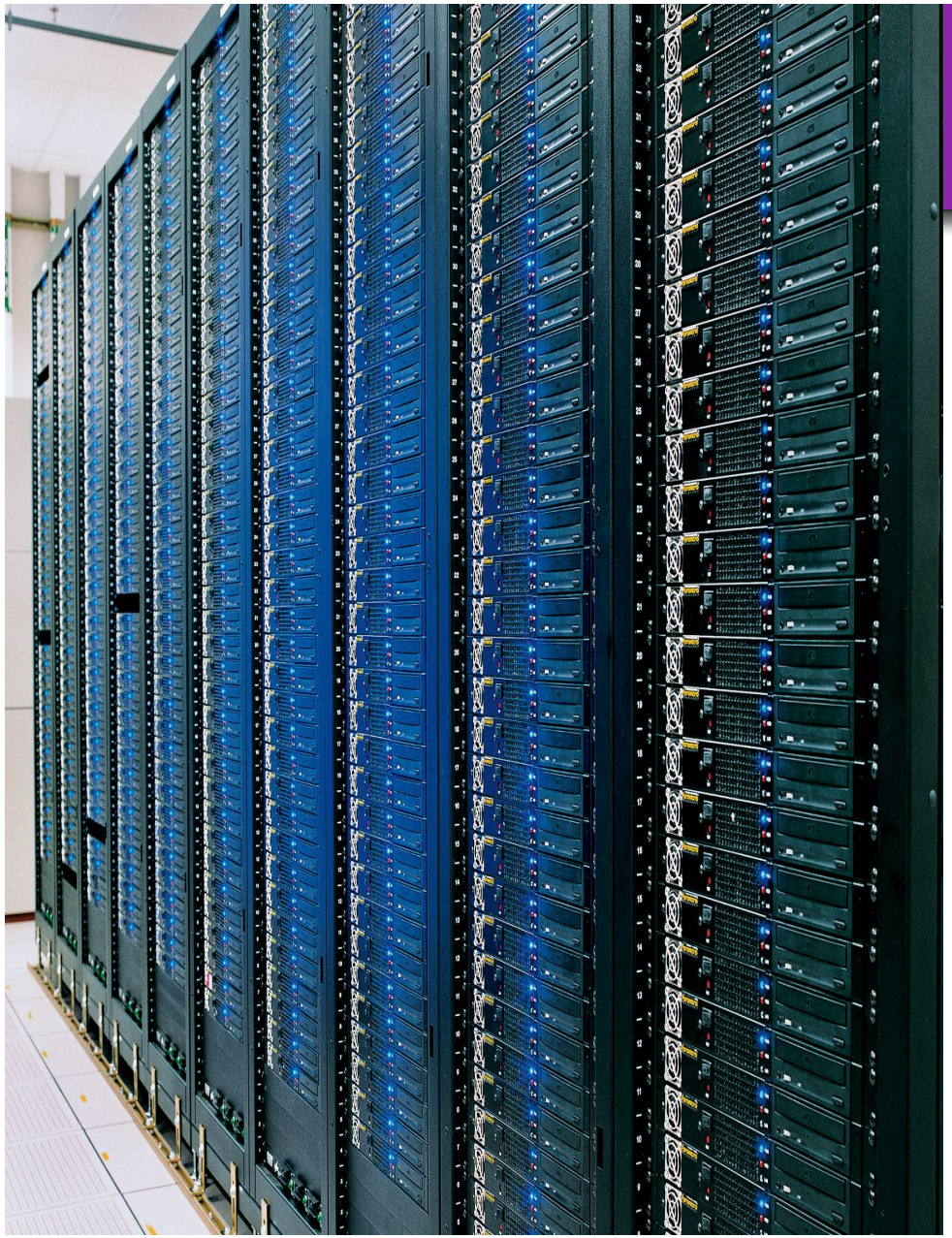


MapReduce Programming with Apache Hadoop

Viraj Bhat

viraj@yahoo-inc.com

Agenda – Session I and II (8-12pm)

- Introduction
- Hadoop Distributed File System (HDFS)
- Hadoop Map-Reduce Programming
- Hadoop Architecture
- Q & A – Break – 10:30-10:45 am
- Programming for Performance - 11:00am – 12pm

About Me

- Part of the Yahoo! Grid Team since May 2008
- PhD from Rutgers University, NJ
  - Specialization in Data Streaming, Grid, Autonomic Computing
- Worked on streaming data from live simulations executing in NERSC (CA), ORNL (TN) to Princeton Plasma Physics Lab (PPPL - NJ)
  - Library introduce less then 5% overhead on computation
- PhD Thesis on In-Transit data processing for peta-scale simulation workflows
- Developed CorbaCoG kit for Globus
- Active contributor to Hadoop Apache and developer of Hadoop Vaidya

The Challenge:
Very large scale data processing

Internet Scale Generates *BigData*

- Yahoo is the 3rd most Visited Site on the Internet
  - 600M+ Unique Visitors per Month
  - Billions of Page Views per Day
  - Billions of Searches per Month
  - Billions of Emails per Month
  - IMs, Address Books, Buddy Lists, …
  - **Terabytes of Data per Day!**
- And we crawl the Web
  - 100+ Billion Pages
  - 5+ Trillion Links
  - **Petabytes of data**

…terabytes, 100s of terabytes, petabytes, 10s of petabytes…

A huge amount of work is needed to unlock that value in that data…

- Search Marketing
  - Model ad performance and query patterns
  - Use to improve ad and page relevance

- Display & Contextual Advertising
  - Trace user activity to discover short- and long-term behavioral trends
  - Combine with demographic and geographic data to select far more targeted and effective advertisements

- Connect People to Content
  - Build models from page views
  - Personalize pages via models (e.g., MyYahoo)
  - Increases engagement, which you then monetize (see display advertising)

How to Process BigData?

- Just reading 100 terabytes of data can be overwhelming
  - Takes ~11 days to read on a standard computer
  - Takes a day across a 10Gbit link (very high end storage solution)
  - But it only takes 15 minutes on 1000 standard computers!
- Using clusters of standard computers, you get
  - Linear scalability
  - Commodity pricing

But there are certain problems…

- Reliability problems
  - In large clusters, computers fail every day
    - Data is corrupted or lost
    - Computations are disrupted
- Absent a good framework, programming clusters is **very hard**
  - A programmer worries about errors, data motion, communication…
  - Traditional debugging and performance tools don't apply
  - Most programs don't handle hardware failure well
- We need a standard set of tools to handle this complexity
  - Dean and Ghemawat described such a solution
    - MapReduce: Simplified Data Processing on Large Clusters
    - http://labs.google.com/papers/mapreduce-osdi04.pdf

Apache Hadoop – The solution!

- Hadoop brings MapReduce to everyone
  - It's an Open Source Apache project
  - Written in Java
    - Runs on Linux, Mac OS/X, Windows, and Solaris
    - Commodity hardware
- Hadoop vastly simplifies cluster programming
  - Distributed File System - distributes data
  - Map/Reduce - distributes application

A Brief History of Hadoop

- Pre-history (2002-2004)
  - Doug Cutting funded the Nutch open source search project
- Gestation (2004-2006)
  - Added DFS &Map-Reduce implementation to Nutch
  - Scaled to several 100M web pages
  - Still distant from web-scale (20 computers * 2 CPUs)
  - Yahoo! hired Doug Cutting
  - Hadoop project split out of Nutch
- Yahoo! commits to Hadoop (2006-2008)
  - Yahoo commits team to scaling Hadoop for production use (2006)
    - Eric Baldeschwieler builds team
  - Hadoop used by science and development teams (2007)
  - Web-scale production in early 2008! (2000 computers * 8 CPUs)

Hadoop At Yahoo! (Some Statistics)

- 30,000 + nodes in 10+ clusters
- Node: 4x1TB Disk, 8 Cores, 16 GB RAM
- Largest cluster is 4,000 nodes
- 6+ Petabytes of data (compressed, un-replicated)
- 1000+ users
- 100,000+ jobs/day

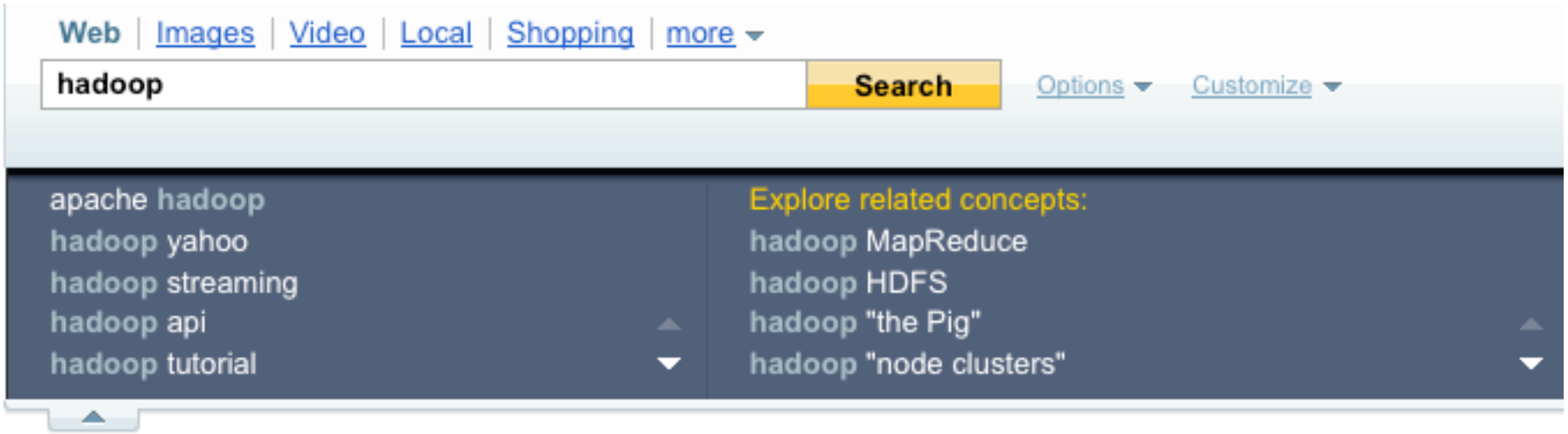
Sample Applications

- Data analysis is the inner loop of Web 2.0
  - Data ⇒ Information ⇒ Value
- Log processing: analytics, reporting, buzz
- Search index
- Content optimization, Spam filters
- Computational Advertising

Prominent Hadoop Users

- Yahoo!
- Amazon
- EHarmony
- Facebook
- Twitter
- Google/IBM
- Quantcast
- LinkedIn
- Baidu
- Adobe
- New York Times
- Powerset/Microsoft

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))
```

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:
  - 100 MM users
  - Average connections per user is 100

You Might Also Know

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

foreach u in UserList : // 100 MM
    foreach x in Connections(u) : // 100
        foreach y in Connections(x) : // 100
            if (y not in Connections(u)) :
                Count(u, y)++; // 1 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
- 100 MM users
  - 100 days on a single machine

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: 0.20.x
- Major contributors: Yahoo!, Facebook, Powerset, Cloudera

Apache Hadoop

- Reliable, Performant Distributed file system
- MapReduce Programming framework
- Sub-Projects: HBase, Hive, Pig, Zookeeper, Chukwa
- 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 on nodes
- Economical: Commodity components only
- Reliable
  - Engineering reliability into every application is expensive

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

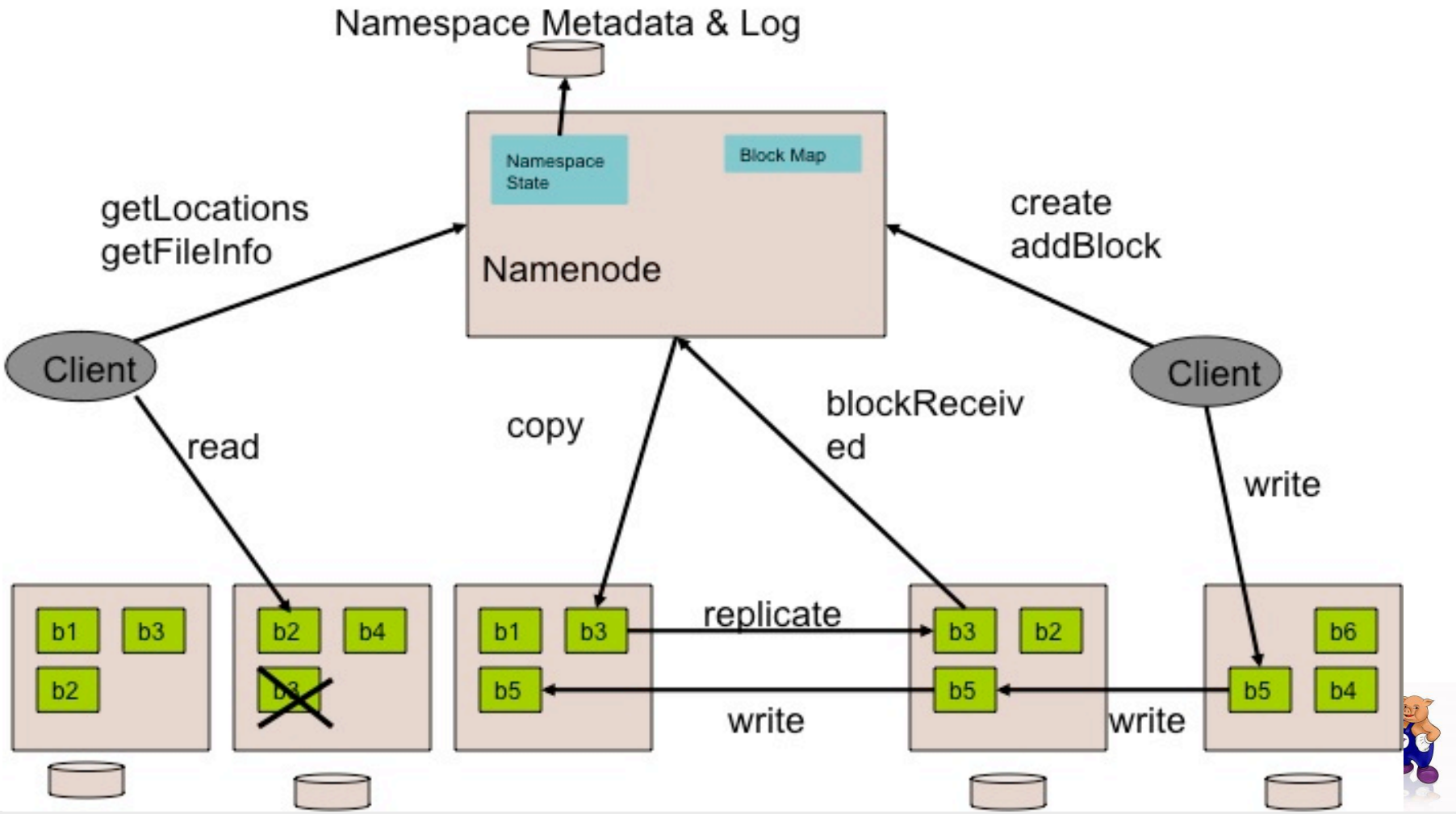
Namenode

- 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);
```

Hadoop MapReduce

- 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: List(Key_2, $Value_2$)
- Output
  - Sort(Partition(List(Key_2, List($Value_2$))))
- Provided by Hadoop

Reduce

- Input: List(Key_2, List($Value_2$))
- Output: List(Key_3, $Value_3$)
- 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)
```

Java Mapper

```
public static class Map extends MapReduceBase implements
                    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, OutputCollector<Text,
   IntWritable> output,
                    Reporter reporter) throws IOException
    {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
              word.set(tokenizer.nextToken());
              output.collect(word, one);
            }
     }
}
```

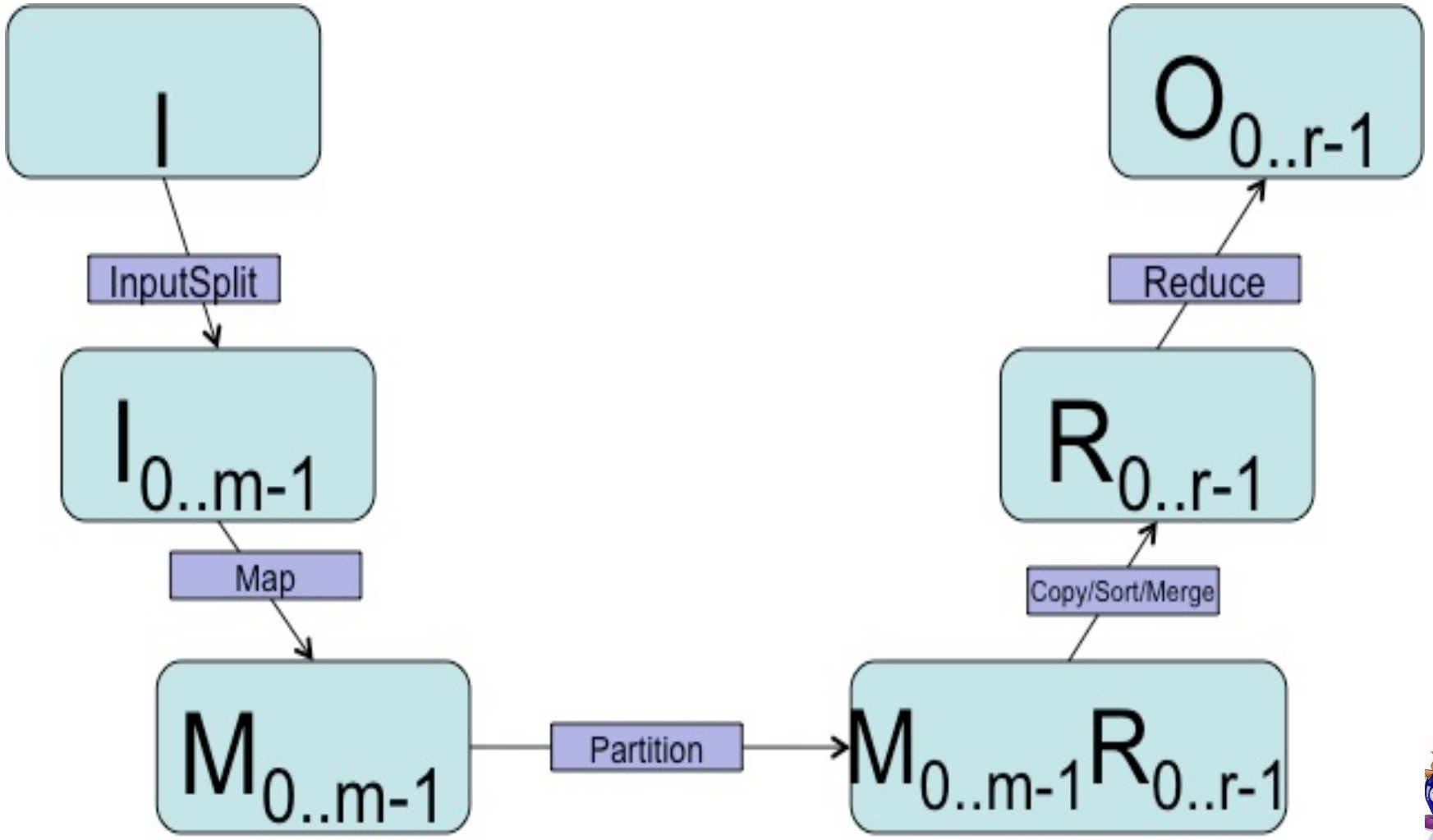
Java Reducer

```
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));
        }
    }
```

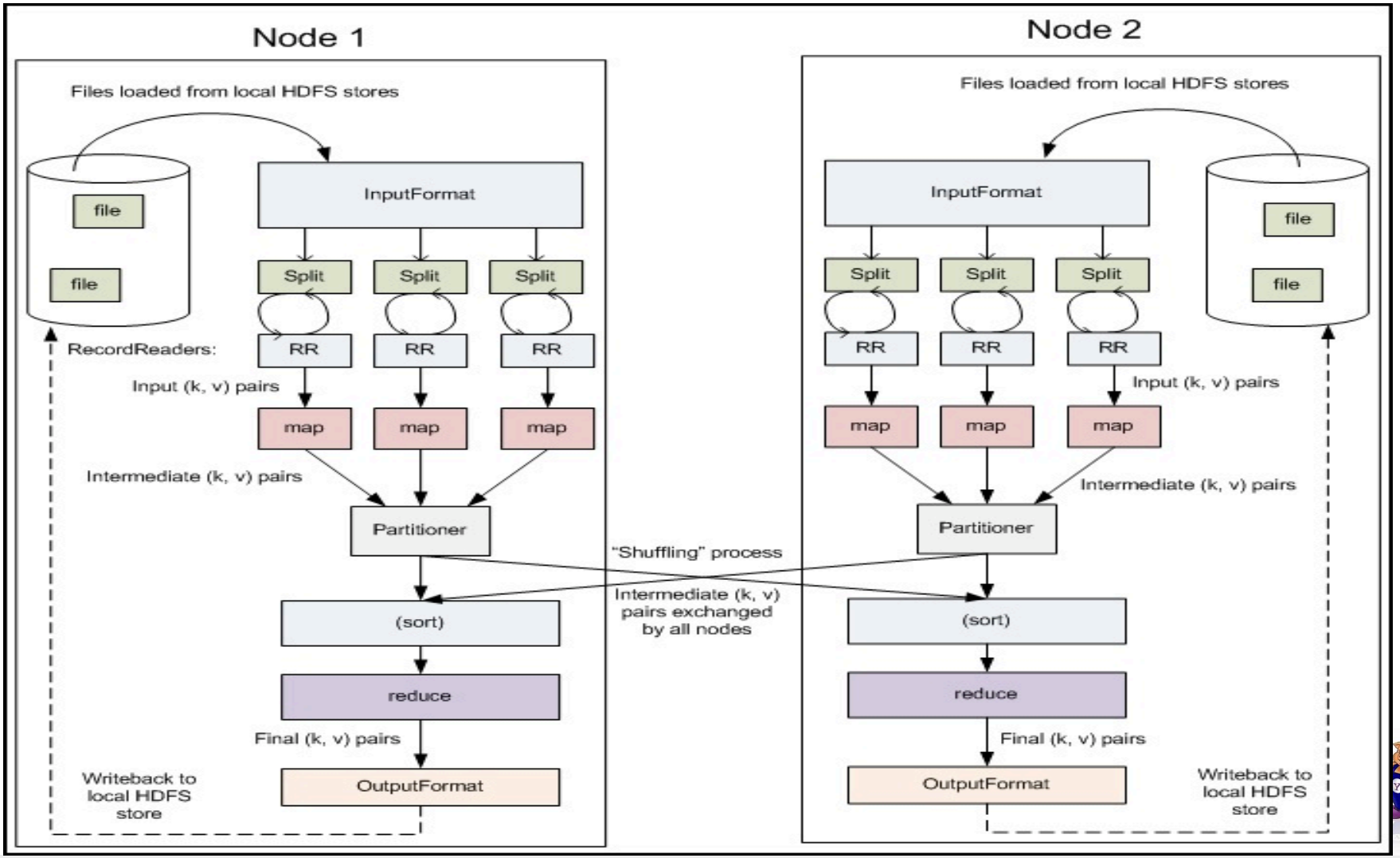
Java Main

```
public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(Unigrams.class);
        conf.setJobName(”unigrams");

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);

        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);

        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        JobClient.runJob(conf);
}
```

MR Dataflow


I
InputSplit
$I_{0..m-1}$
Map
$M_{0..m-1}$
Partition
$M_{0..m-1}R_{0..r-1}$
Copy/Sort/Merge
$R_{0..r-1}$
Reduce
$O_{0..r-1}$

Pipeline Details



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

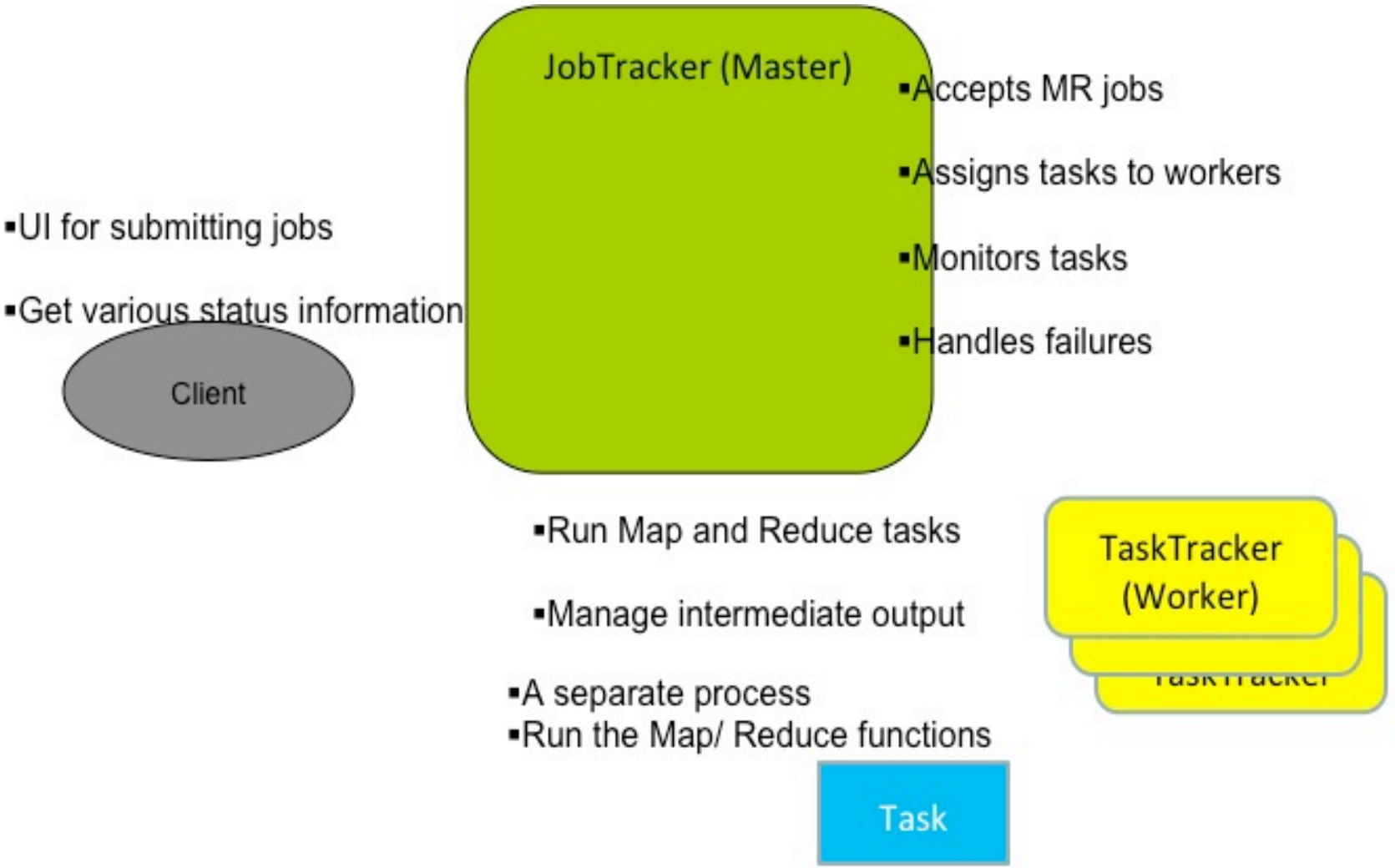
Hadoop Streaming

- Thin Java wrapper 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

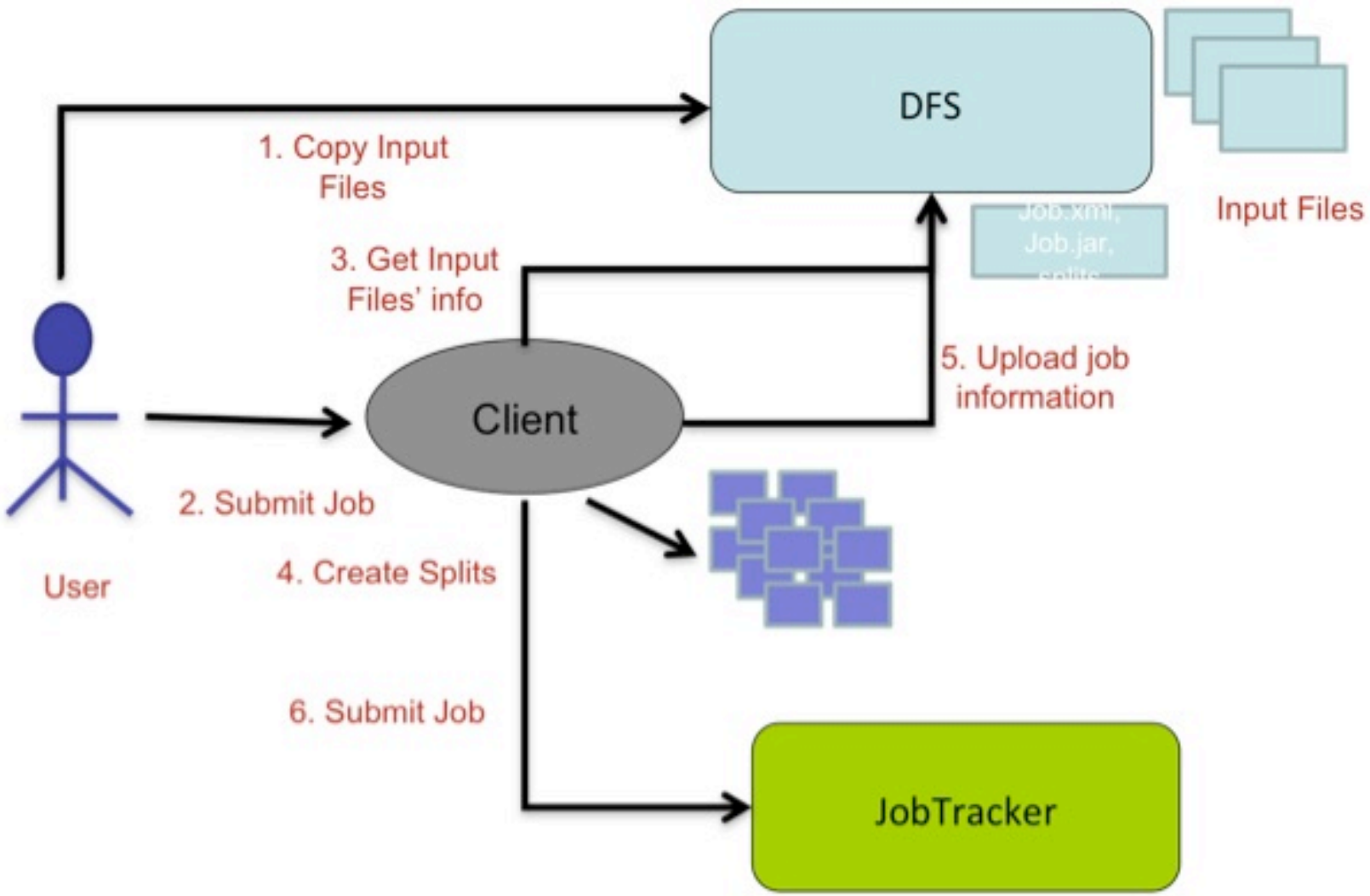
Hadoop Streaming

```
$ 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}'
```

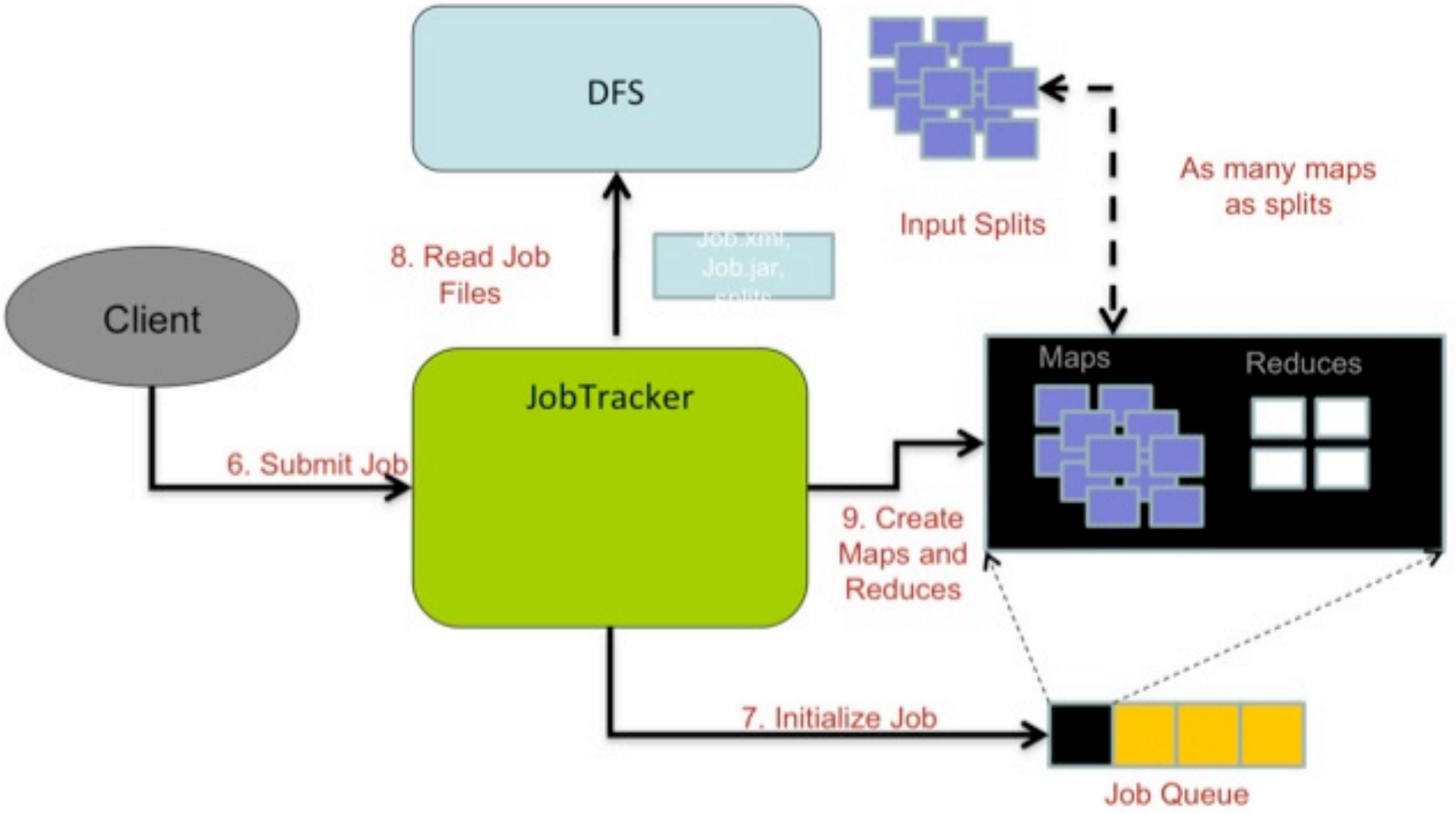
MapReduce Architecture



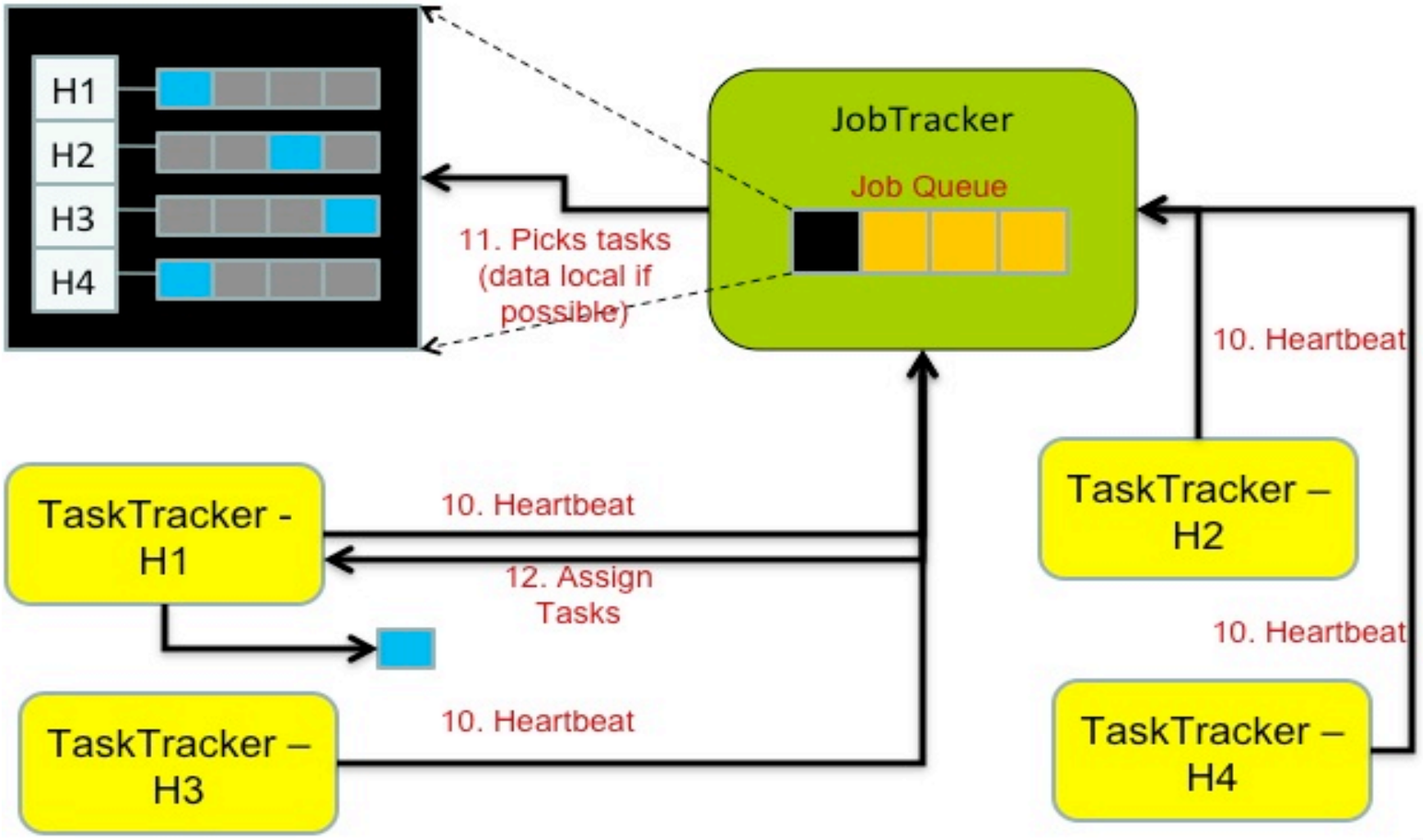
Job Submission



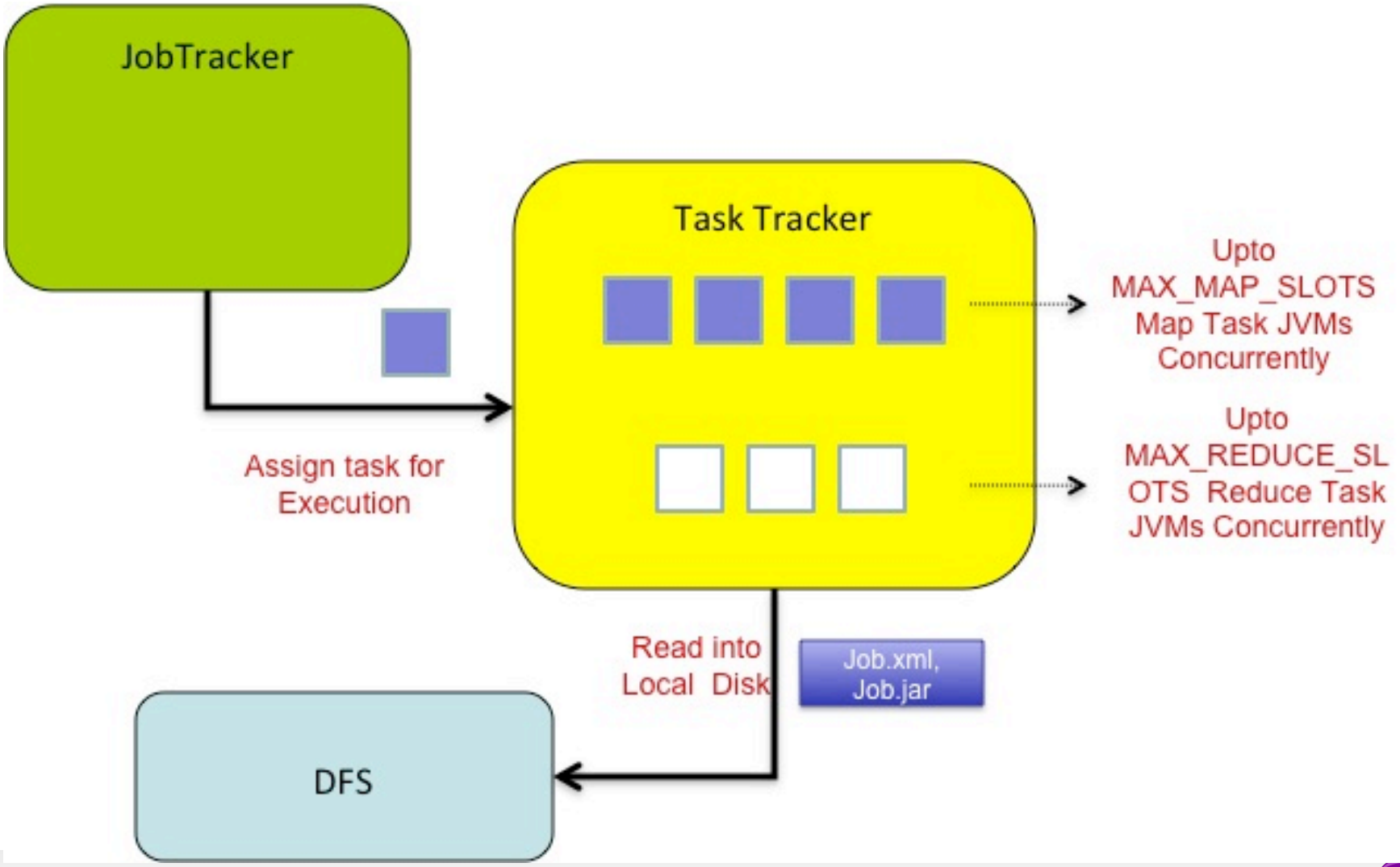
Initialization



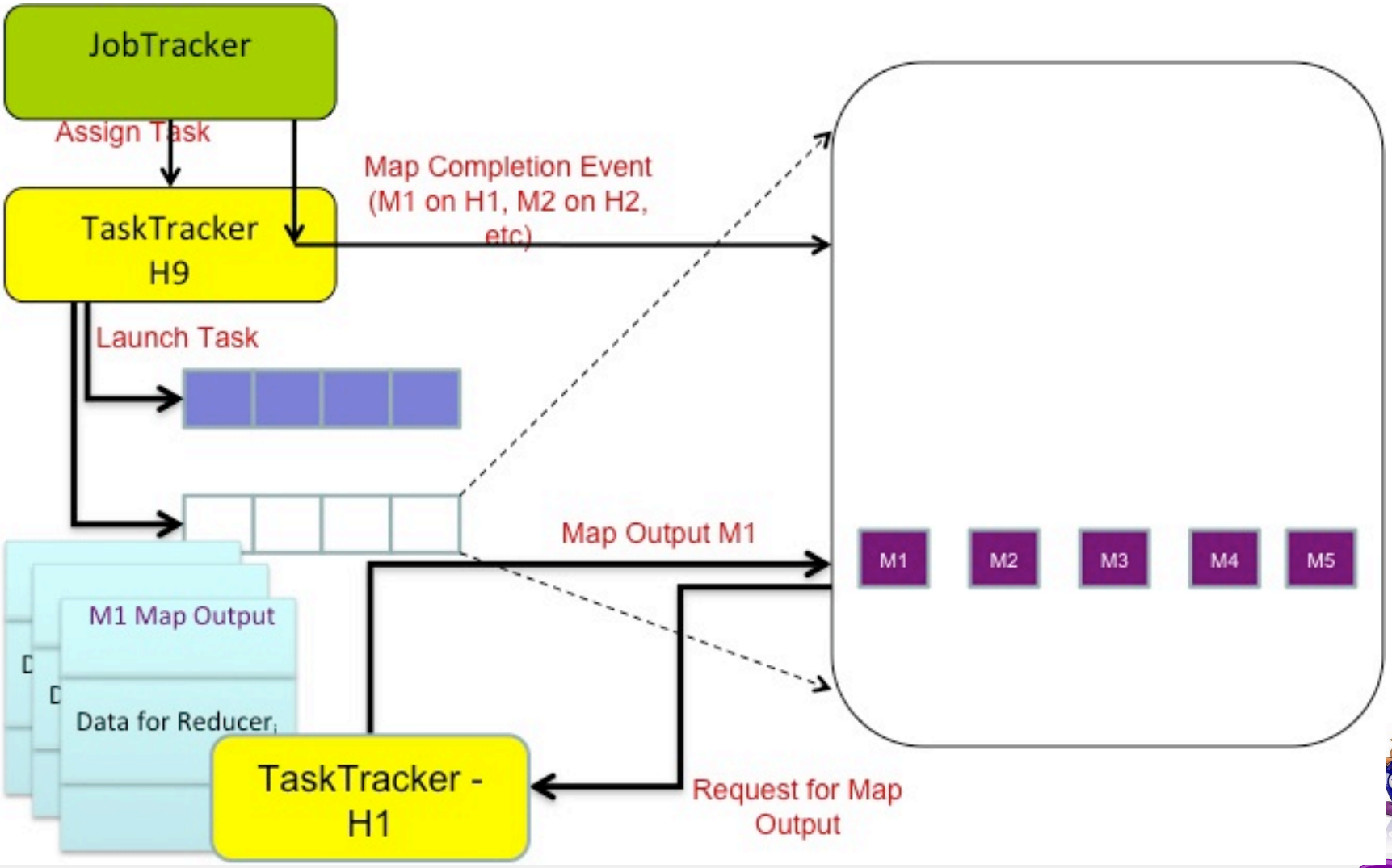
Scheduling


H1
H2
H3
H4
JobTracker
Job Queue
11. Picks tasks (data local if possible)
10. Heartbeat
TaskTracker - H1
10. Heartbeat
12. Assign Tasks
TaskTracker – H2
10. Heartbeat
TaskTracker – H3
10. Heartbeat
TaskTracker – H4

Execution


JobTracker
Task Tracker
Upto
MAX_MAP_SLOTS
Map Task JVMs
Concurrently
Upto
MAX_REDUCE_SL
OTS Reduce Task
JVMs Concurrently
Assign task for
Execution
Read into
Local Disk
Job.xml,
Job.jar
DFS

Reduce Task


JobTracker
Assign Task
TaskTracker
H9
Map Completion Event
(M1 on H1, M2 on H2,
etc)
Launch Task
Map Output M1
M1
M2
M3
M4
M5
M1 Map Output
Data for Reducer
TaskTracker -
H1
Request for Map
Output

Hadoop Scheduler

Capacity Scheduler

- Multiple Queues for users to submit a job to the JobTracker
- Multiple users (ACL) can be associated with a queue to submit one or more jobs to the queue.
- Queues are guaranteed a fraction of the total capacity of the cluster
  - Capacity is measured in terms of total map/reduce slots
  - All jobs submitted to a queue will have access to the capacity guaranteed to the queue
  - Total guaranteed capacity across all queues should be 100%
- Free resources can be allocated to any queue beyond its guaranteed capacity.
  - If needed, excess resources allocated will be reclaimed to meet the guaranteed capacity needs of the other queues.
- The scheduler guarantees that excess resources taken from a queue will be restored to it within N minutes of its need for them
- Queues optionally support job priorities
  - Preemption of low priority jobs is not done to serve high priority jobs
- Support for maximum % resource limit per user per queue
- Support for memory-intensive jobs

Capacity Scheduler

- How it picks a Job (map/reduce task from the job) to run
  - When a slot on a TaskTracker is free
  - Select a queue
    - One which is running with lower than guaranteed capacity for relatively longer time
    - If no such queue then use one with more free space (i.e. ratio of running slots to guaranteed capacity is lowest)
  - Select a job by <priority, submit-time> order
    - If Job user is not currently using more than its max. resource limit
    - If TaskTracker has enough memory to spare for the current job's task (if job has special memory requirements)
- How it reclaims the capacity
  - Periodically, it notes the amount of resources to be reclaimed and the reclaim time remaining for any queue
    - if it is short of its guaranteed capacity and at least one task is pending to execute
  - If queue has not received its guaranteed capacity and reclaim time is about to expire, it starts killing the latest tasks from over capacity queues.

Capacity Scheduler Configuration

- Keep the <hadoop-*-capacity-scheduler.jar> in class path or in $HADOOP_HOME/lib
- Set the following parameters in site configuration
  - mapred.jobtracker.taskScheduler = org.apache.hadoop.mapred.CapacityTaskScheduler
  - mapred.queue.names=<comma separated queue names>
  - mapred.acls.enabled=<true/false>
  - mapred.queue.*queue-name.acl-submit-job=<list of users/groups for queue>*
  - mapred.queue.queue-name.acl-administer-job=<list of admin user/groups>
- To Configure queue properties
  - Use $HADOOP_HOME/conf/capacity-scheduler.xml
  - mapred.capacity-scheduler.queue.<queue-name>.guaranteed-capacity
  - mapred.capacity-scheduler.queue.<queue-name>.reclaim-time-limit (in Secs)
  - mapred.capacity-scheduler.queue.<queue-name>.supports-priority (true/false)
  - mapred.capacity-scheduler.queue.<queue-name>.minimum-user-limit-percent
- To configure scheduler behavior
  - mapred.capacity-scheduler.reclaimCapacity.interval (default 5 Secs)

Fair Scheduler

- One large static cluster with one instance of M/R & HDFS daemons
  - Divided into multiple logical pools (aka. queues)
  - Each pool has minimum guaranteed resource capacity.
  - Multiple users are associated with a pool can submit one or more jobs to the pool to share the pool capacity
    - Jobs submitted to any pool may use the unutilized resource capacity from other pools.
  - It is possible to configure maximum number of running jobs per pool and per user within a pool.
- All jobs submitted to the cluster should on an average get equal share of resources over the time
  - Default JobTracker Scheduler forms a queue of jobs
  - Fair scheduler allows small jobs to finish in reasonable time while not starving long jobs.

Fair Scheduler – Implementation

- Job Weight ~= Fn (Job Priority, Job Size, Pool Weight)
  - Job Priority = Normal, High, Very High
- Fair Share = Job Weight * Cluster Capacity
- Actual Share = Num Job Tasks / Cluster Capacity (over last 100 milliseconds)
- Deficit = Fair Share – Actual Share
- Job to Schedule
  - Next freed up slot allocated to a job w/ max deficit
  - If any jobs not getting minimum guaranteed capacity of their pool are prioritized (again based on their deficit)
- Jobs to run
  - Sorted list of jobs per <priority, submit time>
  - Limited by max number of jobs per pool & user.

Hadoop Software Ecosystem

- Pig – Yahoo!
  - Parallel Programming Language and Runtime
- Zookeeper – Yahoo!
  - High-Availability Directory and Configuration Service
- Oozie
  - Oozie is the workflow/scheduling solution for the Grid.
- Hbase – Powerset.com/Miscrosoft
  - TableStore layered on HDFS
- JACL – IBM
  - JSON / SQL inspired programming Hadoop language
- Mahout – Individual apache members
  - Machine learning libraries for Hadoop
- Tashi – Intel, CMU
  - Virtual machine provisioning service (soon)
- Hive – Facebook.com
  - Data warehousing framework on Hadoop (soon)

What's ahead

- Improved scalability
  - E.g. 10s K nodes through Federation
  - Federated applications across clusters and data centers
    - Hadoop 22
- Improved performance
  - YARN
- Enhanced features
- Further extensions
  - on-line service grid?
- More applications, from many fields!

The Hadoop eco-system is growing and has the potential to have a lot of impact across the internet industry, and many others!

Hadoop uses beyond Yahoo!

BigData…Not Just for Internet Companies

Wal-Mart

- 267 million items/day, sold at 6,000 stores
- HP building them 4PB data warehouse
- Mine data to manage supply chain, understand market trends, formulate pricing strategies

Sloan Digital Sky Survey
Mapping the Universe

Sloan Digital Sky Survey

- New Mexico telescope captures 200 GB image data / day
- Latest dataset release: 10 TB, 287 million celestial objects
- SkyServer provides SQL access

Slide courtesy Randy Bryant

Hadoop In Action

- GrepTheWebby Amazon
  - Hadoop on AWS, using EC2
  - Mining the Web using the Alexa web content
- Building Ground Models of Southern California
  - Research by Intel and CMU
- Online search for engineering design content by Autodesk
- The New York Times uses it to process their archives
  - http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/
- Rapidly growing number of commercial applications

Research & Education @ Univ of Maryland

- *Cloud Computing* Pilot Course in Spring 2008
- Integration of research and education
- Hadoop cluster from IBM/Google
- 13 students: 3 undergrads, 3 masters, 7 PhDs
- 6 teams working on *web-scale* research problems
  - Machine translation
  - Reference resolution in email archives
  - Language modeling
  - Biomedical text retrieval
  - Text-image separation in children's books
  - Biological sequence alignment

Performance Analysis of Map-Reduce

- MR performance requires
  - Maximizing Map input transfer rate
  - Pipelined writes from Reduce
  - Small intermediate output
  - Opportunity to Load Balance

Performance Example

- Bob wants to count lines in text files totaling several hundred gigabytes
- He uses
  - Identity Mapper (input: text, output: same text)
  - A single Reducer that counts the lines and outputs the total
- What is he doing wrong ?
- This happened, really !
  - I am not kidding !

Intermediate Output

- Almost always the most expensive component
  - M * R Transfers over the network
  - Merging and Sorting
- How to improve performance:
  - Avoid shuffling/sorting if possible
  - Minimize redundant transfers
  - Compress

Avoid shuffling/sorting

- Set number of reducers to zero
  - Known as map-only computations
  - Filters, Projections, Transformations
- Beware of number of files generated
  - Each map task produces a part file
  - Make map produce equal number of output files as input files
    - How?

Minimize Redundant Transfers

- Combiners
- When *maps* produce many repeated keys
  - It is often useful to do a local aggregation following the *map*
  - Done by specifying a *Combiner*
  - Goal is to decrease size of the transient data
  - Combiners have the same interface as Reduces, and often are the same class.
  - Combiners must **not** have side effects, because they run an indeterminate number of times.
  - *conf.setCombinerClass(Reduce.class);*

Compress Output

- Compressing the outputs and intermediate data will often yield huge performance gains
  - Can be specified via a configuration file or set programmatically
  - Set *mapred.output.compress* to *true* to compress job output
  - Set *mapred.compress.map.output* to *true* to compress map outputs
- Compression Types *(mapred.[map.]output.compression.type)*
  - "block" - Group of keys and values are compressed together
  - "record" - Each value is compressed individually
  - Block compression is almost always best
- Compression Codecs *(mapred.[map.]output.compression.codec)*
  - Default (zlib) - slower, but more compression
  - LZO - faster, but less compression

Opportunity to Load Balance

- Load imbalance inherent in the application
  - Imbalance in input splits
  - Imbalance in computations
  - Imbalance in partition sizes
- Load imbalance due to heterogeneous hardware
  - Over time performance degradation
- Give Hadoop an opportunity to do load-balancing
  - How many nodes should I allocate ?

Load Balance (contd.)

- M = total number of simultaneous map tasks
- M = map task slots per task tracker * nodes
- Choose nodes such that total mappers is between 5*M and 10*M.

Configuring Task Slots

- mapred.tasktracker.map.tasks.maximum
- mapred.tasktracker.reduce.tasks.maximum
- Tradeoffs:
  - Number of cores
  - Amount of memory
  - Number of local disks
  - Amount of local scratch space
  - Number of processes
- Also consider resources consumed by Tasktracker & Datanode

Speculative execution

- The framework can run multiple instances of slow tasks
  - Output from instance that finishes first is used
  - Controlled by the configuration variable *mapred.[map|reduce].speculative.execution=[true|false]*
  - Can dramatically bring in long tails on jobs

Performance Summary

- Is your input splittable?
  - Gzipped files are NOT splittable
- Are partitioners uniform?
- Buffering sizes (especially io.sort.mb)
- Do you need to Reduce?
- Only use singleton reduces for very small data
  - Use Partitioners and cat to get a total order
- Memory usage
  - Please do not load all of your inputs into memory!

Debugging & Diagnosis

- Run job with the Local Runner
  - Set mapred.job.tracker to "local"
  - Runs application in a single process and thread
- Run job on a small data set on a 1 node cluster
  - Can be done on your local dev box
- Set *keep.failed.task.files* to true
  - This will keep files from failed tasks that can be used for debugging
  - 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

JobTracker UI

kry1036 Hadoop Map/Reduce Administration

State: RUNNING
Started: Mon Apr 28 16:59:04 UTC 2008
Version: 0.16.3, r647354
Compiled: Fri Apr 11 23:58:10 UTC 2008 by hadoopqa
Identifier: 200804281659

Cluster Summary

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node
0	0	1	9	18	18	4.00

Running Jobs

Running Jobs
none

Completed Jobs

Completed Jobs								
Jobid	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed
job_200804281659_0001	milindb	wordcount	100.00%	41	41	100.00%	4	4

Failed Jobs

Failed Jobs
none

Local logs

Log directory, Job Tracker History

Hadoop, 2007.

Job Details

Hadoop job_200804281659_0001 on kry1036

User: milindb
Job Name: wordcount
Job File: /user/milindb/mapredsystem/271140.kry1640.inktomisearch.com/job_200804281659_0001/job.xml
Status: Succeeded
Started at: Mon Apr 28 17:02:05 UTC 2008
Finished at: Mon Apr 28 17:07:08 UTC 2008
Finished in: 5mins, 3sec

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	41	0	0	41	0	0 / 0
reduce	100.00%	4	0	0	4	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	Launched map tasks	0	0	41
	Launched reduce tasks	0	0	4
Map-Reduce Framework	Map input records	60,490,175	0	60,490,175
	Map output records	579,909,134	0	579,909,134
	Map input bytes	5,447,617,164	0	5,447,617,164
	Map output bytes	7,695,121,884	0	7,695,121,884
	Combine input records	579,909,134	0	579,909,134
	Combine output records	79,958,225	0	79,958,225
	Reduce input groups	0	15,009,577	15,009,577
	Reduce input records	0	79,958,225	79,958,225
	Reduce output records	0	15,009,577	15,009,577

List of reduces, all successful

Hadoop reduce task list for job_200804281659_0001 on kry1036

All Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
tip_200804281659_0001_r_000000	100.00%	reduce > reduce	28-Apr-2008 17:02:11	28-Apr-2008 17:07:07 (4mins, 55sec)		3
tip_200804281659_0001_r_000001	100.00%	reduce > reduce	28-Apr-2008 17:02:11	28-Apr-2008 17:06:57 (4mins, 46sec)		3
tip_200804281659_0001_r_000002	100.00%	reduce > reduce	28-Apr-2008 17:02:11	28-Apr-2008 17:06:57 (4mins, 46sec)		3
tip_200804281659_0001_r_000003	100.00%	reduce > reduce	28-Apr-2008 17:02:11	28-Apr-2008 17:07:08 (4mins, 56sec)		3

Go back to JobTracker
Hadoop, 2007.

Task Details

Job job_200804281659_0001

All Task Attempts

Task Attempts	Machine	Status	Progress	Start Time	Shuffle Finished	Sort Finished	Finish Time	Errors	Task Logs	Counters	Actions
task_200804281659_0001_r_000001_0	kry1595.inktomisearch.com	SUCCEEDED	100.00%	28-Apr-2008 17:02:12	28-Apr-2008 17:06:06 (3mins, 53sec)	28-Apr-2008 17:06:06 (0sec)	28-Apr-2008 17:06:57 (4mins, 45sec)		Last 4KB Last 8KB All	3	

Go back to the job
Go back to JobTracker
Hadoop, 2007.

Task Logs

Task Logs: 'task_200804281659_0001_r_000001_0'

stdout logs

---

stderr logs

---

syslog logs

```
53 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 done copying task_200804281659_0001_m_000040_0 output from kry1355.inktomisearch.com.
2008-04-28 17:05:56,853 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 InMemoryFileSystem ramfs://mapoutput24713456 is 0.61232907 full. Triggering merge
2008-04-28 17:05:56,854 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Thread started: Thread for merging in memory files
2008-04-28 17:05:56,867 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Need 3 map output(s)
2008-04-28 17:05:56,870 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 done copying task_200804281659_0001_m_000039_0 output from kry1475.inktomisearch.com.
2008-04-28 17:05:56,919 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 done copying task_200804281659_0001_m_000037_0 output from kry1235.inktomisearch.com.
2008-04-28 17:05:57,373 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0: Got 0 new map-outputs & 0 obsolete map-outputs from tasktracker and 0 map-outputs from previous failures
2008-04-28 17:05:57,373 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Got 0 known map output location(s); scheduling...
2008-04-28 17:05:57,373 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Scheduled 0 of 0 known outputs (0 slow hosts and 0 dup hosts)
2008-04-28 17:05:57,373 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Need 1 map output(s)
2008-04-28 17:05:58,374 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0: Got 0 new map-outputs & 0 obsolete map-outputs from tasktracker and 0 map-outputs from previous failures
2008-04-28 17:05:58,375 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Got 0 known map output location(s); scheduling...
2008-04-28 17:05:58,375 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Scheduled 0 of 0 known outputs (0 slow hosts and 0 dup hosts)
2008-04-28 17:06:00,885 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Merge of the 4 files in InMemoryFileSystem complete. Local file is /export/crawlspace/kryptonite/hod/tmps/2/kry1036-18814-7
2008-04-28 17:06:03,375 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Need 1 map output(s)
2008-04-28 17:06:03,376 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0: Got 1 new map-outputs & 0 obsolete map-outputs from tasktracker and 0 map-outputs from previous failures
2008-04-28 17:06:03,376 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Got 1 known map output location(s); scheduling...
2008-04-28 17:06:03,377 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Scheduled 1 of 1 known outputs (0 slow hosts and 0 dup hosts)
2008-04-28 17:06:03,377 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Copying task_200804281659_0001_m_000036_0 output from kry1475.inktomisearch.com.
2008-04-28 17:06:03,565 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 done copying task_200804281659_0001_m_000036_0 output from kry1475.inktomisearch.com.
2008-04-28 17:06:03,566 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Copying of all map outputs complete. Initiating the last merge on the remaining files in ramfs://mapoutput24713456
2008-04-28 17:06:06,602 INFO org.apache.hadoop.mapred.ReduceTask: task_200804281659_0001_r_000001_0 Merge of the 3 files in InMemoryFileSystem complete. Local file is /export/crawlspace/kryptonite/hod/tmps/3/kry1036-18814-7
2008-04-28 17:06:57,872 INFO org.apache.hadoop.mapred.TaskRunner: Task 'task_200804281659_0001_r_000001_0' done.
```