Introduction to MapReduce

Instructor: Dr. Weikuan Yu

Computer Sci. & Software Eng.

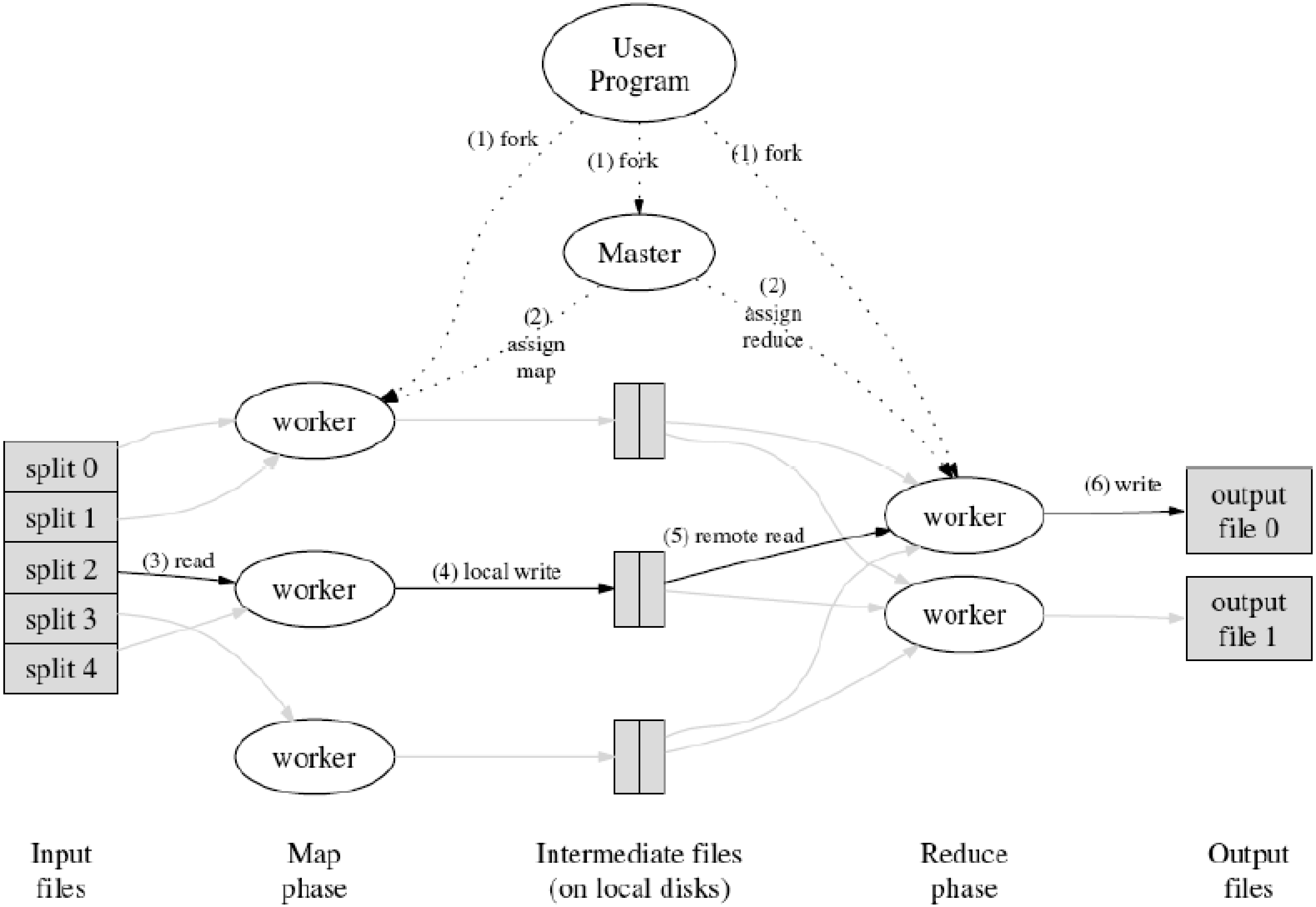
Before MapReduce…

- Large scale data processing was difficult!
  - Managing hundreds or thousands of processors
  - Managing parallelization and distribution
  - I/O Scheduling
  - Status and monitoring
  - Fault/crash tolerance
- MapReduce provides all of these, easily!
  - Introduction based on Google’s paper.

MapReduce Overview

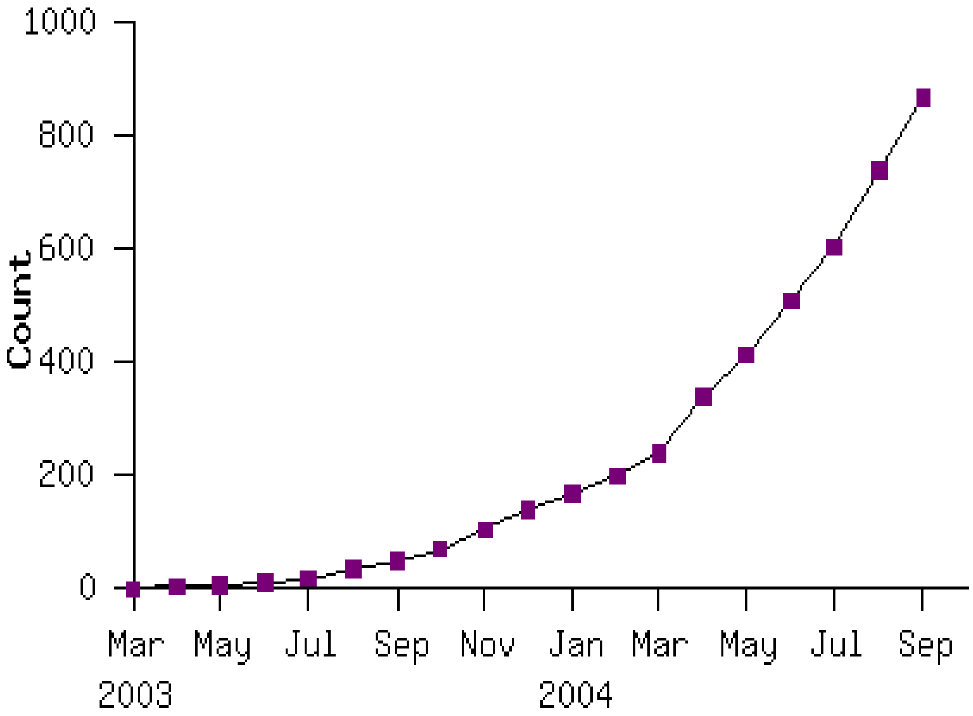
- What is it?
  - Programming model used by Google
  - A combination of the Map and Reduce models with an associated implementation
  - Used for processing and generating large data sets
- How does it solve our previously mentioned problems?
  - MapReduce is highly scalable and can be used across many computers.
  - Many small machines can be used to process jobs that normally could not be processed by a large machine.

Map-Reduce Framework



MapReduce Usage

- Large-Scale Data Processing
  - Can make use of 1000s of CPUs
  - Avoid the hassle of ***managing*** parallelization
- Provide a complete run-time system
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates
- User Growth at Google (2004)



MapReduce Basic Ingredients

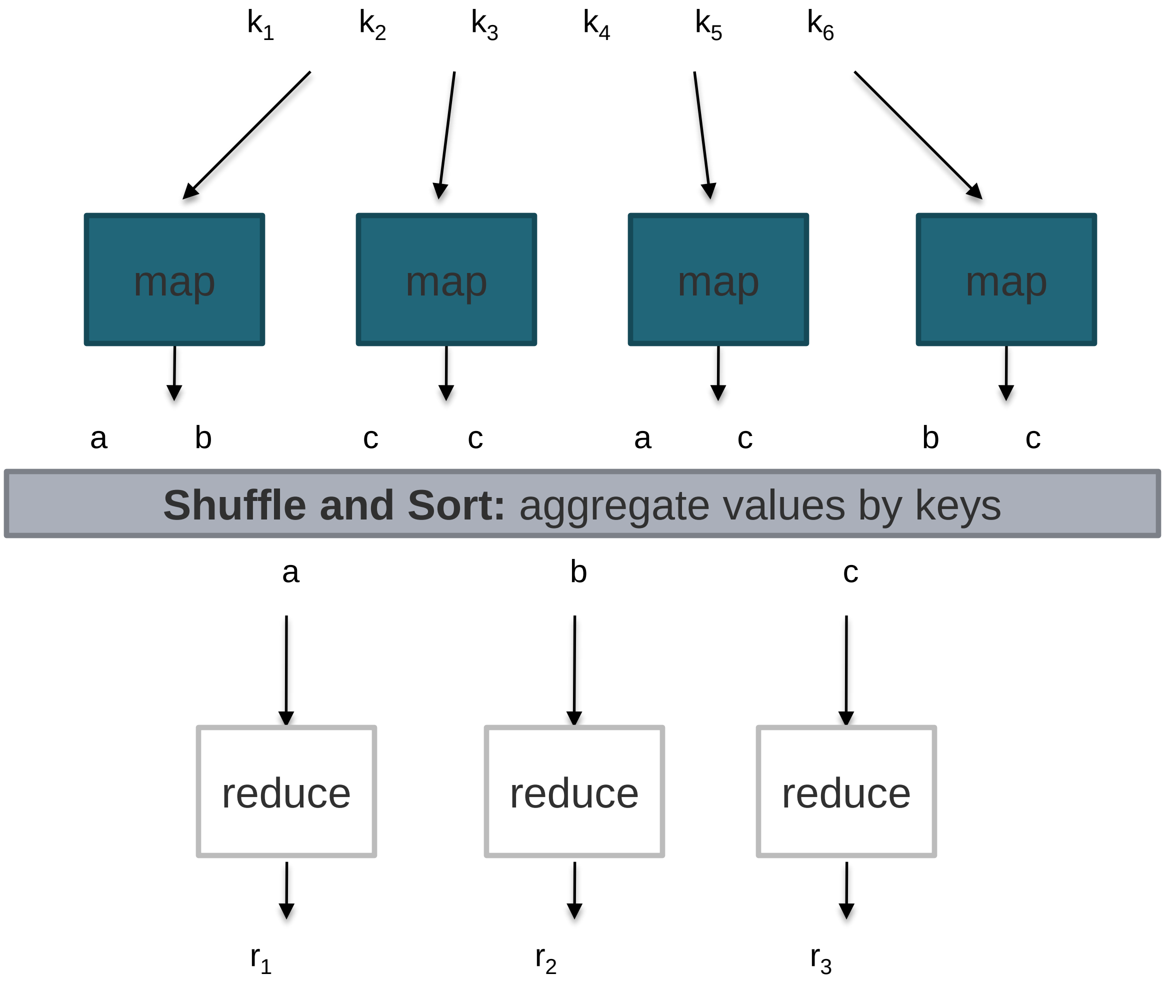
- Programmers specify two functions:

  map (k, v) → <k', v'>*

  reduce (k', v') → <k', v'>*

  – All values with the same key are sent to the same reducer

- The execution framework handles everything else…



Shuffle and Sort: aggregate values by keys

MapReduce

- Programmers specify two functions:

  map (k, v) → <k’, v’>*

  reduce (k’, v’) → <k’, v’>*

  – All values with the same key are reduced together

- The execution framework handles everything else…

- Not quite…usually, programmers also specify:

  partition (k’, number of partitions) → partition for k’

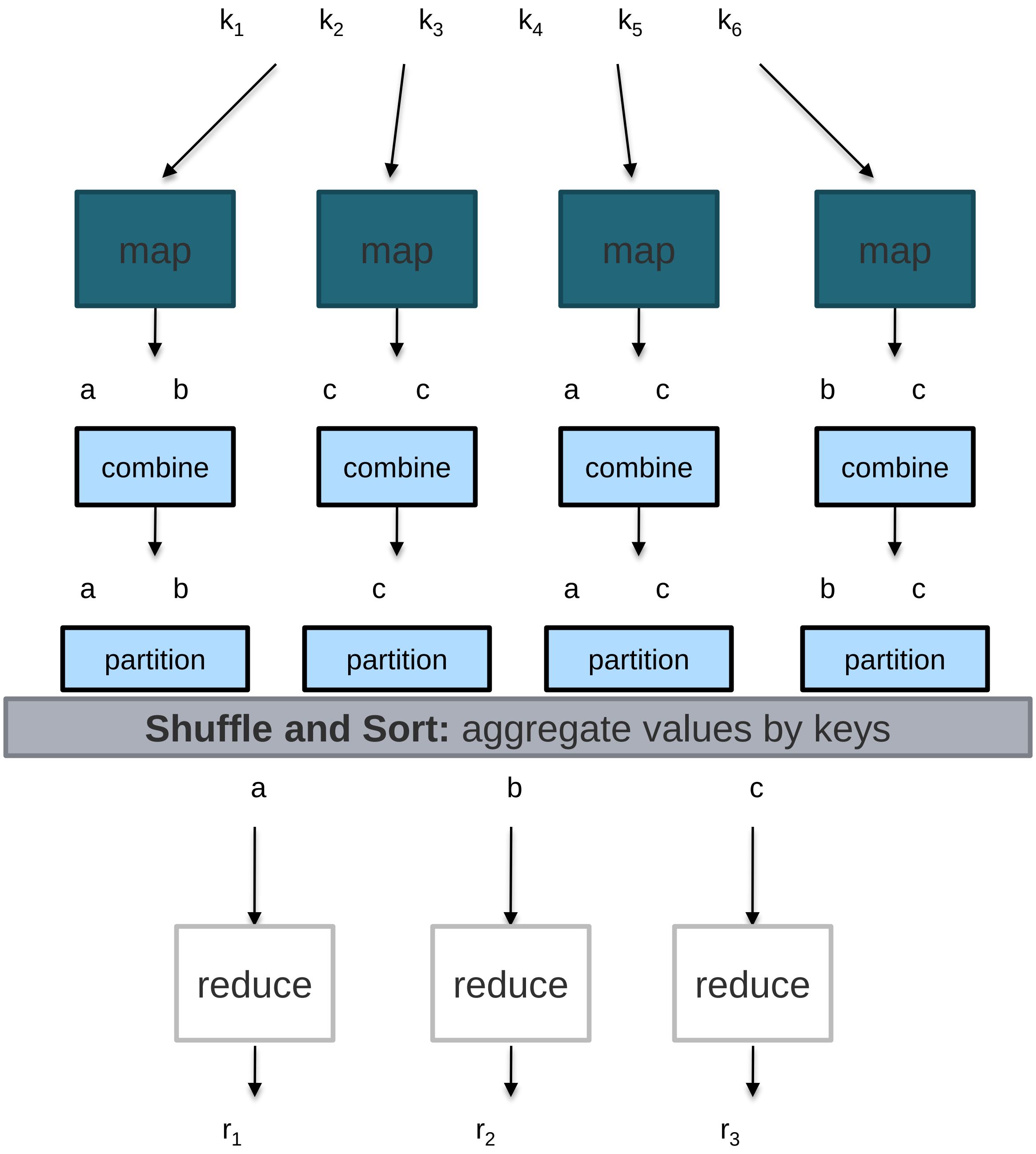
  – Often a simple hash of the key, e.g., hash(k’) mod n

  – Divides up key space for parallel reduce operations

  combine (k’, v’) → <k’, v’>*

  – Mini-reducers that run in memory after the map phase

  – Used as an optimization to reduce network traffic



k_1 k_2 k_3 k_4 k_5 k_6

map map map map

a b c c a c b c

combine combine combine combine

a b c a c b c

partition partition partition partition

Shuffle and Sort: aggregate values by keys

a b c

reduce reduce reduce

r_1 r_2 r_3

Map Abstraction

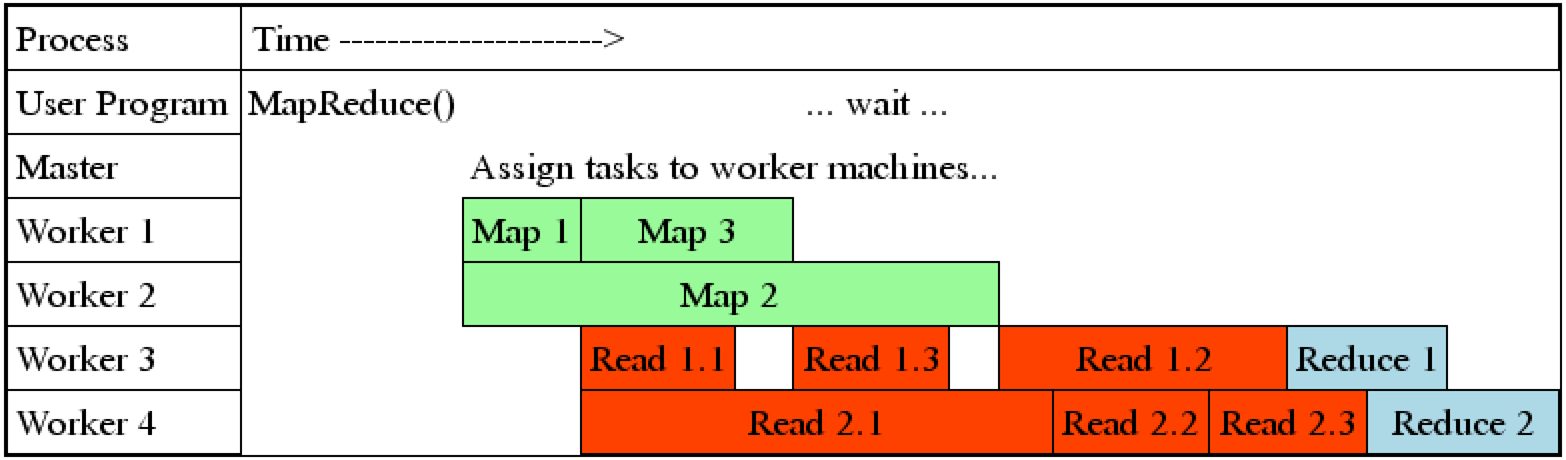
- Inputs a key/value pair
  - Key is a reference to the input value
  - Value is the data set on which to operate
- Evaluation
  - Function defined by user
  - Applies to every value in value input
    - Might need to parse input
- Produces a new list of key/value pairs
  - Can be different type from input pair

Reduce Abstraction

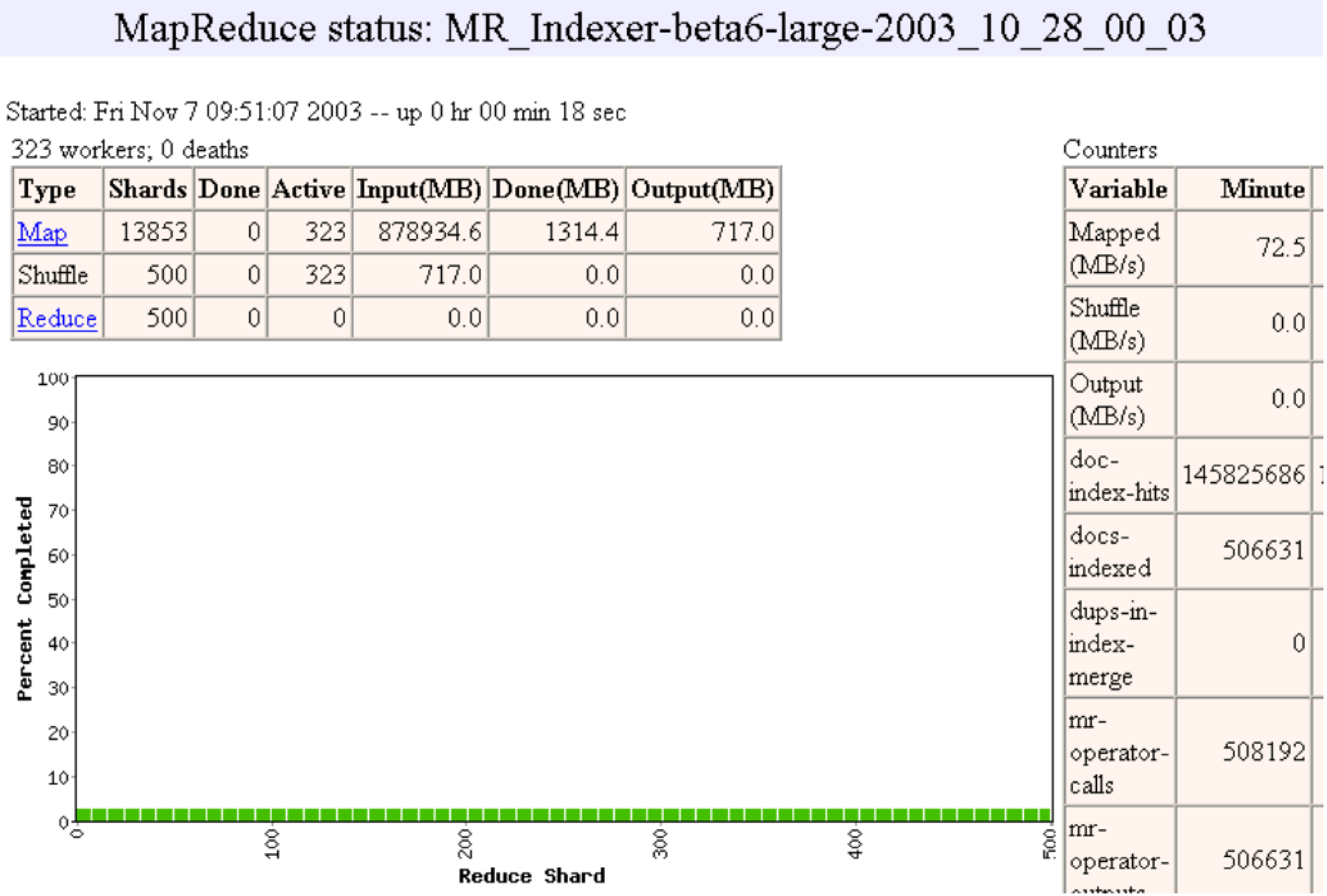
- Typically a function that:
  - Starts with a large number of key/value pairs
    - One key/value for each word in all files being grepped (including multiple entries for the same word)
  - Ends with very few key/value pairs
    - One key/value for each unique word across all the files with the number of instances summed into this entry
- Broken up so a given worker works with input of the same key.

Task Granularity and Pipelining

- Fine granularity tasks: map tasks >> machines
  - Minimizes time for fault recovery
  - Can pipeline shuffling with map execution
  - Better dynamic load balancing
- Often use 200,000 map & 5000 reduce tasks
- Running on 2000 machines



Process	Time ---------------------->
User Program	MapReduce() ... wait ...
Master	Assign tasks to worker machines...
Worker 1	Map 1 Map 3
Worker 2	Map 2
Worker 3	Read 1.1 Read 1.3 Read 1.2 Reduce 1
Worker 4	Read 2.1 Read 2.2 Read 2.3 Reduce 2



MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

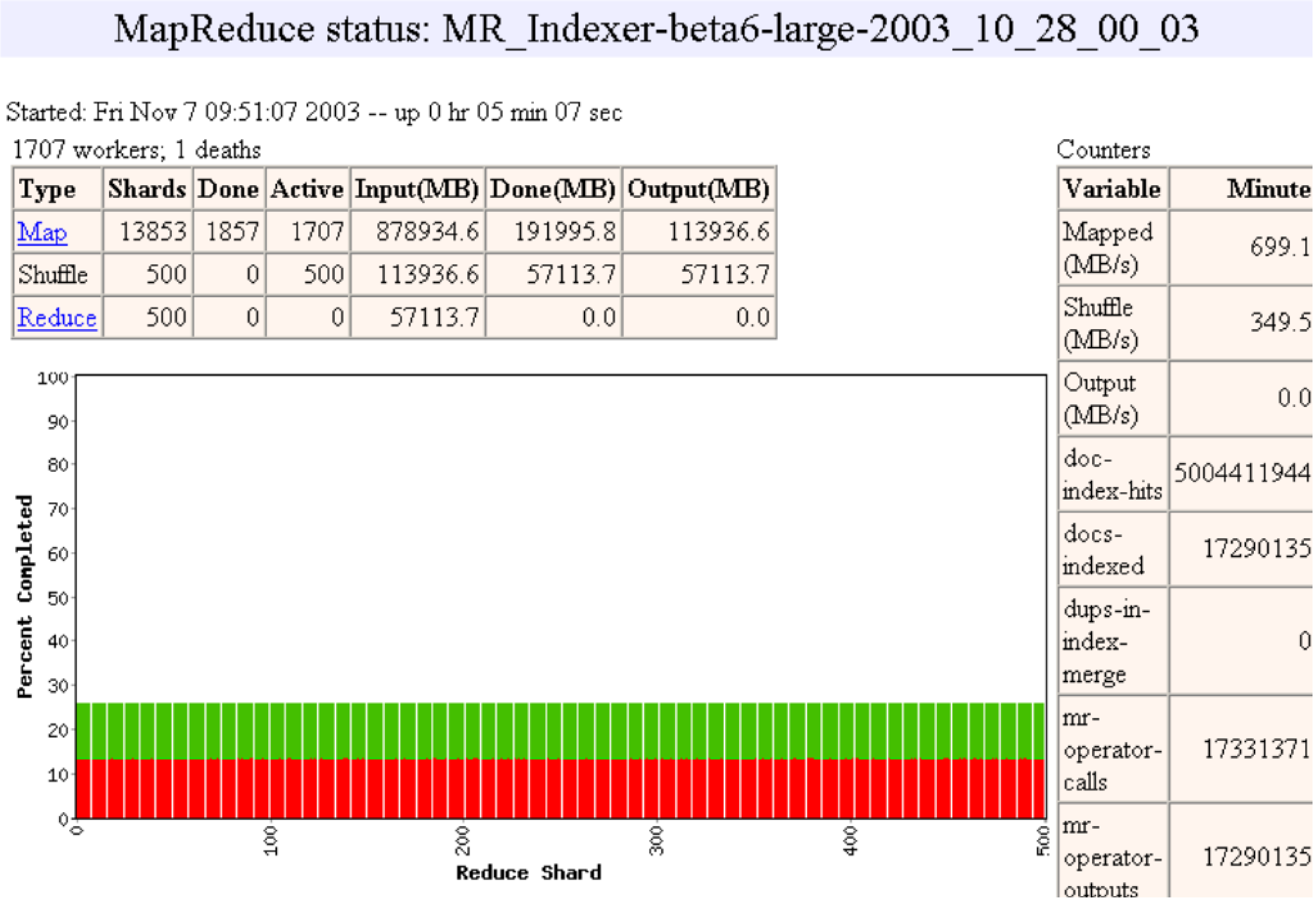
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

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	1
docs-indexed	506631	
dups-in-index-merge	0	
mr-operator-calls	508192	
mr-operator-outputs	506631	



MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

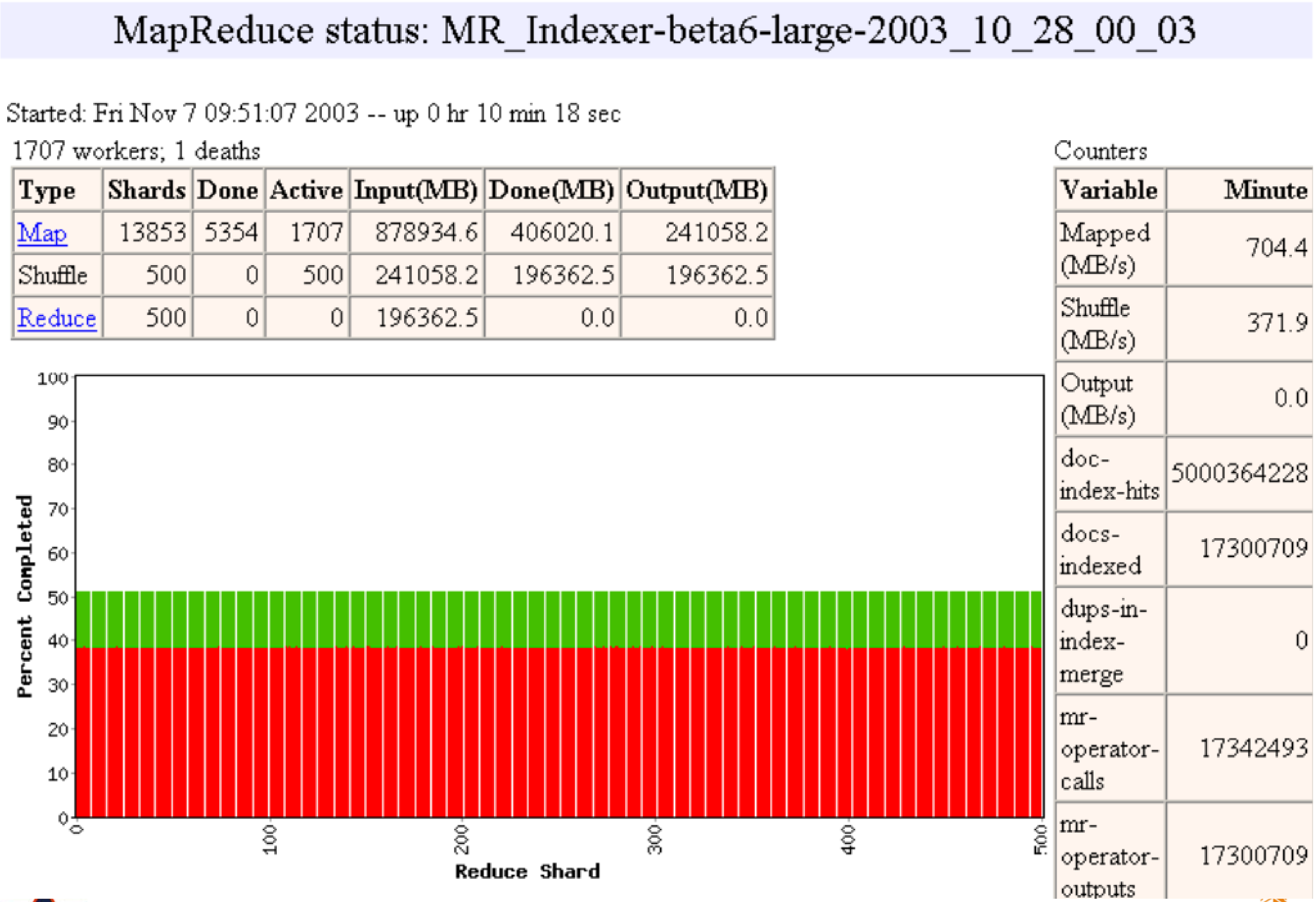
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

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

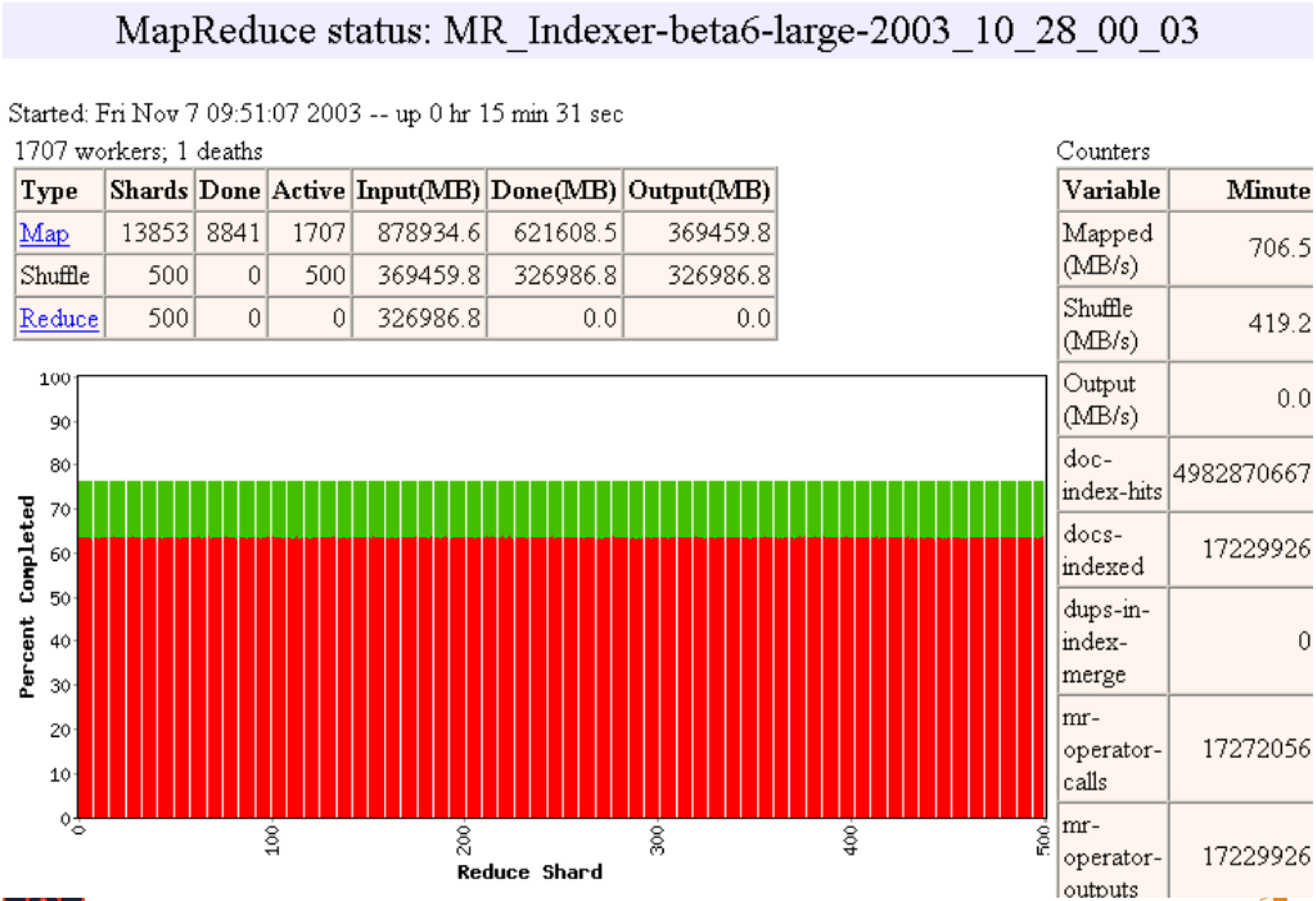
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

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

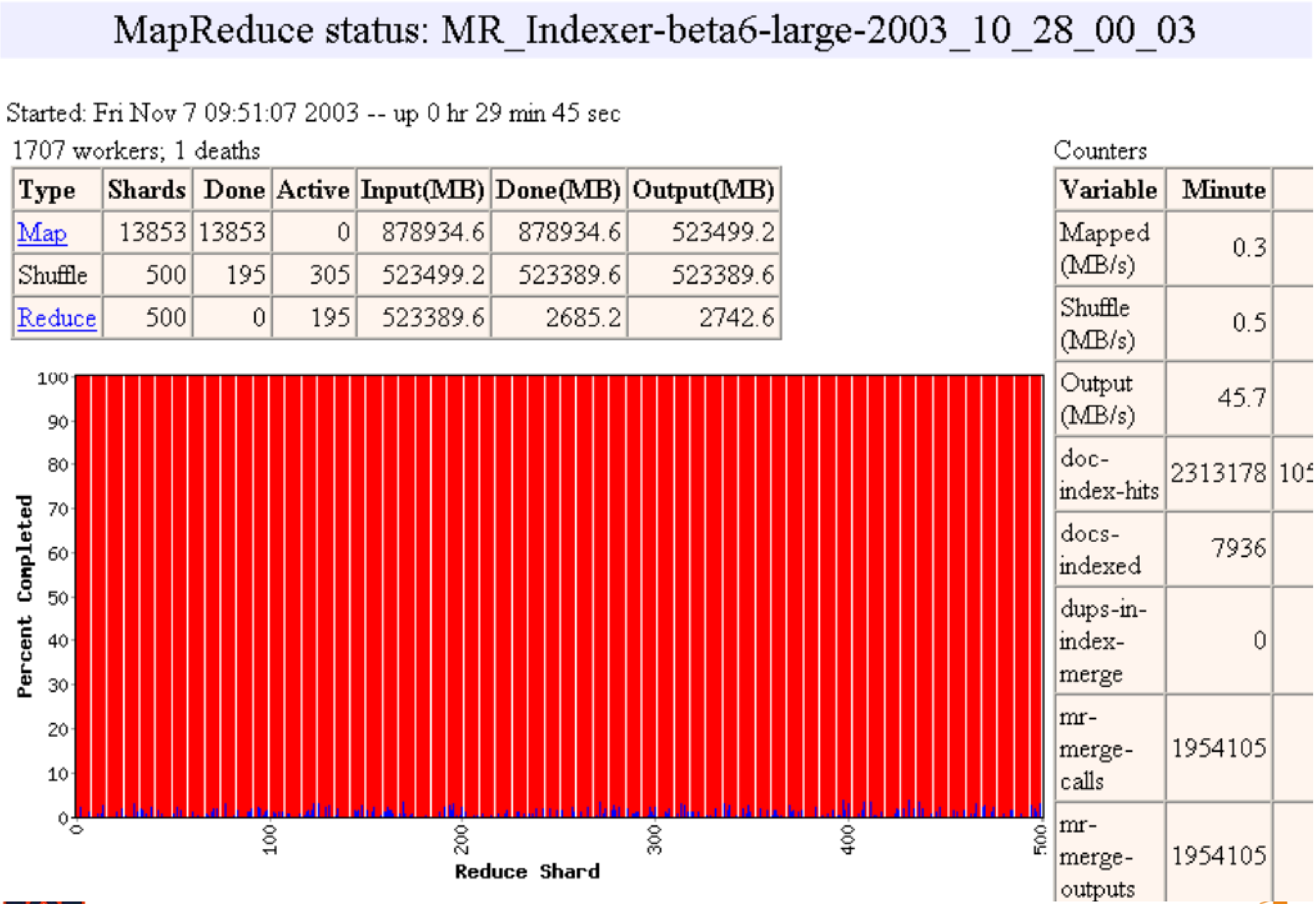
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

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

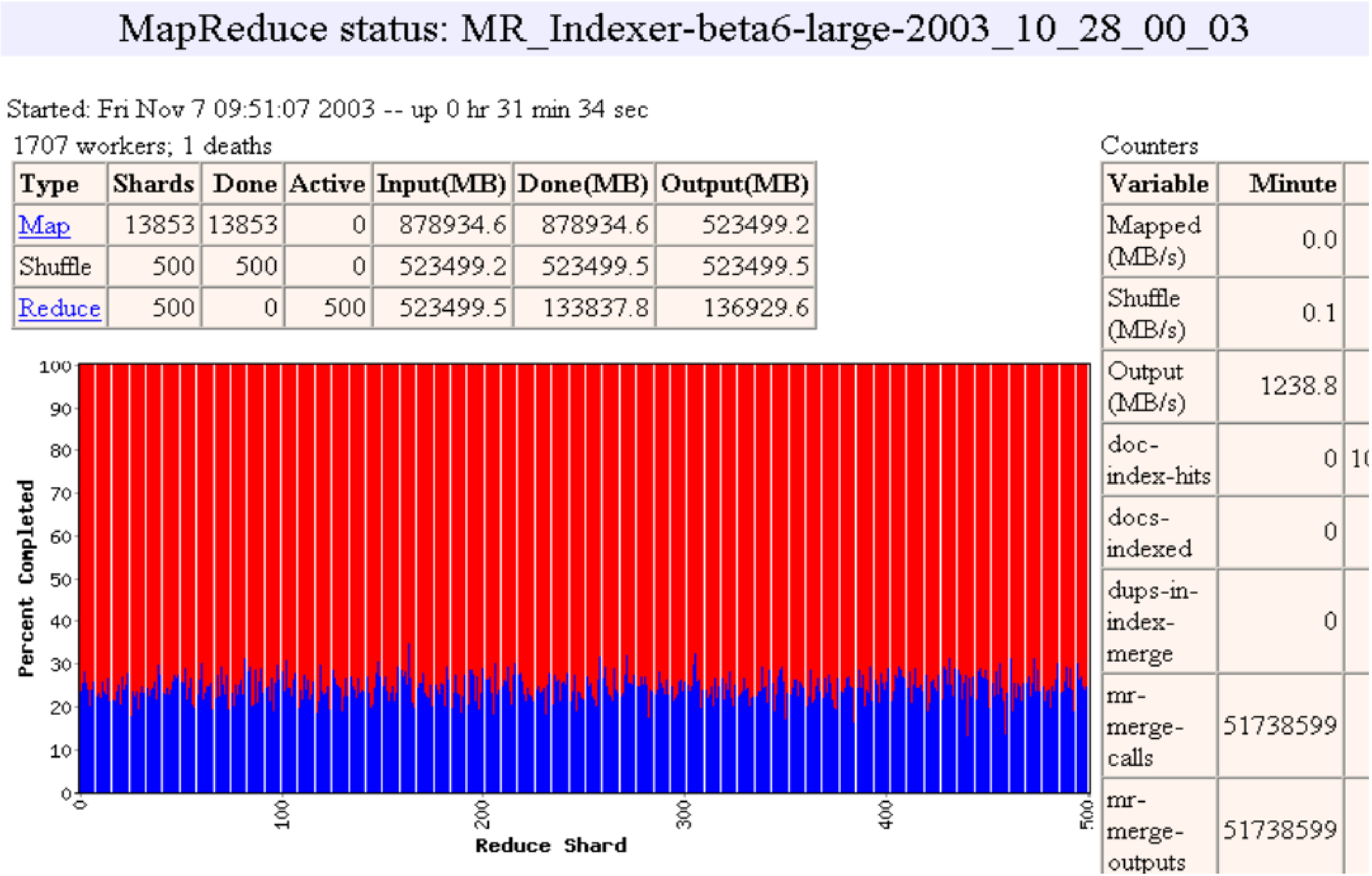
1707 workers; 1 deaths

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	10
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

AUBURN
UNIVERSITY



MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

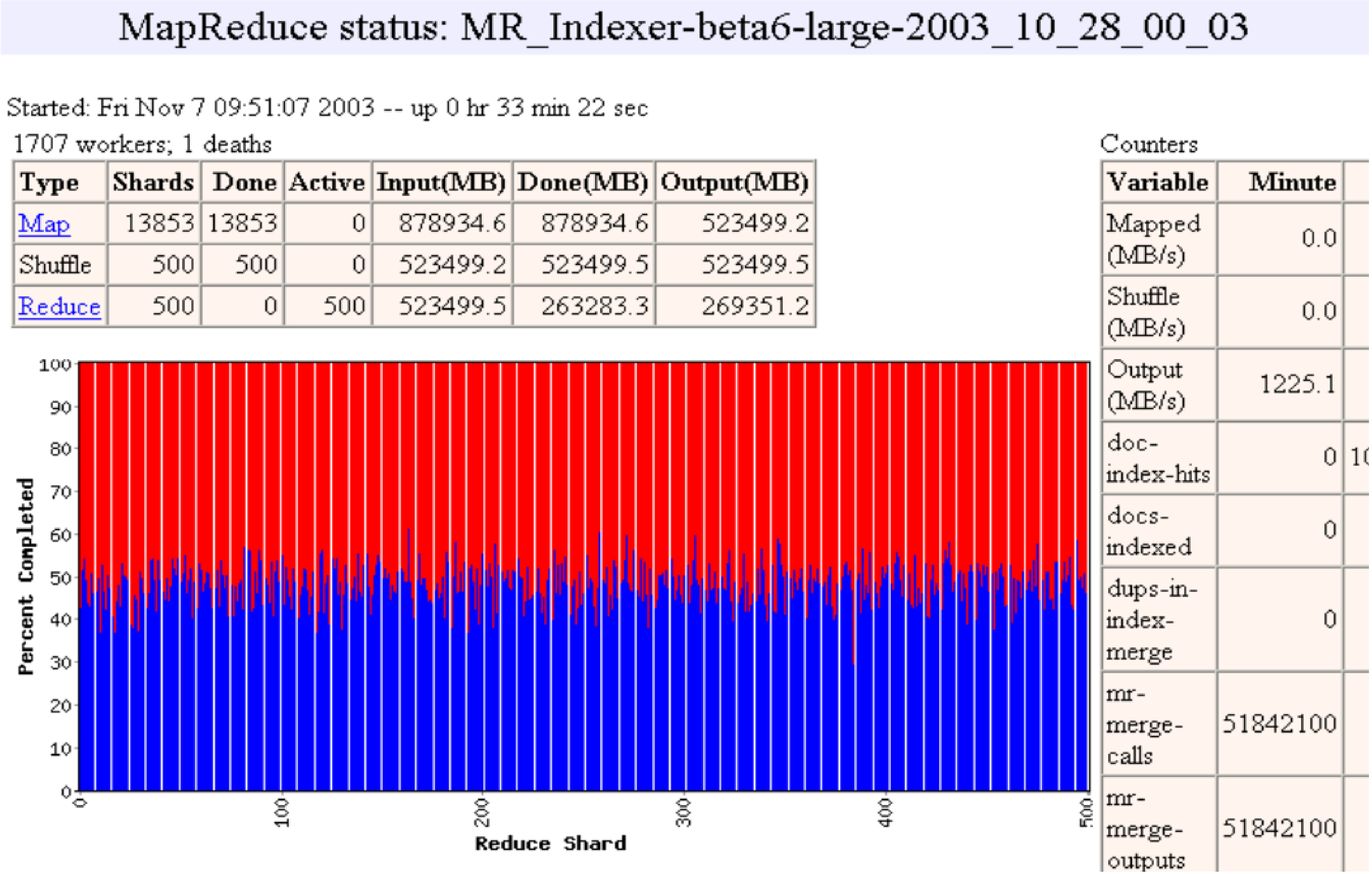
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

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	1
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

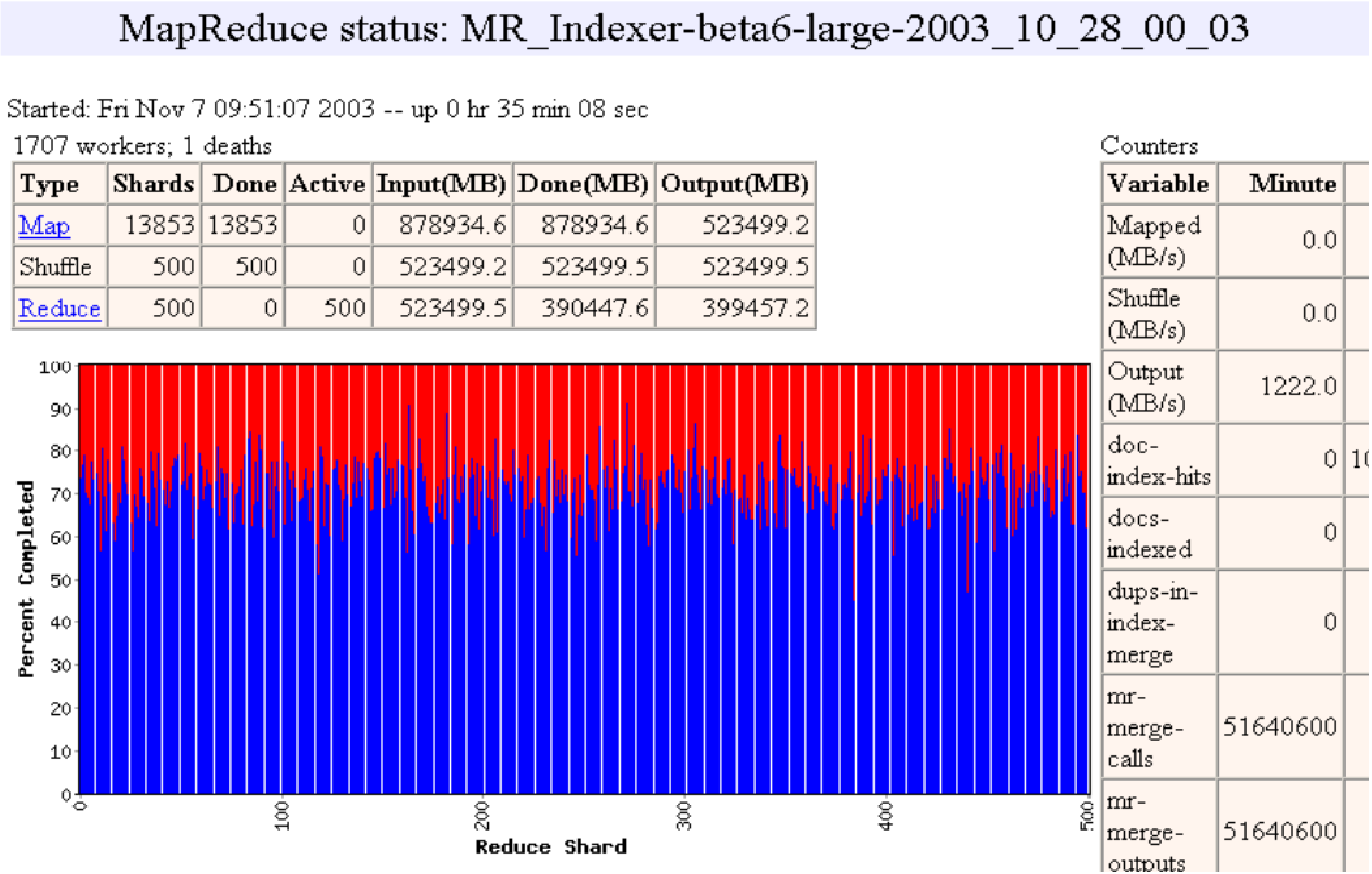
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

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	1
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

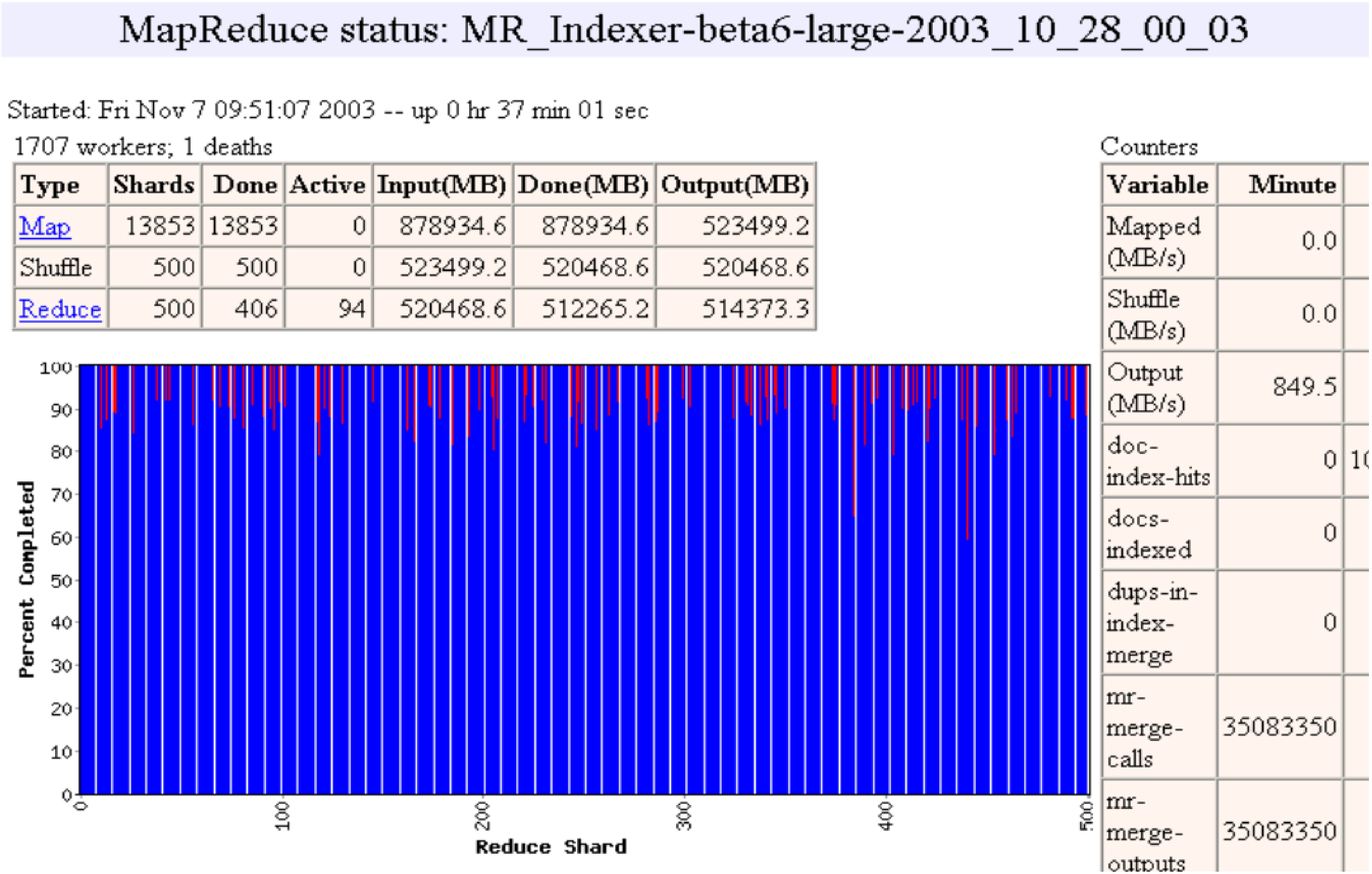
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec

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	1
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

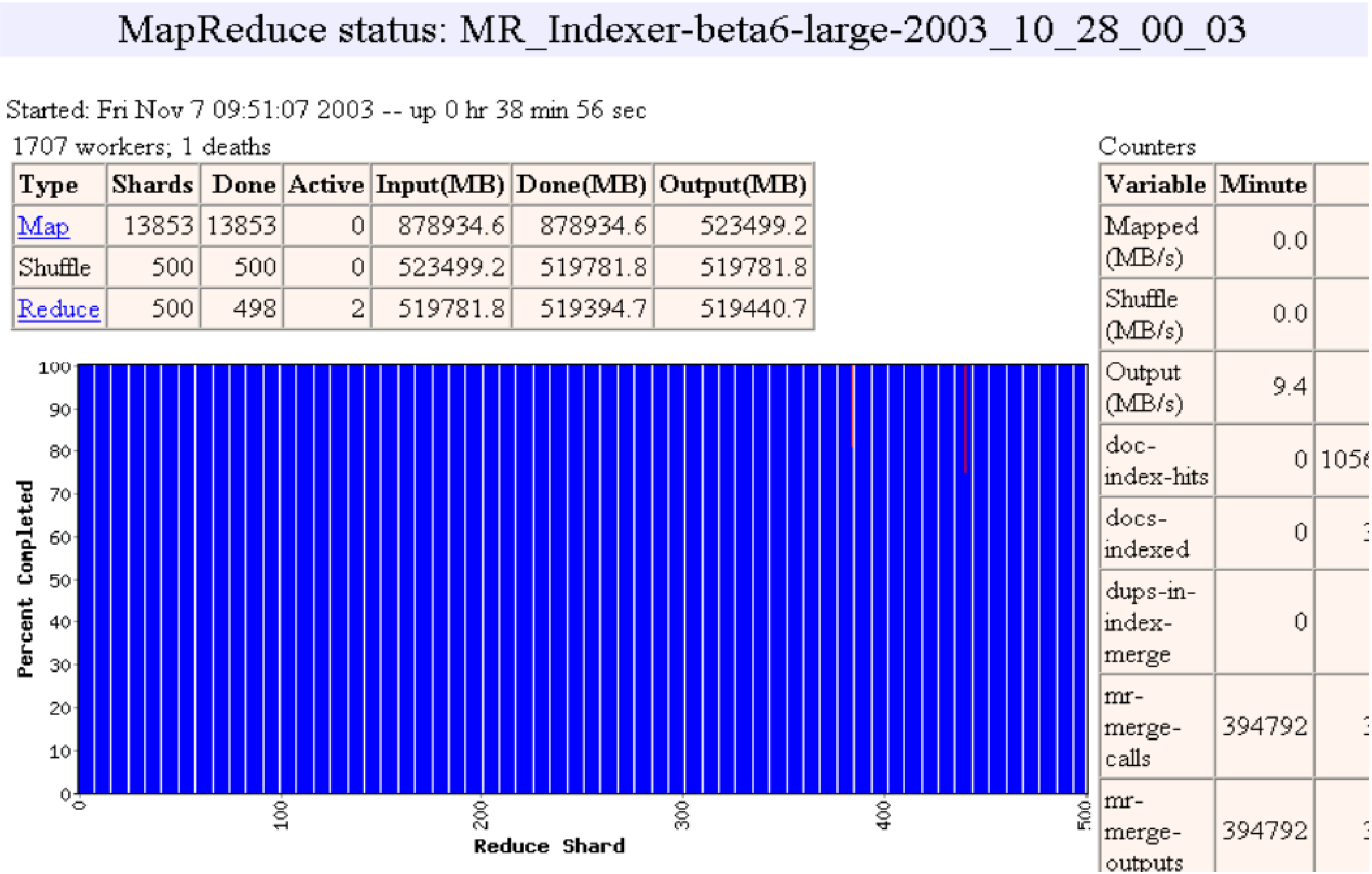
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

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

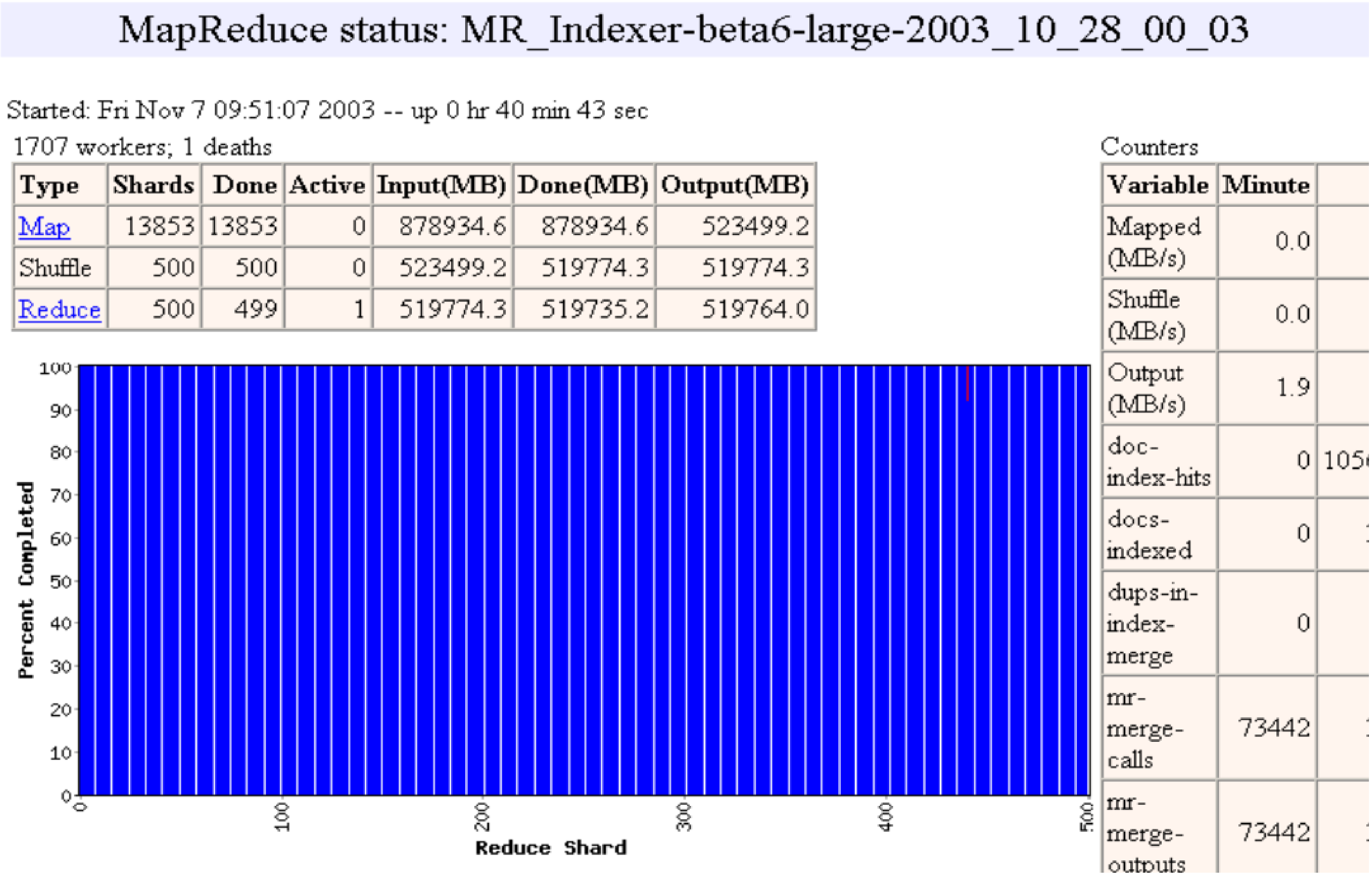
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec

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	

Fault Tolerance

- Workers are periodically pinged by master
  - No response = failed worker
  - Master writes periodic checkpoints
  - On errors, workers send “last gasp” UDP packet to master
  - Detect records that cause deterministic crashes and skips them
- Handled via re-execution
  - Re-execute completed + in-progress map tasks
  - Re-execute in progress reduce tasks
    - **Avoids “stragglers”**
  - Task completion committed through master
- Robustness:
  - Lost 1600/1800 machines once → finished Ok
- Master failure
  - Could handle, but not yet

Refinement: Redundant Execution

- Slow workers significantly delay completion time
  - Other jobs consuming resources on machine
  - Bad disks w/ soft errors transfer data slowly
  - Weird things: processor caches disabled (!!)
- Solution: Near end of phase, spawn backup tasks
  - Whichever one finishes first "wins”
- Dramatically shortens job completion time

Locality Optimization

- Master scheduling policy:
  - – Asks GFS for locations of replicas of input file blocks
  - – Map tasks typically split into 64MB (GFS block size)
  - – Map tasks scheduled so GFS input block replica are on same machine or same rack

- Effect
  - – Thousands of machines read input at local disk speed
    - Without this, rack switches limit read rate

Skipping Bad Records

- Map/Reduce functions sometimes fail for particular inputs
  - Best solution is to debug & fix
    - Not always possible ~ third-party source libraries
  - On segmentation fault:
    - Send UDP packet to master from signal handler
    - Include sequence number of record being processed
  - If master sees two failures for same record:
    - Next worker is told to skip the record

Performance

Tests run on cluster of 1800 machines:

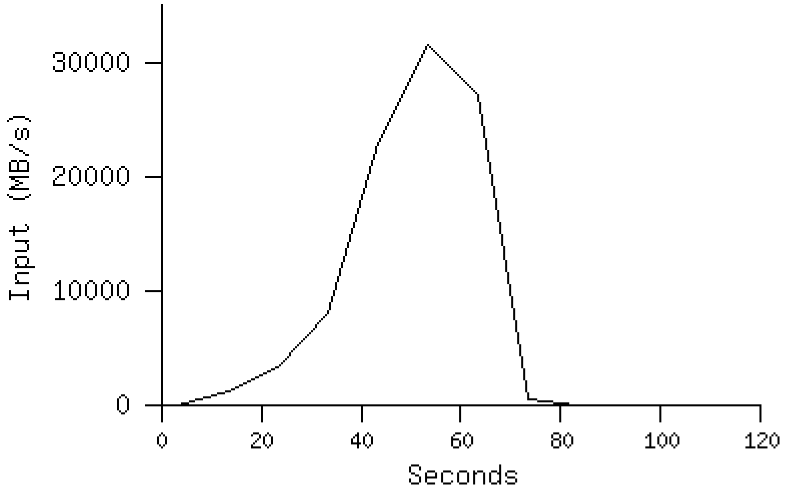
- 4 GB of memory
- Dual-processor 2 GHz Xeons with Hyperthreading
- Dual 160 GB IDE disks
- Gigabit Ethernet per machine
- Bisection bandwidth approximately 100 Gbps

Two benchmarks:

MR_GrepScan 10^{10} 100-byte records to extract records matching a rare pattern (92K matching records)

MR_SortSort 10^{10} 100-byte records (modeled after TeraSort benchmark)

MR_Grep



Locality optimization helps:

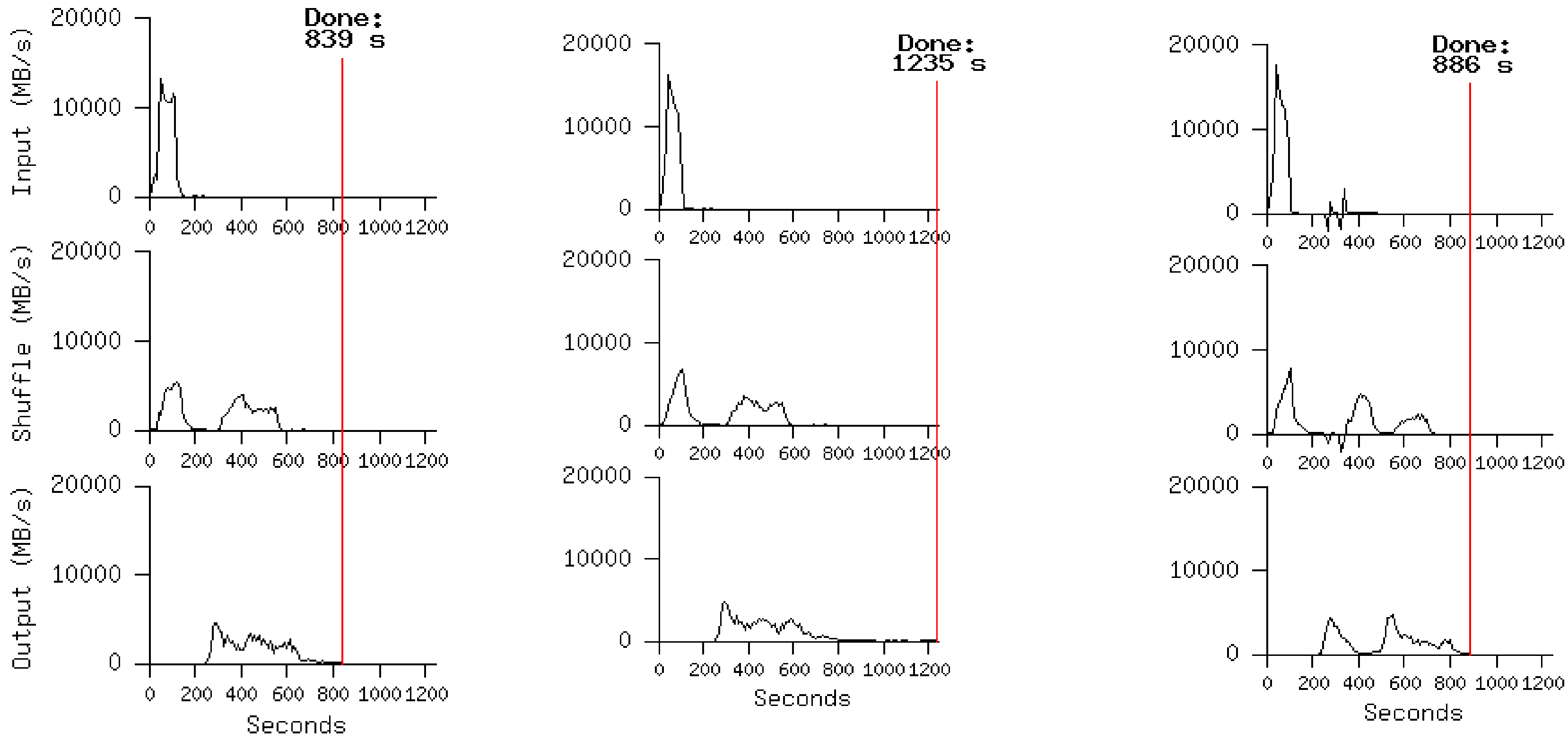
- 1800 machines read 1 TB at peak ~31 GB/s
- W/out this, rack switches would limit to 10 GB/s
- Startup overhead is significant for short jobs

MR_Sort

Normal

No backup tasks

200 processes killed



- Backup tasks reduce job completion time a lot!
- System deals well with failures

Conclusion

- MapReduce has proven to be a useful abstraction
- Greatly simplifies large-scale computations at Google
  - Two phase execution
  - Locality-based scheduling
  - Fault tolerance execution
  - Complex run-time system realized through a MapReduce library
- Fun to use: focus on problem, let library deal w/ messy details

Introduction to Hadoop

Instructor: Dr. Weikuan Yu

Computer Sci. & Software Eng.

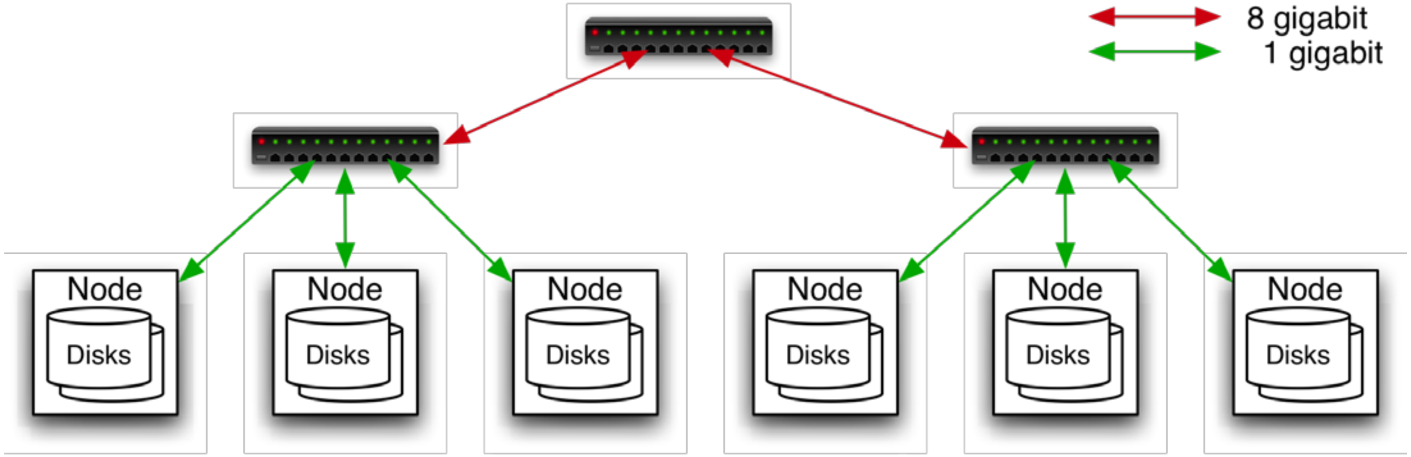
AUBURN

UNIVERSITY

Hadoop

- A framework of MapReduce implementation for large commodity clusters
- Master/Slave relationship
  - JobTracker handles all scheduling & data flow between TaskTrackers
  - TaskTracker handles all worker tasks on a node
  - Individual worker task runs map or reduce operation
- Integrates with Hadoop Distributed File System for data-centric computing

Sample Hadoop Cluster



- Commodity hardware
  - Linux PCs with local 4 disks
- Typically in 2 level architecture
  - 40 nodes/rack
  - Uplink from rack is 8 gigabit
  - Rack-internal is 1 gigabit all-to-all

MapReduce v. Hadoop

	MapReduce	Hadoop
Org	Google	Yahoo/Apache
Impl	C++	Java
Distributed File Sys	GFS	HDFS
Data Base	Bigtable	HBase
Distributed lock mgr	Chubby	ZooKeeper

Hadoop History

- **Dec 2004 –** Google GFS paper published
- **July 2005 –** Nutch uses MapReduce
- **Feb 2006 –** Becomes Lucene subproject
- **Apr 2007 –** Yahoo! on 1000-node cluster
- **Jan 2008 –** An Apache Top Level Project
- **Jul 2008 –** A 4000 node test cluster

Map/Reduce features

- Java, C++, and text-based APIs
  - In Java; use Objects and and C++ bytes
  - Text-based (streaming) great for scripting or legacy apps
  - Higher level interfaces: Pig, Hive, Jaql
- Automatic re-execution on failure
  - In a large cluster, some nodes are always slow or flaky
  - Framework re-executes failed tasks
- Locality optimizations
  - With large data, bandwidth to data is a problem
  - Map-Reduce queries HDFS for locations of input data
  - Map tasks are scheduled close to the inputs when possible

Hadoop Map/Reduce

- The Map-Reduce programming model

  – Framework for distributed processing of large data sets

  – Pluggable user code runs in generic framework

- Common design pattern in data processing

```
 cat * | grep  | sort      | unique -c | cat > file
input | map | shuffle | reduce    | output
```

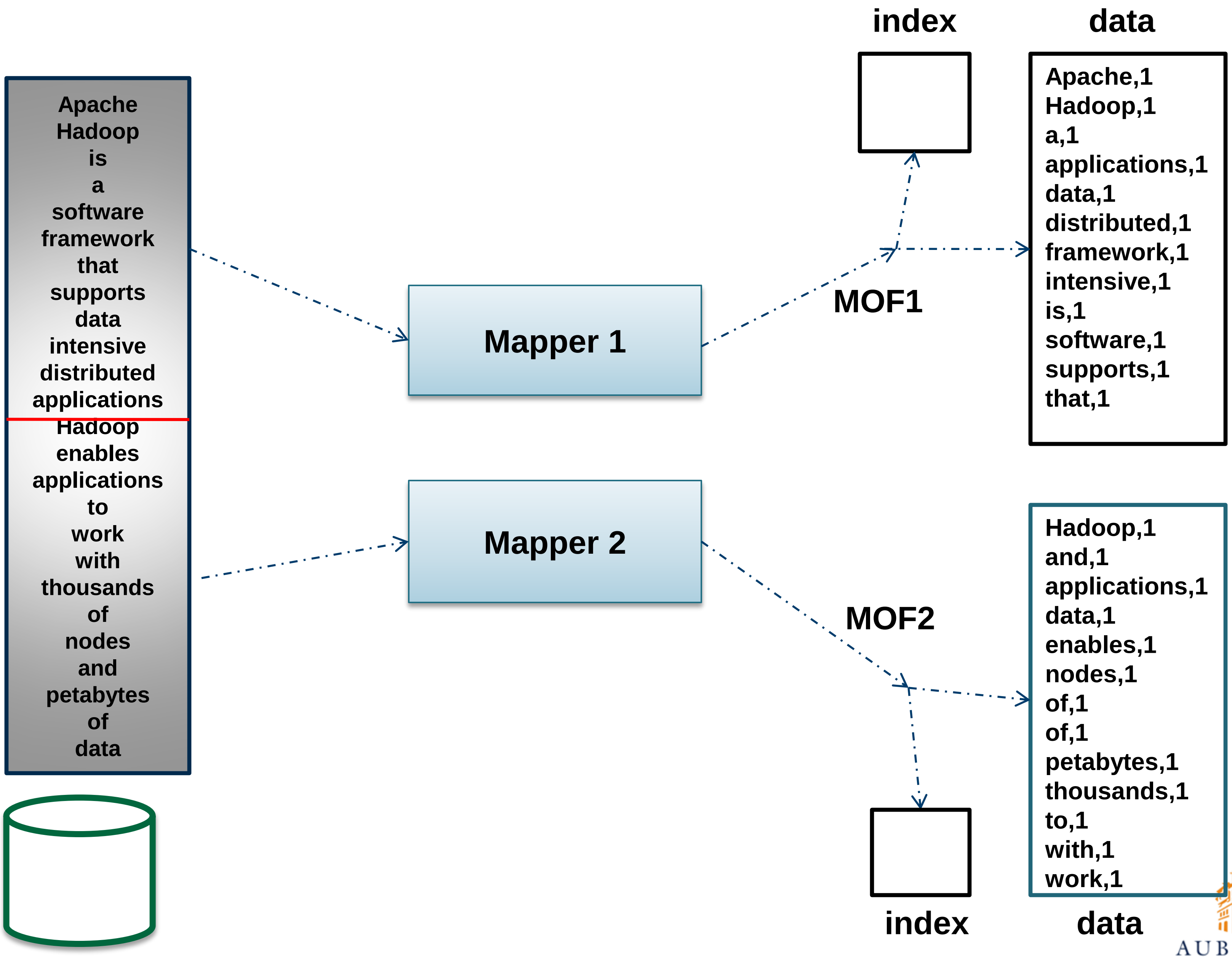
- Natural for:

  – Log processing

  – Web search indexing

  – Ad-hoc queries

WordCount Example: Mapping



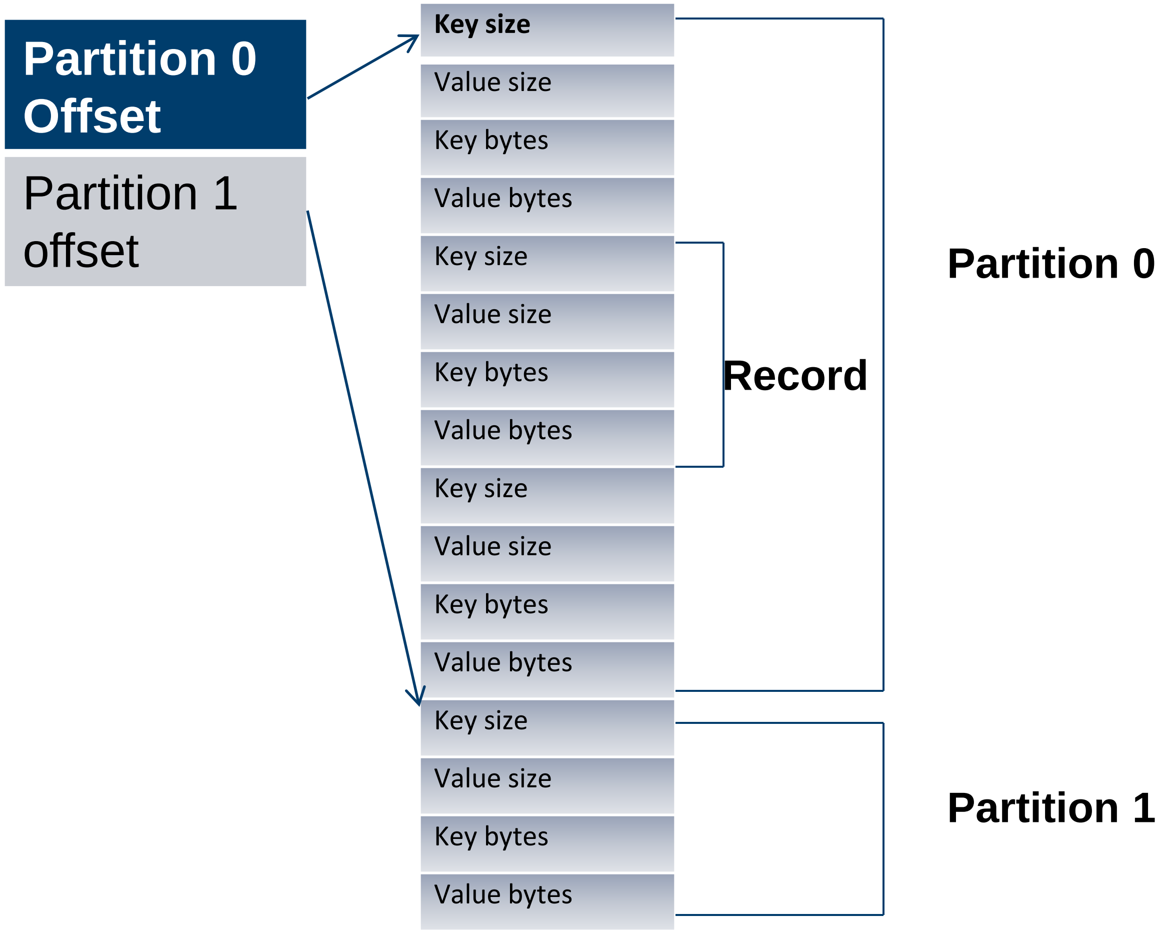
Apache
Hadoop
is
a
software
framework
that
supports
data
intensive
distributed
applications

Hadoop
enables
applications
to
work
with
thousands
of
nodes
and
petabytes
of
data

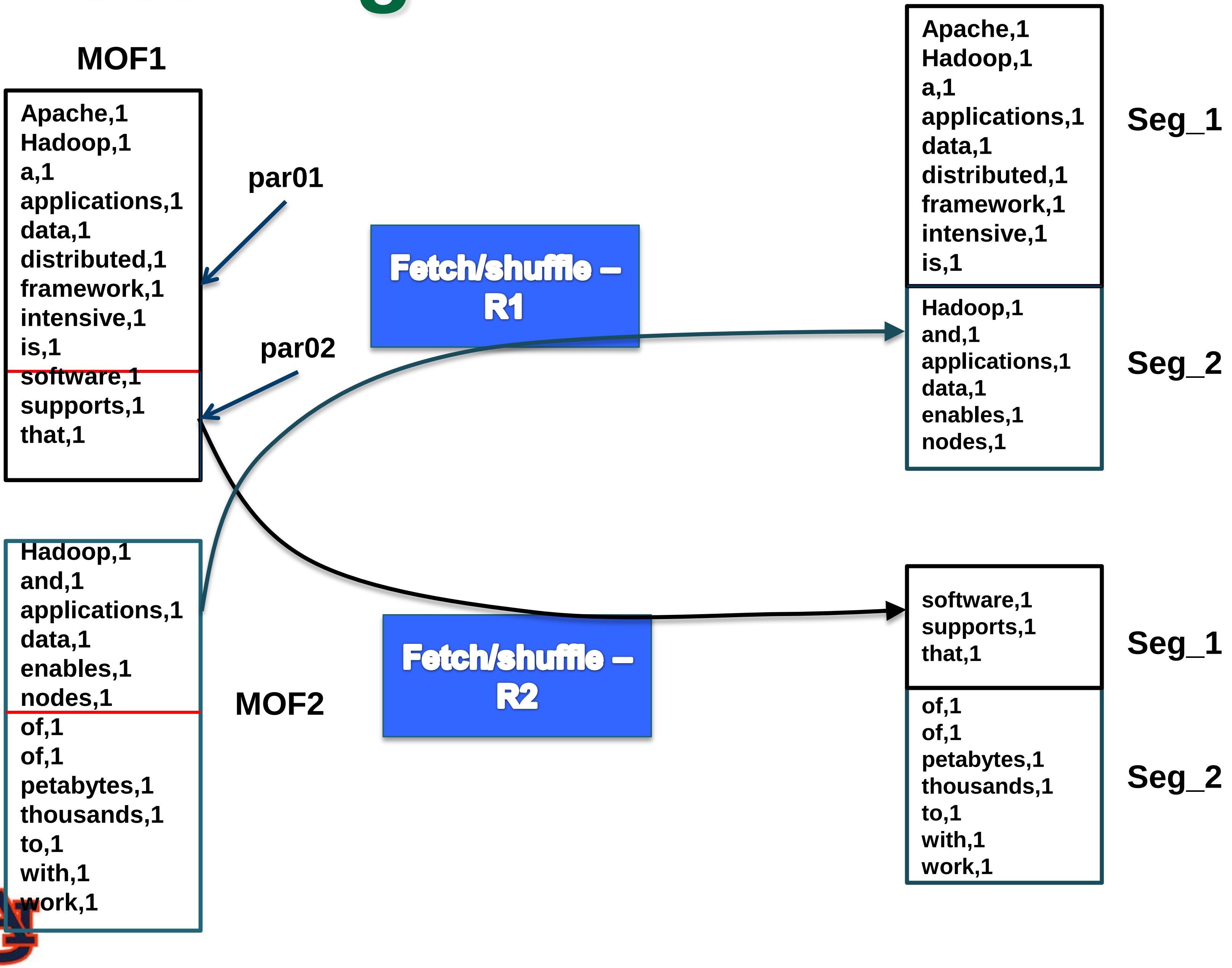
Apache,1
Hadoop,1
a,1
applications,1
data,1
distributed,1
framework,1
intensive,1
is,1
software,1
supports,1
that,1

Hadoop,1
and,1
applications,1
data,1
enables,1
nodes,1
of,1
of,1
petabytes,1
thousands,1
to,1
with,1
work,1

MapOutput File (MOF): Index and Data formats



Fetching



MOF1

Apache,1
Hadoop,1
a,1
applications,1
data,1
distributed,1
framework,1
intensive,1
is,1
software,1
supports,1
that,1

par01

par02

Fetch/shuffle – R1

Hadoop,1
and,1
applications,1
data,1
enables,1
nodes,1
of,1
of,1
petabytes,1
thousands,1
to,1
with,1
work,1

MOF2

Fetch/shuffle – R2

Apache,1
Hadoop,1
a,1
applications,1
data,1
distributed,1
framework,1
intensive,1
is,1

Seg_1

Hadoop,1
and,1
applications,1
data,1
enables,1
nodes,1

Seg_2

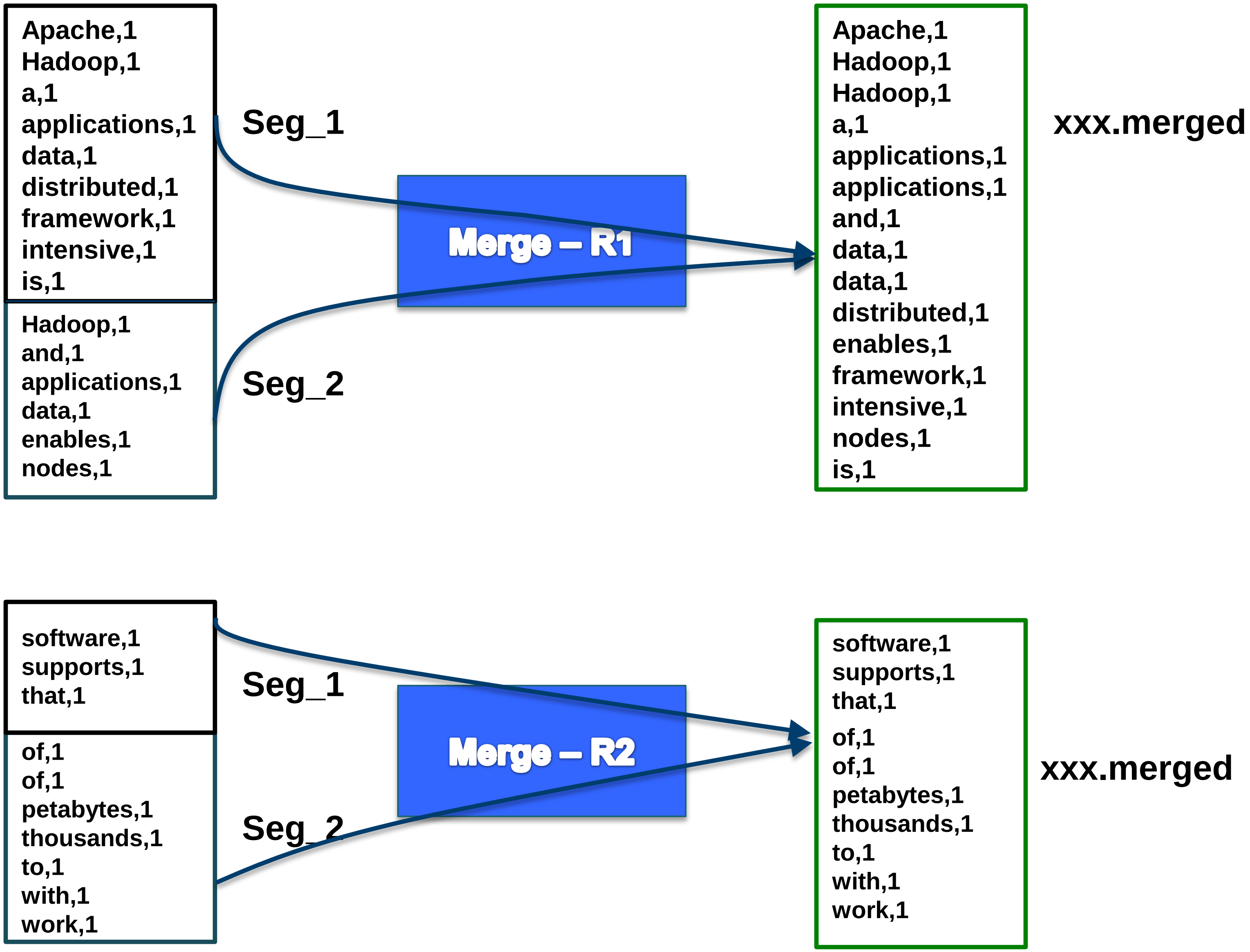
software,1
supports,1
that,1

Seg_1

of,1
of,1
petabytes,1
thousands,1
to,1
with,1
work,1

Seg_2

Merging (sorting)



Apache,1
Hadoop,1
a,1
applications,1
data,1
distributed,1
framework,1
intensive,1
is,1

Seg_1

Hadoop,1
and,1
applications,1
data,1
enables,1
nodes,1

Seg_2

Merge – R1

Apache,1
Hadoop,1
Hadoop,1
a,1
applications,1
applications,1
and,1
data,1
data,1
distributed,1
enables,1
framework,1
intensive,1
nodes,1
is,1

xxx.merged

software,1
supports,1
that,1

Seg_1

of,1
of,1
petabytes,1
thousands,1
to,1
with,1
work,1

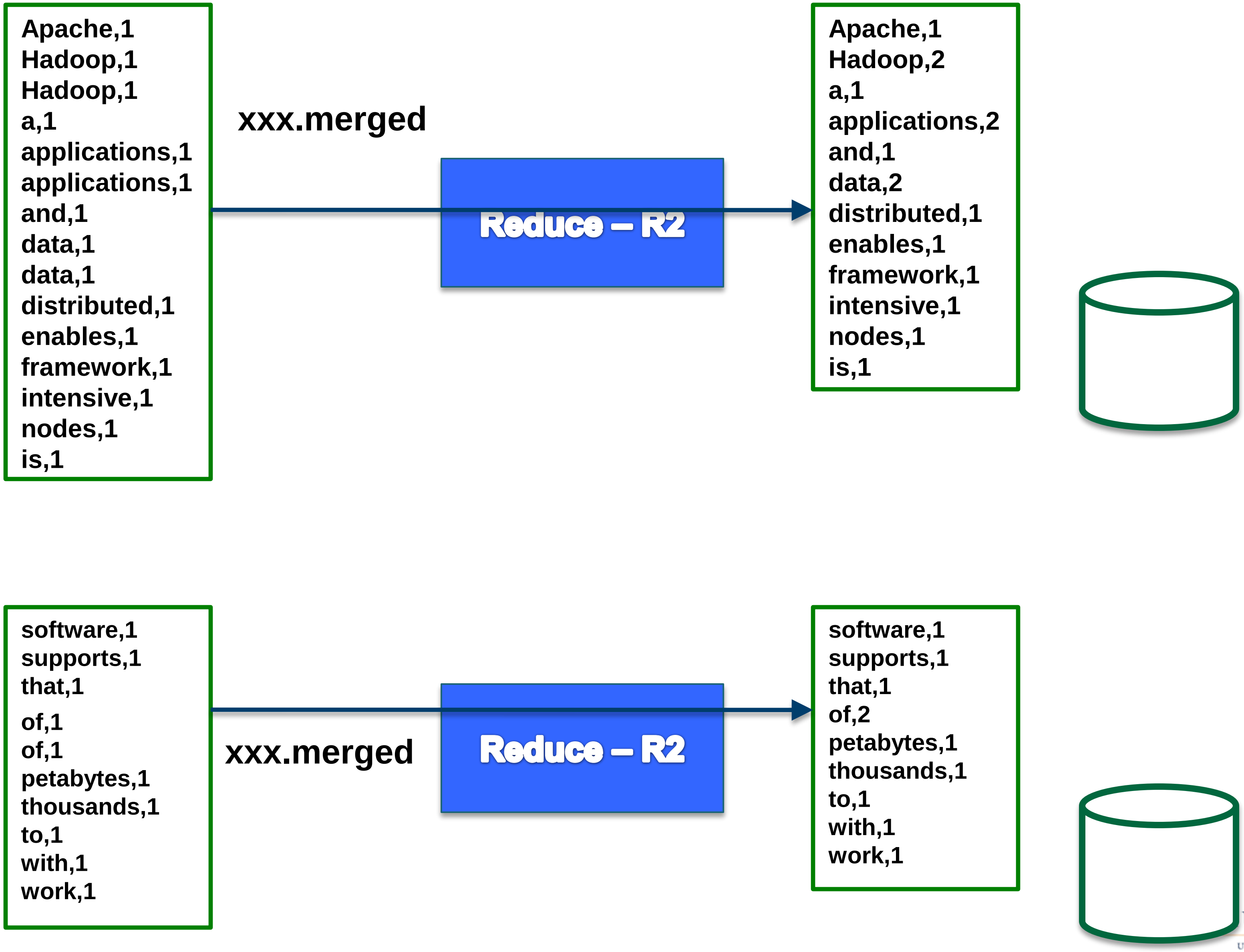
Seg_2

Merge – R2

software,1
supports,1
that,1
of,1
of,1
petabytes,1
thousands,1
to,1
with,1
work,1

xxx.merged

Reduce



Apache,1
Hadoop,1
Hadoop,1
a,1
applications,1
applications,1
and,1
data,1
data,1
distributed,1
enables,1
framework,1
intensive,1
nodes,1
is,1

xxx.merged

Reduce – R2

Apache,1
Hadoop,2
a,1
applications,2
and,1
data,2
distributed,1
enables,1
framework,1
intensive,1
nodes,1
is,1

software,1
supports,1
that,1
of,1
of,1
petabytes,1
thousands,1
to,1
with,1
work,1

xxx.merged

Reduce – R2

software,1
supports,1
that,1
of,2
petabytes,1
thousands,1
to,1
with,1
work,1

Hadoop Impact on Productivity

- Makes Developers & Scientists more productive
  - Key computations solved in days and not months
  - Projects move from research to production in days
  - Easy to learn, even our rocket scientists use it!
- The major factors
  - You don’t need to find new hardware to experiment
  - You can work with all your data!
  - Production and research based on same framework
  - No need for R&D to do IT (it just works)

Apache Hadoop Community

- Owned by the Apache Foundation
  - Provides legal and technical framework for collaboration
  - All code and IP owned by non-profit foundation
- Anyone can join Apache’s meritocracy
  - Users
  - Contributors
    - write patches
  - Committers
    - can commit patches
  - Project Management Committee
    - vote on new committers and releases
    - Representatives from many organizations
- Use, contribution, and diversity is growing
  - But we need and want more!

HDFS: Hadoop Distributed File System

- Designed to scale to petabytes of storage, and run on top of the file systems of the underlying OS.
- Master (“NameNode”) handles replication, deletion, creation
- Slave (“DataNode”) handles data retrieval
- Files stored in many blocks
  - Each block has a block Id
  - Block Id associated with several nodes hostname:port (depending on level of replication)

HDFS Overview

- Single petabyte file system for entire cluster
  - Managed by a single *namenode*.
  - Files are written, read, renamed, deleted, but append-only.
  - Optimized for streaming reads of large files.
- Files are broken in to large blocks.
  - Transparent to the client
  - Data is checksumed with CRC32
  - Replicated to several *datanodes*, for reliability
- Client library talks to both namenode and datanodes
  - Data is not sent through the namenode.
  - Throughput of file system scales nearly linearly.
- Access from Java, C, or command line.

Goals of HDFS

- Very Large Distributed File System

  – 10K nodes, 100 million files, 10 PB

- Assumes Commodity Hardware

  – Files are replicated to handle hardware failure

  – Detect failures and recovers from them

- Optimized for Batch Processing

  – Data locations exposed so that computations can move to where data resides

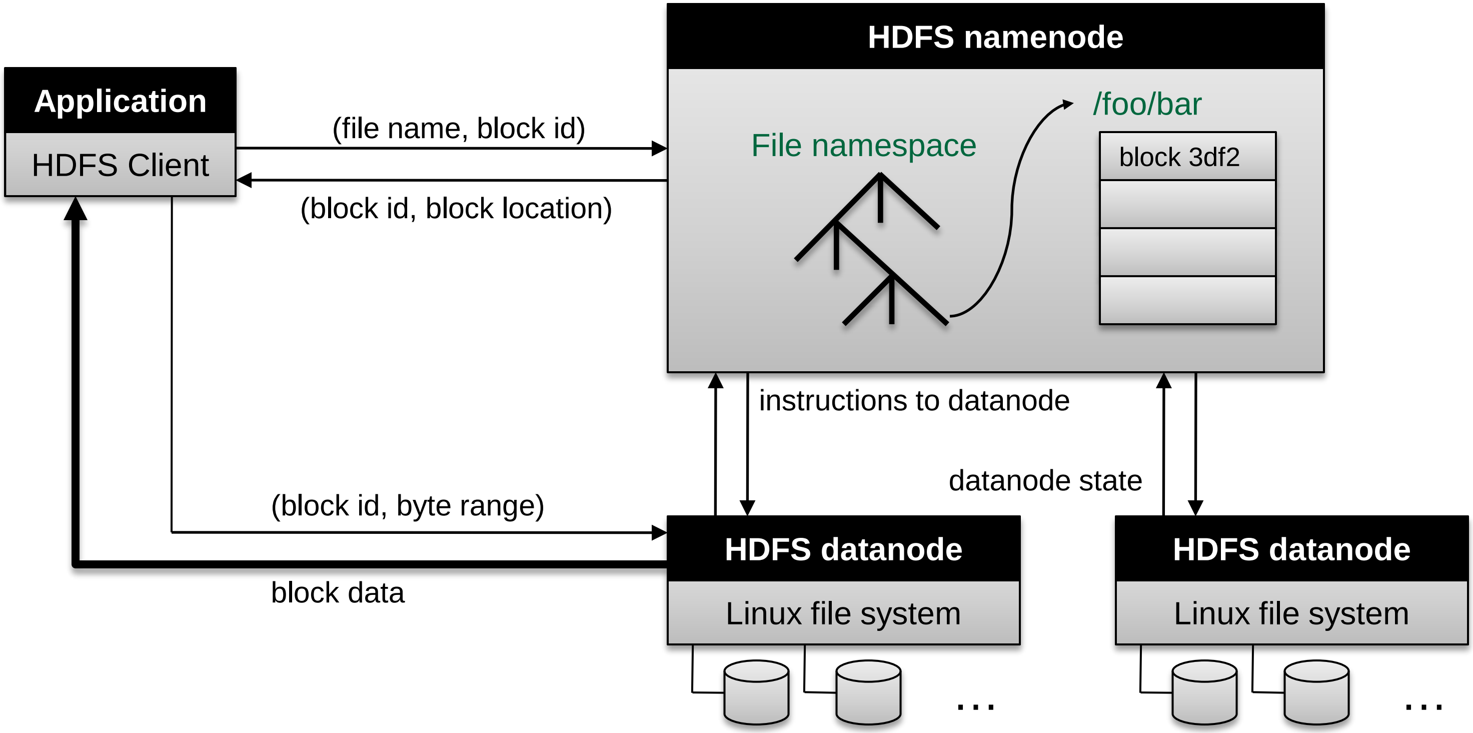
  – Provides very high aggregate bandwidth

- User Space, runs on heterogeneous OS

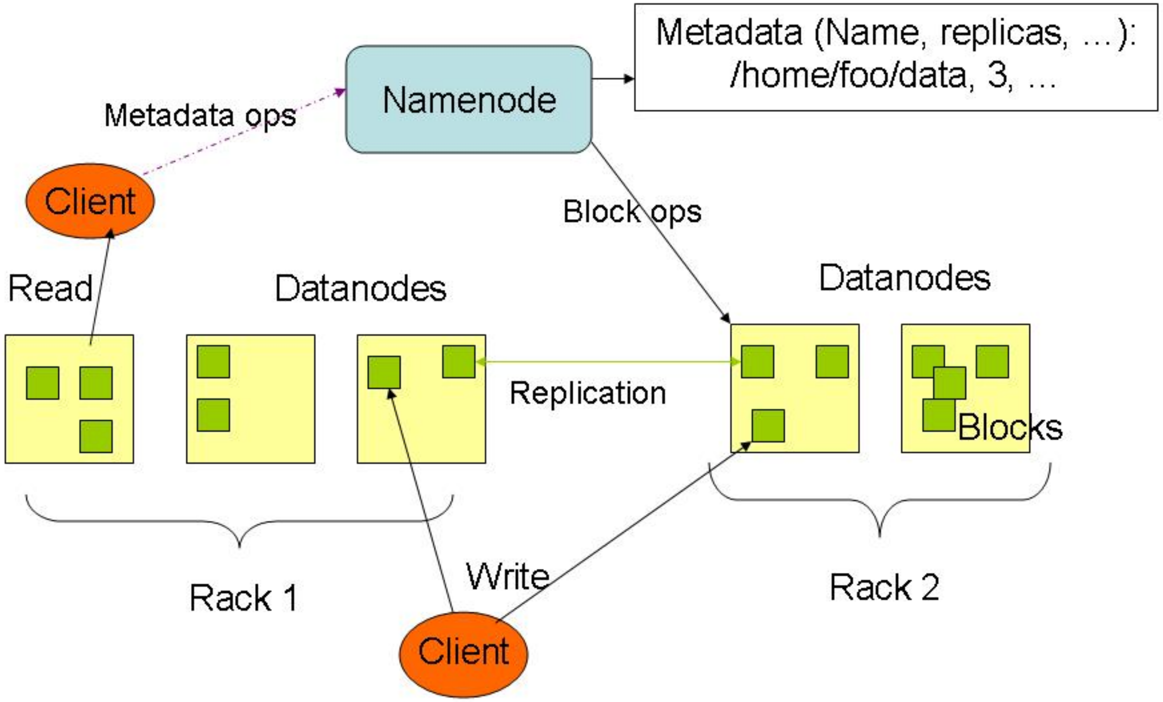
Data-Centric Processing

- Don’t move data to workers… move workers to the data!
  - Store data on the local disks of nodes in the cluster
  - Start up the workers on the node that has the data local
- Why?
  - Not enough RAM to hold all the data in memory
  - Disk access is slow, but disk throughput is reasonable
- A distributed file system is the answer
  - GFS (Google File System) for Google’s MapReduce
  - HDFS (Hadoop Distributed File System) for Hadoop

HDFS Component Architecture



Data Flow of HDFS I/O Operations



NameNode Responsibilities

- Managing the file system namespace:
  - Holds file/directory structure, metadata, file-to-block mapping, access permissions, etc.
- Coordinating file operations:
  - Directs clients to datanodes for reads and writes
  - No data is moved through the namenode
- Maintaining overall health:
  - Periodic communication with the datanodes
  - Block re-replication and rebalancing
  - Garbage collection

DataNode

- A Block Server
  - Stores data in the local file system (e.g. ext3)
  - Stores meta-data of a block (e.g. CRC)
  - Serves data and meta-data to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode
- Facilitates Pipelining of Data
  - Forwards data to other specified DataNodes

Block Placement

- Current Strategy

  -- One replica on local node

  -- Second replica on a remote rack

  -- Third replica on same remote rack

  -- Additional replicas are randomly placed

- Clients read from nearest replica
- Would like to make this policy pluggable

NameNode Failure

- A single point of failure
- Transaction Log stored in multiple directories
  - A directory on the local file system
  - A directory on a remote file system (NFS/CIFS)
- Need to develop a real HA solution

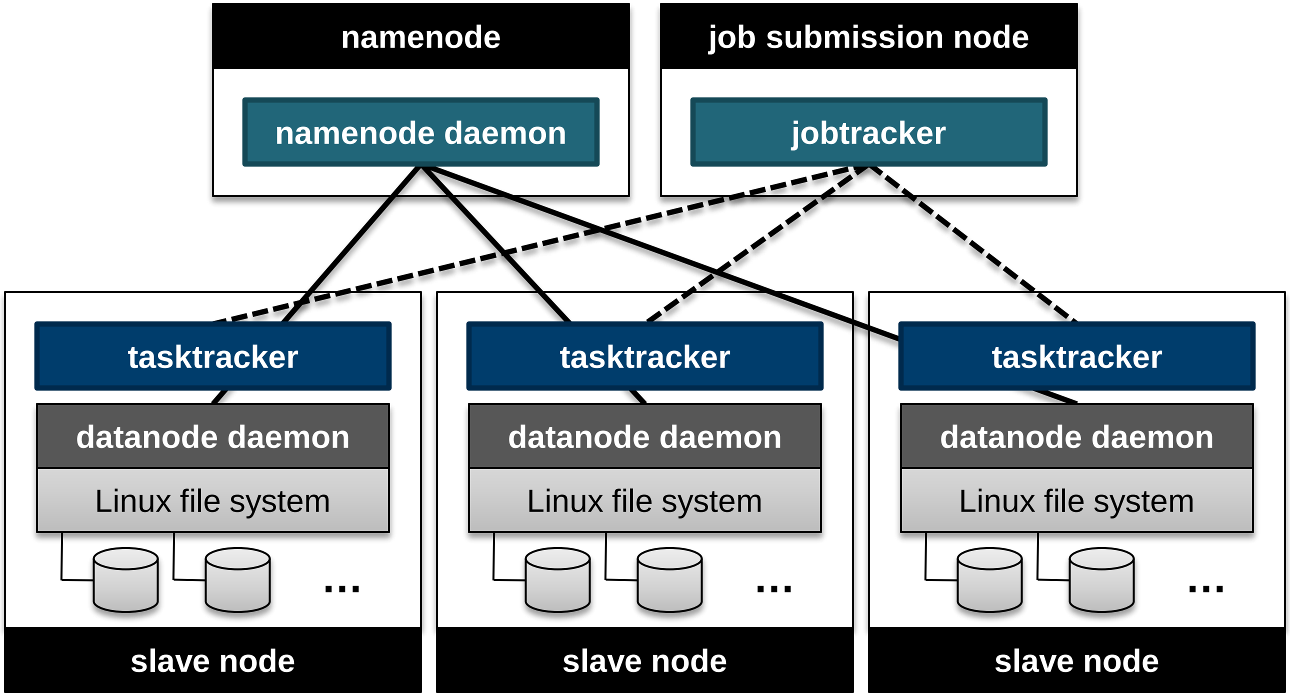
Data Pipelining

- Client retrieves a list of DataNodes on which to place replicas of a block
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next DataNode in the Pipeline
- When all replicas are written, the Client moves on to write the next block in file

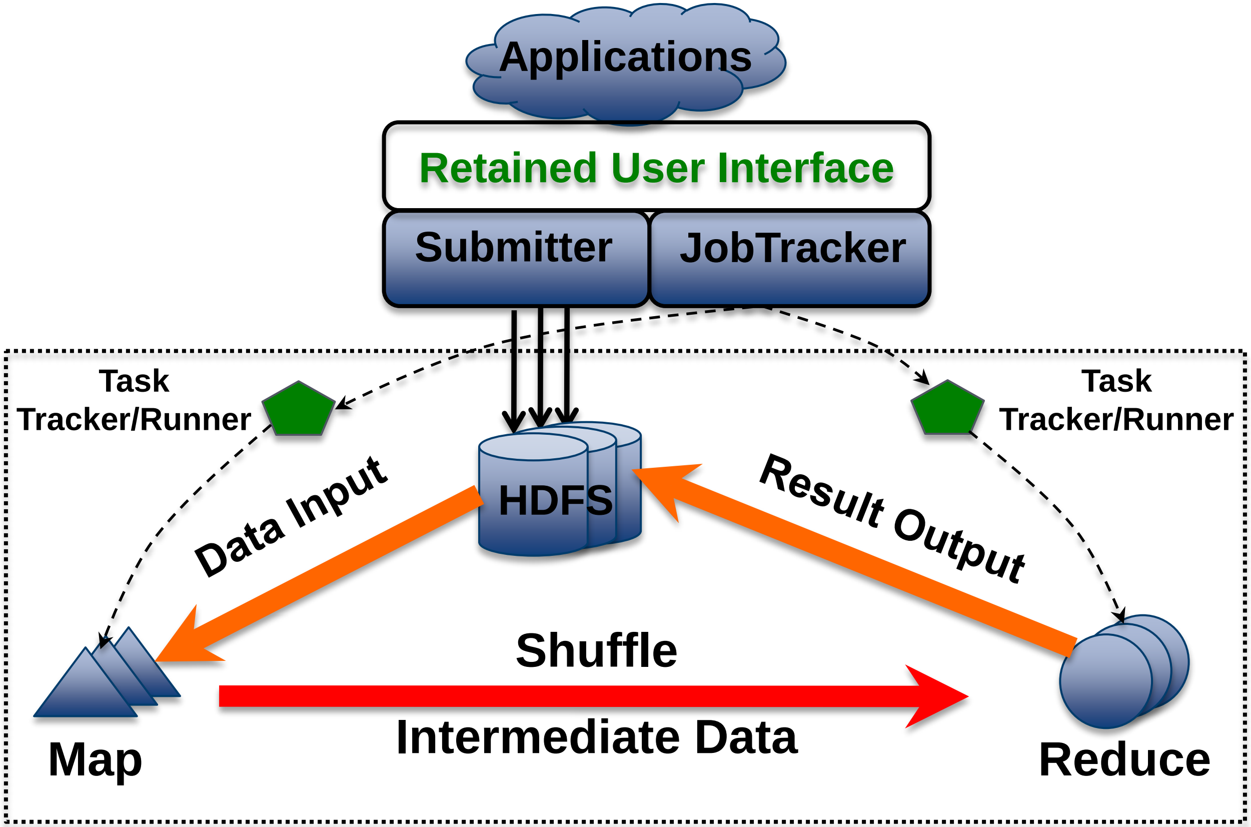
Other Features

- Rebalancer
  - Goal: % disk full on DataNodes should be similar
  - Usually run when new DataNodes are added
  - Cluster is online when Rebalancer is active
  - Rebalancer is throttled to avoid network congestion
  - Command line tool
- Rack Awareness
  - optimization which takes into account the geographic clustering of servers
  - network traffic between servers in different geographic clusters is minimized.

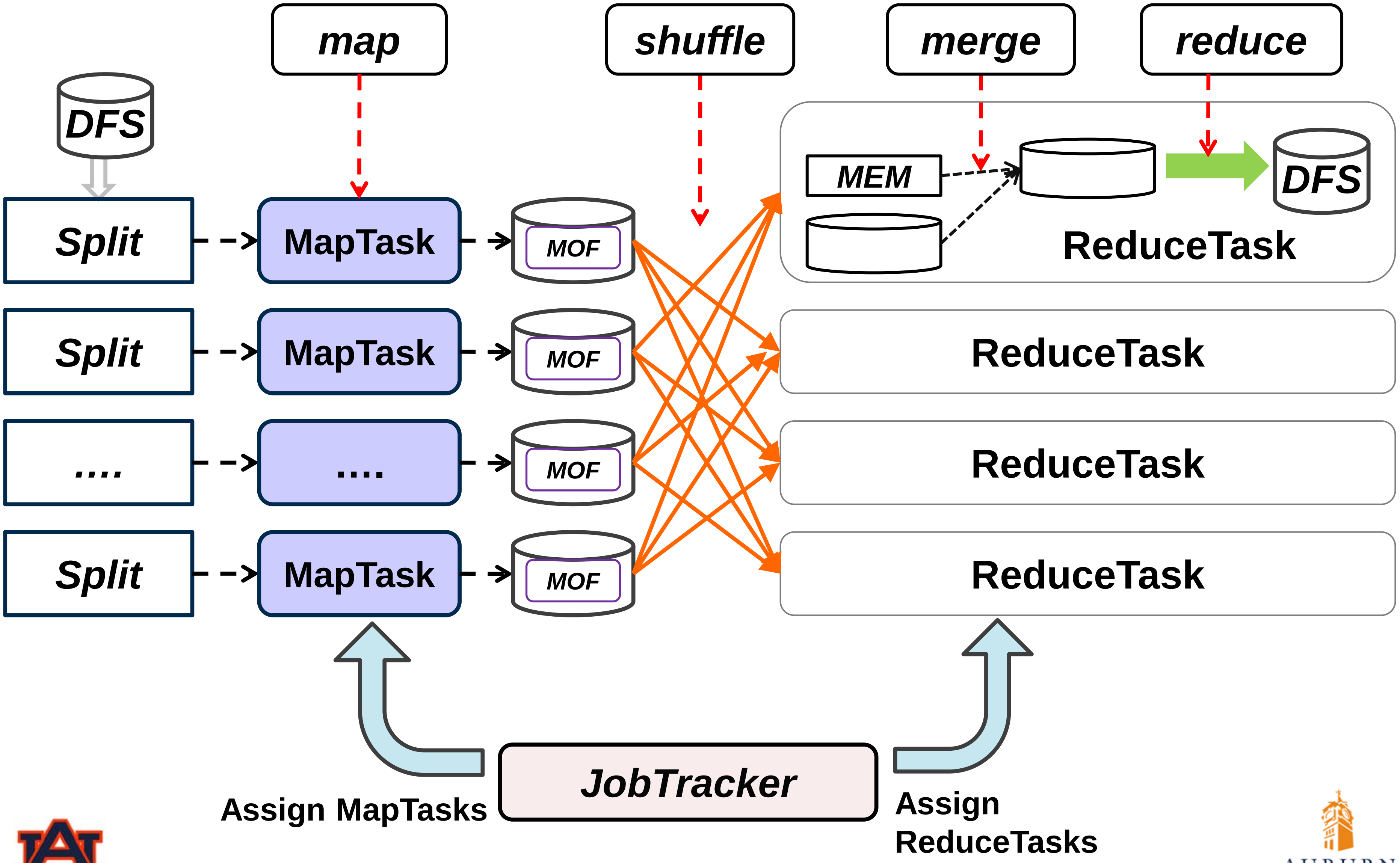
Recap of Hadoop and HDFS



Research for Hadoop Acceleration

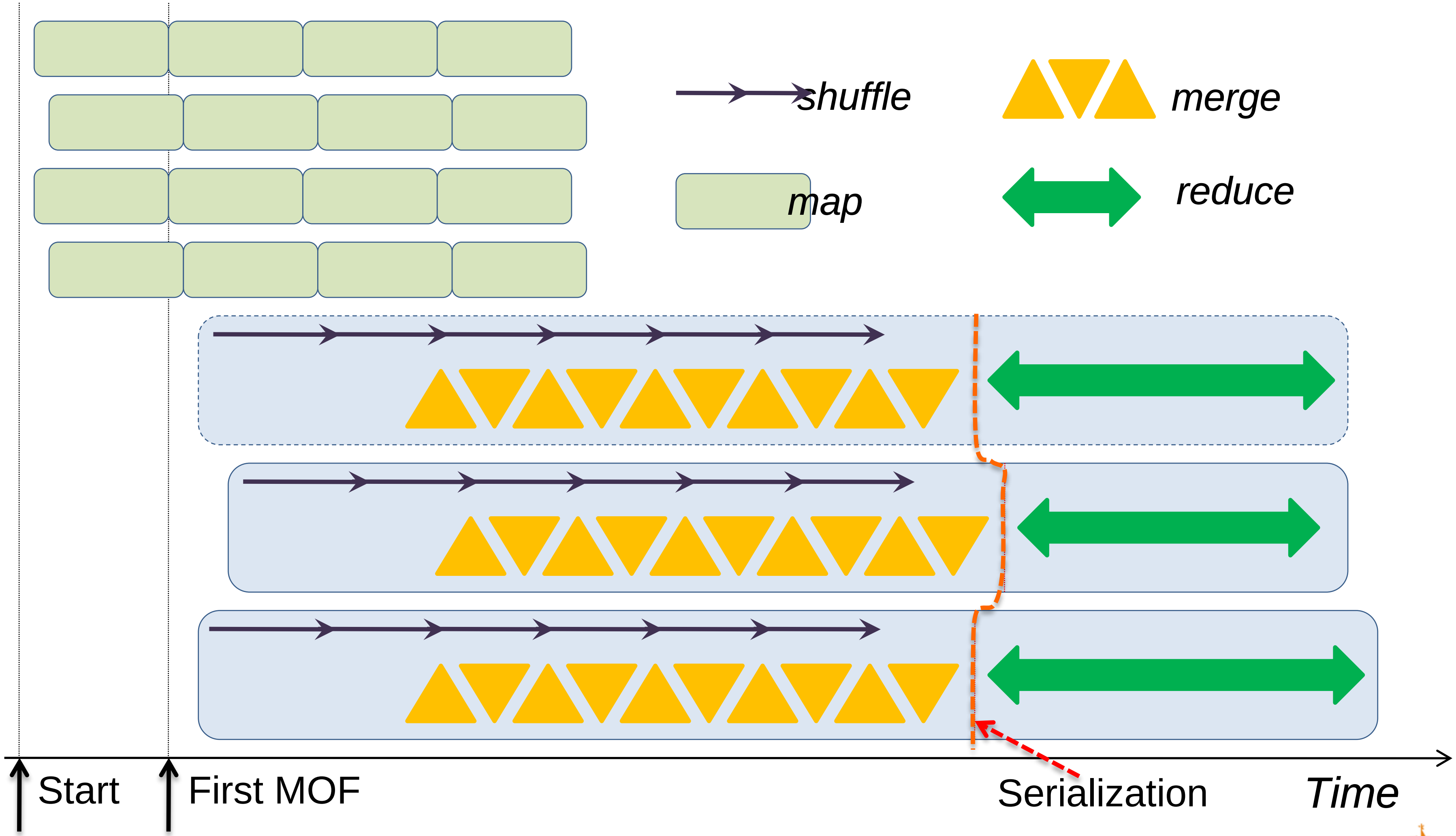


Data Movement in Hadoop MapReduce Framework

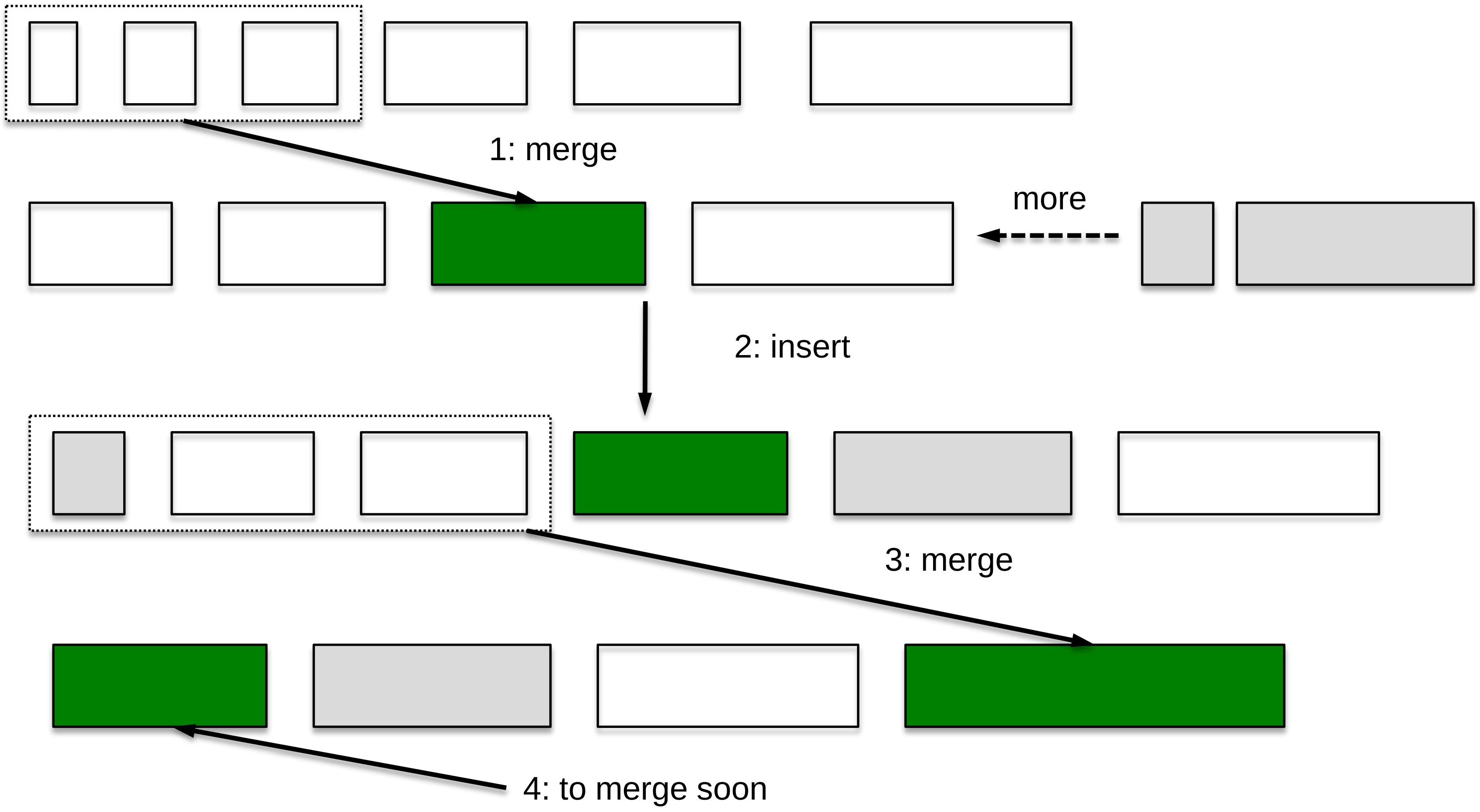


Further Issues in Hadoop Data Movement

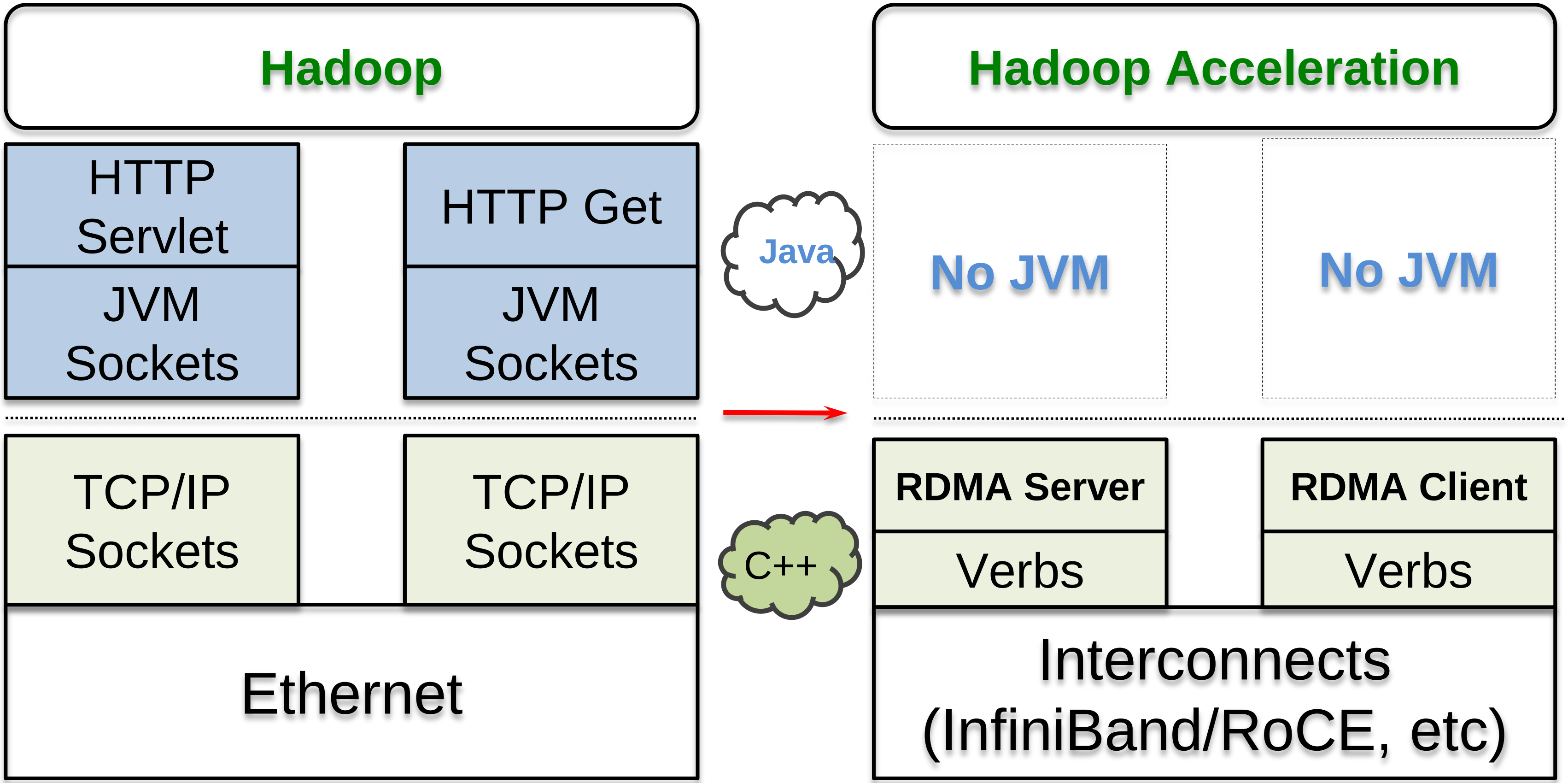
1: Serialization between shuffle/merge and reduce phases



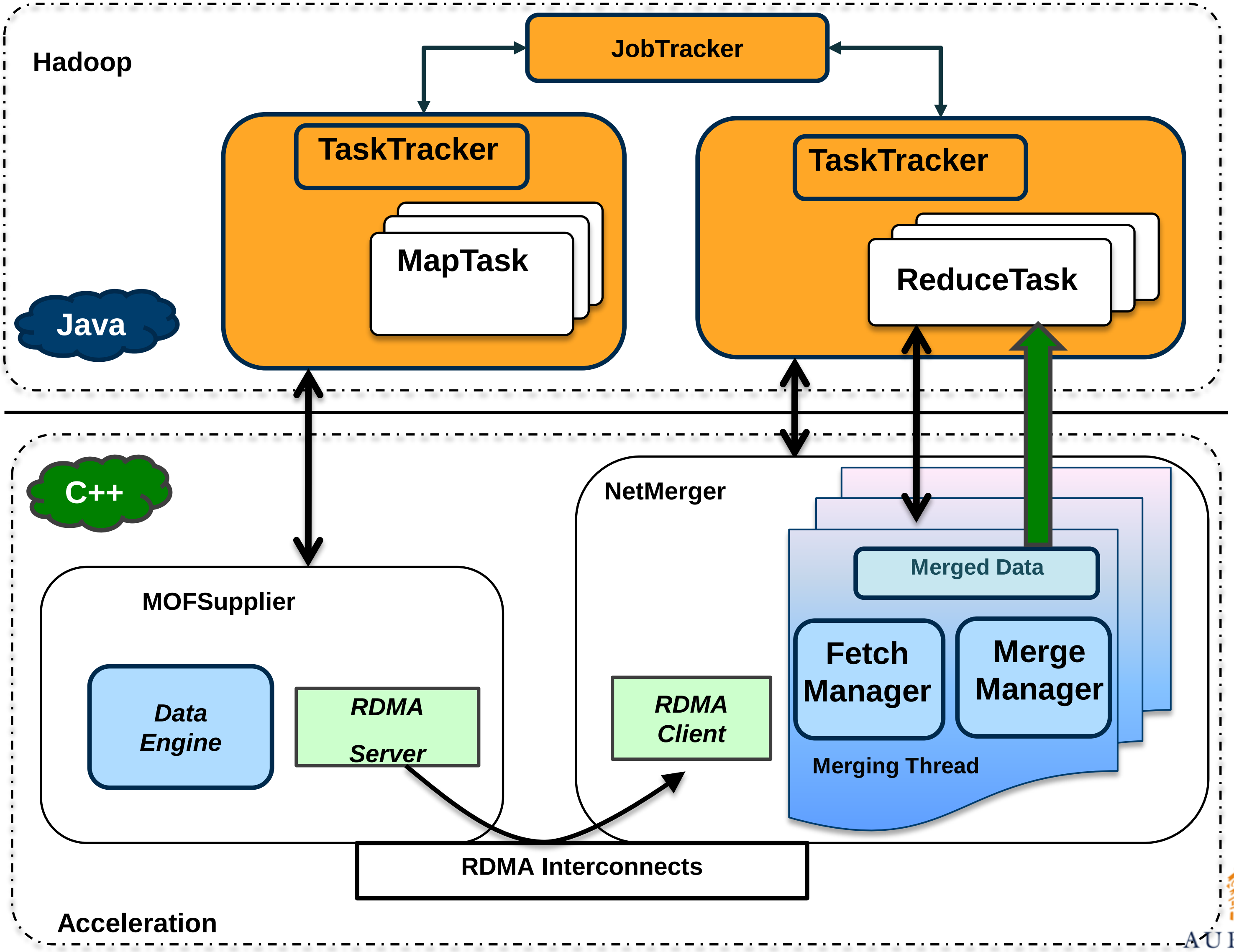
Repetitive Data Merge and Disk I/O



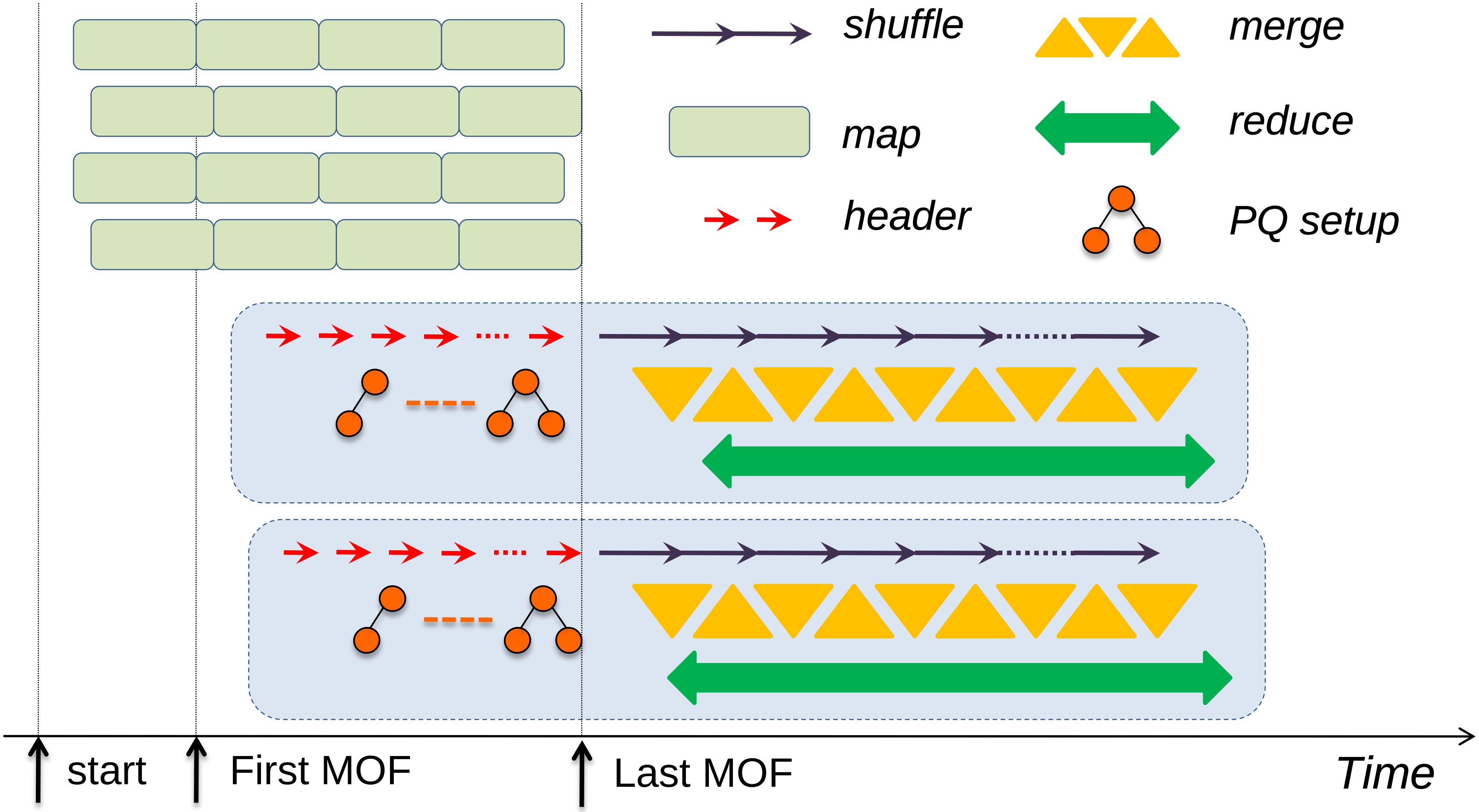
Lack of Support for High-Speed Interconnects



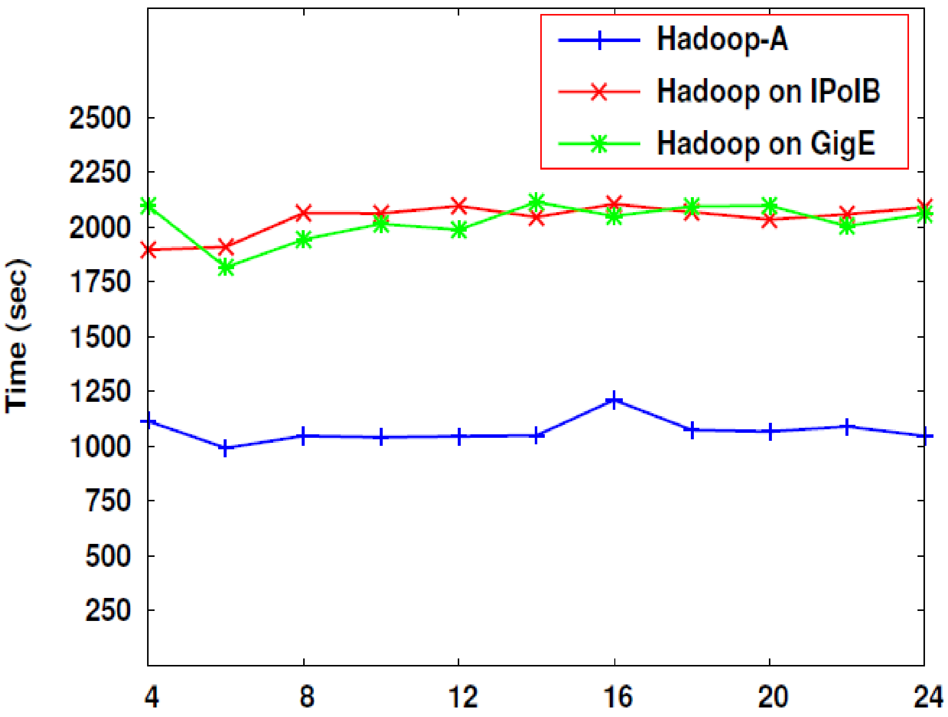
Hadoop Acceleration – UDA (Unstructured Data Accelerator)



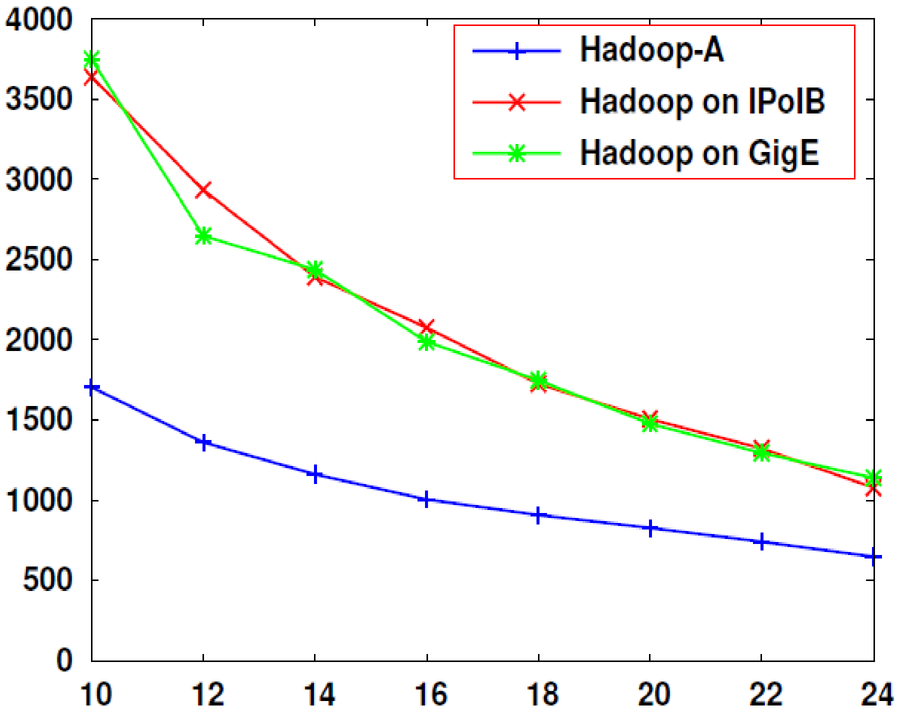
Overlapped Data Shuffle, Merge and Reduce



Improved Data Processing Scalability with UDA

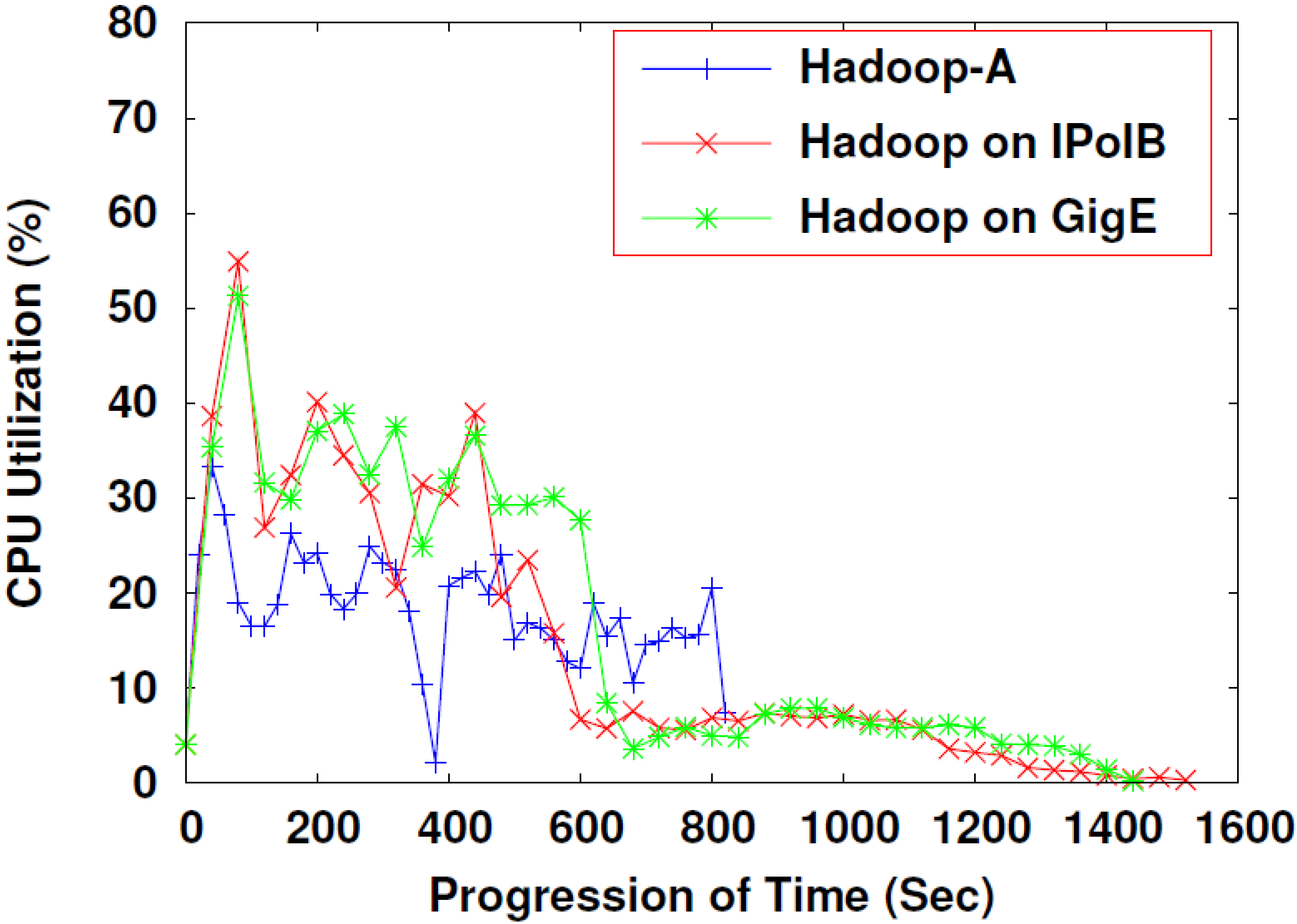


Execution Time with Fixed Dataset Per Reducer



Execution Time with Fixed Data Size Per Job

Reduced CPU Utilization with UDA



Lessons Learned From the Acceleration in UDA

- **challenging to achieve high scalability with limited memory resource for RDMA operations.**
- **Complete overlapping of MapTasks and ReduceTasks is very complicated.**
- **Need the support for heterogeneous networks**
- **Scalable network architecture is critical for Hadoop on large clusters**
- **Better merging algorithms are essential for scalability and efficiency**