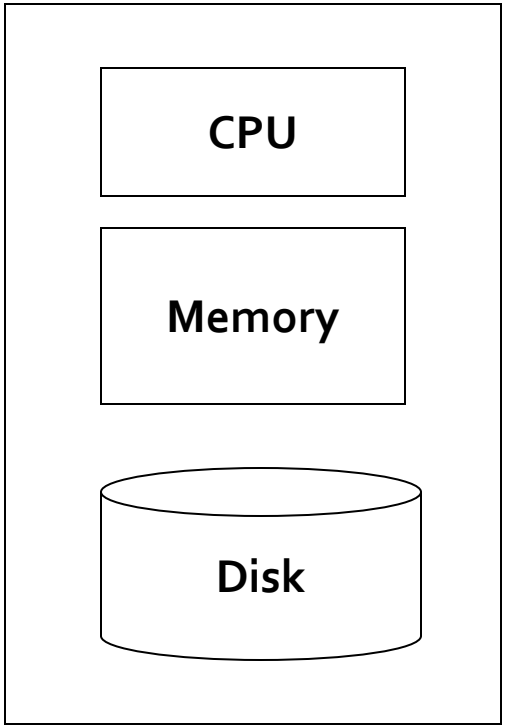
MapReduce

CS246: Mining Massive Datasets

Jure Leskovec, Stanford University

http://cs246.stanford.edu

Single-node architecture



Machine Learning, Statistics

"Classical" Data Mining

Motivation (Google example)

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web
- Even more to **do** something with the data

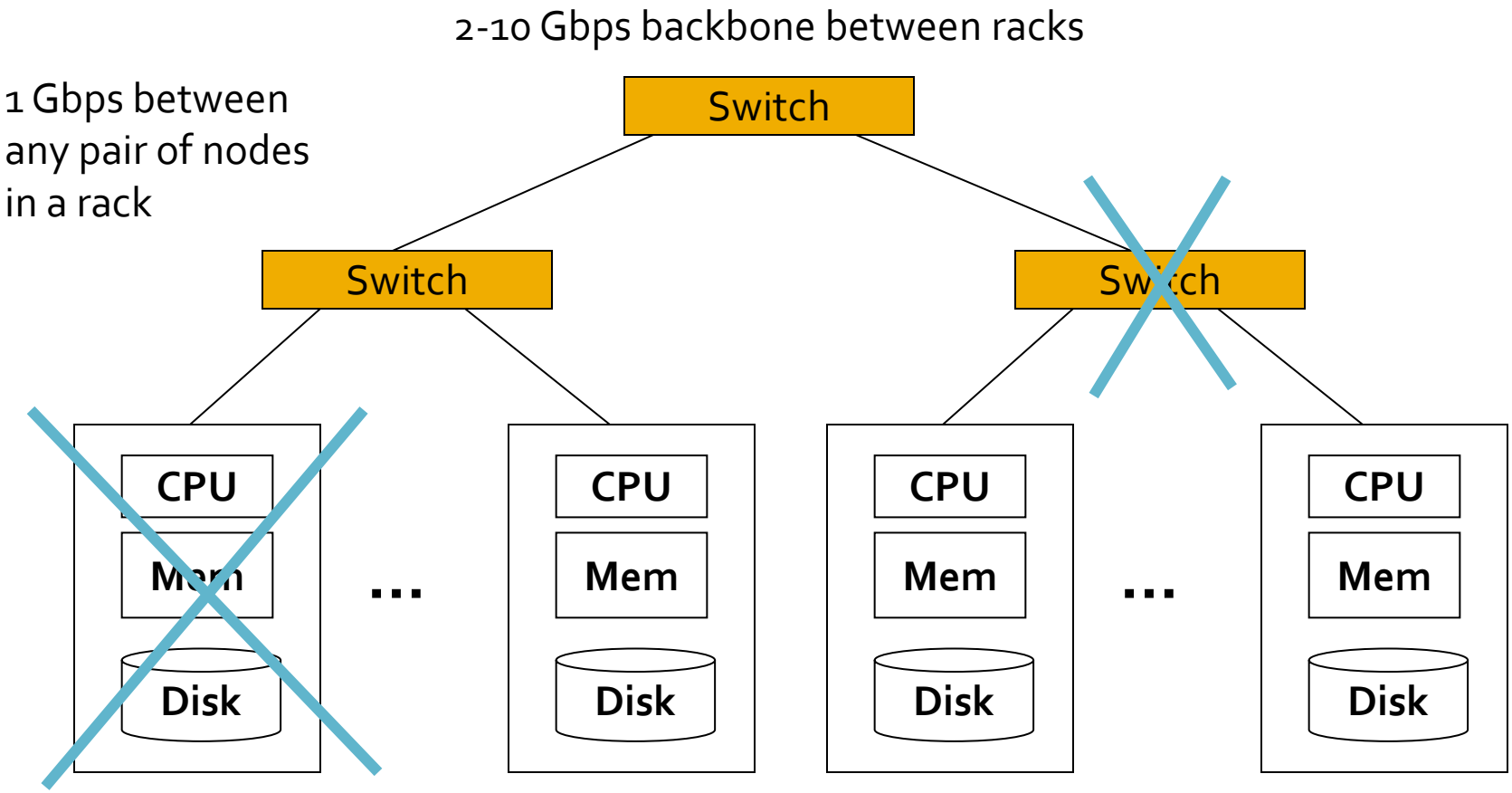
Commodity Clusters

- Web data sets are massive
  - Tens to hundreds of terabytes
- Cannot mine on a single server
- Standard architecture emerging:
  - Cluster of commodity Linux nodes
  - Gigabit ethernet interconnect
- How to organize computations on this architecture?
  - Mask issues such as hardware failure

Big computation – Big machines

- Traditional big-iron box (circa 2003)
  - 8 2GHz Xeons
  - 64GB RAM
  - 8TB disk
  - 758,000 USD
- Prototypical Google rack (circa 2003)
  - 176 2GHz Xeons
  - 176GB RAM
  - ~7TB disk
  - 278,000 USD
- In Aug 2006 Google had ~450,000 machines

Cluster Architecture



Each rack contains 16-64 nodes

M45: Open Academic Cluster

- Yahoo M45 cluster:
  - Datacenter in a Box (DiB)
  - 1000 nodes, 4000 cores, 3TB RAM, 1.5PB disk
  - High bandwidth connection to Internet
  - Located on Yahoo! campus
  - World's top 50 supercomputer

Large scale computing

- **Large scale computing** for **data mining** problems on **commodity hardware:**
  - PCs connected in a network
  - Process huge datasets on many computers
- Challenges:
  - How do you distribute computation?
  - Distributed/parallel programming is hard
  - Machines fail
- **Map-reduce** addresses all of the above
  - Google's computational/data manipulation model
  - Elegant way to work with big data

Implications

- Implications of such computing environment:
  - Single machine performance does not matter
    - Add more machines
  - Machines break:
    - One server may stay up 3 years (1,000 days)
    - If you have 1,0000 servers, expect to loose 1/day
- How can we make it easy to write distributed programs?

Idea and solution

- **Idea:**
  - Bring computation close to the data
  - Store files multiple times for reliability
- **Need:**
  - Programming model
    - Map-Reduce
  - Infrastructure – File system
    - Google: GFS
    - Hadoop: HDFS

Stable storage

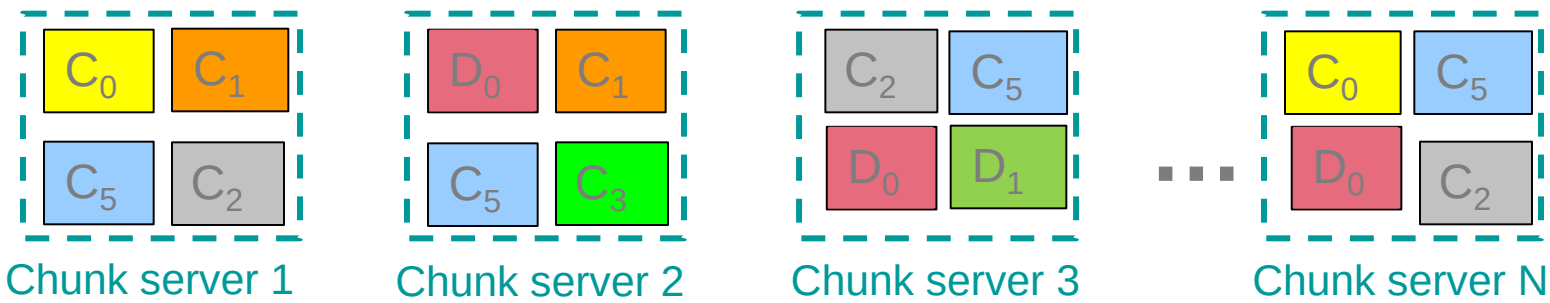
- Problem:
  - If nodes fail, how to store data persistently?
- Answer:
  - Distributed File System:
    - Provides global file namespace
    - Google GFS; Hadoop HDFS; Kosmix KFS
- Typical usage pattern
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

Distributed File System

- Chunk Servers:
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- Master node:
  - a.k.a. Name Nodes in Hadoop's HDFS
  - Stores metadata
  - Might be replicated
- Client library for file access:
  - Talks to master to find chunk servers
  - Connects directly to chunkservers to access data

Distributed File System

- Reliable distributed file system for petabyte scale
- Data kept in "chunks" spread across thousands of machines
- Each chunk replicated on different machines
  - Seamless recovery from disk or machine failure



Bring computation directly to the data!

Warm up: Word Count

- We have a large file of words:
  - one word per line
- Count the number of times each distinct word appears in the file
- Sample application:
  - Analyze web server logs to find popular URLs

Warm up: Word Count (2)

- Case 1:
  - Entire file fits in memory
- Case 2:
  - File too large for memory, but all <word, count> pairs fit in memory
- Case 3:
  - File on disk, too many distinct words to fit in memory:
    - `sort datafile | uniq -c`

Warm up: Word Count (3)

- Suppose we have a large corpus of documents
- Count occurrences of words:
  - `words(docs/*) | sort | uniq -c`
    - where `words` takes a file and outputs the words in it, one per a line
- Captures the essence of MapReduce
  - Great thing is it is naturally parallelizable

Map-Reduce: Overview

- Read a lot of data
- **Map:**
  - Extract something you care about
- Shuffle and Sort
- **Reduce:**
  - Aggregate, summarize, filter or transform
- Write the result

Outline stays the same, **map** and **reduce** change to fit the problem

More specifically

- Program specifies two primary methods:
  - Map(k,v) → <k', v'>*
  - Reduce(k', <v'>*) → <k', v''>*

- All values v' with same key k' are reduced together and processed in v' order

Map-Reduce: Word counting

Provided by the programmer

MAP:
reads input and produces a set of key value pairs

Group by key:
Collect all pairs with same key

Provided by the programmer

Reduce:
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/machine partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need to do to build any work station or habitat structure on the moon or Mars," said Allard Beutel.

Big document

(the, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....

(key, value)

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
...

(key, value)

(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
...

(key, value)

Only sequential reads

Word Count using MapReduce

```
map(key, value):
// key: document name; value: text of document
  for each word w in value:
     emit(w, 1)


reduce(key, values):
// key: a word; value: an iterator over counts
      result = 0
      for each count v in values:
            result += v
      emit(result)
```

Map-Reduce: Environment

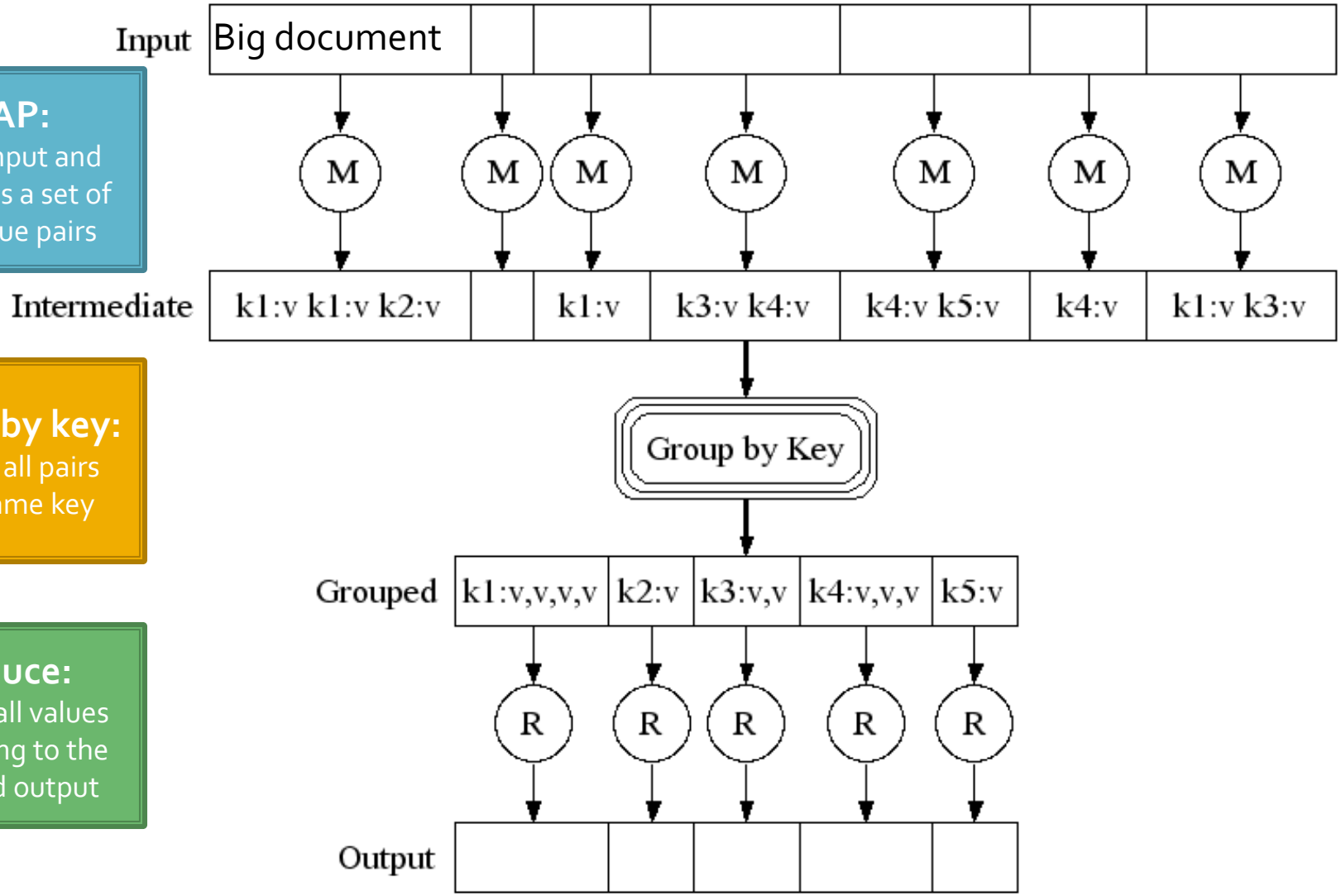
- Map-Reduce environment takes care of:
  - Partitioning the input data
  - Scheduling the program's execution across a set of machines
  - Handling machine failures
  - Managing required inter-machine communication
- Allows programmers without a PhD in parallel and distributed systems to use large distributed clusters

Map-Reduce: A diagram

MAP:
reads input and produces a set of key value pairs

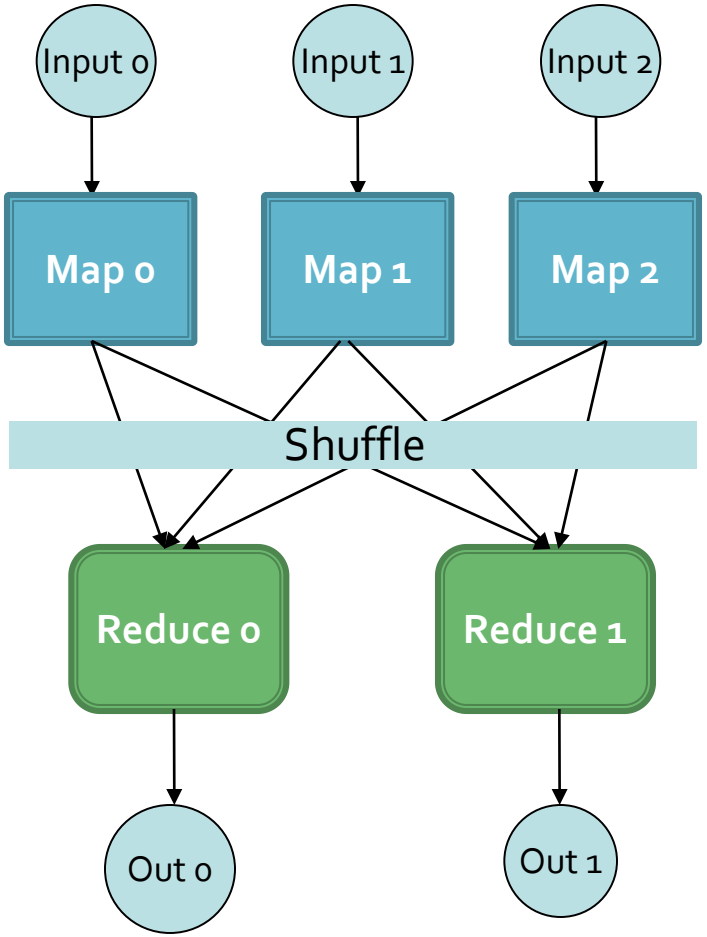
Group by key:
Collect all pairs with same key

Reduce:
Collect all values belonging to the key and output

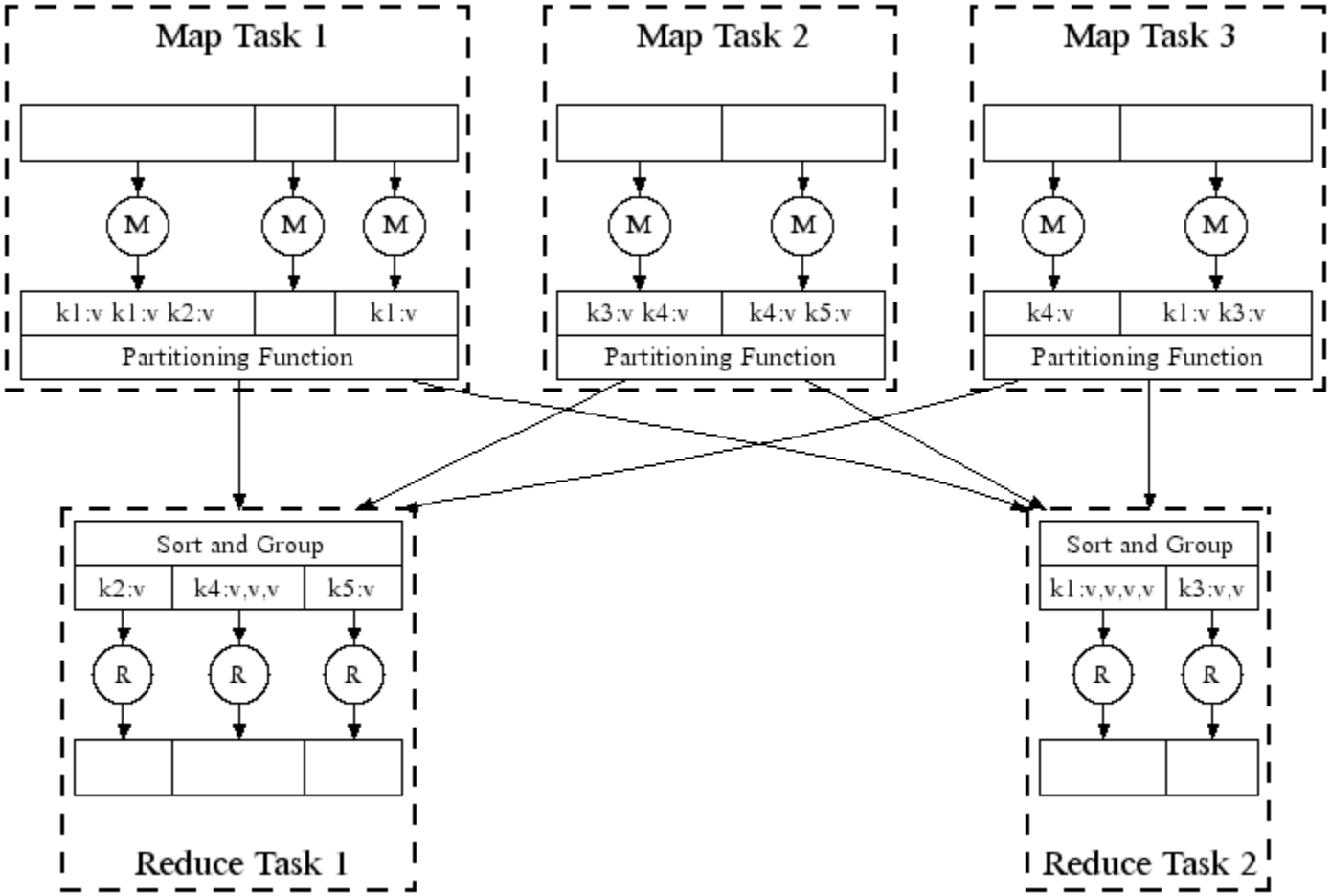


Map-Reduce

- Programmer specifies:
  - Map and Reduce and input files
- **Workflow:**
  - Read inputs as a set of key-value-pairs
  - **Map** transforms input kv-pairs into a new set of k'v'-pairs
  - Sorts & Shuffles the k'v'-pairs to output nodes
  - All k'v'-pairs with a given k' are sent to the same **reduce**
  - **Reduce** processes all k'v'-pairs grouped by key into new k''v''-pairs
  - Write the resulting pairs to files
- All phases are distributed with many tasks doing the work



Map-Reduce: in Parallel



Data flow

- Input and final output are stored on a distributed file system:
  - Scheduler tries to schedule map tasks "close" to physical storage location of input data
- Intermediate results are stored on local FS of map and reduce workers
- Output is often input to another map reduce task

Coordination

- Master data structures:
  - Task status: (idle, in-progress, completed)
  - Idle tasks get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its R intermediate files, one for each reducer
  - Master pushes this info to reducers
- Master pings workers periodically to detect failures

Failures

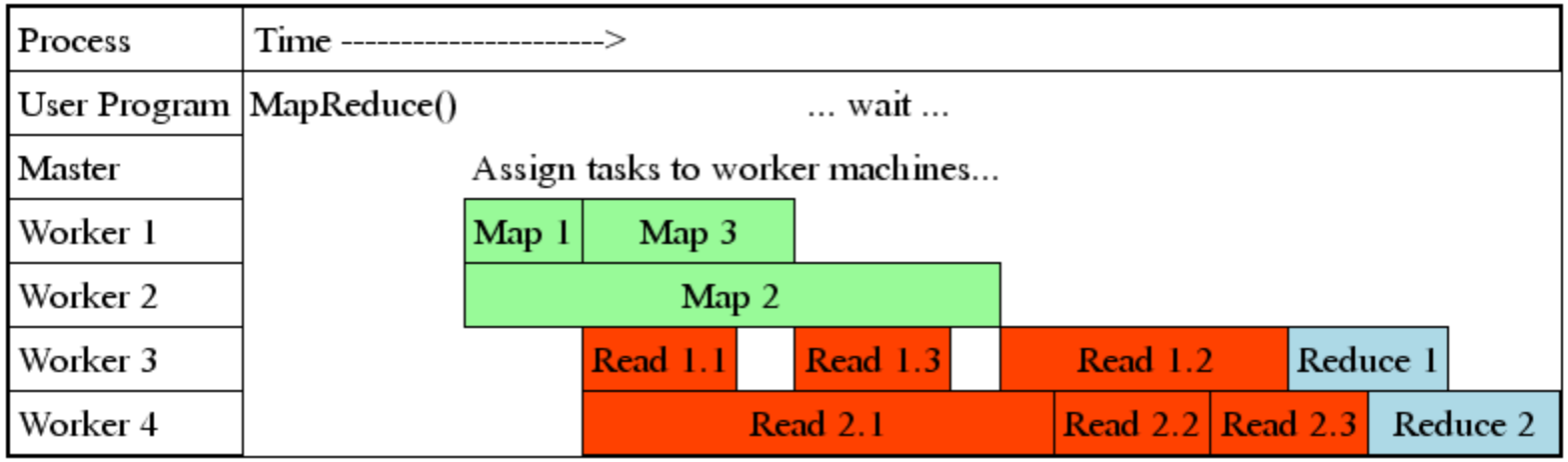
- Map worker failure
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker
- Reduce worker failure
  - Only in-progress tasks are reset to idle
- Master failure
  - MapReduce task is aborted and client is notified

How many Map and Reduce jobs?

- M map tasks, R reduce tasks
- Rule of thumb:
  - Make M and R much larger than the number of nodes in cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds recovery from worker failure
- Usually R is smaller than M
  - because output is spread across R files

Task Granularity & Pipelining

- Fine granularity tasks: map tasks >> machines
  - Minimizes time for fault recovery
  - Can pipeline shuffling with map execution
  - Better dynamic load balancing

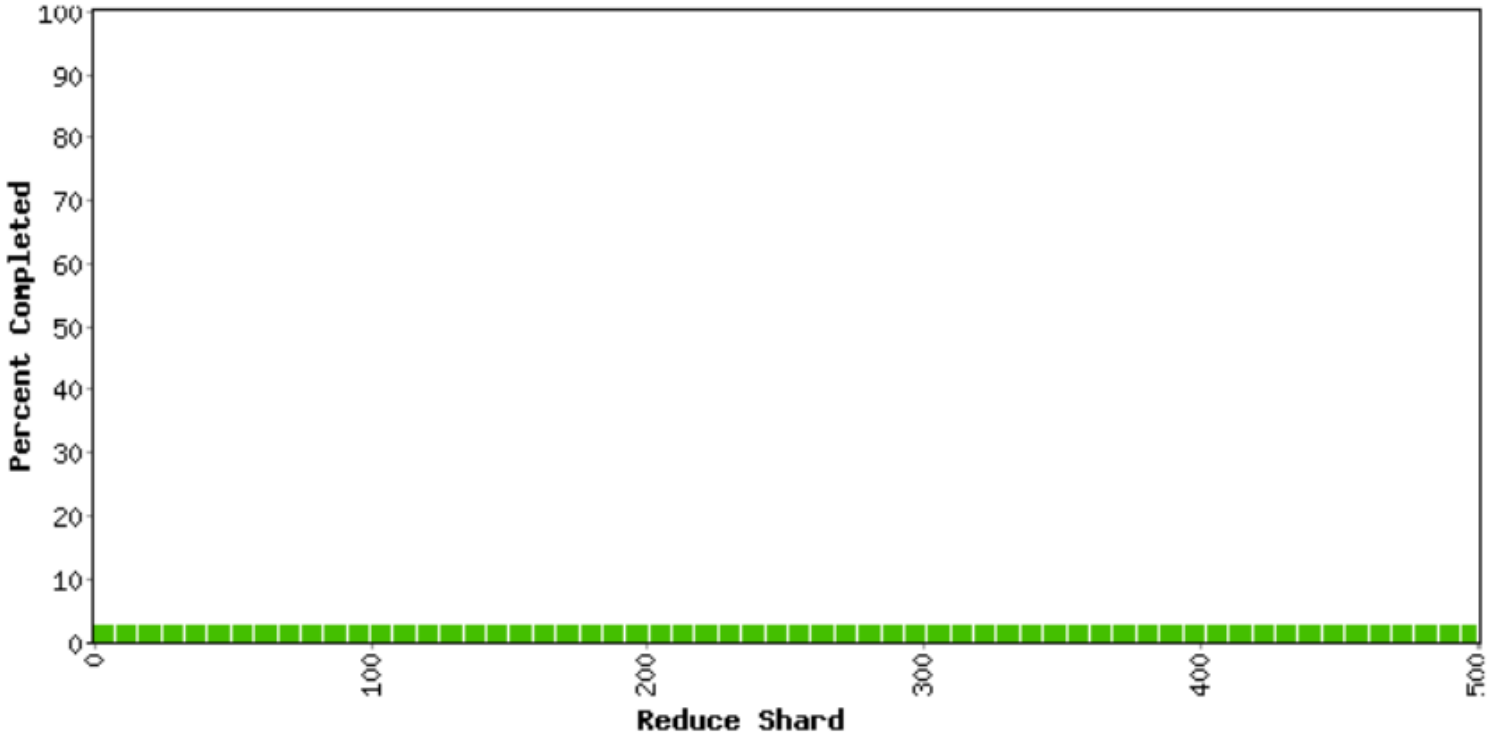


MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	0	323	878934.6	1314.4	717.0
Shuffle	500	0	323	717.0	0.0	0.0
Reduce	500	0	0	0.0	0.0	0.0



Counters

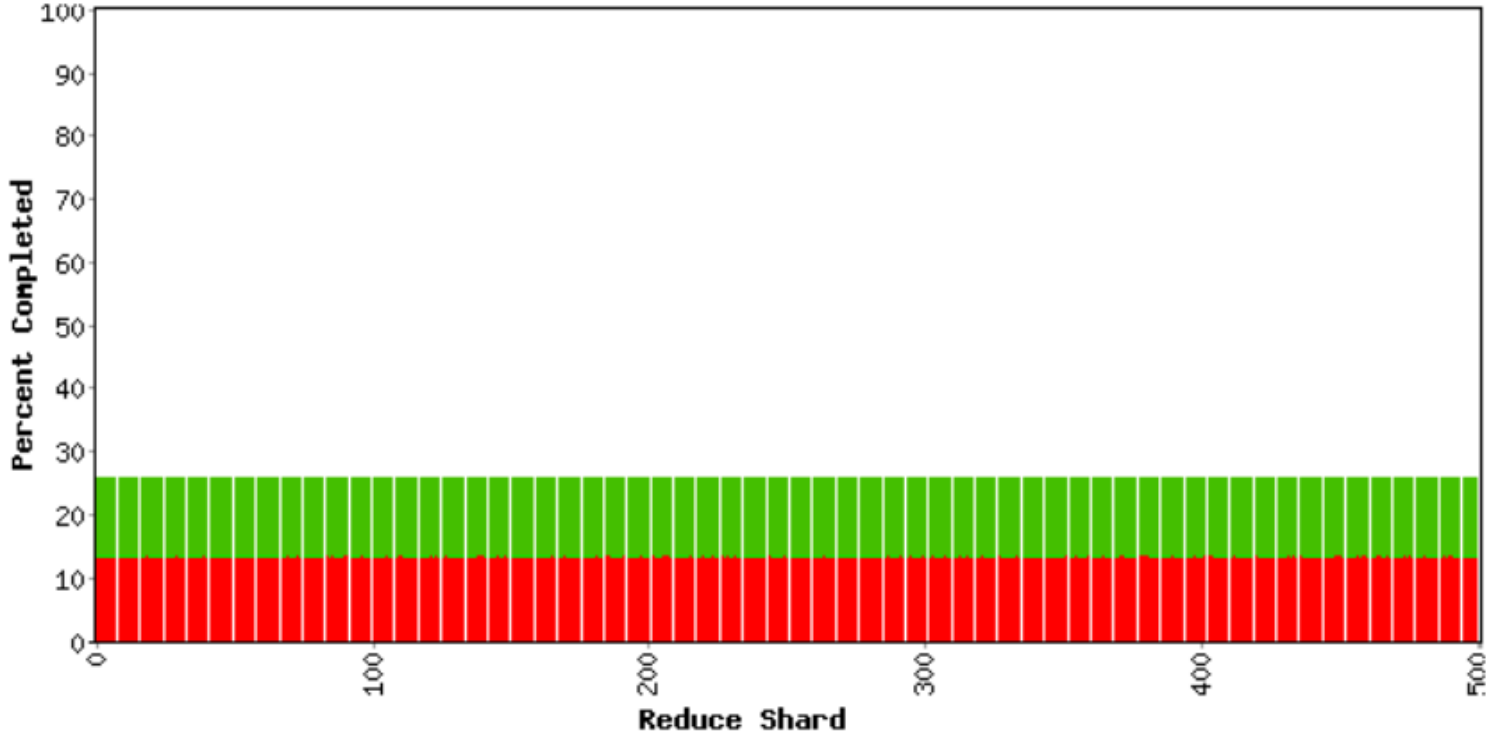
Variable	Minute
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-	506631

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	1857	1707	878934.6	191995.8	113936.6
Shuffle	500	0	500	113936.6	57113.7	57113.7
Reduce	500	0	0	57113.7	0.0	0.0



Counters

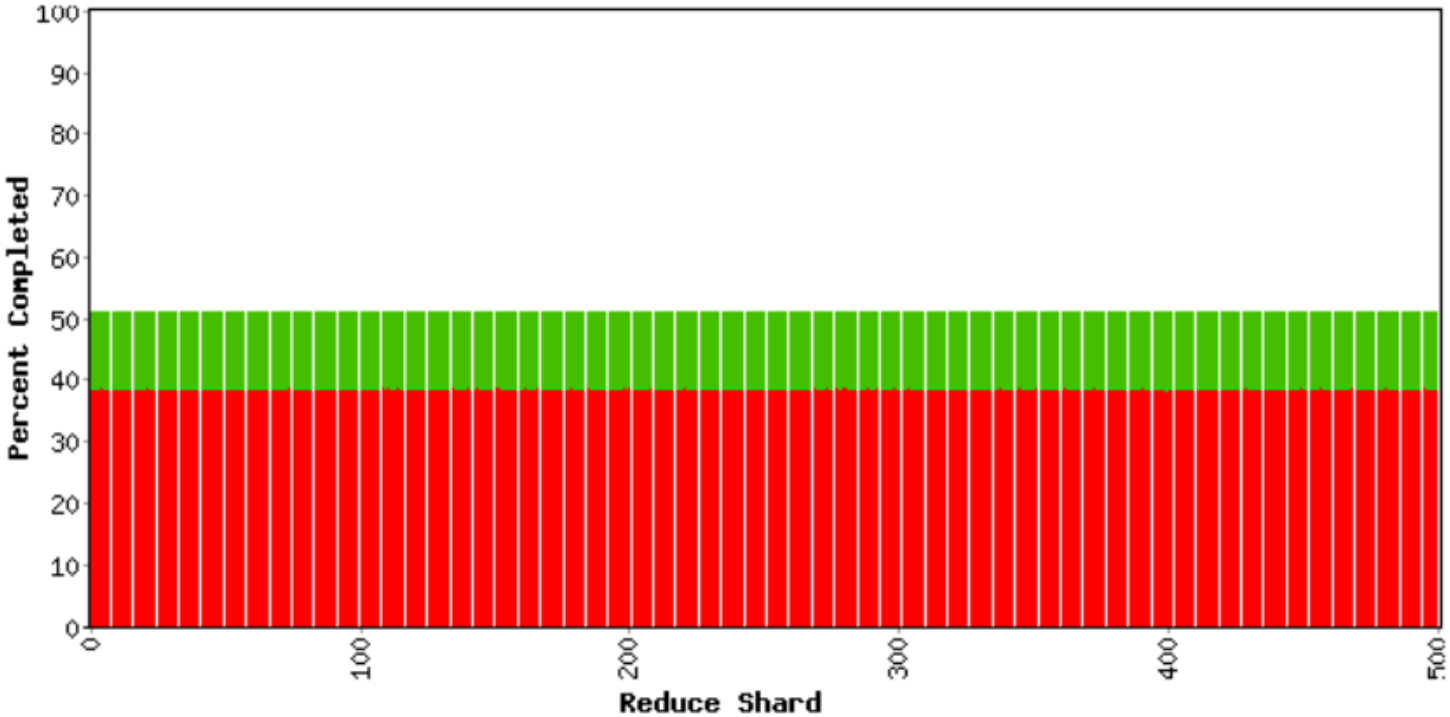
Variable	Minute
Mapped (MB/s)	699.1
Shuffle (MB/s)	349.5
Output (MB/s)	0.0
doc-index-hits	5004411944
docs-indexed	17290135
dups-in-index-merge	0
mr-operator-calls	17331371
mr-operator-outputs	17290135

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	5354	1707	878934.6	406020.1	241058.2
Shuffle	500	0	500	241058.2	196362.5	196362.5
Reduce	500	0	0	196362.5	0.0	0.0



Counters

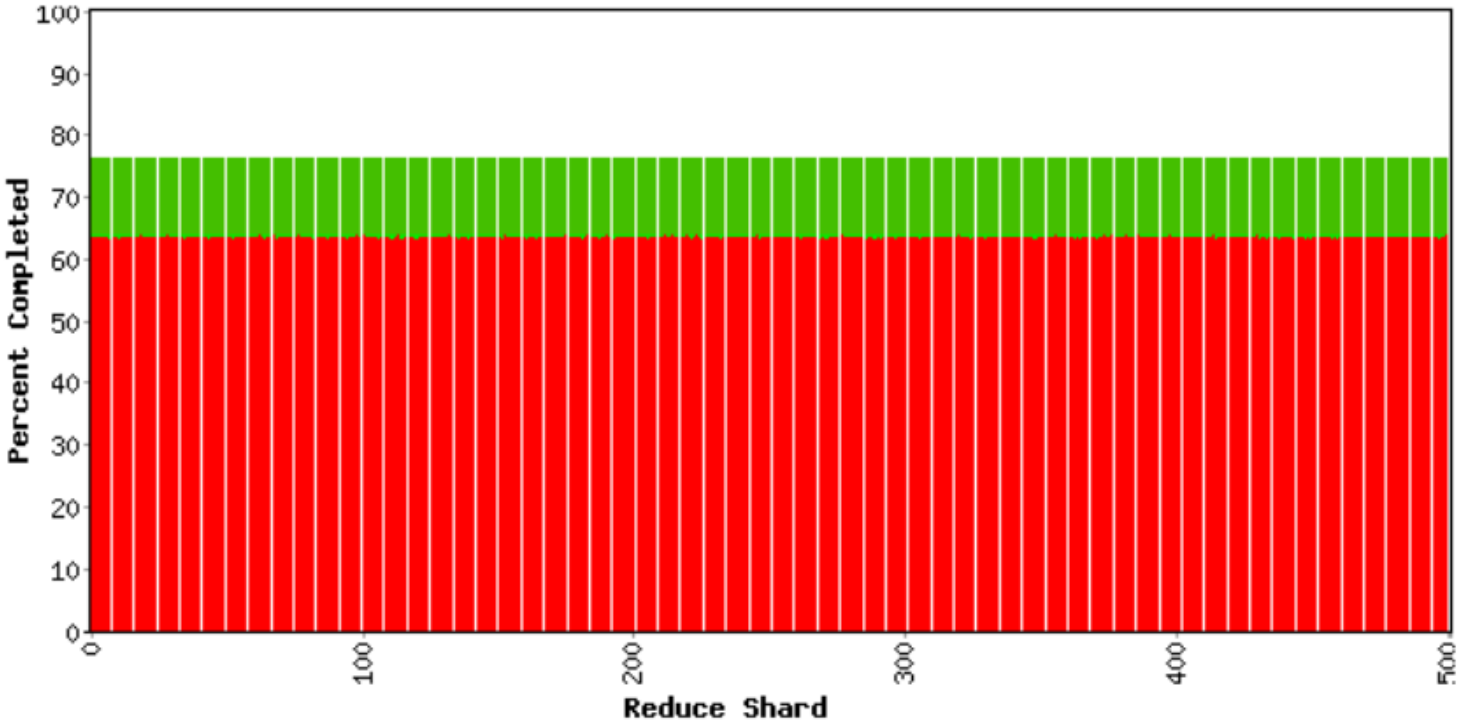
Variable	Minute
Mapped (MB/s)	704.4
Shuffle (MB/s)	371.9
Output (MB/s)	0.0
doc-index-hits	5000364228
docs-indexed	17300709
dups-in-index-merge	0
mr-operator-calls	17342493
mr-operator-outputs	17300709

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	8841	1707	878934.6	621608.5	369459.8
Shuffle	500	0	500	369459.8	326986.8	326986.8
Reduce	500	0	0	326986.8	0.0	0.0



Counters

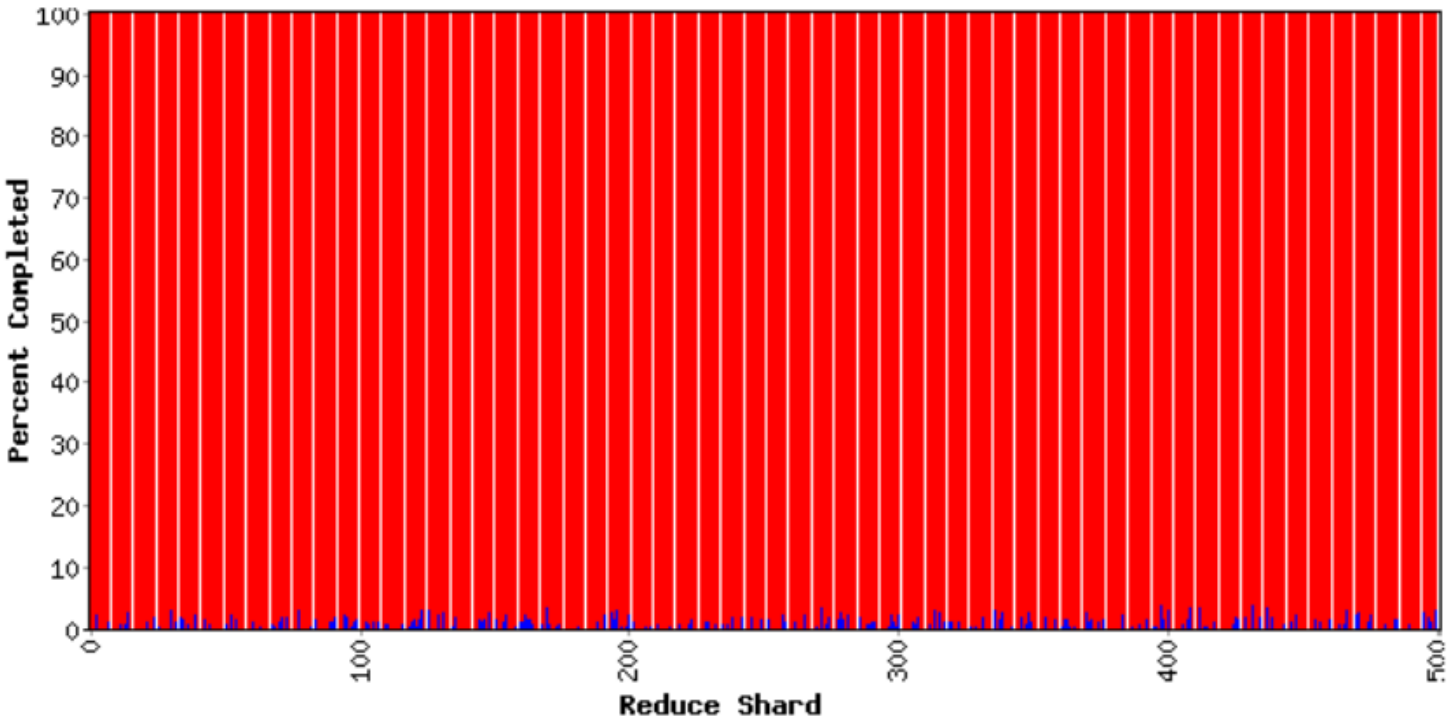
Variable	Minute
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	195	305	523499.2	523389.6	523389.6
Reduce	500	0	195	523389.6	2685.2	2742.6



Counters

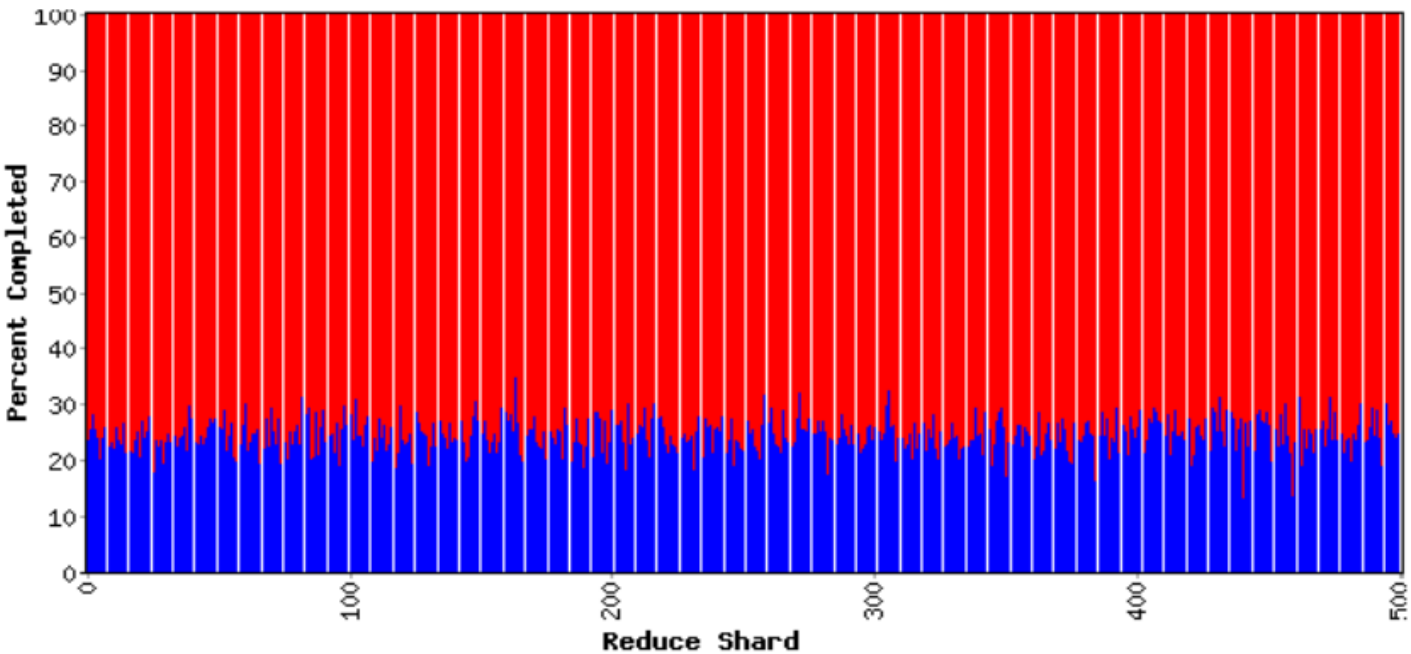
Variable	Minute	
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	105
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	133837.8	136929.6



Counters

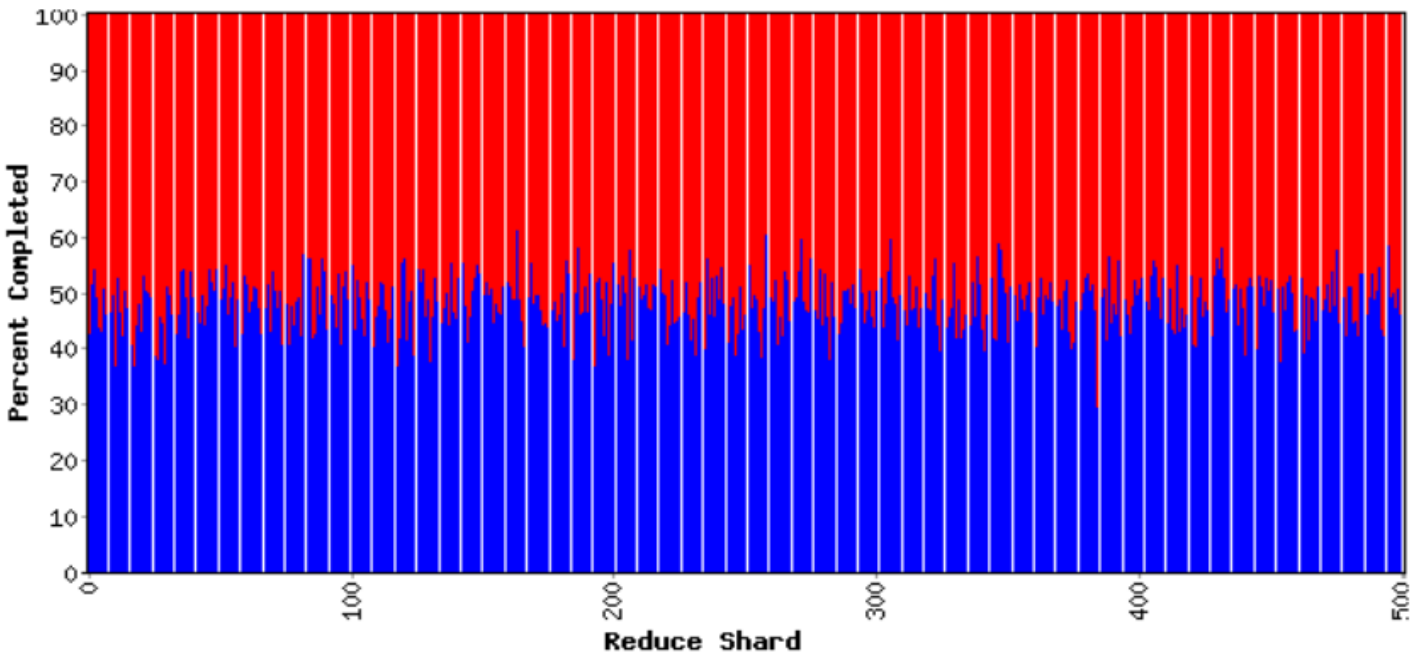
Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.1
Output (MB/s)	1238.8
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51738599
mr-merge-outputs	51738599

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	263283.3	269351.2



Counters

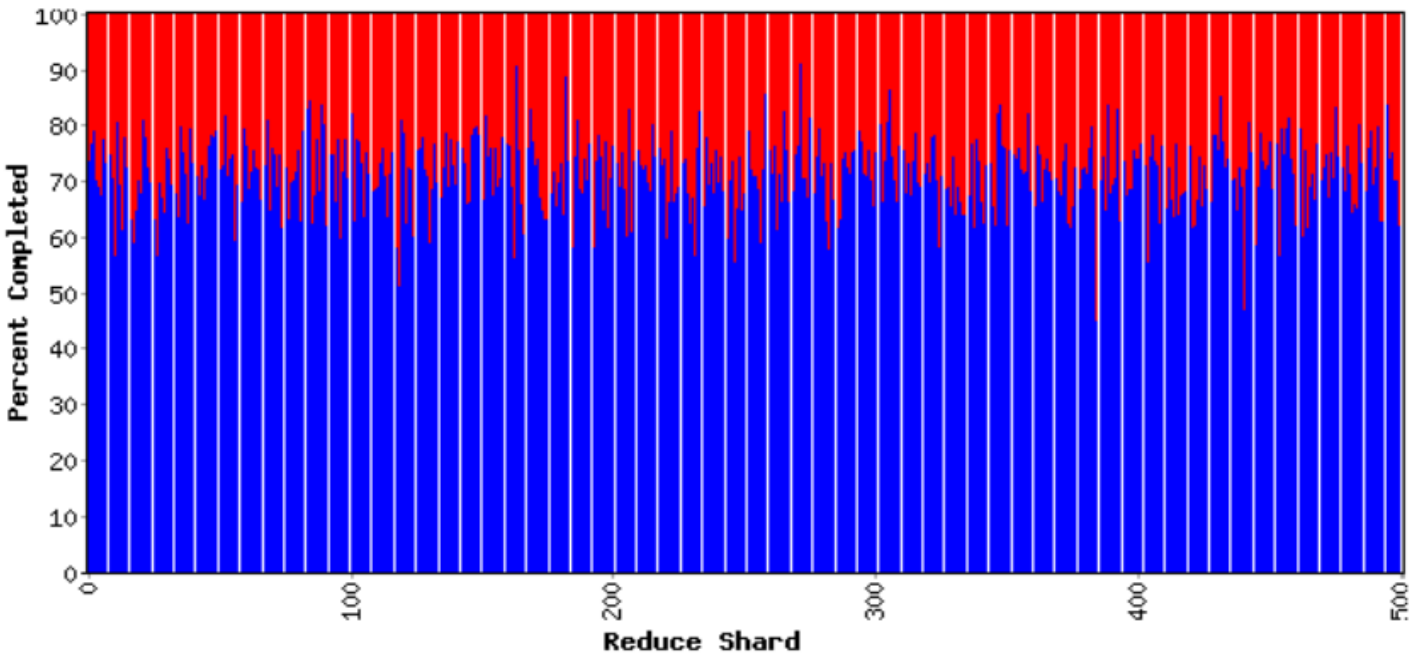
Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1225.1
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51842100
mr-merge-outputs	51842100

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	390447.6	399457.2



Counters

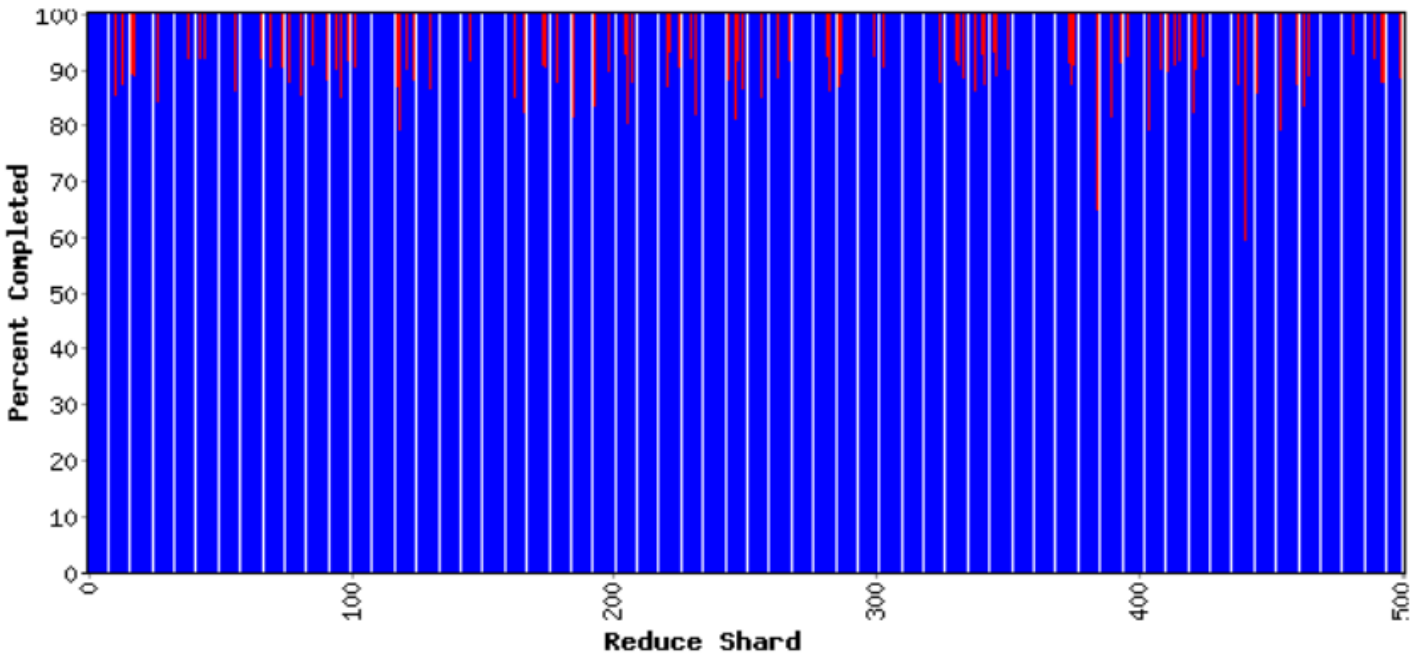
Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1222.0
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51640600
mr-merge-outputs	51640600

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	520468.6	520468.6
Reduce	500	406	94	520468.6	512265.2	514373.3



Counters

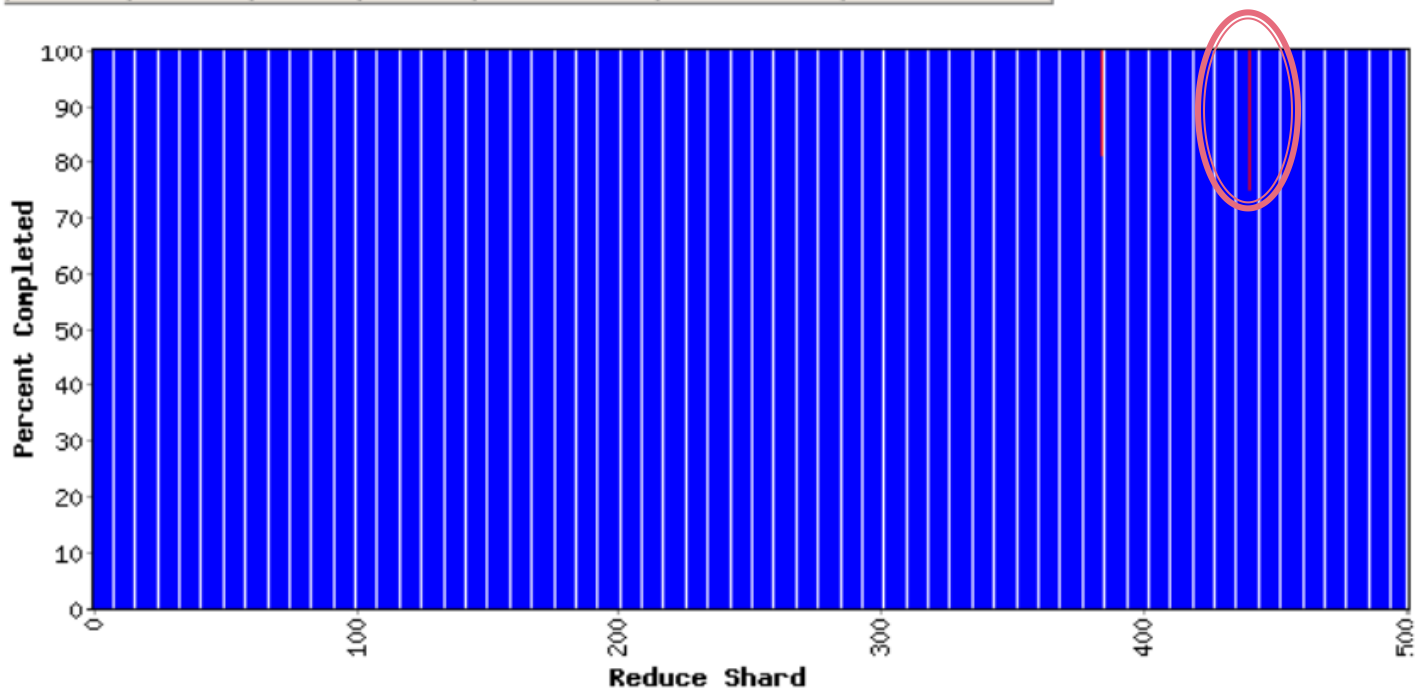
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	849.5	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	35083350	
mr-merge-outputs	35083350	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519781.8	519781.8
Reduce	500	498	2	519781.8	519394.7	519440.7



Counters

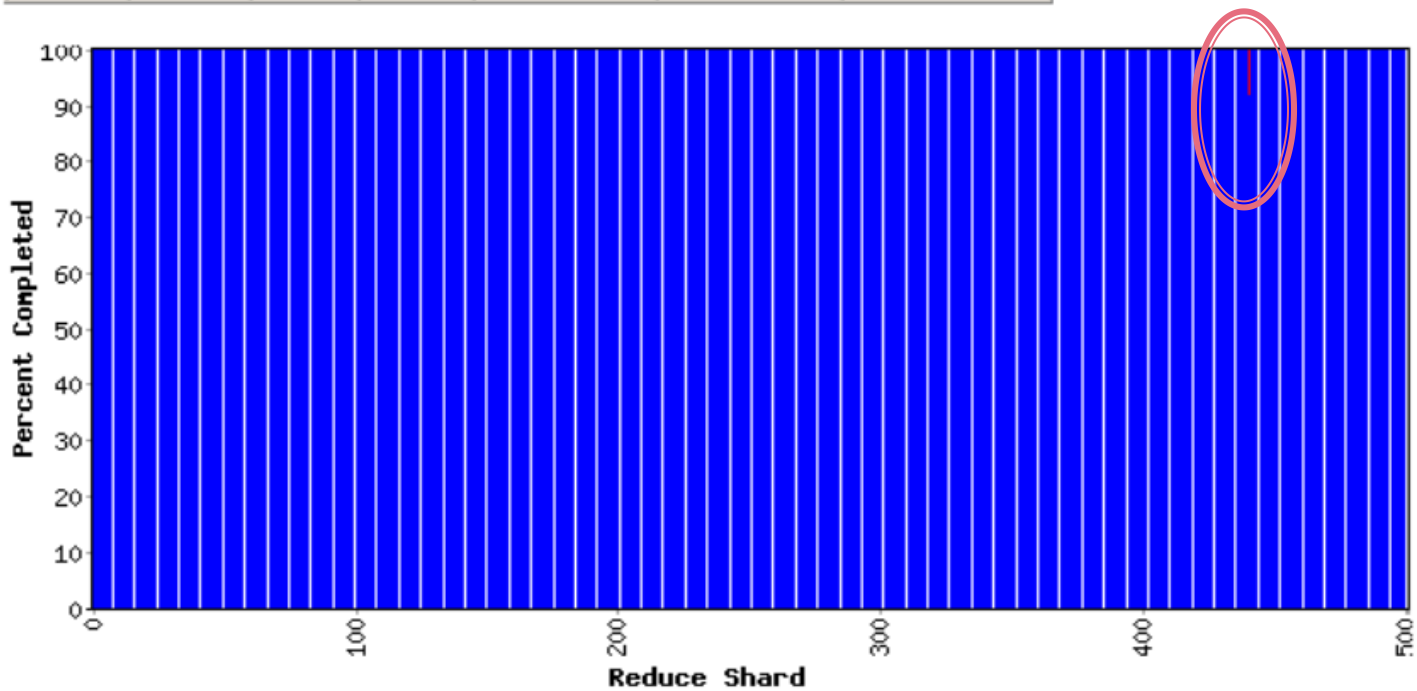
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	1056
docs-indexed	0	3
dups-in-index-merge	0	
mr-merge-calls	394792	3
mr-merge-outputs	394792	3

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519774.3	519774.3
Reduce	500	499	1	519774.3	519735.2	519764.0



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1.9	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	73442	
mr-merge-outputs	73442	

Problems suited for Map-Reduce

- Want to simulate disease spreading in a network
- **Input**
  - Each line: node id, virus parameters
- **Map**
  - Reads a line of input and simulate the virus
  - Output: triplets (node id, virus id, hit time)
- **Reduce**
  - Collect the node IDs and see which nodes are most vulnerable

Example 2: Language model

- Statistical machine translation:
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents
- Easy with MapReduce:
  - Map:
    - Extract (5-word sequence, count) from document
  - Reduce:
    - Combine counts

Example 3: Host size

- Suppose we have a large web corpus
- Look at the metadata file
  - Lines of the form (URL, size, date, …)
- For each host, find the total number of bytes
  - i.e., the sum of the page sizes for all URLs from that host
- Other examples:
  - Link analysis and graph processing
  - Machine Learning algorithms

Implementations

- Google
  - Not available outside Google
- Hadoop
  - An open-source implementation in Java
  - Uses HDFS for stable storage
  - Download: http://lucene.apache.org/hadoop/
- Aster Data
  - Cluster-optimized SQL Database that also implements MapReduce

Cloud Computing

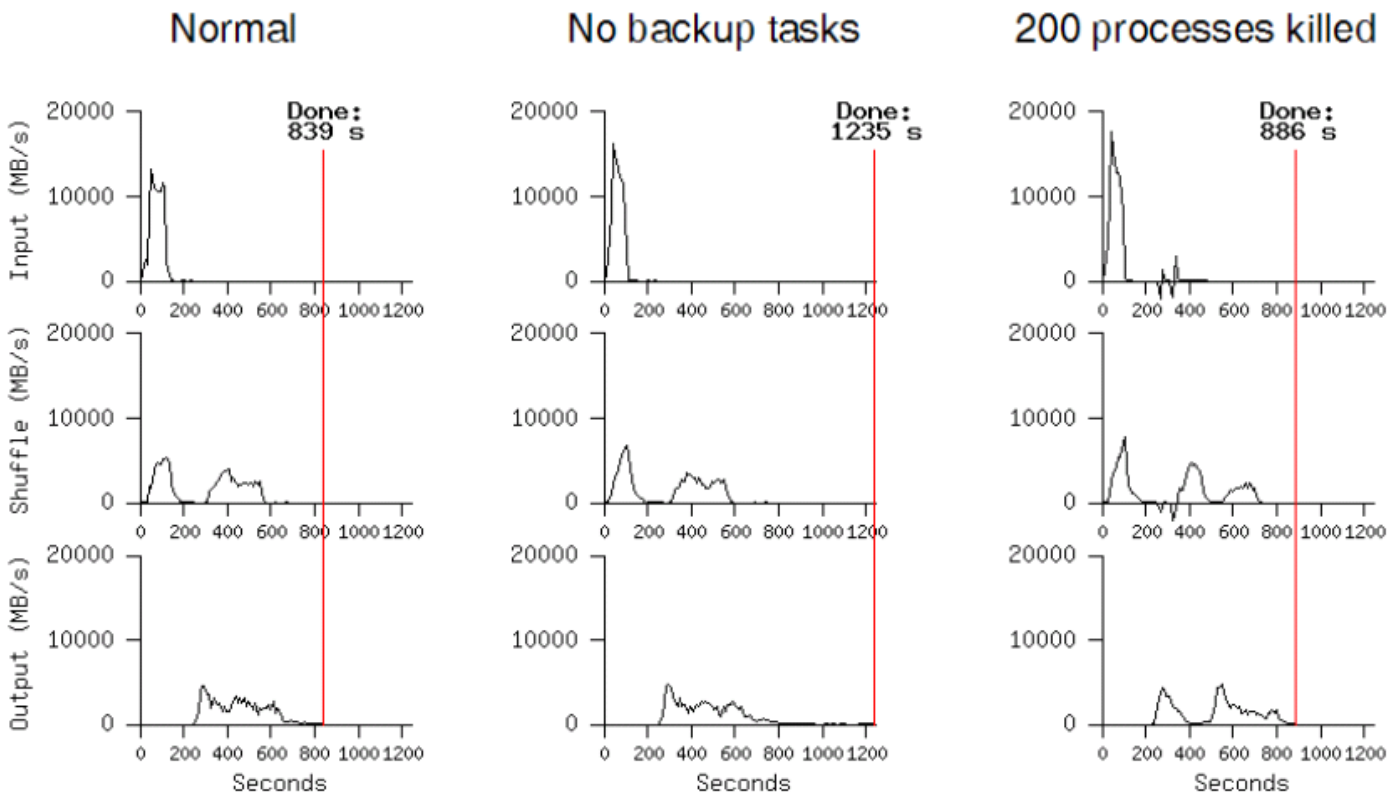
- Ability to rent computing by the hour
  - Additional services e.g., persistent storage
- Amazon's "Elastic Compute Cloud" (EC2)
- Aster Data and Hadoop can both be run on EC2
- For CS345 (offered next quarter) Amazon will provide free access for the class

Refinement: Backup tasks

- Problem:
  - Slow workers significantly lengthen the job completion time:
    - Other jobs on the machine
    - Bad disks
    - Weird things
- Solution:
  - Near end of phase, spawn backup copies of tasks
    - Whichever one finishes first "wins"
- Effect:
  - Dramatically shortens job completion time

Refinements: Backup tasks

- Backup tasks reduce job time
- System deals with failures



Refinements: Combiners

- Often a map task will produce many pairs of the form (k,v1), (k,v2), … for the same key k
  - E.g., popular words in Word Count
- Can save network time by pre-aggregating at mapper:
  - combine(k1, list(v1)) → v2
  - Usually same as the reduce function
- Works only if reduce function is commutative and associative

Refinements: Partition Function

- Inputs to map tasks are created by contiguous splits of input file
- Reduce needs to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function:
  - hash(key) mod R
- Sometimes useful to override:
  - E.g., hash(hostname(URL)) mod R ensures URLs from a host end up in the same output file

Reading

- Jeffrey Dean and Sanjay Ghemawat,

  MapReduce: Simplified Data Processing on Large Clusters
  http://labs.google.com/papers/mapreduce.html

- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, **The Google File System**
  http://labs.google.com/papers/gfs.html

Resources

- **Hadoop Wiki**
  - Introduction
    - http://wiki.apache.org/lucene-hadoop/
  - Getting Started
    - http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop
  - Map/Reduce Overview
    - http://wiki.apache.org/lucene-hadoop/HadoopMapReduce
    - http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses
  - Eclipse Environment
    - http://wiki.apache.org/lucene-hadoop/EclipseEnvironment
- **Javadoc**
  - http://lucene.apache.org/hadoop/docs/api/

Resources

- **Releases from Apache download mirrors**
  - http://www.apache.org/dyn/closer.cgi/lucene/hadoop/
- **Nightly builds of source**
  - http://people.apache.org/dist/lucene/hadoop/nightly/
- **Source code from subversion**
  - http://lucene.apache.org/hadoop/version_control.html

Further reading

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as River's distributed queues
  - River ['99]