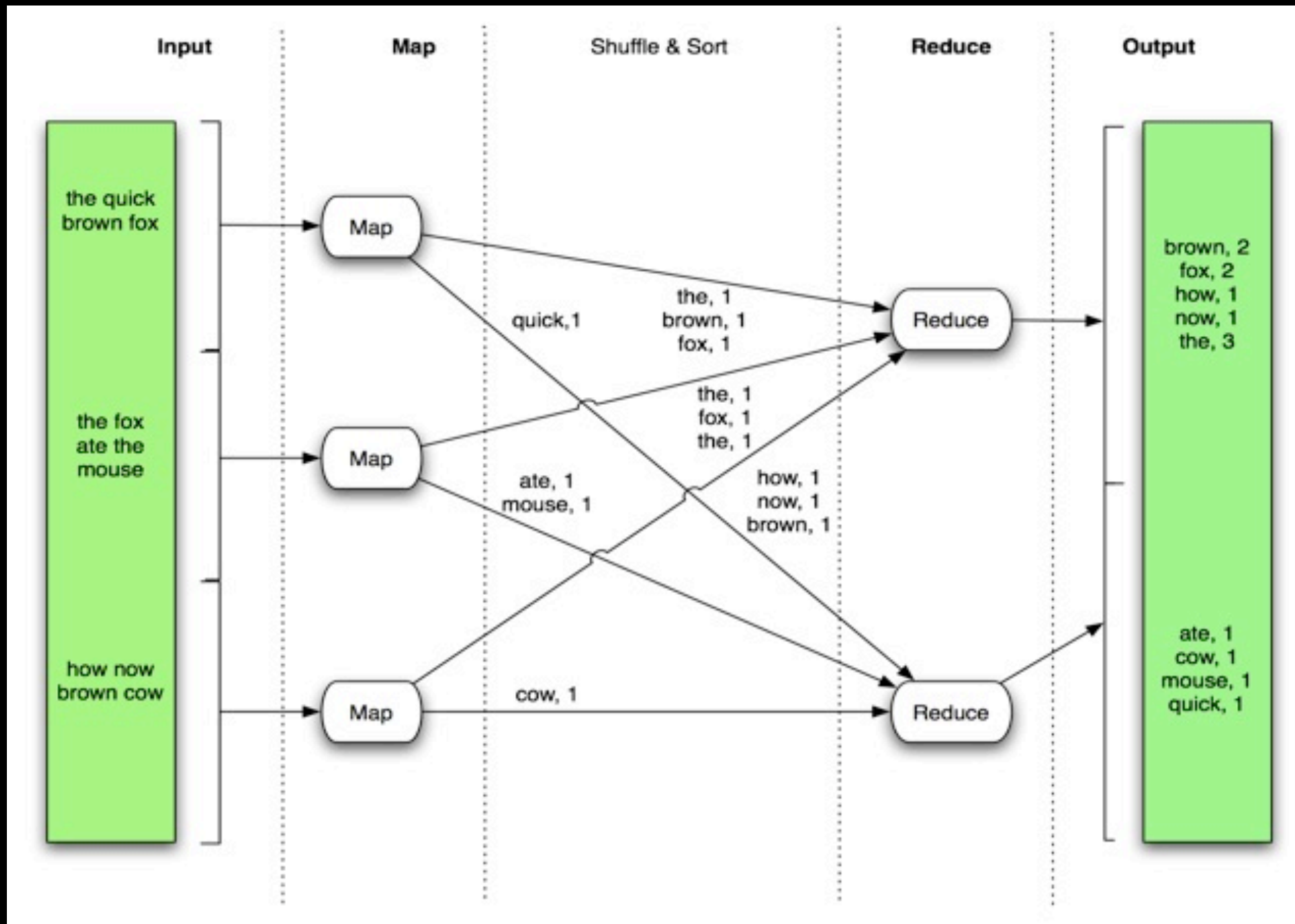


MR for Unigrams

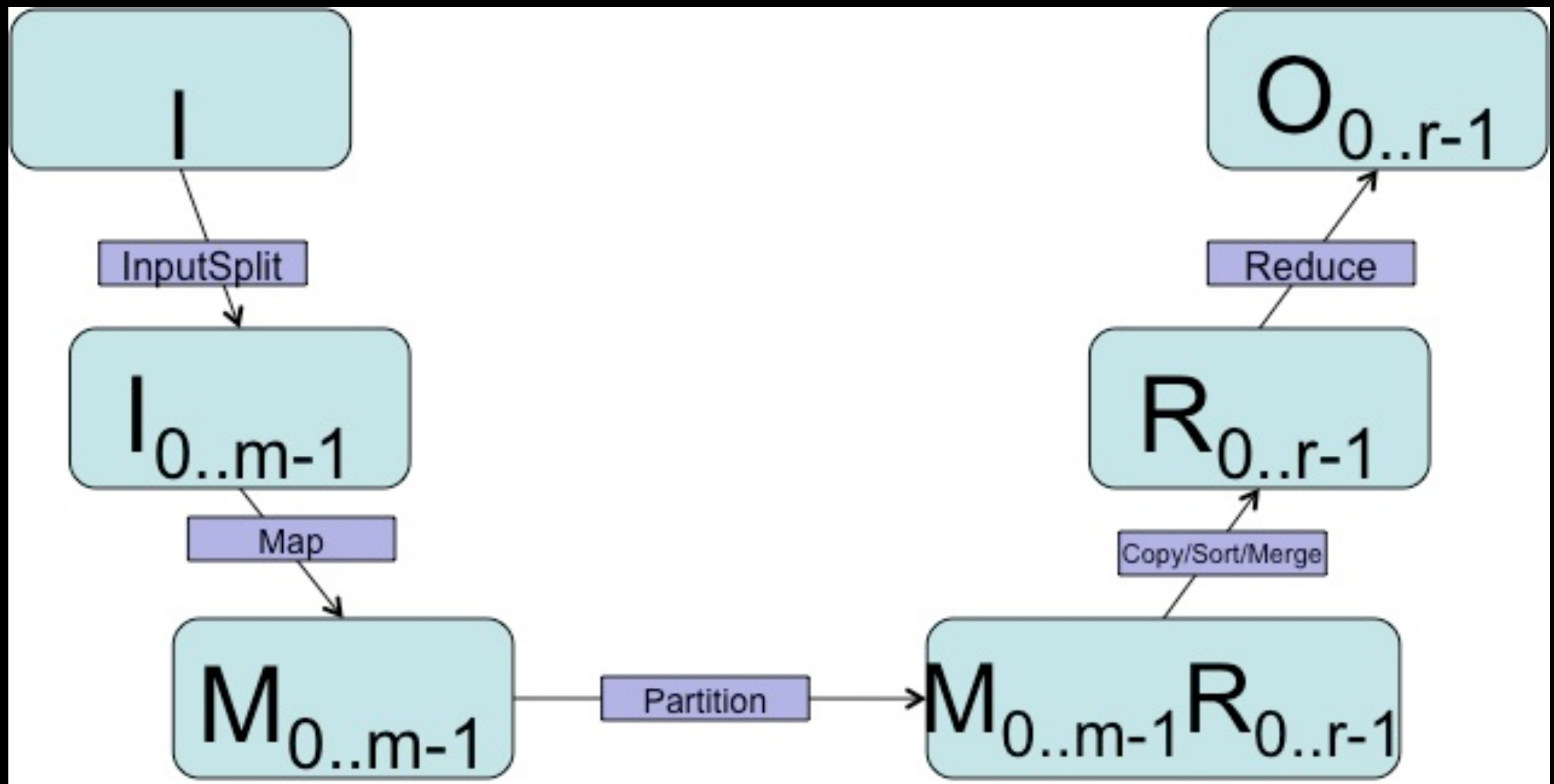
```
mapper (word, frequency):  
    emit (frequency, word)
```

```
reducer (frequency, words):  
    for each word in words:  
        emit (word, frequency)
```





Dataflow



MR Dataflow

Unigrams: Java Mapper

```
public static class MapClass extends MapReduceBase
    implements Mapper
        <LongWritable, Text, Text, IntWritable> {

    public void map(LongWritable key, Text value,
        OutputCollector<Text, IntWritable> output,
        Reporter reporter) throws IOException {
        String line = value.toString();
        StringTokenizer itr = new StringTokenizer(line);
        while (itr.hasMoreTokens()) {
            Text word = new Text(itr.nextToken());
            output.collect(word, new IntWritable(1));
        }
    }
}
```



Unigrams: Java Reducer

```
public static class Reduce extends MapReduceBase
    implements Reducer
        <Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key,
        Iterator<IntWritable> values,
        OutputCollector<Text, IntWritable> output,
        Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```



Unigrams: Driver

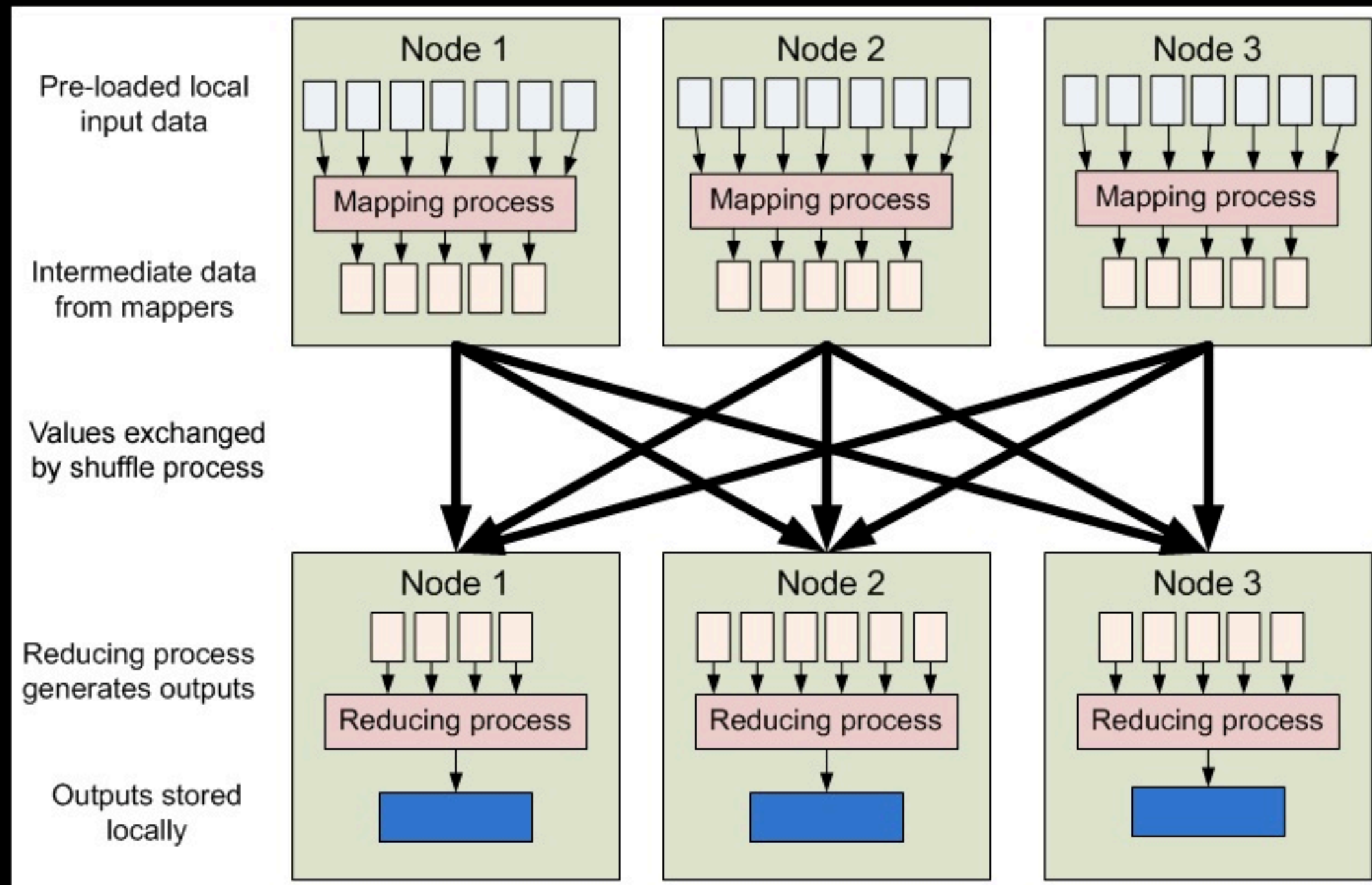
```
public void run(String inputPath, String outputPath)
throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

    conf.setMapperClass(MapClass.class);
    conf.setReducerClass(Reduce.class);

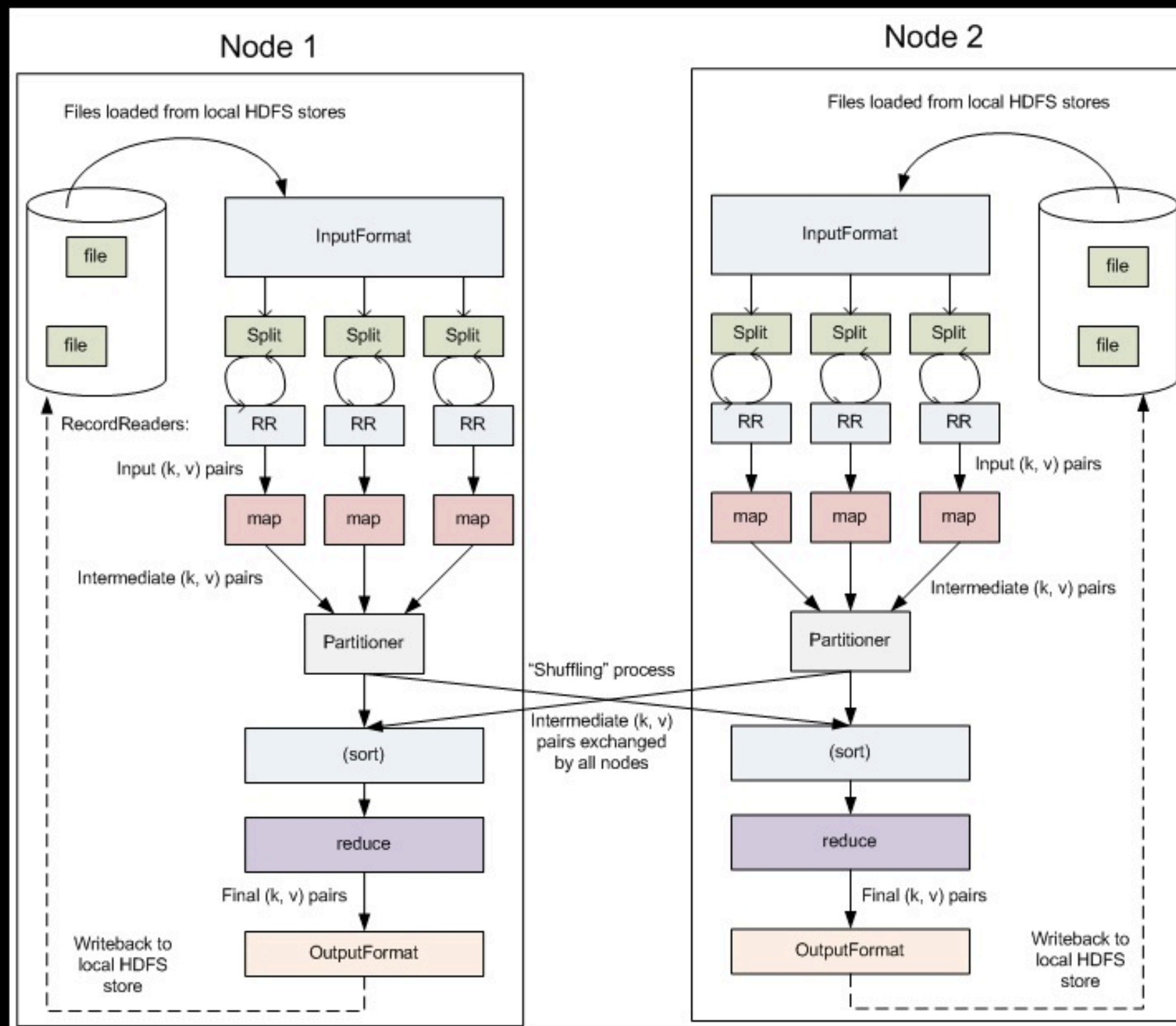
    FileInputFormat.addInputPath(conf,
        new Path(inputPath));
    FileOutputFormat.setOutputPath(conf,
        new Path(outputPath));

    JobClient.runJob(conf);
}
```





MapReduce Pipeline



Pipeline Details

Configuration

- Unified Mechanism for
 - Configuring Daemons
 - Runtime environment for Jobs/Tasks
- Defaults: **-default.xml*
- Site-Specific: **-site.xml*
- *final* parameters



Example

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>head.server.node.com:9001</value>
  </property>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://head.server.node.com:9000</value>
  </property>
  <property>
    <name>mapred.child.java.opts</name>
    <value>-Xmx512m</value>
    <final>true</final>
  </property>
  ....
</configuration>
```



InputFormats

Format	Key Type	Value Type
TextInputFormat (Default)	File Offset	Text Line
KeyValueInputFormat	Text (upto \t)	Remaining Text
SequenceFileInputFormat	User-Defined	User-Defined

OutputFormats

Format	Description
TextOutputFormat (default)	Key \t Value \n
SequenceFileOutputFormat	Binary Serialized keys and values
NullOutputFormat	Discards Output

Hadoop Streaming

- Hadoop is written in Java
 - Java MapReduce code is “native”
- What about Non-Java Programmers ?
 - Perl, Python, Shell, R
 - grep, sed, awk, uniq as Mappers/Reducers
- Text Input and Output



Hadoop Streaming

- Thin Java wrappers for Map & Reduce Tasks
- Forks actual Mapper & Reducer
- IPC via *stdin, stdout, stderr*
- *Key.toString() \t Value.toString() \n*
- Slower than Java programs
 - Allows for quick prototyping / debugging



Hadoop Streaming

```
$ bin/hadoop jar hadoop-streaming.jar \  
    -input in-files -output out-dir \  
    -mapper mapper.sh -reducer reducer.sh
```

```
# mapper.sh
```

```
sed -e 's/ /\n/g' | grep .
```

```
# reducer.sh
```

```
uniq -c | awk '{print $2 "\t" $1}'
```



Hadoop Pipes

- Library for C/C++
- Key & Value are std::string (binary)
- Communication through Unix pipes
- High numerical performance
 - legacy C/C++ code (needs modification)



Pipes Program

```
#include "hadoop/Pipes.hh"
#include "hadoop/TemplateFactory.hh"
#include "hadoop/StringUtils.hh"

int main(int argc, char *argv[]) {
    return
        HadoopPipes::runTask(
            HadoopPipes::TemplateFactory<WordCountMap,
                WordCountReduce>());
}
```



Pipes Mapper

```
class WordCountMap: public HadoopPipes::Mapper {
public:
    WordCountMap(HadoopPipes::TaskContext& context){}
    void map(HadoopPipes::MapContext& context) {
        std::vector<std::string> words =
            HadoopUtils::splitString(
                context.getInputValue(), " ");
        for(unsigned int i=0; i < words.size(); ++i) {
            context.emit(words[i], "1");
        }
    }
};
```



Pipes Reducer

```
class WordCountReduce: public HadoopPipes::Reducer {
public:
    WordCountReduce(HadoopPipes::TaskContext& context){}
    void reduce(HadoopPipes::ReduceContext& context) {
        int sum = 0;
        while (context.nextValue()) {
            sum +=
                HadoopUtils::toInt(context.getInputValue());
        }
        context.emit(context.getInputKey(),
            HadoopUtils::toString(sum));
    }
};
```



Running Pipes

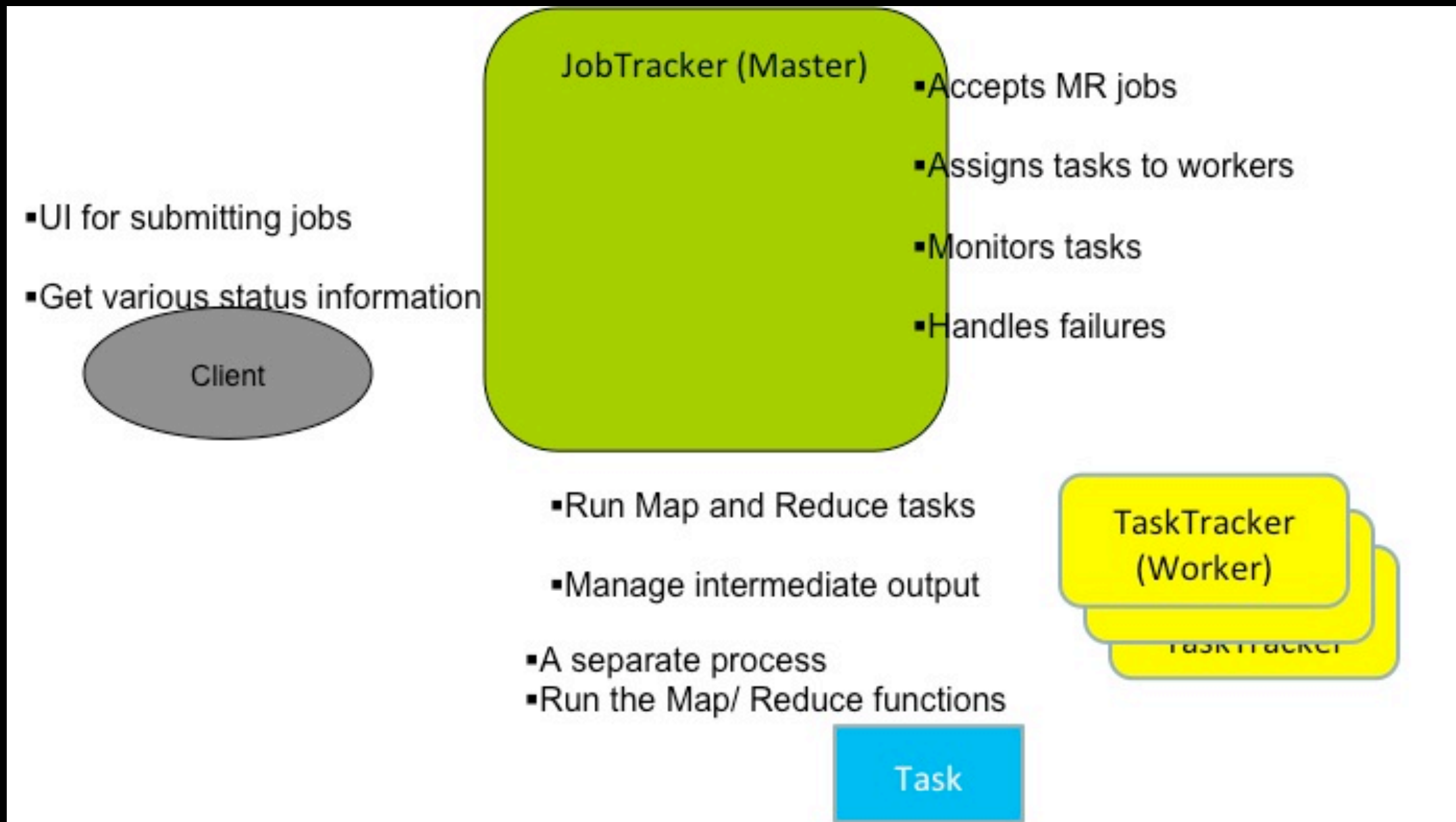
```
# upload executable to HDFS
$ bin/hadoop fs -put wordcount /examples/bin

# Specify configuration
$ vi /tmp/word.xml

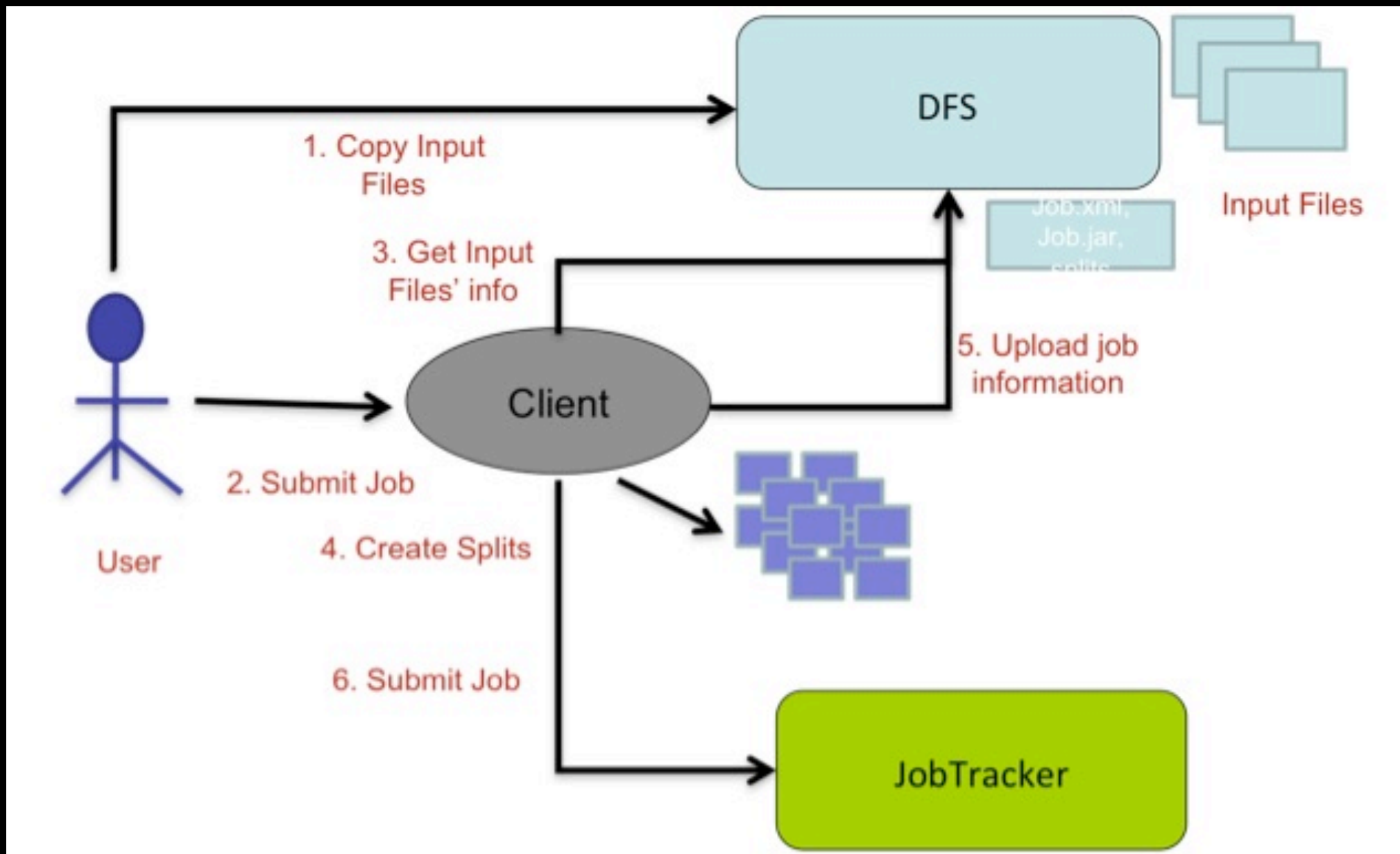
...
// Set the binary path on DFS
<property>
  <name>hadoop.pipes.executable</name>
  <value>/examples/bin/wordcount</value>
</property>

...
# Execute job
# bin/hadoop pipes -conf /tmp/word.xml \
  -input in-dir -output out-dir
```

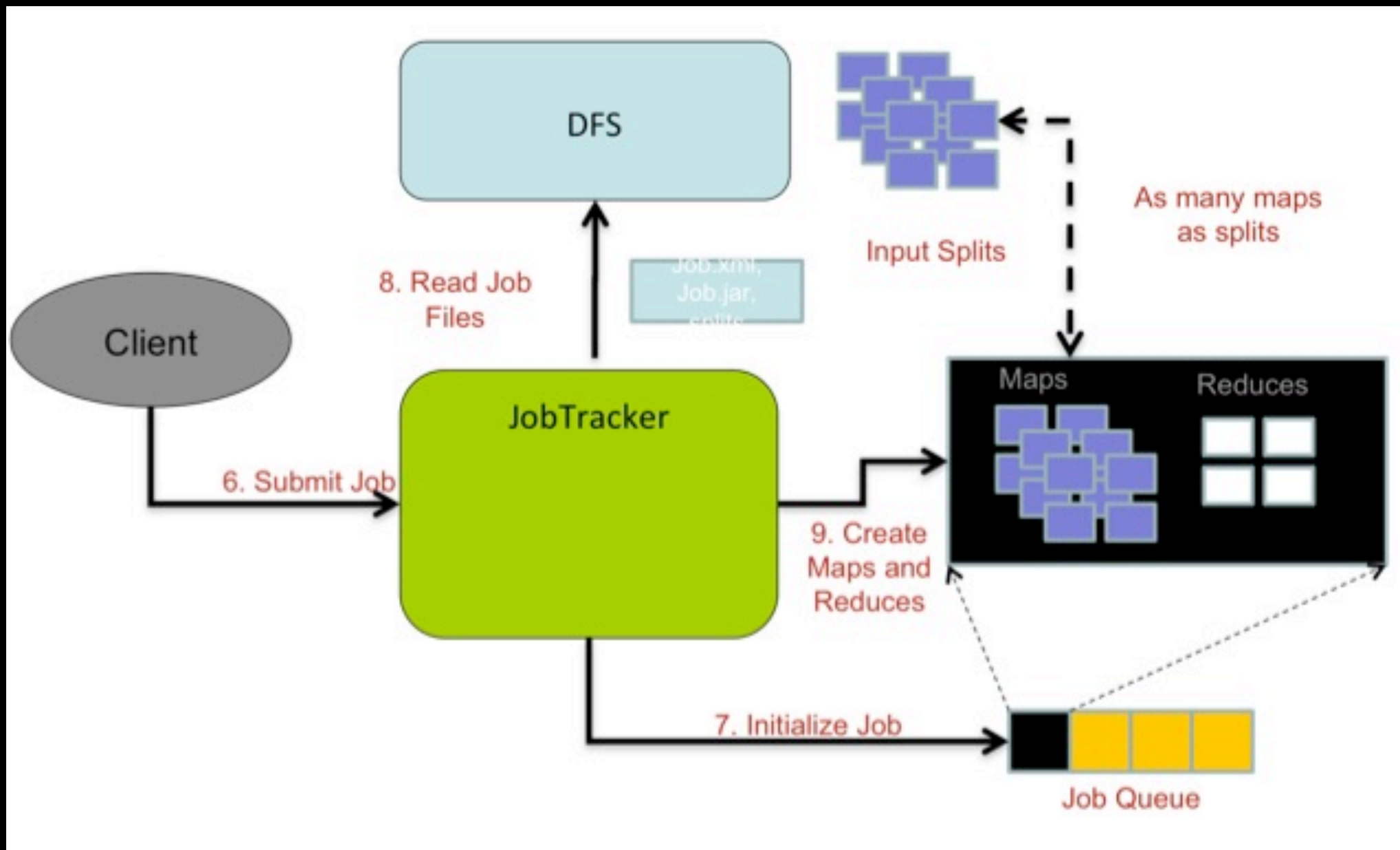




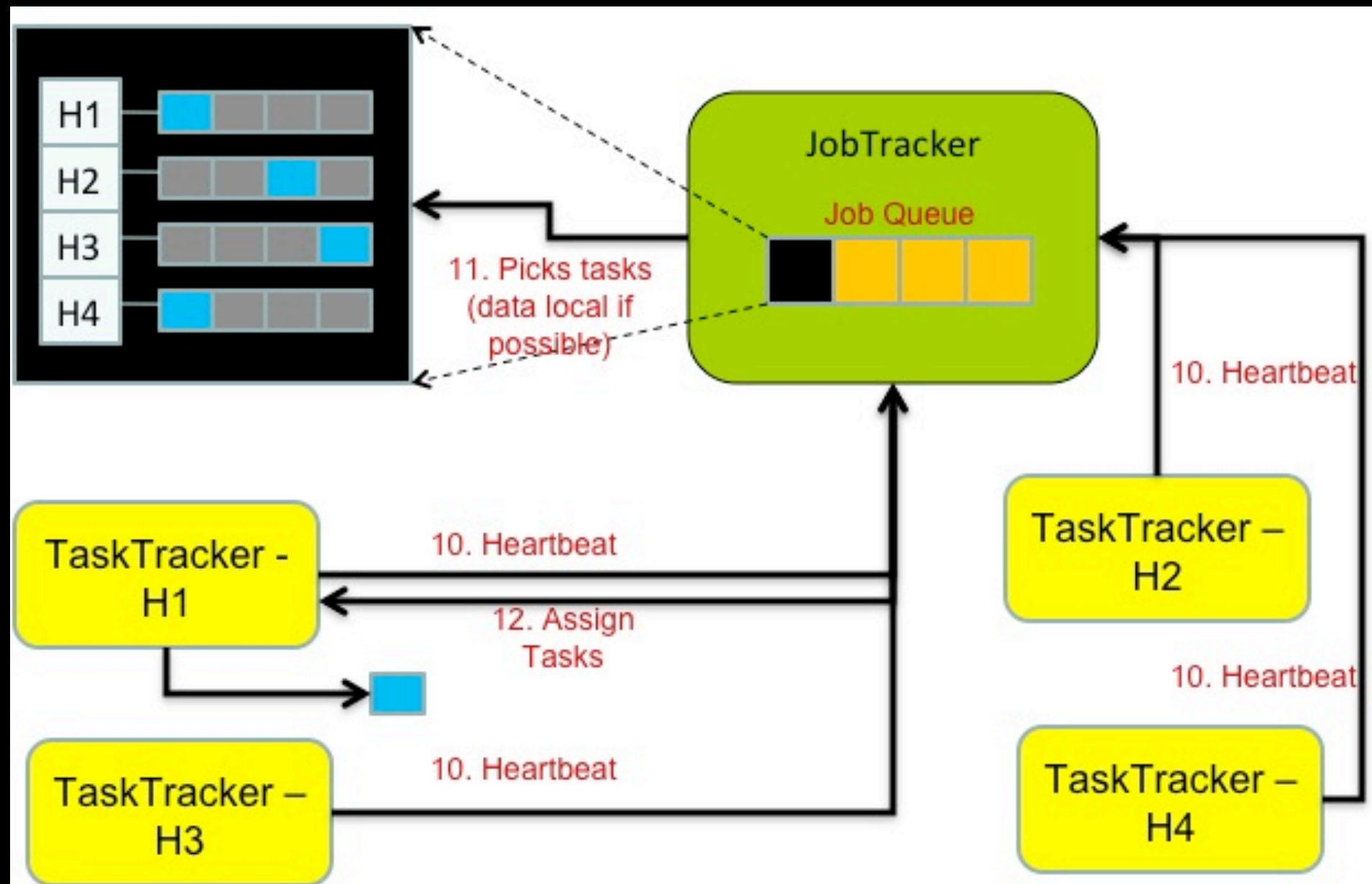
MR Architecture



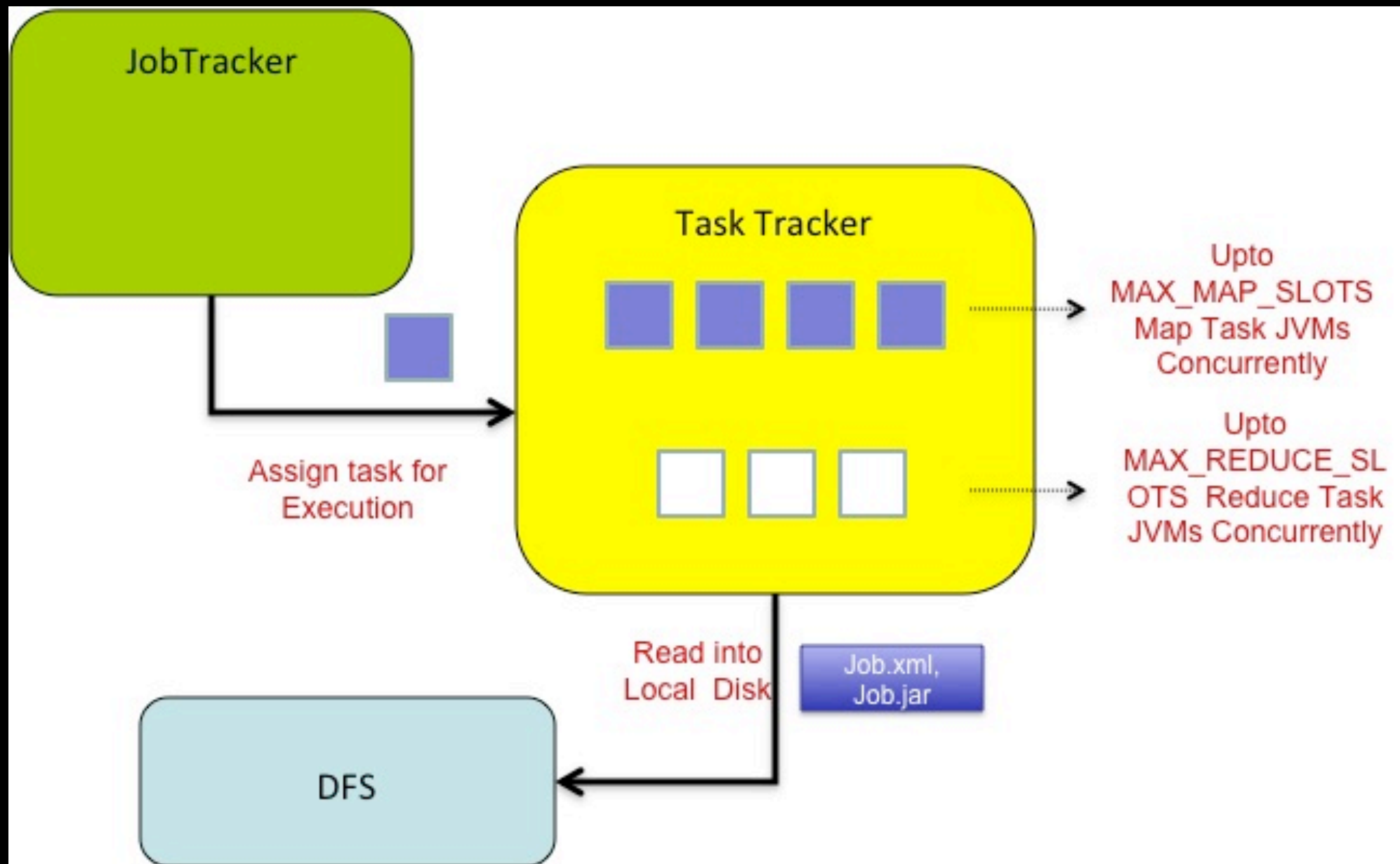
Job Submission



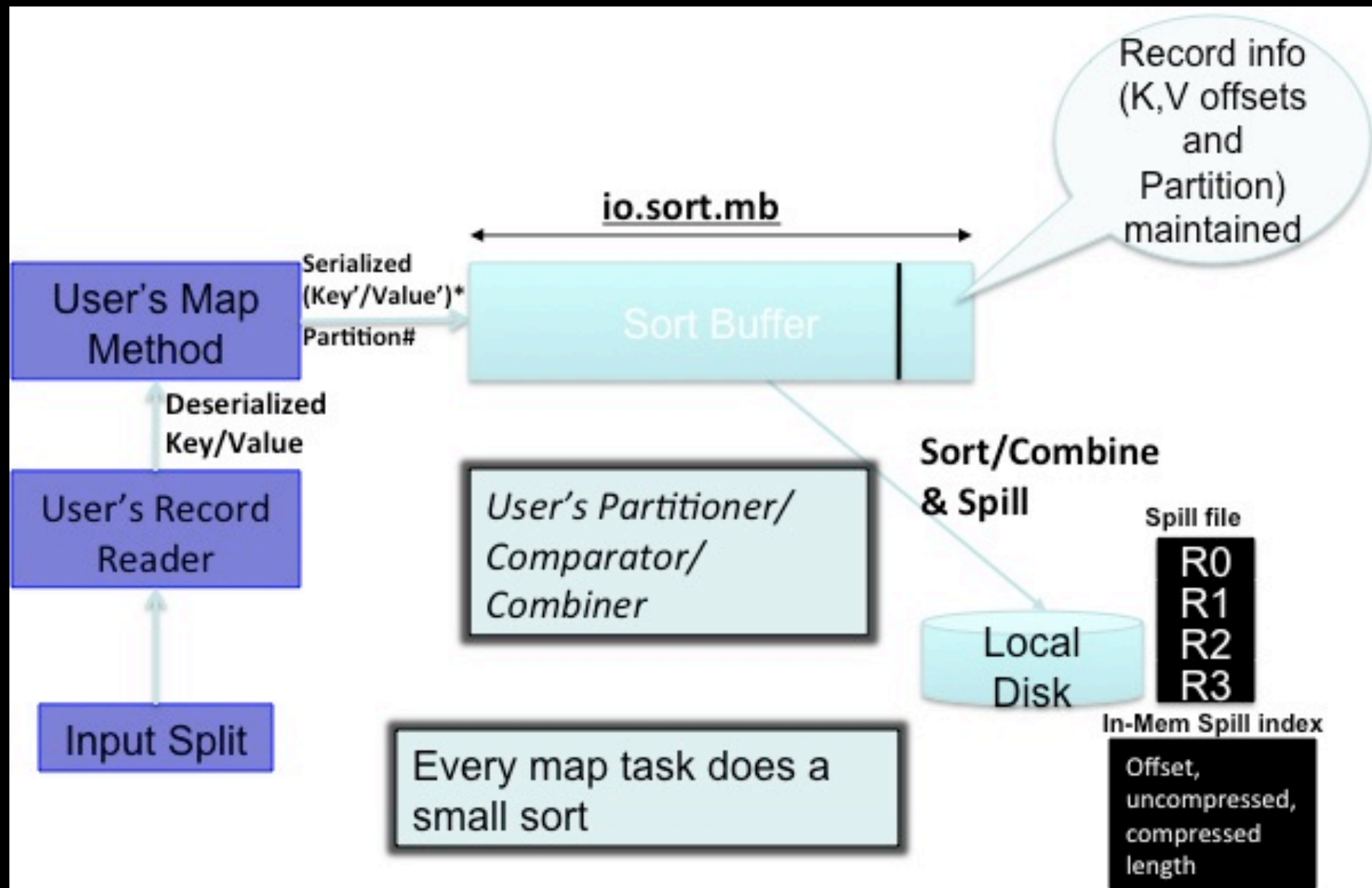
Initialization



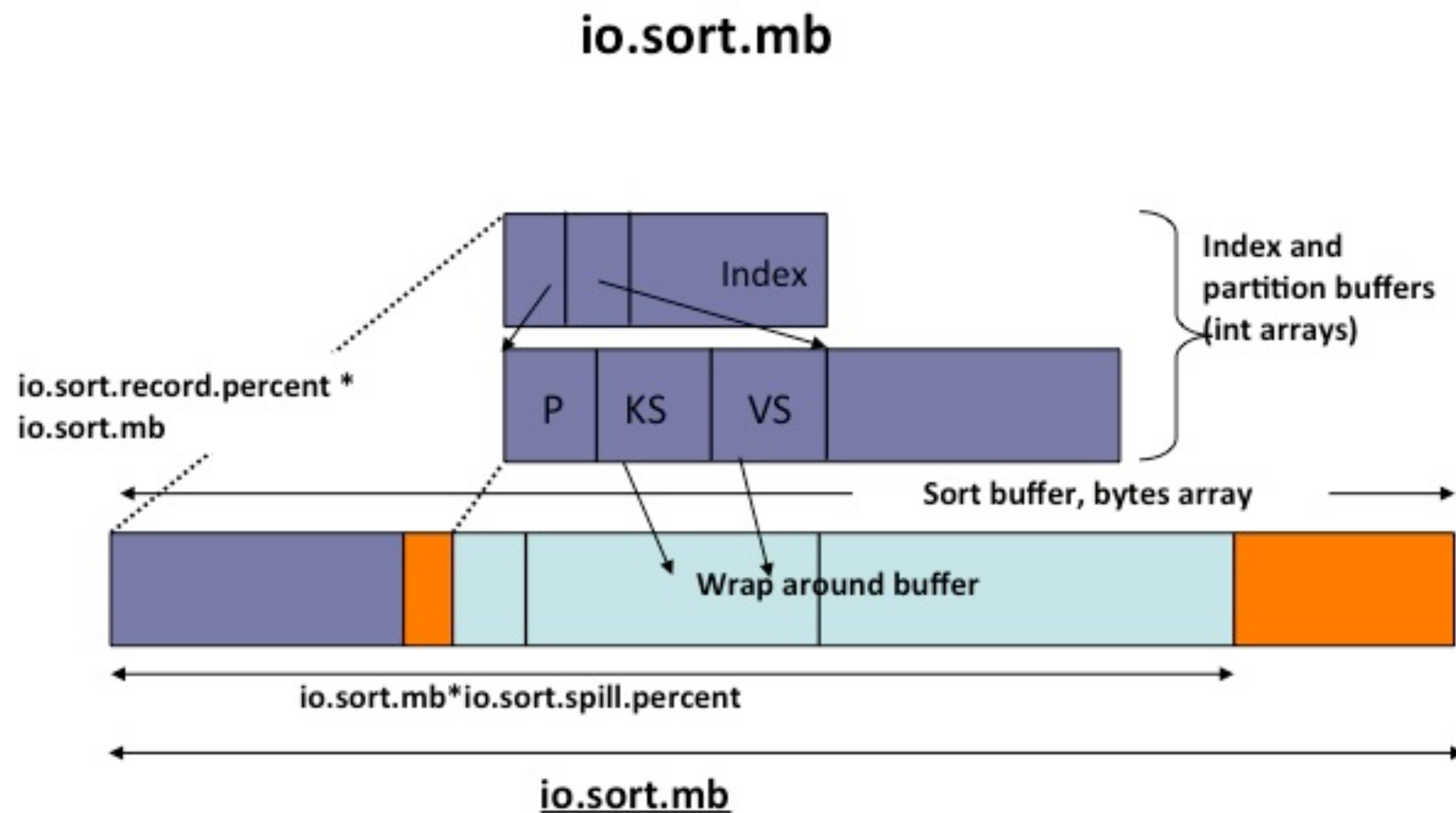
Scheduling



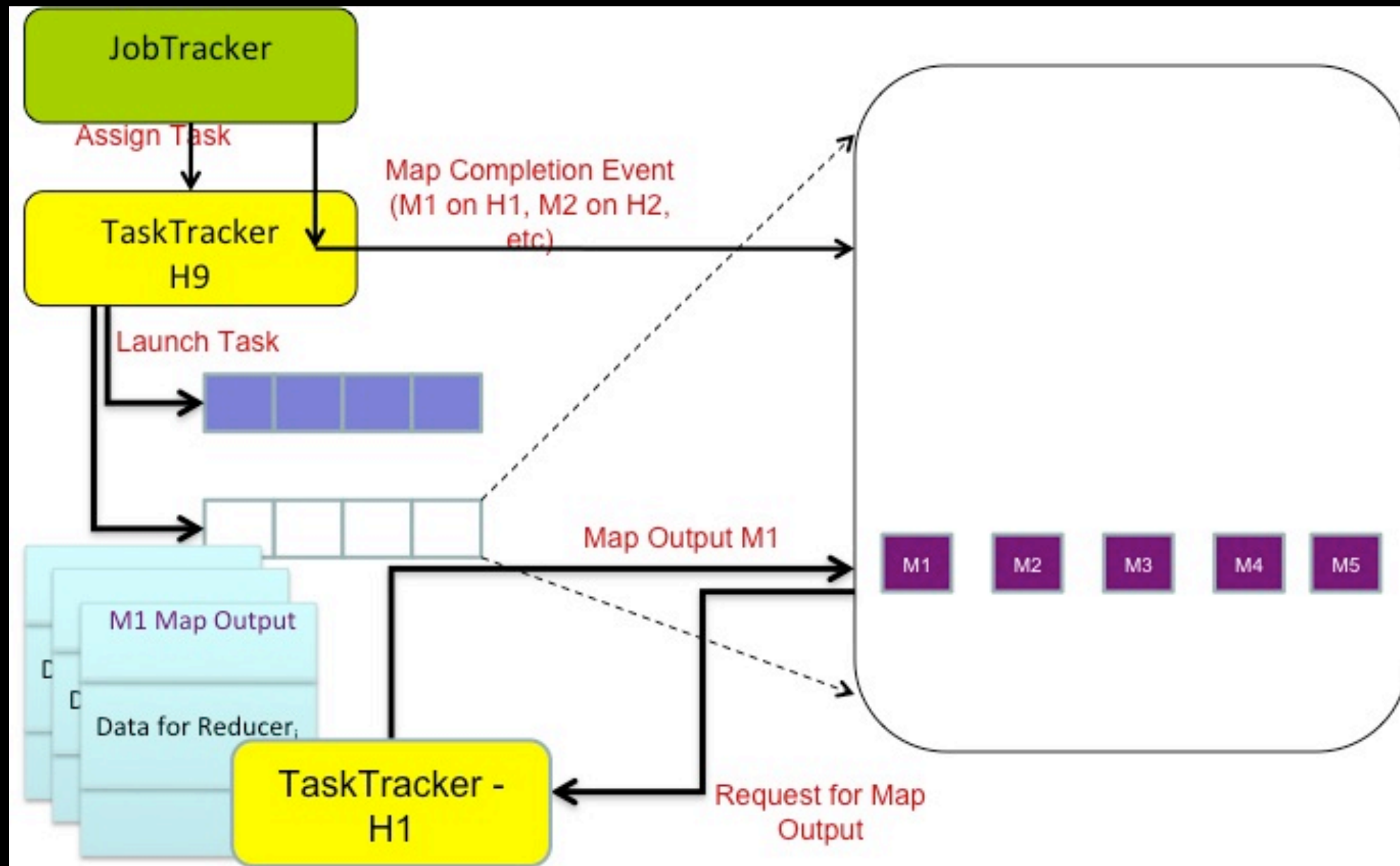
Execution



Map Task



Sort Buffer



Reduce Task



Questions ?

Running Hadoop Jobs



Running a Job

```
[milindb@gateway ~]$ hadoop jar \
$HADOOP_HOME/hadoop-examples.jar wordcount \
/data/newsarchive/20080923 /tmp/newsout
input.FileInputFormat: Total input paths to process : 4
mapred.JobClient: Running job: job_200904270516_5709
mapred.JobClient:  map 0% reduce 0%
mapred.JobClient:  map 3% reduce 0%
mapred.JobClient:  map 7% reduce 0%
....
mapred.JobClient:  map 100% reduce 21%
mapred.JobClient:  map 100% reduce 31%
mapred.JobClient:  map 100% reduce 33%
mapred.JobClient:  map 100% reduce 66%
mapred.JobClient:  map 100% reduce 100%
mapred.JobClient: Job complete: job_200904270516_5709
```



Running a Job

```
mapred.JobClient: Counters: 18
mapred.JobClient:   Job Counters
mapred.JobClient:     Launched reduce tasks=1
mapred.JobClient:     Rack-local map tasks=10
mapred.JobClient:     Launched map tasks=25
mapred.JobClient:     Data-local map tasks=1
mapred.JobClient:   FileSystemCounters
mapred.JobClient:     FILE_BYTES_READ=491145085
mapred.JobClient:     HDFS_BYTES_READ=3068106537
mapred.JobClient:     FILE_BYTES_WRITTEN=724733409
mapred.JobClient:     HDFS_BYTES_WRITTEN=377464307
```



Running a Job

```
mapred.JobClient: Map-Reduce Framework
mapred.JobClient:   Combine output records=73828180
mapred.JobClient:   Map input records=36079096
mapred.JobClient:   Reduce shuffle bytes=233587524
mapred.JobClient:   Spilled Records=78177976
mapred.JobClient:   Map output bytes=4278663275
mapred.JobClient:   Combine input records=371084796
mapred.JobClient:   Map output records=313041519
mapred.JobClient:   Reduce input records=15784903
```



kry-jt1 Hadoop Map/Reduce Administration

State: RUNNING

Started: Mon Apr 27 05:16:52 UTC 2009

Version: 0.20.0-2701599, r

Compiled: Tue Apr 21 18:09:33 UTC 2009 by hadoopqa

Identifier: 200904270516

JobTracker WebUI

Cluster Summary (Heap Size is 16.46 GB/23.14 GB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
5	21	5568	1497	2994	2994	4.00	3

JobTracker Status

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job 200904270516 5719	NORMAL	milindb	word count	0.00% <div></div>	25	0	0.00% <div></div>	1	0	NA

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job 200904270516 5709	NORMAL	milindb	word count	100.00% <div></div>	25	25	100.00% <div></div>	1	1	NA

Jobs Status

Hadoop job_200904270516_5709 on kry-jt1

User: milindb

Job Name: word count

Job File: hdfs://kry-nn1.inktomisearch.com/mapredsystem/hadoop/mapredsystem/job_200904270516_5709/job.xml

Job Setup: [Successful](#)

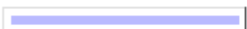
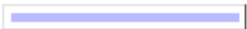
Status: Succeeded

Started at: Sun May 03 21:38:54 UTC 2009

Finished at: Sun May 03 21:42:52 UTC 2009

Finished in: 3mins, 57sec

Job Cleanup: [Successful](#)

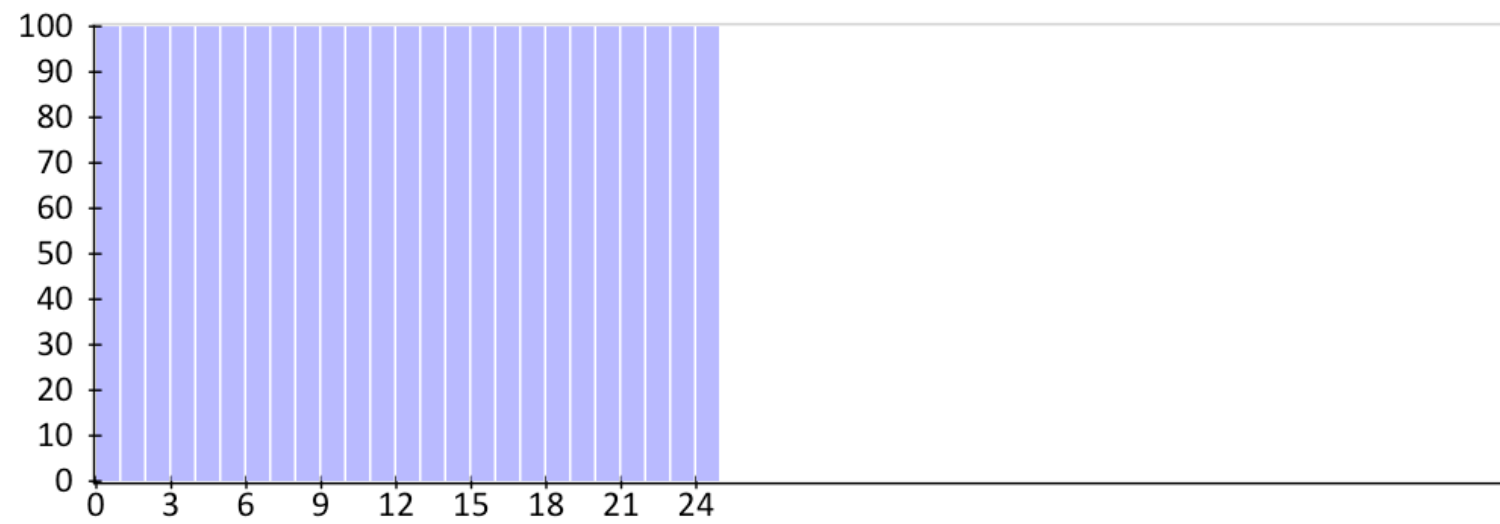
Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00% 	25	0	0	25	0	0 / 0
reduce	100.00% 	1	0	0	1	0	0 / 0

Job Details

	Counter	Map	Reduce	Total
Job Counters	Launched reduce tasks	0	0	1
	Rack-local map tasks	0	0	10
	Launched map tasks	0	0	25
	Data-local map tasks	0	0	1
FileSystemCounters	FILE_BYTES_READ	342,134,961	149,010,124	491,145,085
	HDFS_BYTES_READ	3,068,106,537	0	3,068,106,537
	FILE_BYTES_WRITTEN	575,723,285	149,010,124	724,733,409
	HDFS_BYTES_WRITTEN	0	377,464,307	377,464,307
Map-Reduce Framework	Reduce input groups	0	0	0
	Combine output records	62,393,073	11,435,107	73,828,180
	Map input records	36,079,096	0	36,079,096
	Reduce shuffle bytes	0	233,587,524	233,587,524
	Reduce output records	0	0	0
	Spilled Records	62,393,073	15,784,903	78,177,976
	Map output bytes	4,278,663,275	0	4,278,663,275
	Map output records	313,041,519	0	313,041,519
	Combine input records	350,530,796	20,554,000	371,084,796
	Reduce input records	0	15,784,903	15,784,903

Job Counters

Map Completion Graph - [close](#)



Reduce Completion Graph - [close](#)



Job Progress

All Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_200904270516_5709_m_000000	100.00% <div></div>		3-May-2009 21:39:21	3-May-2009 21:41:06 (1mins, 45sec)		9
task_200904270516_5709_m_000001	100.00% <div></div>		3-May-2009 21:39:21	3-May-2009 21:40:52 (1mins, 30sec)		9
task_200904270516_5709_m_000002	100.00% <div></div>		3-May-2009 21:39:21	3-May-2009 21:40:51 (1mins, 30sec)		9
task_200904270516_5709_m_000003	100.00% <div></div>		3-May-2009 21:39:21	3-May-2009 21:40:51 (1mins, 30sec)		9
task_200904270516_5709_m_000004	100.00% <div></div>		3-May-2009 21:39:21	3-May-2009 21:40:51 (1mins, 30sec)		9
task_200904270516_5709_m_000005	100.00% <div></div>		3-May-2009 21:39:20	3-May-2009 21:40:51 (1mins, 30sec)		9
task_200904270516_5709_m_000006	100.00% <div></div>		3-May-2009 21:39:21	3-May-2009 21:40:51 (1mins, 30sec)		9
task_200904270516_5709_m_000007	100.00% <div></div>		3-May-2009 21:39:20	3-May-2009 21:40:51 (1mins, 30sec)		9
task_200904270516_5709_m_000008	100.00% <div></div>		3-May-2009 21:39:21	3-May-2009 21:41:07 (1mins, 46sec)		9
task_200904270516_5709_m_000009	100.00% <div></div>		3-May-2009 21:39:21	3-May-2009 21:40:51 (1mins, 30sec)		9
task_200904270516_5709_m_000010	100.00% <div></div>		3-May-2009 21:39:21	3-May-2009 21:40:51 (1mins, 30sec)		9
task_200904270516_5709_m_000011	100.00% <div></div>		3-May-2009 21:39:20	3-May-2009 21:40:51 (1mins, 30sec)		9

All Tasks

All Task Attempts

Task Attempts	Machine	Status	Progress	Start Time	Finish Time	Errors	Task Logs	Counters	Actions
attempt_200904270516_5709_m_000000_0	/72.30.127.192 /kry1540.inktomisearch.com	SUCCEEDED	100.00% <div></div>	3-May-2009 21:39:20	3-May-2009 21:40:54 (1mins, 34sec)		Last 4KB Last 8KB All	9	

Input Split Locations

/72.30.127.192/kry1545.inktomisearch.com
/72.30.62.192/kry50414.inktomisearch.com
/72.30.62.192/kry50413.inktomisearch.com

Task Details

Counters for attempt_200904270516_5709_m_000000_0

FileSystemCounters

FILE_BYTES_READ	15,201,458
HDFS_BYTES_READ	143,611,125
FILE_BYTES_WRITTEN	25,452,741

Map-Reduce Framework

Combine output records	2,755,686
Map input records	1,717,449
Spilled Records	2,755,686
Map output bytes	200,041,337
Combine input records	16,226,718
Map output records	14,560,207

Task Counters

Task Logs: 'attempt_200904270516_5709_m_000000_0'

stdout logs

stderr logs

syslog logs

```
ed-local/taskTracker/jobcache/job_200904270516_5709/attempt_200904270516_5709_m_000000_0/job.xml:a attempt to override final parameter
2009-05-03 21:39:23,594 WARN org.apache.hadoop.conf.Configuration: /grid/3/tmp/mapred-local/taskTracker/jobcache/job_200904270516_5709
2009-05-03 21:39:23,595 WARN org.apache.hadoop.conf.Configuration: /grid/3/tmp/mapred-local/taskTracker/jobcache/job_200904270516_5709
2009-05-03 21:39:23,663 INFO org.apache.hadoop.mapred.MapTask: io.sort.mb = 256
2009-05-03 21:39:24,131 INFO org.apache.hadoop.mapred.MapTask: data buffer = 204010944/214748368
2009-05-03 21:39:24,131 INFO org.apache.hadoop.mapred.MapTask: record buffer = 3187670/3355443
2009-05-03 21:39:24,138 INFO org.apache.hadoop.util.NativeCodeLoader: Loaded the native-hadoop library
2009-05-03 21:39:24,140 INFO org.apache.hadoop.io.compress.LzoCodec: Successfully loaded & initialized native-lzo library
2009-05-03 21:39:32,030 INFO org.apache.hadoop.mapred.MapTask: Spilling map output: record full = true
2009-05-03 21:39:32,030 INFO org.apache.hadoop.mapred.MapTask: bufstart = 0; bufend = 44050661; bufvoid = 214748368
2009-05-03 21:39:32,030 INFO org.apache.hadoop.mapred.MapTask: kvstart = 0; kvend = 3187670; length = 3355443
2009-05-03 21:39:40,615 INFO org.apache.hadoop.io.compress.CodecPool: Got brand-new compressor
2009-05-03 21:39:43,963 INFO org.apache.hadoop.mapred.MapTask: Finished spill 0
2009-05-03 21:39:50,580 INFO org.apache.hadoop.mapred.MapTask: Spilling map output: record full = true
2009-05-03 21:39:50,580 INFO org.apache.hadoop.mapred.MapTask: bufstart = 44050661; bufend = 86154575; bufvoid = 214748368
2009-05-03 21:39:50,580 INFO org.apache.hadoop.mapred.MapTask: kvstart = 3187670; kvend = 3019896; length = 3355443
2009-05-03 21:40:02,417 INFO org.apache.hadoop.mapred.MapTask: Finished spill 1
```

Task Logs

Debugging

- Run job with the Local Runner
 - Set *mapred.job.tracker* to “local”
 - Runs application in a single thread
- Run job on a small data set on a 1 node cluster



Debugging

- Set *keep.failed.task.files* to keep files from failed tasks
- Use the IsolationRunner to run just the failed task
- Java Debugging hints
 - Send a kill -QUIT to the Java process to get the call stack, locks held, deadlocks



Hadoop Performance Tuning



Example

- “Bob” wants to count records in AdServer logs (several hundred GB)
- Used Identity Mapper & Single counting reducer
- What is he doing wrong ?
- This happened, really !



MapReduce Performance

- Reduce intermediate data size
 - map outputs + reduce inputs
- Maximize map input transfer rate
- Pipelined writes from reduce
- Opportunity to load balance



Shuffle

- Often the most expensive component
- $M * R$ Transfers over the network
- Sort map outputs (intermediate data)
- Merge reduce inputs



Improving Shuffle

- Avoid shuffling/sorting if possible
- Minimize redundant transfers
- Compress intermediate data



Avoid Shuffle

- Set *mapred.reduce.tasks* to zero
 - Known as map-only computations
 - Filters, Projections, Transformations
- Number of output files = number of input splits = number of input blocks
- May overwhelm namenode



Minimize Redundant Transfers

- Combiners
- Intermediate data compression



Combiners

- When Maps produce many repeated keys
- Combiner: Local aggregation after Map & before Reduce
- Side-effect free
- Same interface as Reducers, and often the same class



Compression

- Often yields huge performance gains
- Set *mapred.output.compress* to true to compress job output
- Set *mapred.compress.map.output* to true to compress map outputs
- Codecs: Java zlib (default), LZO, bzip2, native gzip



Load Imbalance

- Inherent in application
 - Imbalance in input splits
 - Imbalance in computations
 - Imbalance in partitions
- Heterogenous hardware
 - Degradation over time



Optimal Number of Nodes

- T_m = Map slots per TaskTracker
- N = optimal number of nodes
- $S_m = N * T_m$ = Total Map slots in cluster
- M = Map tasks in application
- Rule of thumb: $5 * S_m < M < 10 * S_m$



Configuring Task Slots

- *mapred.tasktracker.map.tasks.maximum*
- *mapred.tasktracker.reduce.tasks.maximum*
- Tradeoffs: Number of cores, RAM, number and size of disks
- Also consider resources consumed by TaskTracker & DataNode



Speculative Execution

- Runs multiple instances of slow tasks
- Instance that finishes first, succeeds
- *mapred.map.speculative.execution=true*
- *mapred.reduce.speculative.execution=true*
- Can dramatically bring in long tails on jobs



Hadoop Examples



Example: Standard Deviation

- Takeaway: Changing algorithm to suit architecture yields the best implementation

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2}$$

Implementation I

- Two Map-Reduce stages
- First stage computes Mean
- Second stage computes standard deviation



Stage 1: Compute Mean

- Map Input (x_i for $i = 1 \dots N_m$)
- Map Output ($N_m, \text{Mean}(x_{1..N_m})$)
- Single Reducer
- Reduce Input ($\text{Group}(\text{Map Output})$)
- Reduce Output ($\text{Mean}(x_{1..N})$)



Stage 2: Compute Standard Deviation

- Map Input (x_i for $i = 1 \dots N_m$) & $\text{Mean}(x_{1..N})$
- Map Output $(\text{Sum}(x_i - \text{Mean}(x))^2 \text{ for } i = 1 \dots N_m$
- Single Reducer
- Reduce Input (Group (Map Output)) & N
- Reduce Output (σ)



Standard Deviation

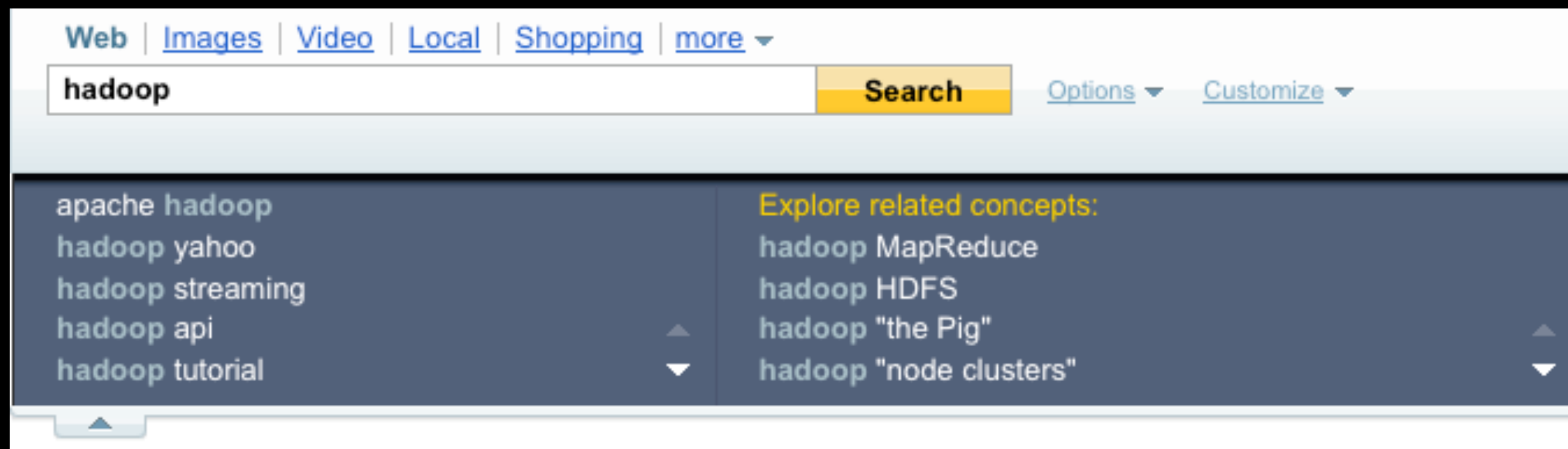
- Algebraically equivalent
- Be careful about numerical accuracy, though

$$\sigma = \sqrt{\frac{1}{N} \left(\sum_{i=1}^N x_i^2 - N \bar{x}^2 \right)}$$

Implementation 2

- Map Input (x_i for $i = 1 \dots N_m$)
- Map Output (N_m ,
[$\text{Sum}(x^2_{1..N_m}), \text{Mean}(x_{1..N_m})$])
- Single Reducer
- Reduce Input (Group (Map Output))
- Reduce Output (σ)





NGrams



Bigrams

- Input: A large text corpus
- Output: $\text{List}(\text{word}_1, \text{Top}_K(\text{word}_2))$
- Two Stages:
 - Generate all possible bigrams
 - Find most frequent K bigrams for each word



Bigrams: Stage I

Map

- Generate all possible Bigrams
- Map Input: Large text corpus
- Map computation
 - In each sentence, or each “word₁ word₂”
 - Output (word₁, word₂), (word₂, word₁)
- Partition & Sort by (word₁, word₂)



pairs.pl

```
while(<STDIN>) {  
    chomp;  
    $_ =~ s/^[^a-zA-Z]+//g ;  
    $_ =~ s/^\s+//g ;  
    $_ =~ s/\s+$//g ;  
    $_ =~ tr/A-Z/a-z/;  
    my @words = split(/\s+/, $_);  
    for (my $i = 0; $i < $#words - 1; ++$i) {  
        print "$words[$i]:$words[$i+1]\n";  
        print "$words[$i+1]:$words[$i]\n";  
    }  
}
```



Bigrams: Stage I

Reduce

- Input: List(word₁, word₂) sorted and partitioned
- Output: List(word₁, [freq, word₂])
- Counting similar to Unigrams example



count.pl

```
$_ = <STDIN>; chomp;  
my ($pw1, $pw2) = split(/:/, $_);  
$count = 1;  
while(<STDIN>) {  
    chomp;  
    my ($w1, $w2) = split(/:/, $_);  
    if ($w1 eq $pw1 && $w2 eq $pw2) {  
        $count++;  
    } else {  
        print "$pw1:$count:$pw2\n";  
        $pw1 = $w1;  
        $pw2 = $w2;  
        $count = 1;  
    }  
}  
print "$pw1:$count:$pw2\n";
```



Bigrams: Stage 2

Map

- Input: List(word₁, [freq, word₂])
- Output: List(word₁, [freq, word₂])
- Identity Mapper (/bin/cat)
- Partition by word₁
- Sort descending by (word₁, freq)



Bigrams: Stage 2

Reduce

- Input: $\text{List}(\text{word}_1, [\text{freq}, \text{word}_2])$
 - partitioned by word_1
 - sorted descending by $(\text{word}_1, \text{freq})$
- Output: $\text{Top}_K(\text{List}(\text{word}_1, [\text{freq}, \text{word}_2]))$
- For each word, throw away after K records



firstN.pl

```
$N = 5;
$_ = <STDIN>; chomp;
my ($pw1, $count, $pw2) = split(/:/, $_);
$idx = 1;
$out = "$pw1\t$pw2,$count;";
while(<STDIN>) {
    chomp;
    my ($w1, $c, $w2) = split(/:/, $_);
    if ($w1 eq $pw1) {
        if ($idx < $N) {
            $out .= "$w2,$c;";
            $idx++;
        }
    } else {
        print "$out\n";
        $pw1 = $w1;
        $idx = 1;
        $out = "$pw1\t$w2,$c;";
    }
}
print "$out\n";
```



Partitioner

- By default, evenly distributes keys
 - $\text{hashCode}(\text{key}) \% \text{NumReducers}$
- Overriding partitioner
 - Skew in map-outputs
 - Restrictions on reduce outputs
 - All URLs in a domain together



Partitioner

```
// JobConf.setPartitionerClass(className)
```

```
public interface Partitioner <K, V>  
    extends JobConfigurable {
```

```
    int getPartition(K key, V value, int maxPartitions);
```

```
}
```



Fully Sorted Output

- By contract, reducer gets input sorted on key
- Typically reducer output order is the same as input order
 - Each output file (part file) is sorted
- How to make sure that Keys in part i are all less than keys in part $i+1$?



Fully Sorted Output

- Use single reducer for small output
- Insight: Reducer input must be fully sorted
- Partitioner should provide fully sorted reduce input
- Sampling + Histogram equalization



Number of Maps

- Number of Input Splits
 - Number of HDFS blocks
- *mapred.map.tasks*
- Minimum Split Size (*mapred.min.split.size*)
- $split_size = \max(\min(hdfs_block_size, data_size/\#maps), min_split_size)$



Parameter Sweeps

- External program processes data based on command-line parameters
- *`./prog -params="0.1,0.3" < in.dat > out.dat`*
- Objective: Run an instance of *`./prog`* for each parameter combination
- Number of Mappers = Number of different parameter combinations



Parameter Sweeps

- Input File: *params.txt*
 - Each line contains one combination of parameters
- Input format is *NLineInputFormat* (N=1)
- Number of maps = Number of splits = Number of lines in *params.txt*



Auxiliary Files

- *-file auxFile.dat*
- Job submitter adds file to *job.jar*
- Unjarred on the task tracker
- Available to task as *\$cwd/auxFile.dat*
- Not suitable for large / frequently used files



Auxiliary Files

- Tasks need to access “side” files
 - Read-only Dictionaries (such as for porn filtering)
 - Dynamically linked libraries
- Tasks themselves can fetch files from HDFS
 - Not Always ! (Hint: Unresolved symbols)



Distributed Cache

- Specify “side” files via *–cacheFile*
- If lot of such files needed
 - Create a *tar.gz* archive
 - Upload to HDFS
 - Specify via *–cacheArchive*



Distributed Cache

- TaskTracker downloads these files “once”
- Untars archives
- Accessible in task’s *\$cwd* before task starts
- Cached across multiple tasks
- Cleaned up upon exit



Joining Multiple Datasets

- Datasets are streams of key-value pairs
- Could be split across multiple files in a single directory
- Join could be on Key, or any field in Value
- Join could be inner, outer, left outer, cross product etc
- **Join is a natural Reduce operation**



Example

- $A = (\text{id}, \text{name}), B = (\text{name}, \text{address})$
- A is in */path/to/A/part-**
- B is in */path/to/B/part-**
- *Select A.name, B.address where A.name == B.name*



Map in Join

- Input: $(Key_1, Value_1)$ from A or B
 - *map.input.file* indicates A or B
 - *MAP_INPUT_FILE* in Streaming
- Output: $(Key_2, [Value_2, A|B])$
 - Key_2 is the Join Key



Reduce in Join

- Input: Groups of $[Value_2, A|B]$ for each Key_2
- Operation depends on which kind of join
 - Inner join checks if key has values from both A & B
- Output: $(Key_2, JoinFunction(Value_2, \dots))$



MR Join Performance

- Map Input = Total of A & B
- Map output = Total of A & B
- Shuffle & Sort
- Reduce input = Total of A & B
- Reduce output = Size of Joined dataset
- Filter and Project in Map



Join Special Cases

- Fragment-Replicate
 - 100GB dataset with 100 MB dataset
- Equipartitioned Datasets
 - Identically Keyed
 - Equal Number of partitions
 - Each partition locally sorted



Fragment-Replicate

- Fragment larger dataset
 - Specify as Map input
- Replicate smaller dataset
 - Use Distributed Cache
- Map-Only computation
 - No shuffle / sort



Equipartitioned Join

- Available since Hadoop 0.16
- Datasets joined “before” input to mappers
- Input format: *CompositeInputFormat*
- *mapred.join.expr*
- Simpler to use in Java, but can be used in Streaming



Example

```
mapred.join.expr =  
  inner (  
    tbl (  
      ....SequenceFileInputFormat.class,  
      "hdfs://namenode:8020/path/to/data/A"  
    ),  
    tbl (  
      ....SequenceFileInputFormat.class,  
      "hdfs://namenode:8020/path/to/data/B"  
    )  
  )  
)
```





Questions ?



Apache Pig

What is Pig?

- System for processing large semi-structured data sets using Hadoop MapReduce platform
- Pig Latin: High-level procedural language
- Pig Engine: Parser, Optimizer and distributed query execution



Pig vs SQL

- Pig is procedural
- Nested relational data model
- Schema is optional
- Scan-centric analytic workloads
- Limited query optimization
- SQL is declarative
- Flat relational data model
- Schema is required
- OLTP + OLAP workloads
- Significant opportunity for query optimization

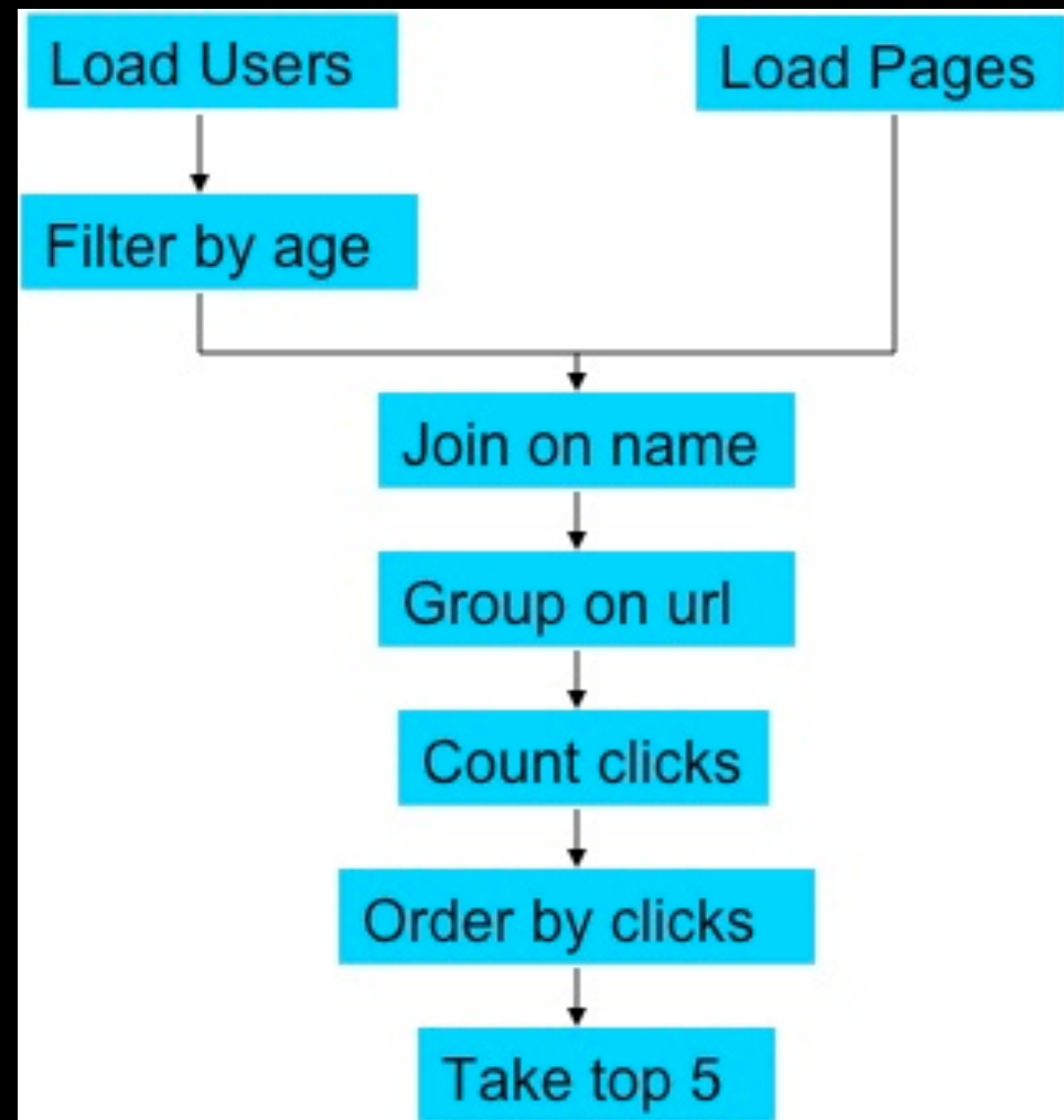
Pig vs Hadoop

- Increases programmer productivity
- Decreases duplication of effort
- Insulates against Hadoop complexity
 - Version Upgrades
 - *JobConf* configuration tuning
 - Job Chains



Example

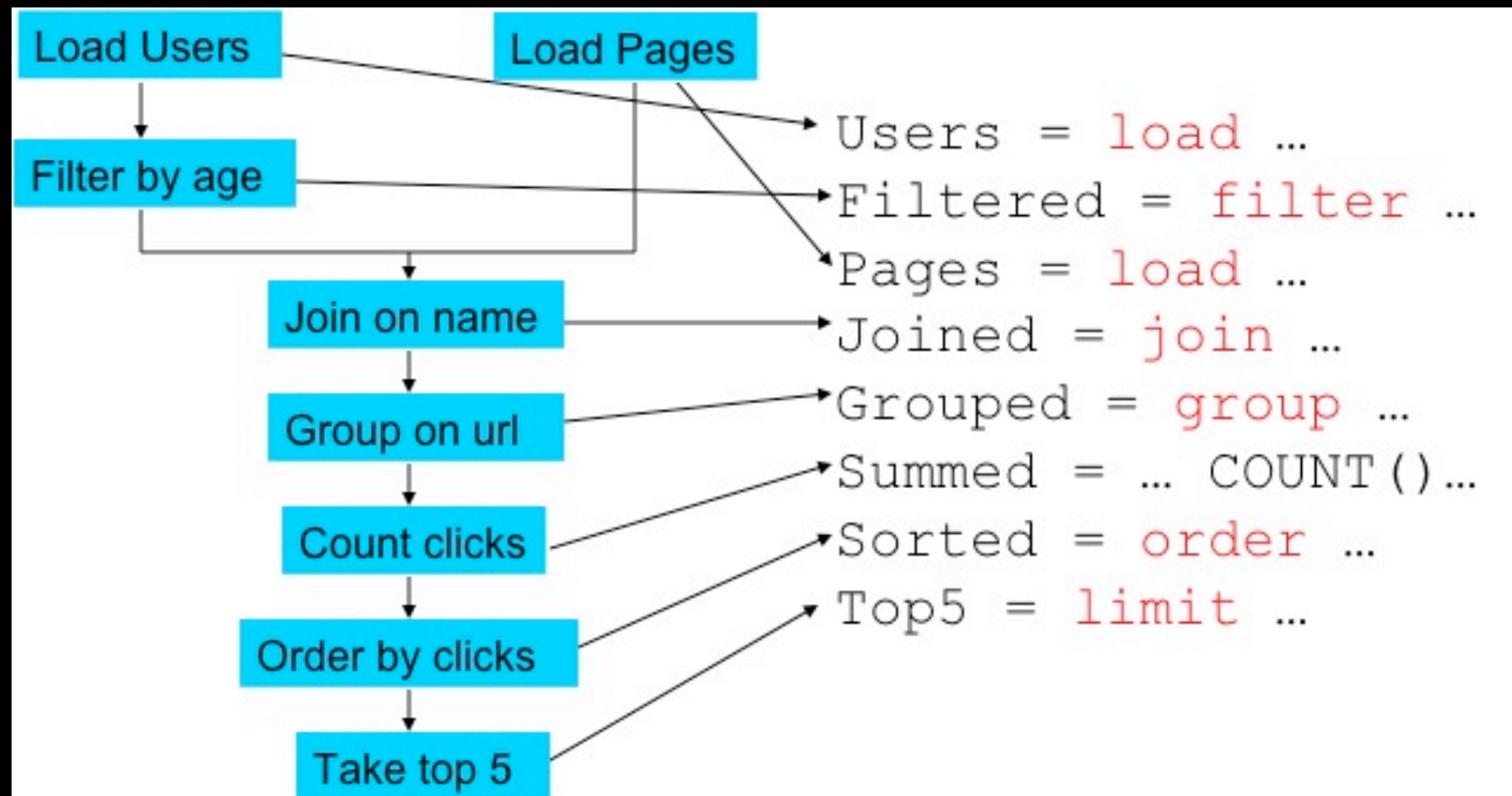
- Input: User profiles, Page visits
- Find the top 5 most visited pages by users aged 18-25



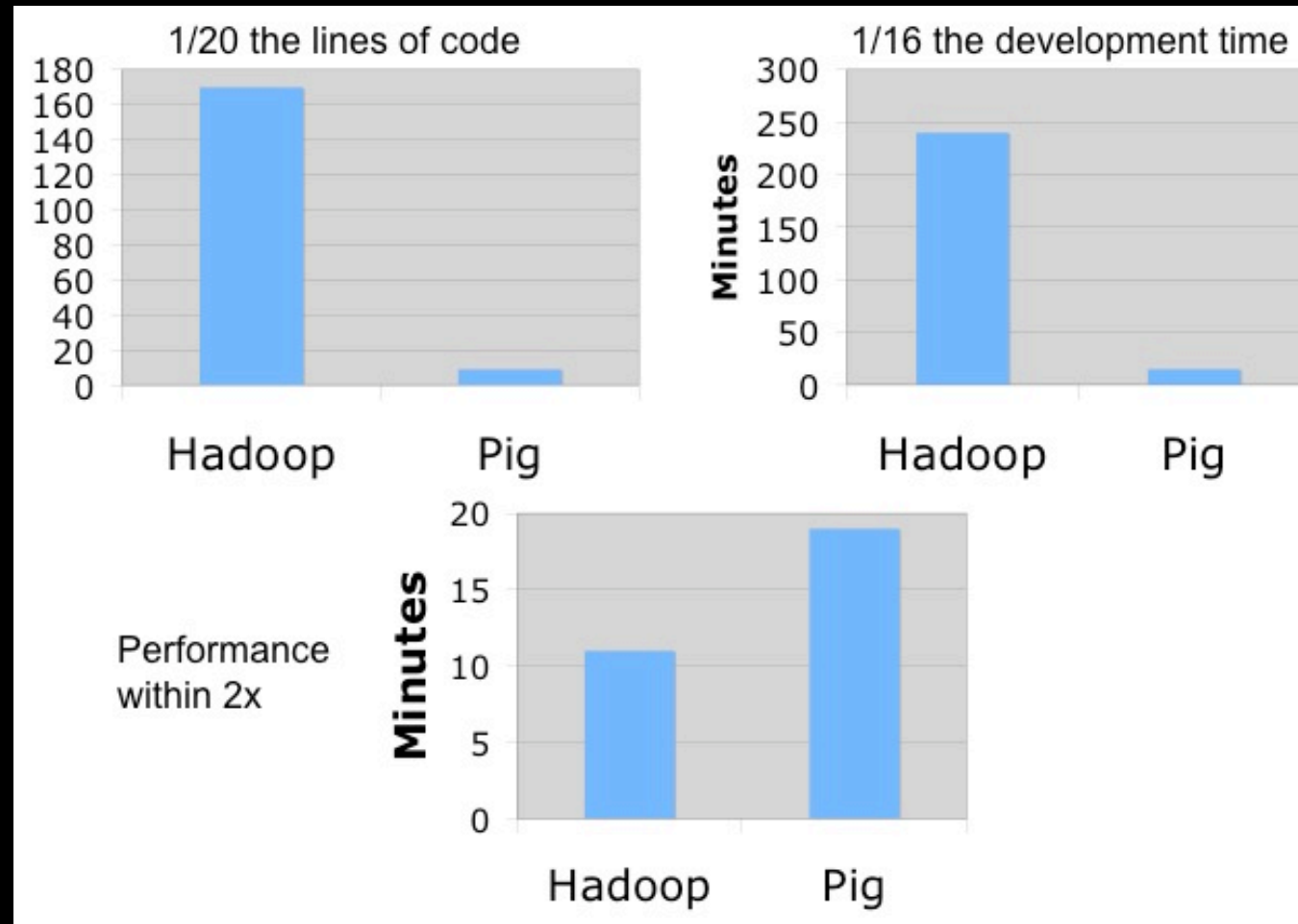
In Pig

```
Users = load 'users' as (name, age);
Filtered = filter Users by age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Joined = join Filtered by name, Pages by user;
Grouped = group Joined by url;
Summed = foreach Grouped generate group,
    COUNT(Joined) as clicks;
Sorted = order Summed by clicks desc;
Top5 = limit Sorted 5;
store Top5 into 'top5sites';
```





Natural Fit



Comparison

Flexibility & Control

- Easy to plug-in user code
- Metadata is not mandatory
- Does not impose a data model
- Fine grained control
- Complex data types



Pig Data Types

- Tuple: Ordered set of fields
 - Field can be simple or complex type
 - Nested relational model
- Bag: Collection of tuples
 - Can contain duplicates
- Map: Set of (key, value) pairs



Simple data types

- *int* : 42
- *long* : 42L
- *float* : 3.1415f
- *double* : 2.7182818
- *chararray* : UTF-8 String
- *bytearray* : blob



Expressions

```
A = LOAD 'data.txt' AS  
    (f1:int , f2:{t:(n1:int, n2:int)}, f3: map[] )
```

```
A =  
{  
    ( 1,                -- A.f1 or A.$0  
      { (2, 3), (4, 6) }, -- A.f2 or A.$1  
      [ 'yahoo' #'mail' ] -- A.f3 or A.$2  
    )  
}
```



Pig Unigrams

- Input: Large text document
- Process:
 - Load the file
 - For each line, generate word tokens
 - Group by word
 - Count words in each group



Load

```
myinput = load '/user/milindb/text.txt'  
        USING TextLoader() as (myword:chararray);
```

```
{  
  (program program)  
  (pig pig)  
  (program pig)  
  (hadoop pig)  
  (latin latin)  
  (pig latin)  
}
```



Tokenize

```
words = FOREACH myinput GENERATE FLATTEN(TOKENIZE(*));
```

```
{  
  (program) (program) (pig) (pig) (program) (pig)  
  (hadoop) (pig) (latin) (latin) (pig) (latin)  
}
```



Group

```
grouped = GROUP words BY $0;
```

```
{  
  (pig, {(pig), (pig), (pig), (pig), (pig)})  
  (latin, {(latin), (latin), (latin)})  
  (hadoop, {(hadoop)})  
  (program, {(program), (program), (program)})  
}
```



Count

```
counts = FOREACH grouped GENERATE group, COUNT(words);
```

```
{  
  (pig, 5L)  
  (latin, 3L)  
  (hadoop, 1L)  
  (program, 3L)  
}
```



Store

```
store counts into '/user/milindb/output'  
using PigStorage();
```

```
pig      5  
latin    3  
hadoop   1  
program  3
```



Example: Log Processing

```
-- use a custom loader
Logs = load '/var/log/access_log' using
    CommonLogLoader() as (addr, logname,
        user, time, method, uri, p, bytes);
-- apply your own function
Cleaned = foreach Logs generate addr,
    canonicalize(uri) as url;
Grouped = group Cleaned by url;
-- run the result through a binary
Analyzed = stream Grouped through
    'urlanalyzer.py';
store Analyzed into 'analyzedurls';
```



Schema on the fly

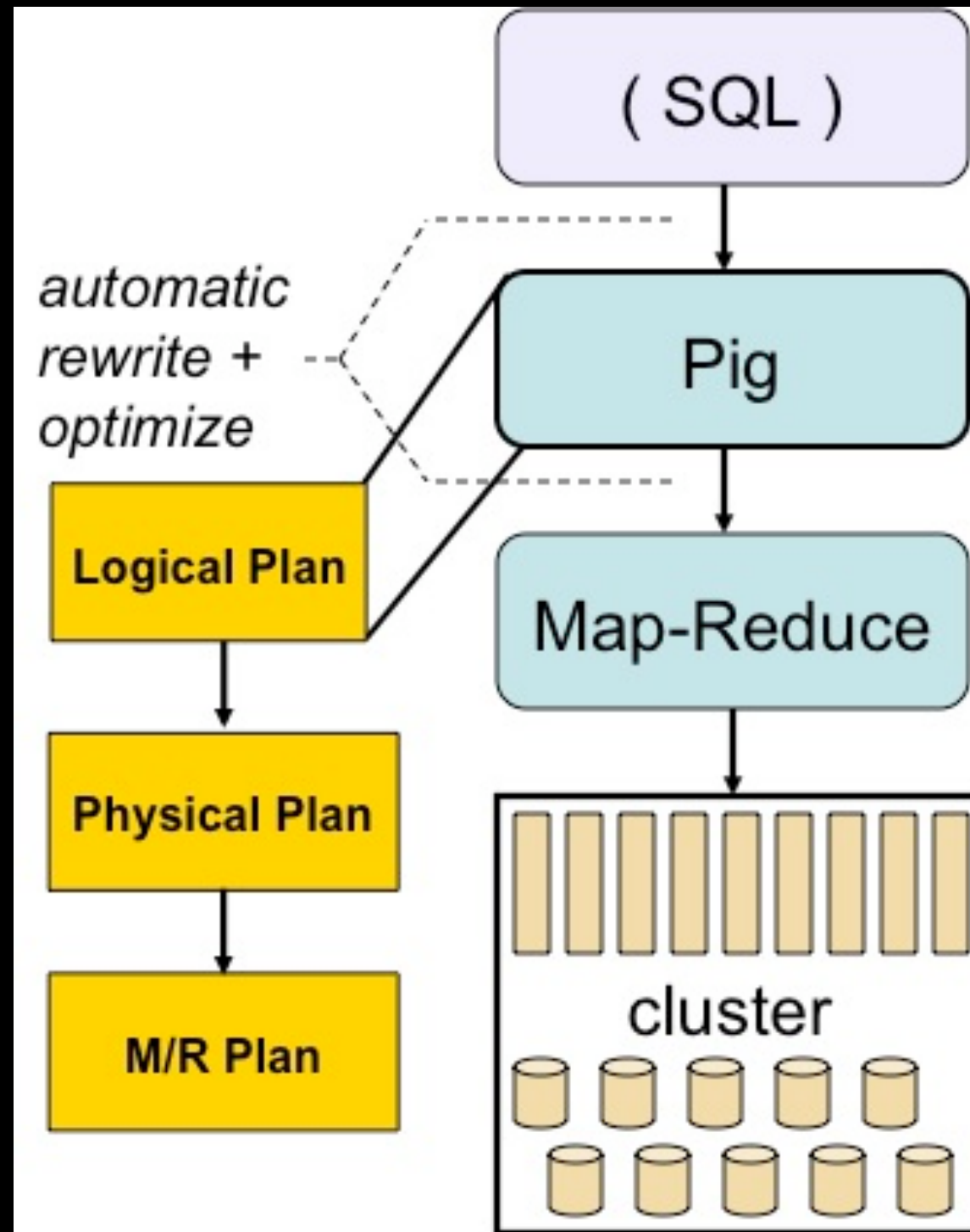
```
-- declare your types
Grades = load 'studentgrades' as
  (name: chararray, age: int,
   gpa: double);
Good = filter Grades by age > 18
      and gpa > 3.0;
-- ordering will be by type
Sorted = order Good by gpa;
store Sorted into 'smartgrownups';
```



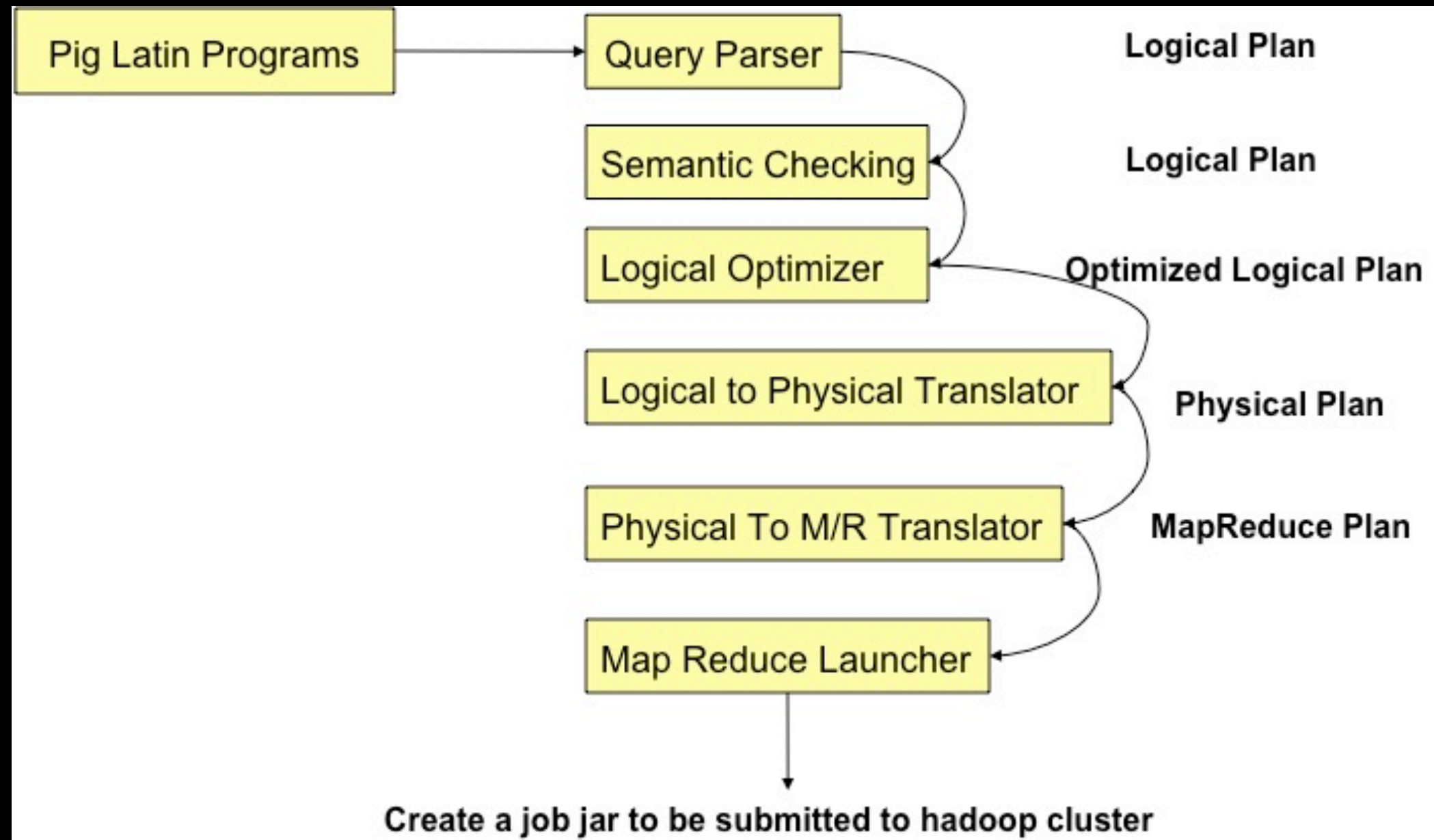
Nested Data

```
Logs = load 'weblogs' as (url, userid);
Grouped = group Logs by url;
-- Code inside {} will be applied to each
-- value in turn.
DistinctCount = foreach Grouped {
    Userid = Logs.userid;
    DistinctUsers = distinct Userid;
    generate group, COUNT(DistinctUsers);
}
store DistinctCount into 'distinctcount';
```





Pig Architecture



Pig Stages

Logical Plan

- Directed Acyclic Graph
 - Logical Operator as Node
 - Data flow as edges
- Logical Operators
 - One per Pig statement
 - Type checking with Schema



Pig Statements

Load	Read data from the file system
Store	Write data to the file system
Dump	Write data to <i>stdout</i>

Pig Statements

Foreach..Generate	Apply expression to each record and generate one or more records
Filter	Apply predicate to each record and remove records where false
Stream..through	Stream records through user-provided binary

Pig Statements

Group/CoGroup	Collect records with the same key from one or more inputs
Join	Join two or more inputs based on a key
Order.by	Sort records based on a key

Physical Plan

- Pig supports two back-ends
 - Local
 - Hadoop MapReduce
- 1:1 correspondence with most logical operators
 - Except Distinct, Group, Cogroup, Join etc



MapReduce Plan

- Detect Map-Reduce boundaries
 - Group, Cogroup, Order, Distinct
- Coalesce operators into Map and Reduce stages
- *Job.jar* is created and submitted to Hadoop *JobControl*



Lazy Execution

- Nothing really executes until you request output
- Store, Dump, Explain, Describe, Illustrate
- Advantages
 - In-memory pipelining
 - Filter re-ordering across multiple commands



Parallelism

- Split-wise parallelism on Map-side operators
- By default, 1 reducer
- PARALLEL keyword
 - group, cogroup, cross, join, distinct, order



Running Pig

```
$ pig
grunt > A = load 'students' as (name, age, gpa);
grunt > B = filter A by gpa > '3.5';
grunt > store B into 'good_students';
grunt > dump A;
(jessica thompson, 73, 1.63)
(victor zipper, 23, 2.43)
(rachel hernandez, 40, 3.60)
grunt > describe A;
A: (name, age, gpa )
```



Running Pig

- Batch mode
 - `$ pig myscript.pig`
- Local mode
 - `$ pig -x local`
- Java mode (embed pig statements in java)
 - Keep pig.jar in the class path



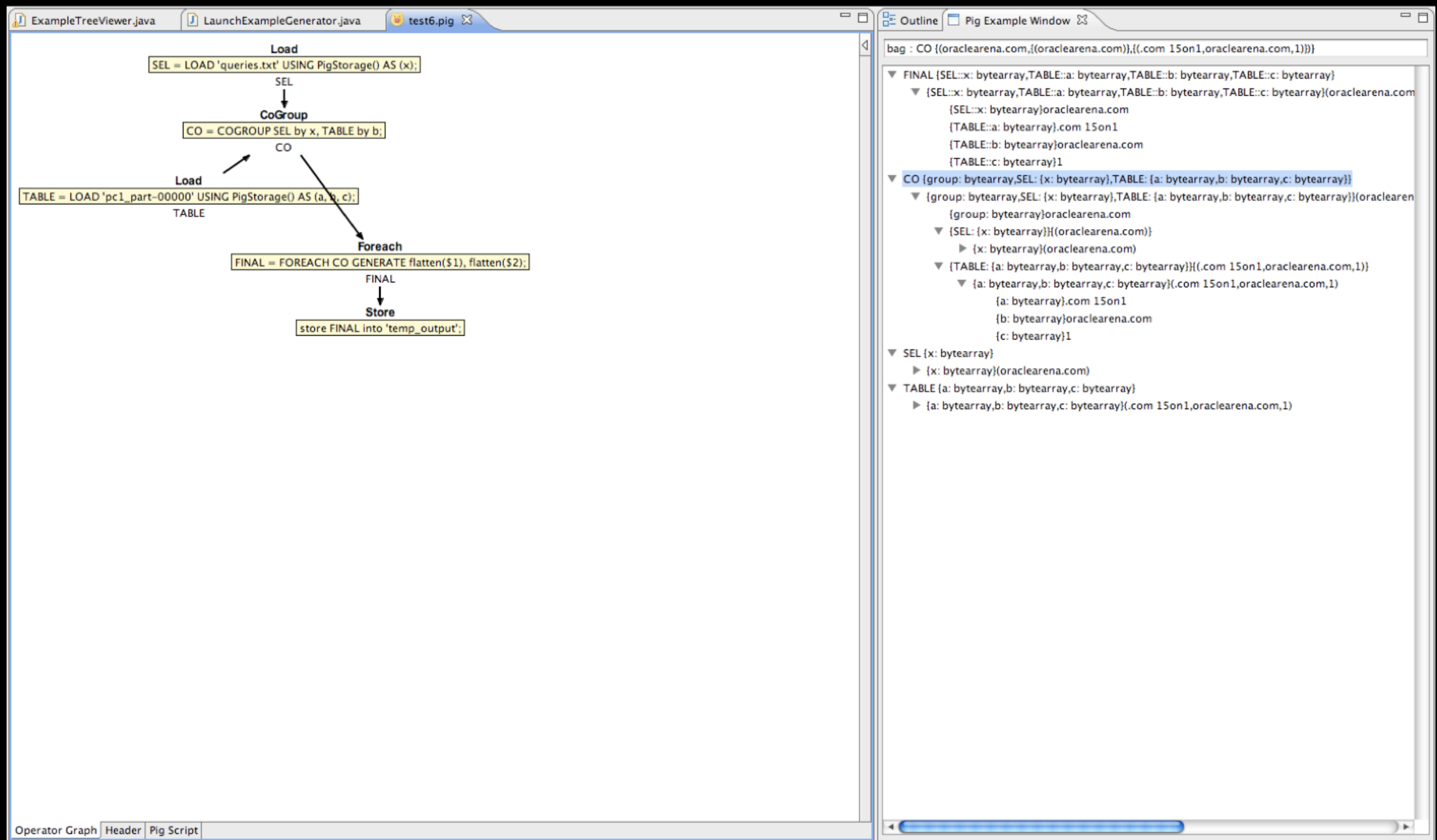
The screenshot displays the PigPen IDE interface. The main editor window on the left contains a Pig script named 'test6.pig'. The script is as follows:

```
1|
2SEL = LOAD 'queries.txt' USING PigStorage() AS (x);
3TABLE = LOAD 'pc1_part-00000' USING PigStorage() AS (a, b, c);
4CO = COGROUP SEL by x, TABLE by b;
5FINAL = FOREACH CO GENERATE flatten($1), flatten($2);
6store FINAL into 'temp_output';
7|
```

The right-hand pane, titled 'Outline', shows the execution plan for the script. It is a hierarchical tree structure representing the data flow and operations. The root of the plan is 'bag : CO {(oraclearena.com, {(oraclearena.com)}, {(com 15on1, oraclearena.com, 1)}}'. The plan details the execution of the 'COGROUP', 'FOREACH', and 'store' operators, showing the intermediate data structures and the final output path.

At the bottom of the IDE, there are tabs for 'Operator Graph', 'Header', and 'Pig Script', with 'Pig Script' currently selected.

PigPen



PigPen

Pig for SQL Programmers



SQL to Pig

SQL	Pig
...FROM MyTable...	A = LOAD 'MyTable' USING PigStorage('\t') AS (col1:int, col2:int, col3:int);
SELECT col1 + col2, col3 ...	B = FOREACH A GENERATE col1 + col2, col3;
...WHERE col2 > 2	C = FILTER B by col2 > 2;

SQL to Pig

SQL	Pig
<pre>SELECT col1, col2, sum(col3) FROM X GROUP BY col1, col2</pre>	<pre>D = GROUP A BY (col1, col2) E = FOREACH D GENERATE FLATTEN(group), SUM(A.col3);</pre>
<pre>...HAVING sum(col3) > 5</pre>	<pre>F = FILTER E BY \$2 > 5;</pre>
<pre>...ORDER BY col1</pre>	<pre>G = ORDER F BY \$0;</pre>

SQL to Pig

SQL	Pig
<pre>SELECT DISTINCT col1 from X</pre>	<pre>I = FOREACH A GENERATE col1; J = DISTINCT I;</pre>
<pre>SELECT col1, count(DISTINCT col2) FROM X GROUP BY col1</pre>	<pre>K = GROUP A BY col1; L = FOREACH K { M = DISTINCT A.col2; GENERATE FLATTEN(group), count(M); }</pre>

SQL to Pig

SQL	Pig
<pre>SELECT A.col1, B.col3 FROM A JOIN B USING (col1)</pre>	<pre>N = JOIN A by col1 INNER, B by col1 INNER; O = FOREACH N GENERATE A.col1, B.col3; -- Or N = COGROUP A by col1 INNER, B by col1 INNER; O = FOREACH N GENERATE flatten(A), flatten(B); P = FOREACH O GENERATE A.col1, B.col3</pre>



Questions ?