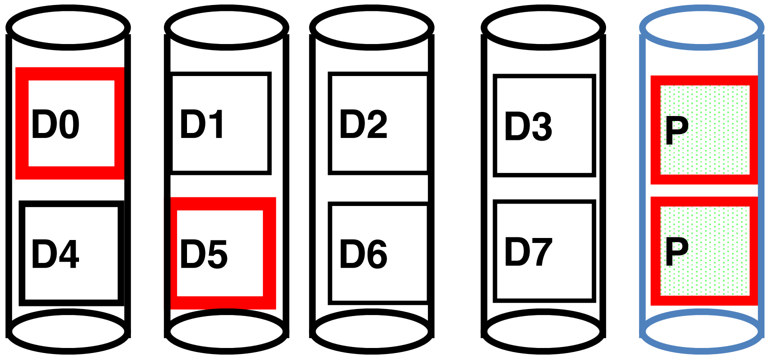
Warehouse Scale Computing

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

Inspiration for RAID 5

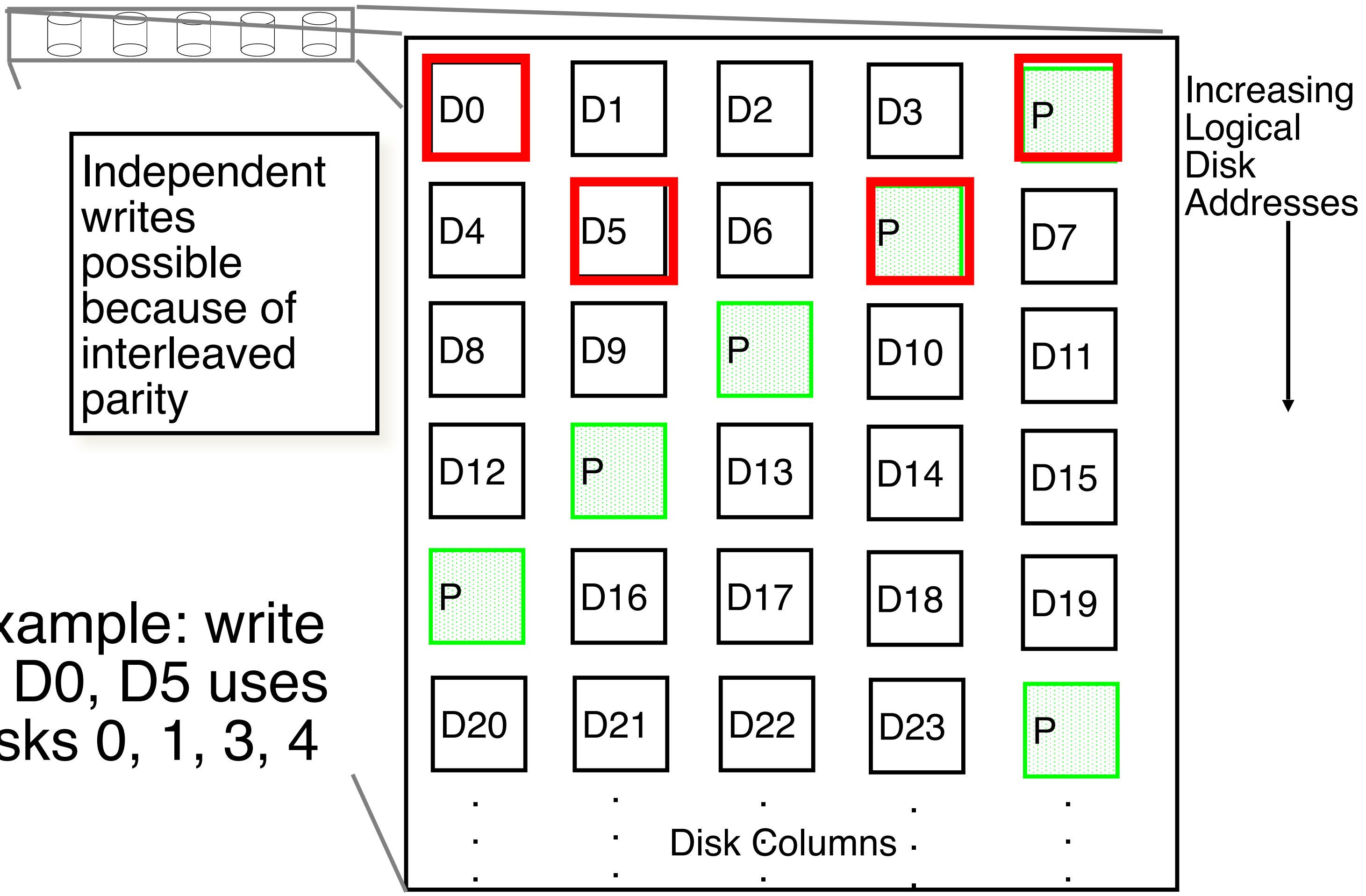
- RAID 4 works well for reads, but
- Parity Disk is the bottleneck for writes: Write to D0, D5 both also write to P disk



RAID 5: High I/O Rate Interleaved Parity

Independent writes possible because of interleaved parity

Example: write to D0, D5 uses disks 0, 1, 3, 4

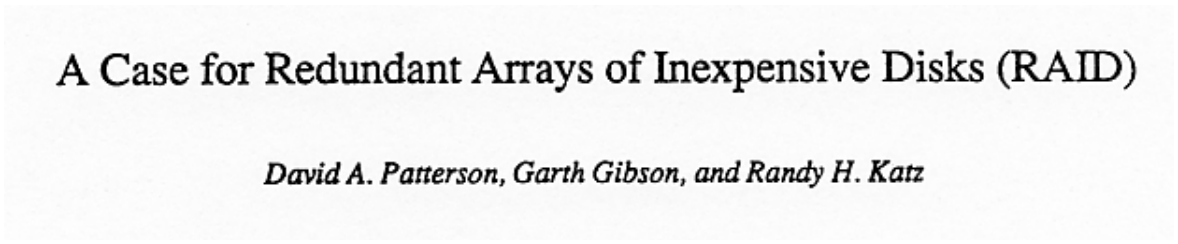


Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

RAID 6

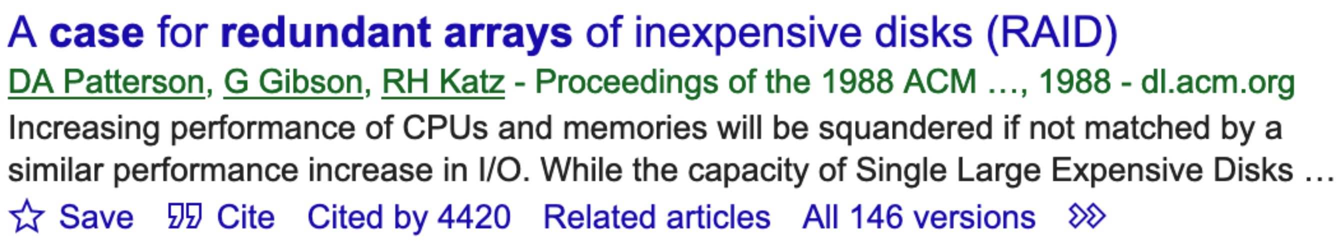
- RAID 5 is no longer the “gold standard”
- Can experience 1 disk failure and continue operation
  - RAID array is in a “degraded” state
- But disk failures are not actually independent!
  - When one disk has failed, there’s a decent chance another will fail soon
- RAID 6: Add another parity block per stripe
  - Now 2 blocks per stripe rather than 1
  - Sacrifice capacity for increased redundancy
  - Now the array can tolerate **2** disk failures and continue operating

Berkeley's Role in Definition of RAID (December 1987)



A Case for Redundant Arrays of Inexpensive Disks (RAID)

David A. Patterson, Garth Gibson, and Randy H. Katz

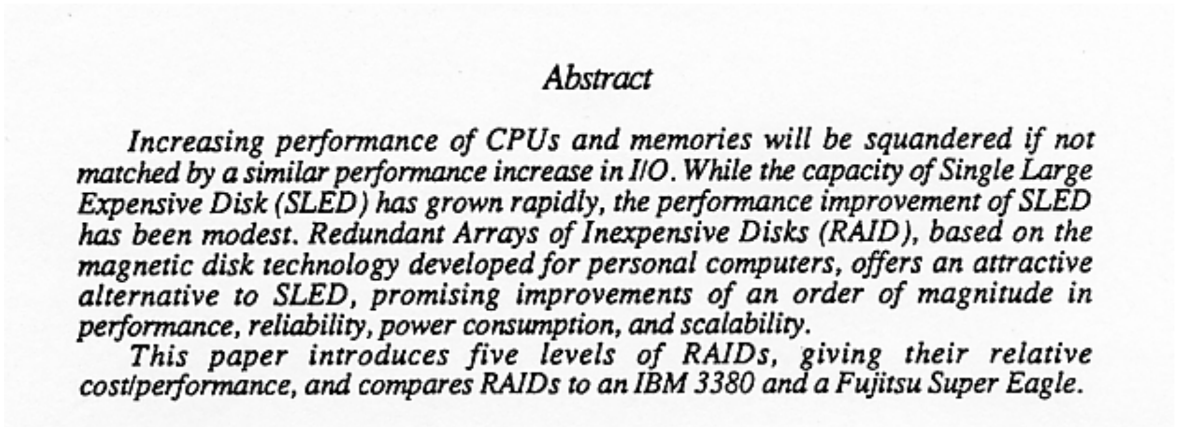


A **case** for **redundant arrays** of inexpensive disks (RAID)

DA Patterson, G Gibson, RH Katz - Proceedings of the 1988 ACM …, 1988 - dl.acm.org

Increasing performance of CPUs and memories will be squandered if not matched by a similar performance increase in I/O. While the capacity of Single Large Expensive Disks …

☆ Save 99 Cite Cited by 4420 Related articles All 146 versions



Abstract

Increasing performance of CPUs and memories will be squandered if not matched by a similar performance increase in I/O. While the capacity of Single Large Expensive Disk (SLED) has grown rapidly, the performance improvement of SLED has been modest. Redundant Arrays of Inexpensive Disks (RAID), based on the magnetic disk technology developed for personal computers, offers an attractive alternative to SLED, promising improvements of an order of magnitude in performance, reliability, power consumption, and scalability.

This paper introduces five levels of RAIDs, giving their relative cost/performance, and compares RAIDs to an IBM 3380 and a Fujitsu Super Eagle.

RAID Version 1

- RAID-I (1989)
  - Consisted of a Sun 4/280 workstation:
    128 MB of DRAM,
    four dual-string SCSI controllers,
    28 5.25-inch SCSI disks and
    specialized disk striping software

RAID Version 2

- 1990-1993
- Early Network Attached Storage (NAS) System running a Log Structured File System (LFS)
- Impact:
  - $25 Billion/year in 2002
  - Over $150 Billion in RAID device sold since 1990-2002
  - 200+ RAID companies (at the peak)
  - Software RAID a standard component of modern OSs

RAID Is Not Enough By Itself

- You don't just have one disk die...
  - You can have more die in a short period of time
  - Thank both the "bathtub curve" and common environmental conditions
- If you care about your data, RAID isn't sufficient
  - You need to also consider a separate backup solution
  - Especially true in the modern days of ***ransomware***:
    Bad guys encrypting your data
- A good practice in clusters/warehouse scale computers:
  - RAID-6 in each cluster node with auto-failover and a hot spare
  - Distributed filesystem on top
    - Replicates amongst the cluster nodes so that nodes can fail
  - And then distribute to a different WSC...

In Conclusion …

- We have methods to mitigate faults in electronic systems:
  - Design bugs, Manufacturing defects, and Runtime Faults
- Dependability Measures let us quantify
- Dealing with Runtime Faults requires redundancy
  - either more hardware (cost) or more time (performance)
- Redundancy most commonly used in memory systems (DRAM, SRAM, Disks, SSD), also for communications

Agenda

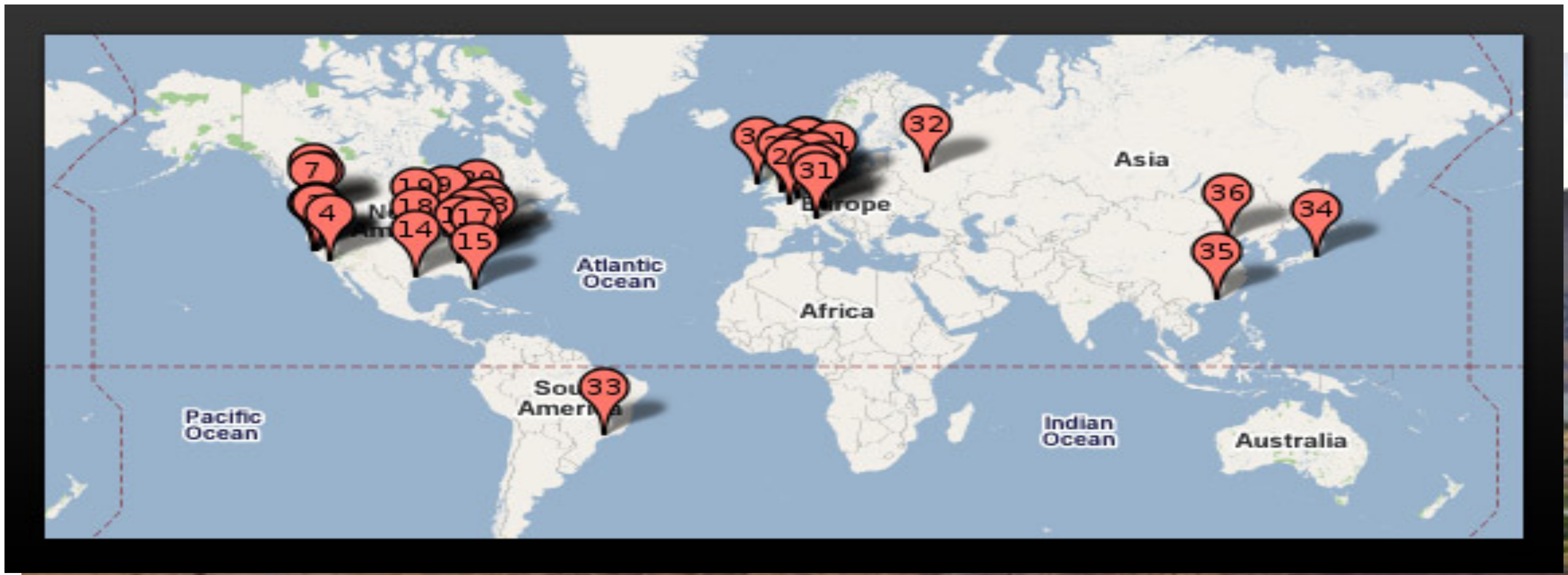
- Warehouse-Scale Computing
- Cloud Computing
- Request-Level Parallelism (RLP)
- Map-Reduce Data Parallelism

GETTING CLOSER TO *Infinity!*
IBM
ELECTRONIC BUSINESS MACHINES
International Business Machines

Businessmen, engineers, and scientists now are solving problems in scientific and industrial data processing which, a few short years ago, would have been considered well-nigh infinite.

IBM Electronic Business Machines are making an important contribution to this progress. These machines accomplish once-overwhelming tasks with incredible speed and accuracy . . . freeing thousands of valuable minds for creative effort.

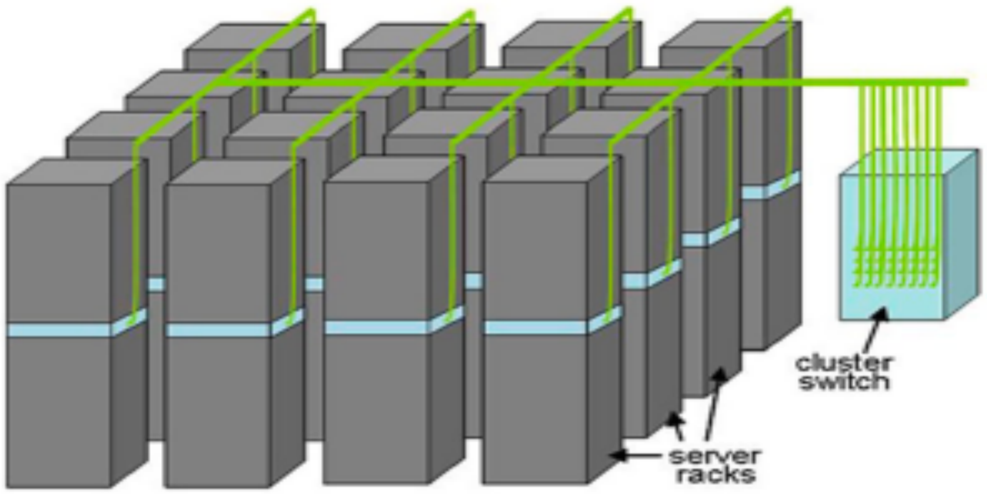
Google’s WSCs



WSC Architecture

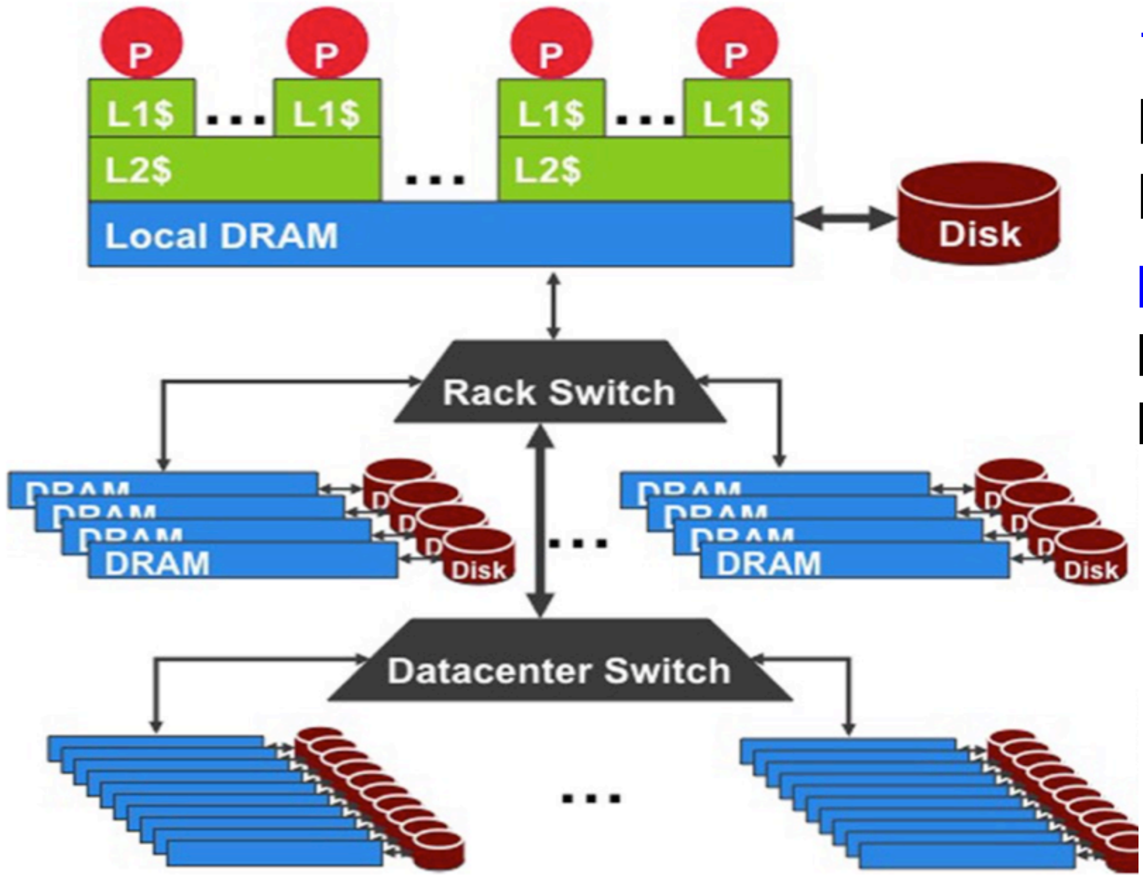
1U Server:
8 cores, 16 GiB DRAM,
4x1 TB disk

Rack:
40-80 servers,
Local Ethernet (1-10Gbps) switch
(30$/1Gbps/server)



Array (aka cluster):
16-32 racks
Expensive switch
(10X bandwidth → 100x cost)

WSC Storage Hierarchy

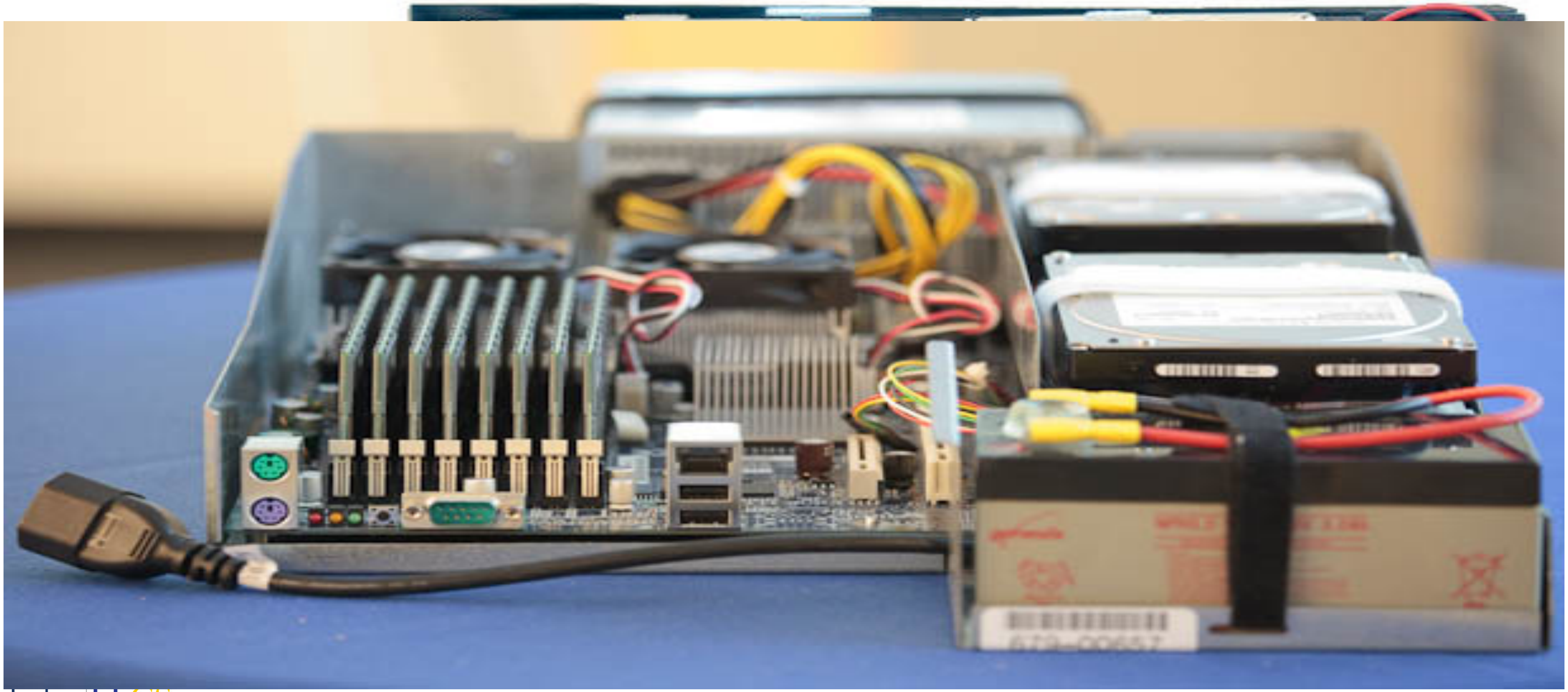


1U Server:
DRAM: 64GB, 100ns
Disk: 10TB, 10ms

Rack (80 severs):
DRAM: 5TB, 300µs
Disk: 800TB, 11ms

Array (30 racks):
DRAM: 150TB, 500µs
Disk: 24PB, 12ms

Google Server Internals



Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

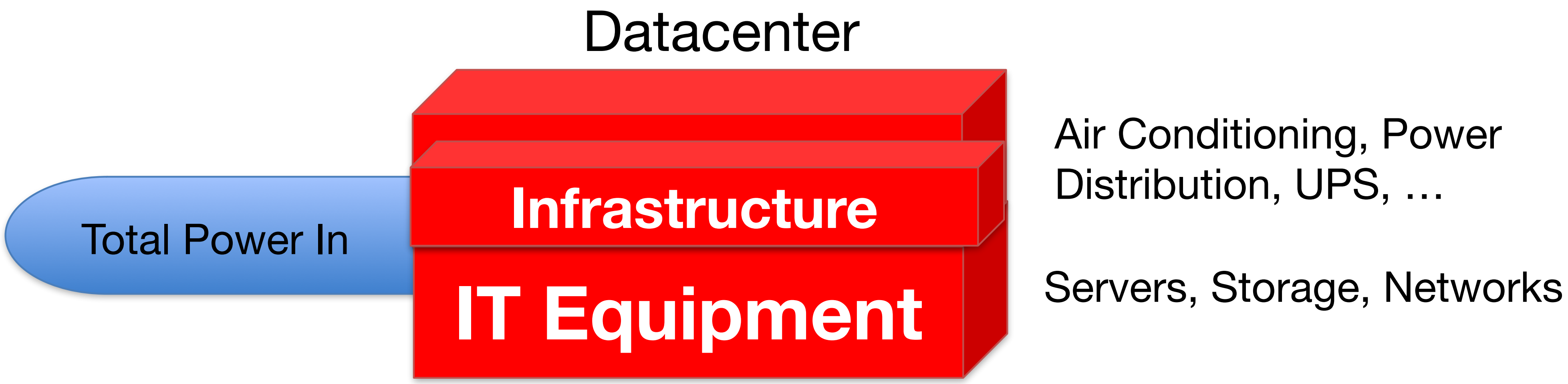
Power Usage Effectiveness

- Energy efficiency
  - Primary concern in the design of WSC
  - Important component of the total cost of ownership
- Power Usage Effectiveness (PUE):

$$\frac{\textbf{Total Building Power}}{\textbf{IT equipment Power}}$$

  - Power efficiency measure for WSC
  - Not considering efficiency of servers, networking
  - Perfection = 1.0
  - Google WSC's PUE = 1.2

Power Usage Effectiveness


Datacenter
Total Power In
Infrastructure
IT Equipment
Air Conditioning, Power Distribution, UPS, …
Servers, Storage, Networks

PUE = Total Power/IT Power

PUE = 1.5

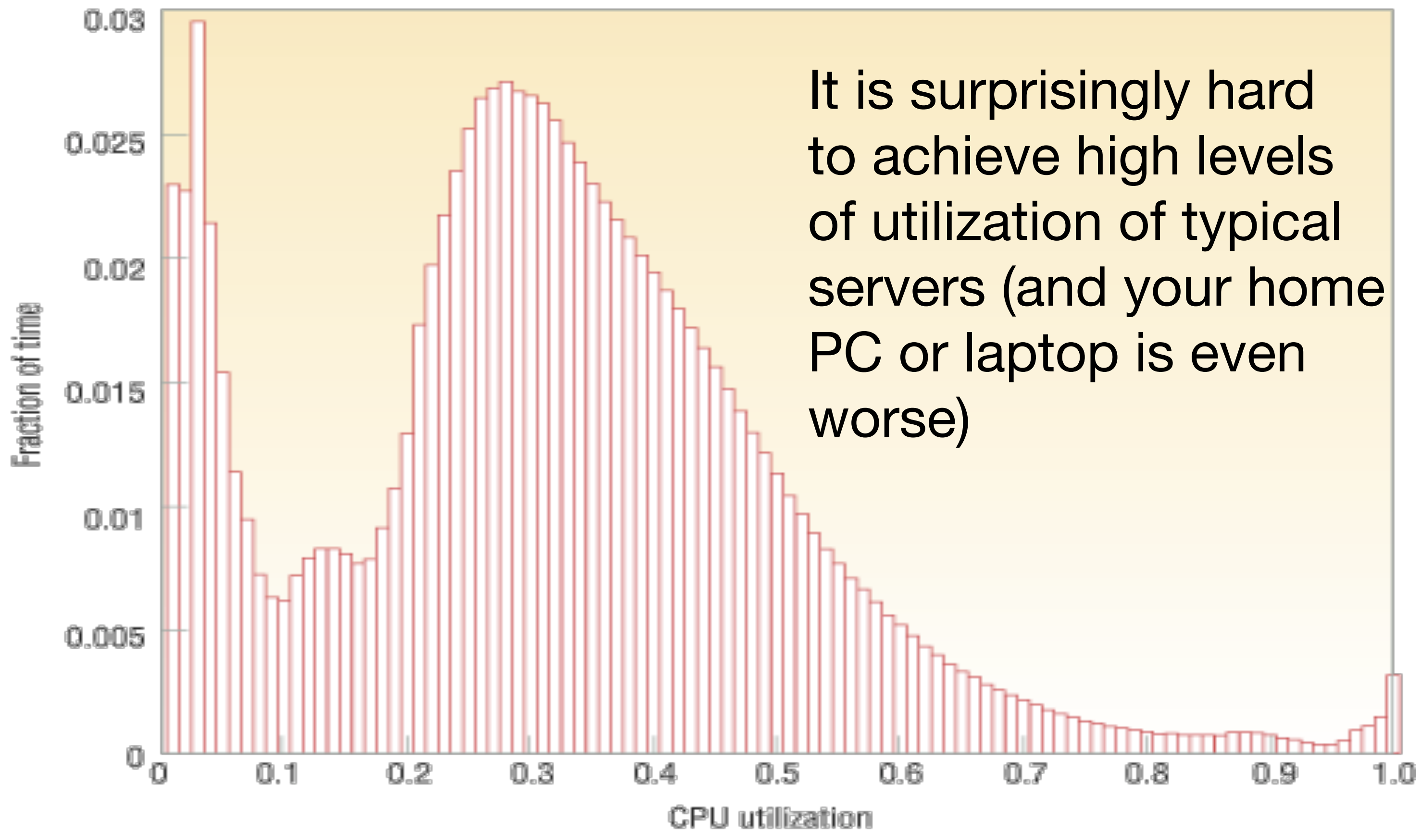
PUE = 2

Cheating on Cooling

- Normally cooling the air requires big air-conditioning units
  - These suck a lot of power and still consume a lot of water
    - Evaporation of water to dissipate the energy
- Cheat #1: Heat-exchange to a water source
  - Locate your data center on a river or the ocean
  - Or even just put it in a sealed container dropped onto the sea bottom
- Cheat #2: Just have things open to the air!
  - Ups the failure rate, but if the power savings exceed the costs incurred by additional machines dying, it becomes worth it

Energy Proportionality

"The Case for Energy-Proportional Computing," Luiz André Barroso, Urs Hölzle, *IEEE Computer* December 2007



It is surprisingly hard to achieve high levels of utilization of typical servers (and your home PC or laptop is even worse)

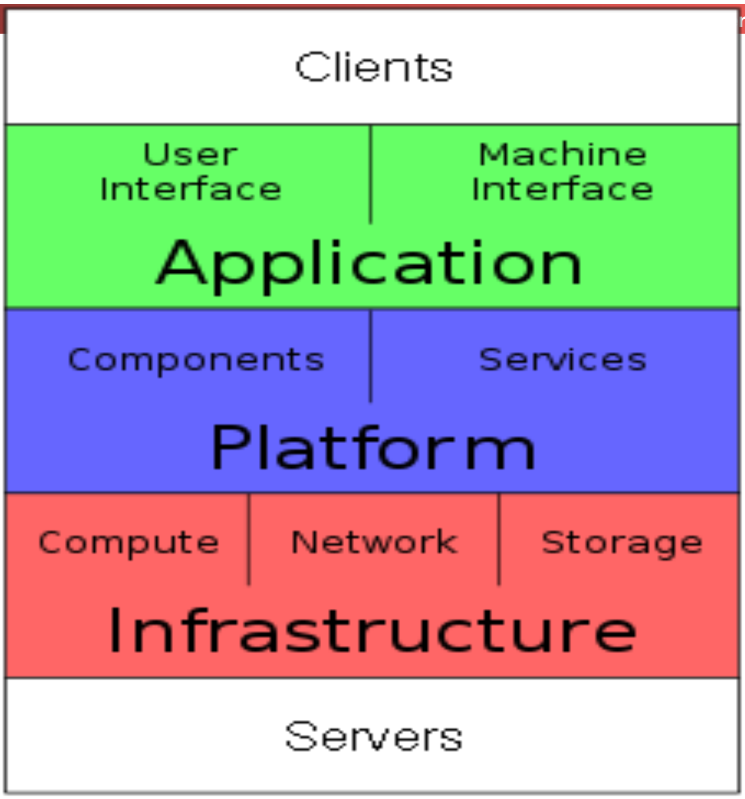
Figure 1. Average CPU utilization of more than 5,000 servers during a six-month period. Servers are rarely completely idle and seldom operate near their maximum utilization, instead operating most of the time at between 10 and 50 percent of their maximum

Cloud Distinguished by …

- Shared platform with illusion of isolation
  - Collocation with other tenants
  - Exploits technology of VMs and hypervisors
  - At best “fair” allocation of resources, but not true isolation
- Attraction of low-cost cycles
  - Economies of scale driving move to consolidation
  - Statistical multiplexing to achieve high utilization/efficiency of resources
- Elastic service
  - Pay for what you need, get more when you need it
  - But no performance guarantees: assumes uncorrelated demand for resources

Cloud Services

- **SaaS**: deliver apps over Internet, eliminating need to install/run on customer's computers, simplifying maintenance and support
  - E.g., Google Docs, Win Apps in the Cloud
- **PaaS**: Deliver computing "stack" as a service, using cloud infrastructure to implement apps. Deploy apps without cost/complexity of buying and managing underlying layers
  - E.g., Hadoop on EC2, Apache Spark on GCP
- **IaaS**: Rather than purchasing servers, software, data center space or net equipment, clients buy resources as an outsourced service. Billed on utility basis. Amount of resources consumed/cost reflect level of activity
  - E.g., Amazon Elastic Compute Cloud, Google Compute Platform

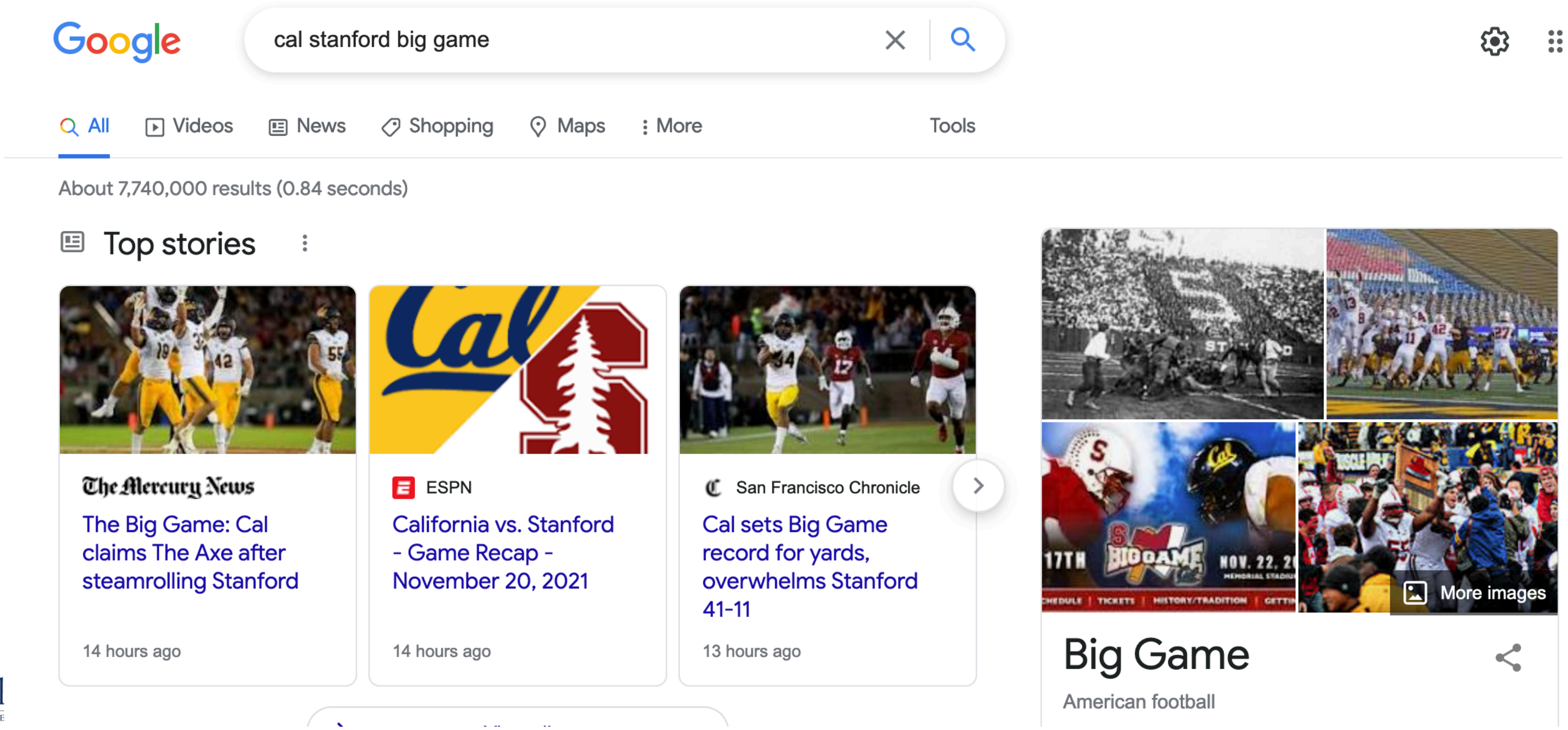


Cloud Computing Stack

Request-Level Parallelism (RLP)

- Hundreds of thousands of requests per second
  - Popular Internet services like web search, social networking, …
  - Such requests are largely independent
    - Often involve read-mostly databases
    - Rarely involve read-write sharing or synchronization across requests
- Computation easily partitioned across different requests and even within a request
  - Can often "load balance" just at the DNS level:
    Just tell different people to use a different computer

Scaled Communities, Processing, and Data



cal stanford big game

About 7,740,000 results (0.84 seconds)

Top stories

The Mercury News
The Big Game: Cal claims The Axe after steamrolling Stanford
14 hours ago

ESPN
California vs. Stanford - Game Recap - November 20, 2021
14 hours ago

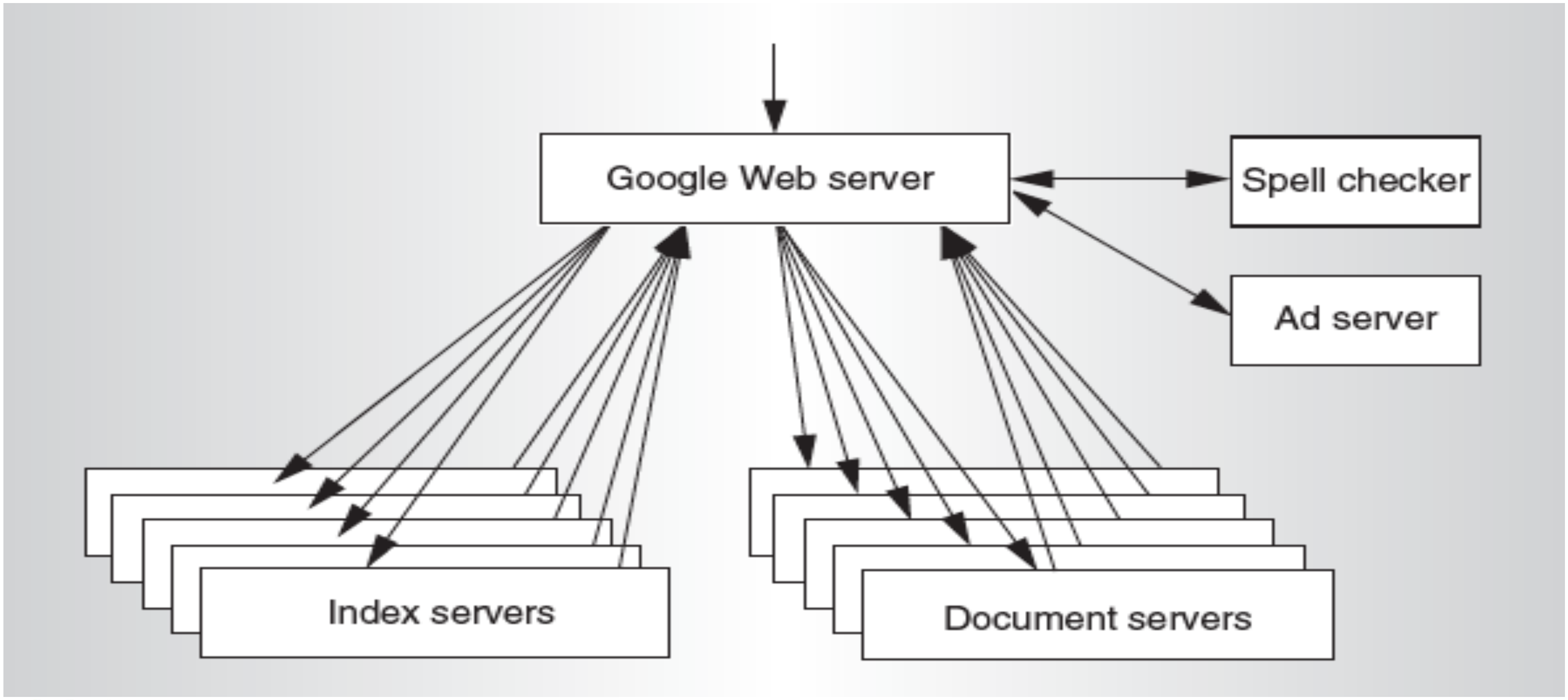
San Francisco Chronicle
Cal sets Big Game record for yards, overwhelms Stanford 41-11
13 hours ago

Big Game

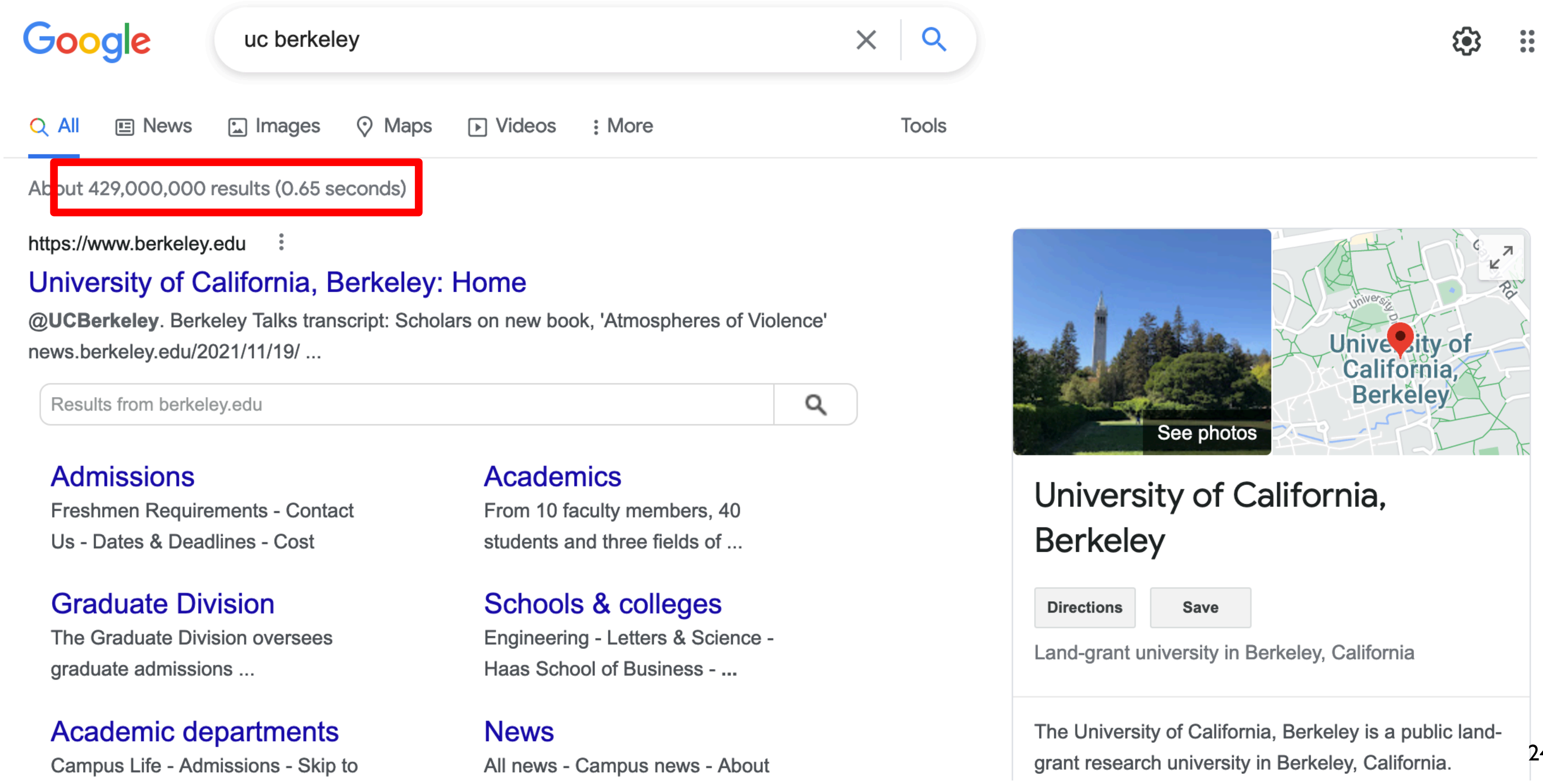
American football

Berkeley ELECTRICAL ENGINEERING

Google Query-Serving Architecture



Web Search Result



uc berkeley

All News Images Maps Videos More Tools

About 429,000,000 results (0.65 seconds)

https://www.berkeley.edu

University of California, Berkeley: Home

@UCBerkeley. Berkeley Talks transcript: Scholars on new book, 'Atmospheres of Violence' news.berkeley.edu/2021/11/19/ ...

Results from berkeley.edu

Admissions
Freshmen Requirements - Contact Us - Dates & Deadlines - Cost

Academics
From 10 faculty members, 40 students and three fields of ...

Graduate Division
The Graduate Division oversees graduate admissions ...

Schools & colleges
Engineering - Letters & Science - Haas School of Business - ...

Academic departments
Campus Life - Admissions - Skip to

News
All news - Campus news - About

University of California, Berkeley

Directions Save

Land-grant university in Berkeley, California

The University of California, Berkeley is a public land-grant research university in Berkeley, California.

Anatomy of a Web Search (1/3)

- Google “UC Berkeley”

1. Direct request to “closest” Google Warehouse-Scale Computer
2. Front-end load balancer directs request to one of many clusters of servers within WSC
3. Within cluster, select one of many Google Web Servers (GWS) to handle the request and compose the response pages
4. GWS communicates with Index Servers to find documents that contain the search words, “UC”, “Berkeley”, uses location of search as well as user information
5. Send information about this search to the node in charge of tracking you
6. Return document list with associated relevance score

Anatomy of a Web Search (2/3)

- In parallel,
  - Ad system: if anyone has bothered to advertise for you
  - Customization based on your account
- Use docids (document IDs) to access indexed documents to get snippets of stuff
- Compose the page
  - Result document extracts (with keyword in context) ordered by relevance score
  - A bunch of advertisements (along the top and side)
    - Initially they were easy to see...
      But now they are almost indistinguishable from the desired content

Anatomy of a Web Search (3/3)

- Implementation strategy
  - Randomly distribute the entries
  - Make many copies of data (aka “replicas”)
  - Load balance requests across replicas
- ***Redundant copies*** of indices and documents
  - Breaks up hot spots — especially popular queries
  - Increases opportunities for ***request-level parallelism***
  - Makes the system more ***tolerant of failures***

Data-Level Parallelism (DLP)

- SIMD
  - Supports data-level parallelism in a single machine
  - Additional instructions & hardware (e.g., AVX)

  e.g., Matrix multiplication in memory

- DLP on WSC
  - Supports data-level parallelism across ***multiple machines***
  - MapReduce & scalable file systems

Problem Statement

- How process large amounts of raw data (crawled documents, request logs, …) every day to compute derived data (inverted indices, page popularity, …)
  - Each computation is relatively simple but the input data is huge (petabytes) and distributed across 100s or 1000s of servers
- Challenge: Parallelize computation, distribute data, tolerate faults without obscuring simple computation with complex code to deal with issues

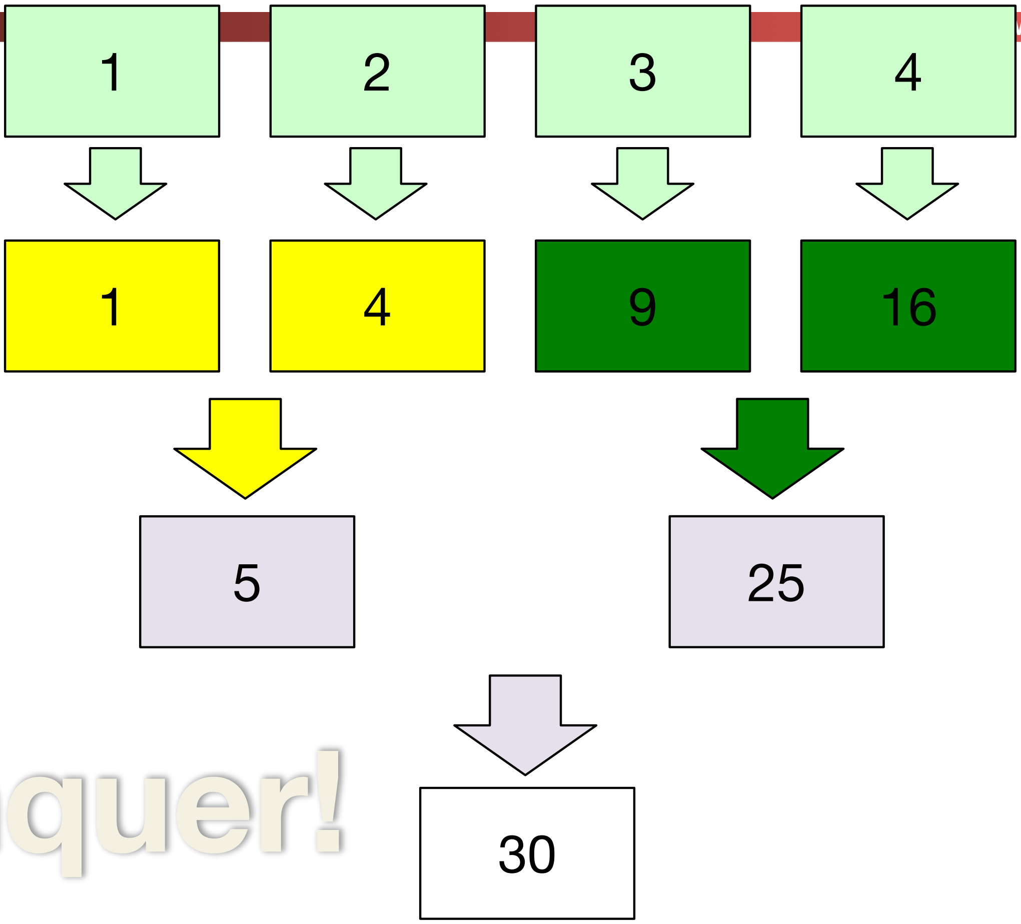
Solution: MapReduce

- Simple data-parallel ***programming model*** and ***implementation*** for processing large datasets
- Users specify the computation in terms of
  - a ***map*** function, and
  - a ***reduce*** function
- Underlying runtime system
  - Automatically ***parallelize*** the computation across large scale clusters of machines
  - ***Handles*** machine ***failure***
  - ***Schedule*** inter-machine communication to make efficient use of the networks

Inspiration: Map & Reduce Functions, ex: Python

Calculate : $$\sum_{n=1}^{4} n^2$$

```
A = [1, 2, 3, 4]
def square(x):
  return x * x
def sum(x, y):
  return x + y
reduce(sum, map(square,
A))
```



Divide and Conquer!

MapReduce Programming Model

- ***Map***: `(in_key, in_value) → list(interm_key, interm_val)`

```
map(in_key, in_val):
  // DO WORK HERE
  emit(interm_key,interm_val)
```

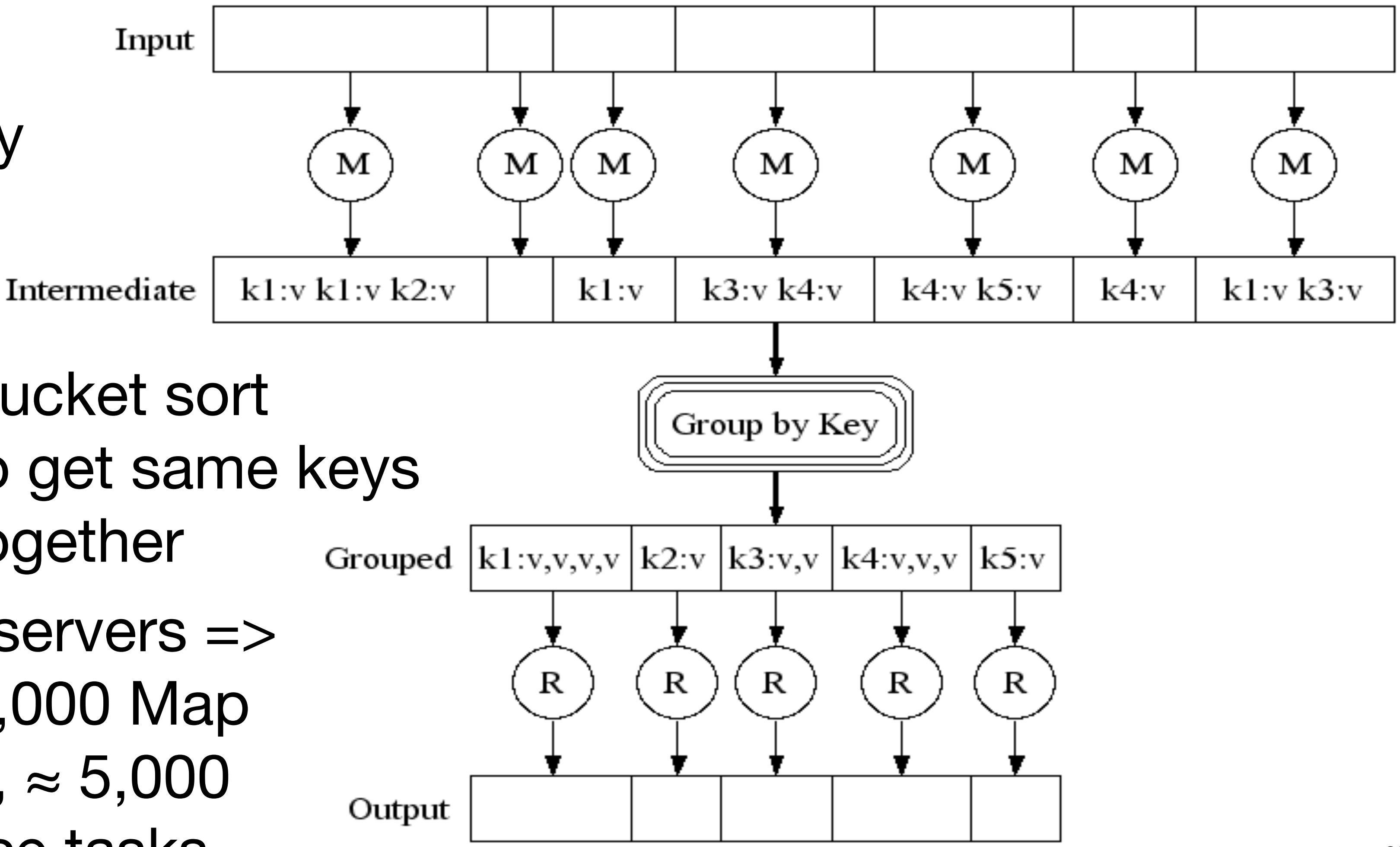
  - Slice data into “shards” or “splits” and distribute to workers
  - Compute set of intermediate key/value pairs

- ***Reduce***: `(interm_key, list(interm_value)) → list(out_value)`

```
reduce(interm_key, list(interm_val)):
  // DO WORK HERE
  emit(out_key, out_val)
```

  - Combines all intermediate values for a particular key
  - Produces a set of merged output values (usually just one)

MapReduce Execution



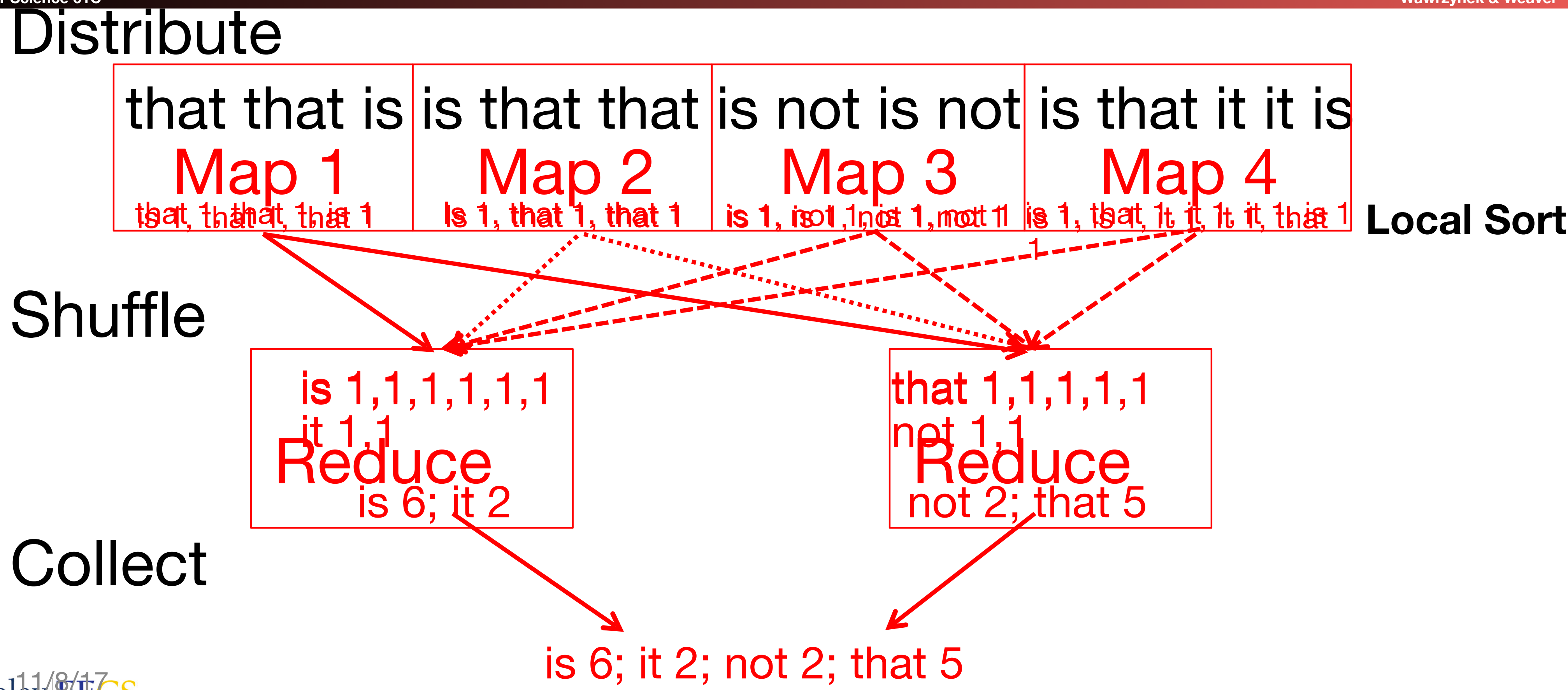
Fine granularity tasks: many more map tasks than machines

Bucket sort to get same keys together

2000 servers => ≈ 200,000 Map Tasks, ≈ 5,000 Reduce tasks

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

MapReduce Word Count Example



MapReduce Word Count Example

User-written ***Map*** function reads the document data and

parses the words. For each word, it writes the (key, value) pair of (word, 1). The word is treated as the intermediate key and the associated value of 1 means that we saw the word once.

Map phase: (doc name, doc contents) → list(word, count)

```
// "I do I learn" → [("I",1),("do",1),("I",1),("learn",1)]
```

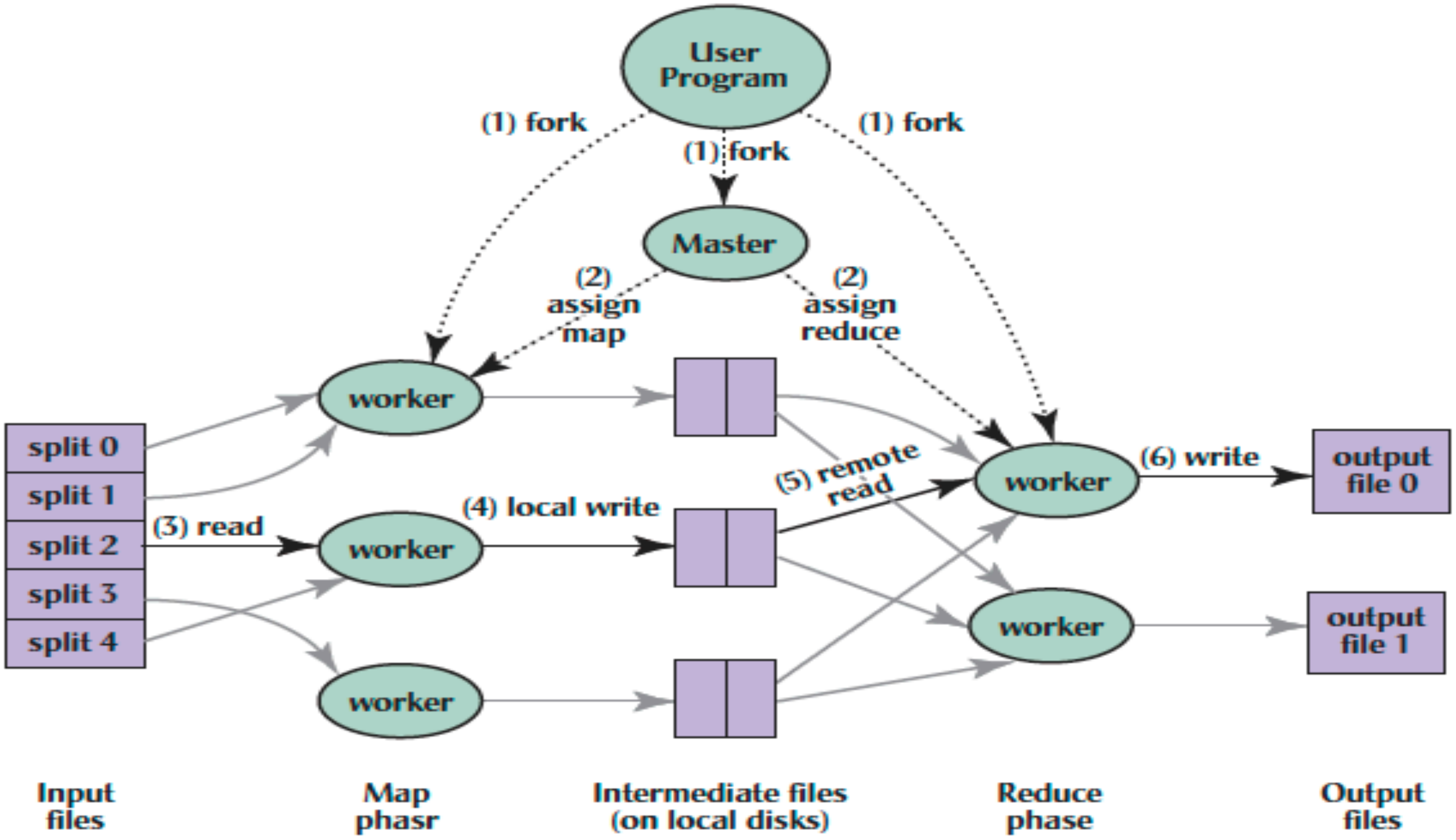
```
map(key, value):
  for each word w in value:
    emit(w, 1)
```

MapReduce Word Count Example

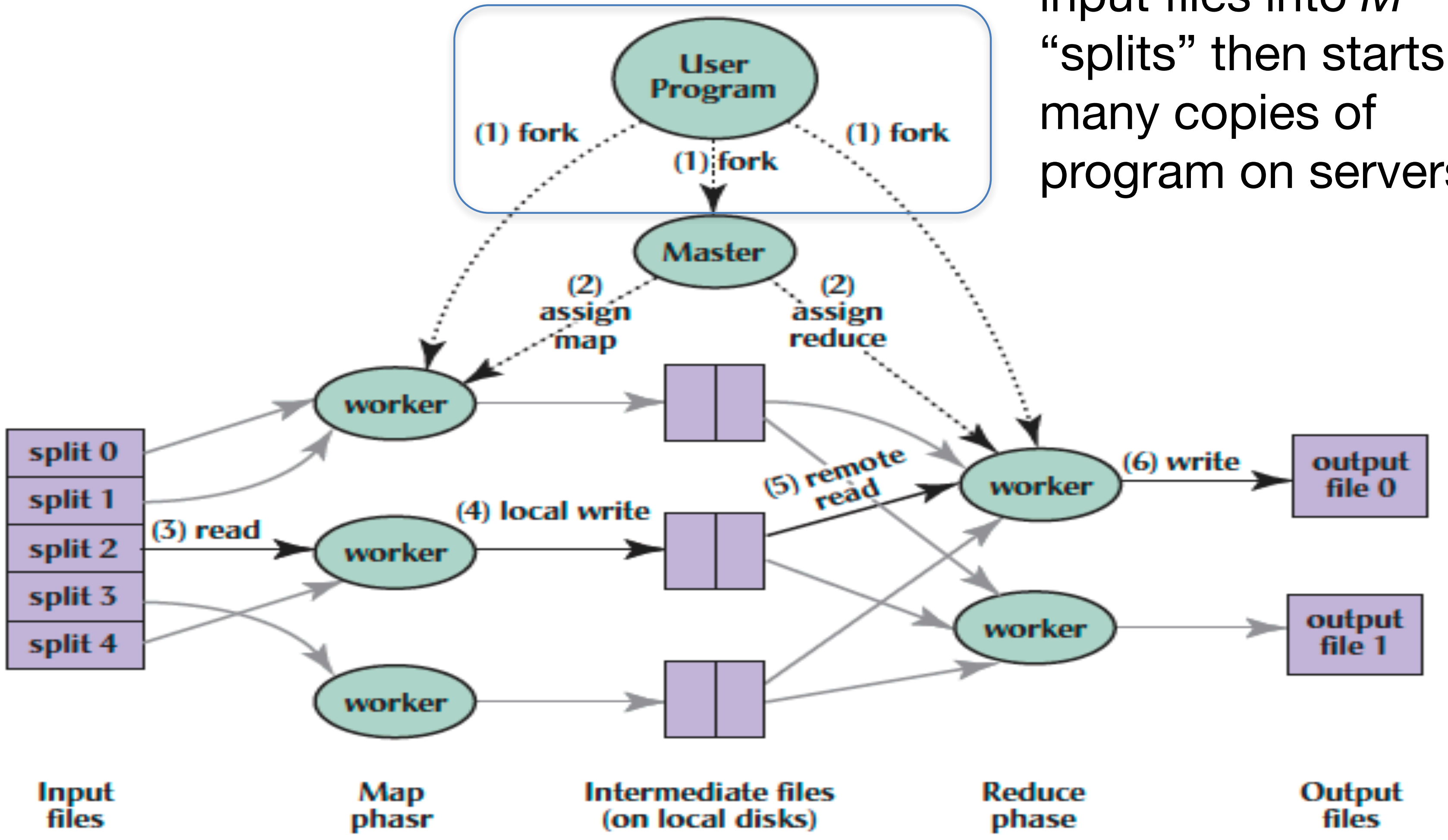
Intermediate data is then sorted by MapReduce by keys and the user's ***Reduce*** function is called for each unique key. In this case, Reduce is called with a list of a "1" for each occurrence of the word that was parsed from the document. The function adds them up to generate a total word count for that word.

Reduce phase: (word, list(counts)) → (word, count_sum)

```
 // ("I", [1,1]) → ("I",2)
reduce(key, values):
  result = 0
  for each v in values:
    result += v
  emit(key, result)
```

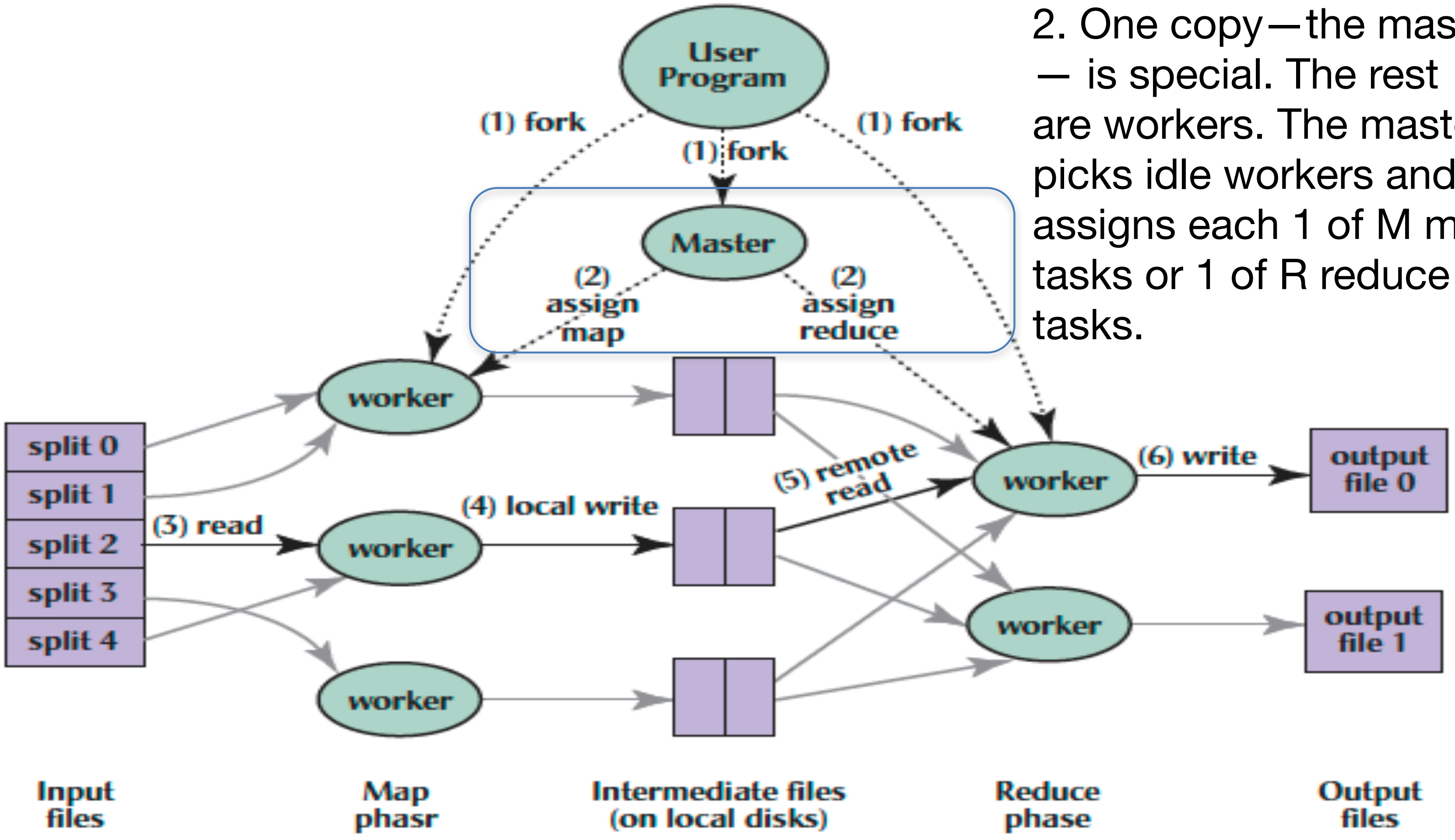

User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
(6) write
output file 0
output file 1
Input files
Map phasr
Intermediate files (on local disks)
Reduce phase
Output files

MapReduce Processing



1. MR 1st splits the input files into *M* “splits” then starts many copies of program on servers

MapReduce Processing



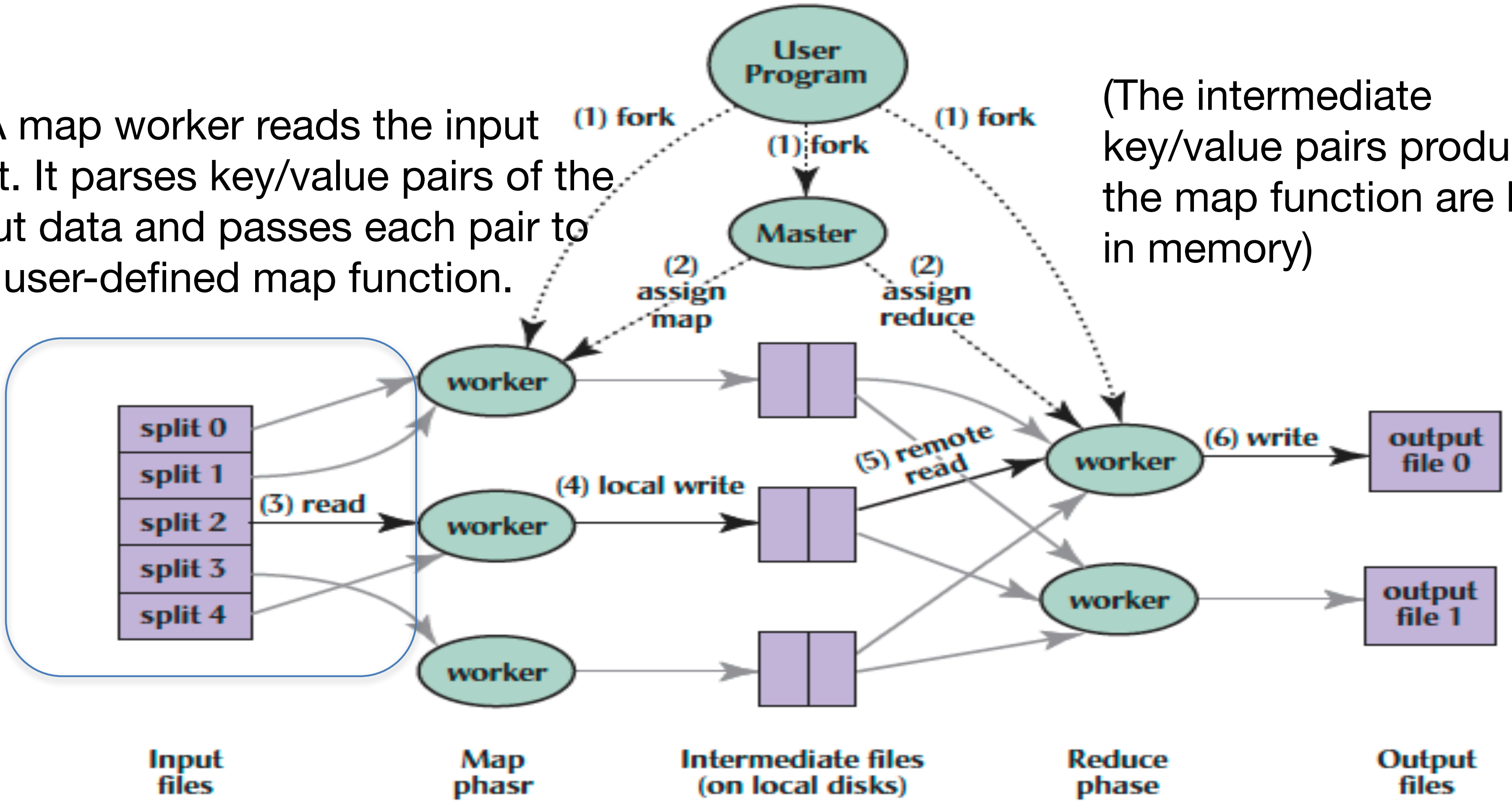
2. One copy—the master — is special. The rest are workers. The master picks idle workers and assigns each 1 of M map tasks or 1 of R reduce tasks.

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

MapReduce Processing

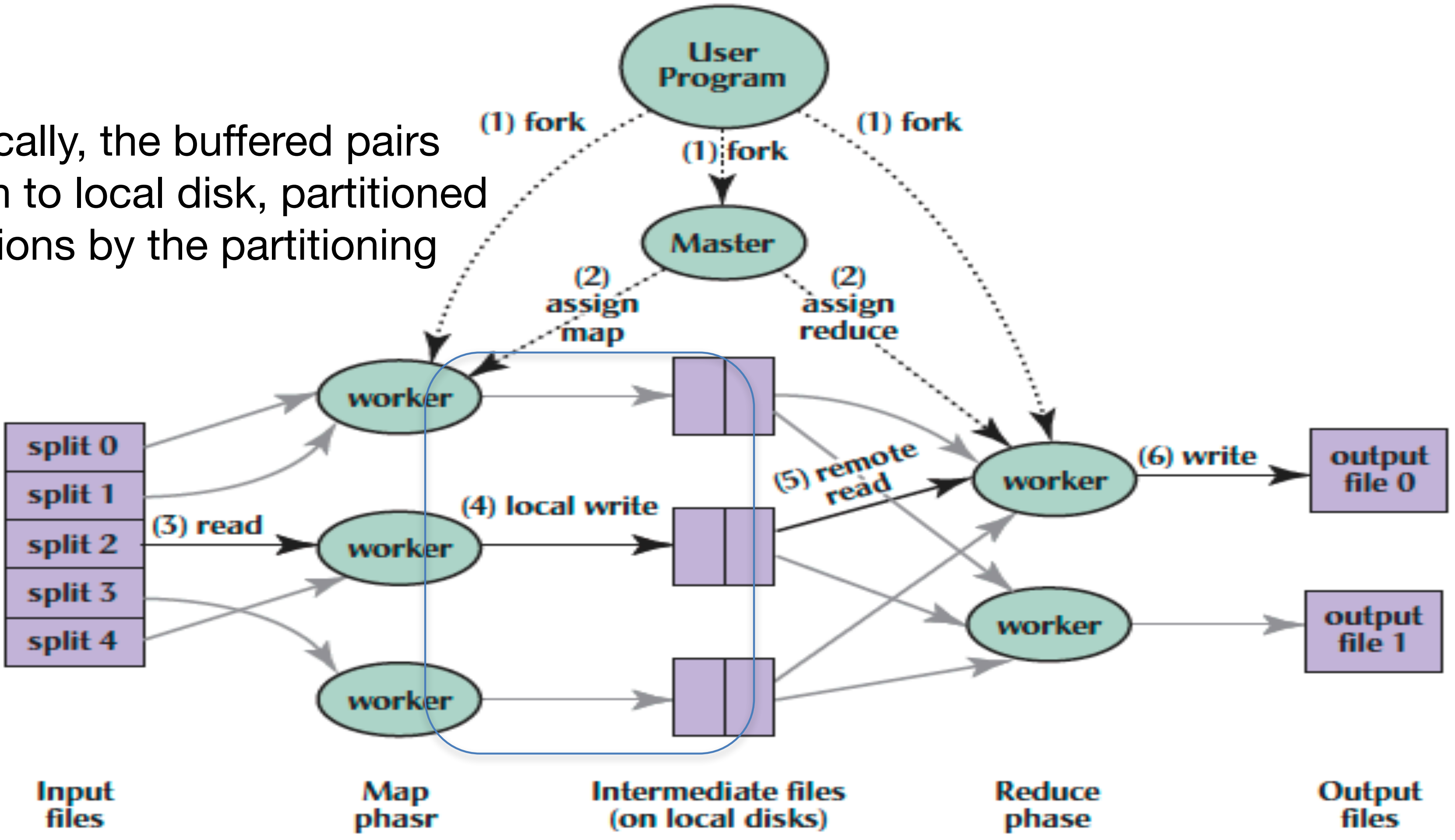
3. A map worker reads the input split. It parses key/value pairs of the input data and passes each pair to the user-defined map function.

(The intermediate key/value pairs produced by the map function are buffered in memory)



MapReduce Processing

4. Periodically, the buffered pairs are written to local disk, partitioned into *R* regions by the partitioning function.

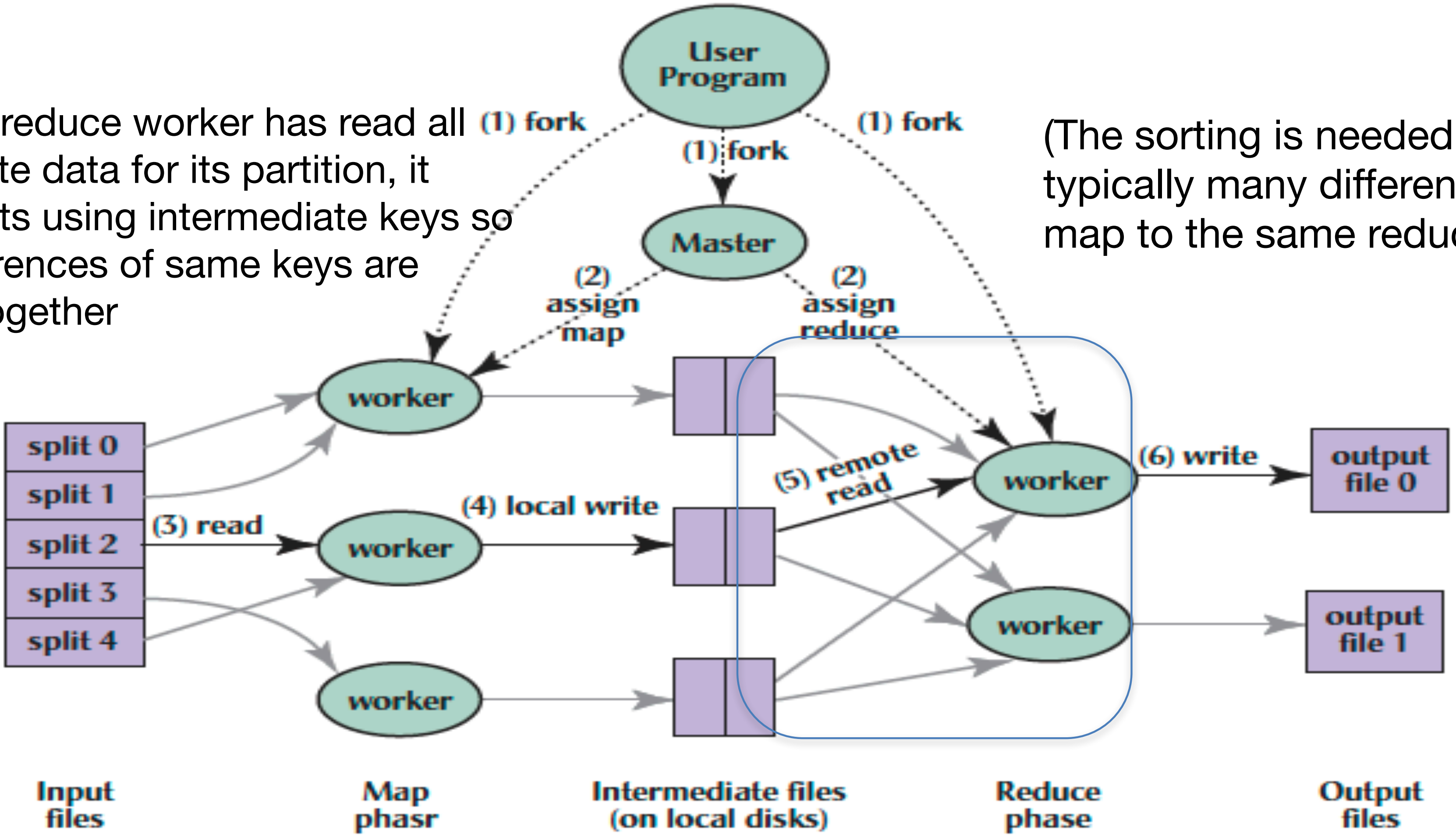


Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

MapReduce Processing

5. When a reduce worker has read all intermediate data for its partition, it bucket sorts using intermediate keys so that occurrences of same keys are grouped together

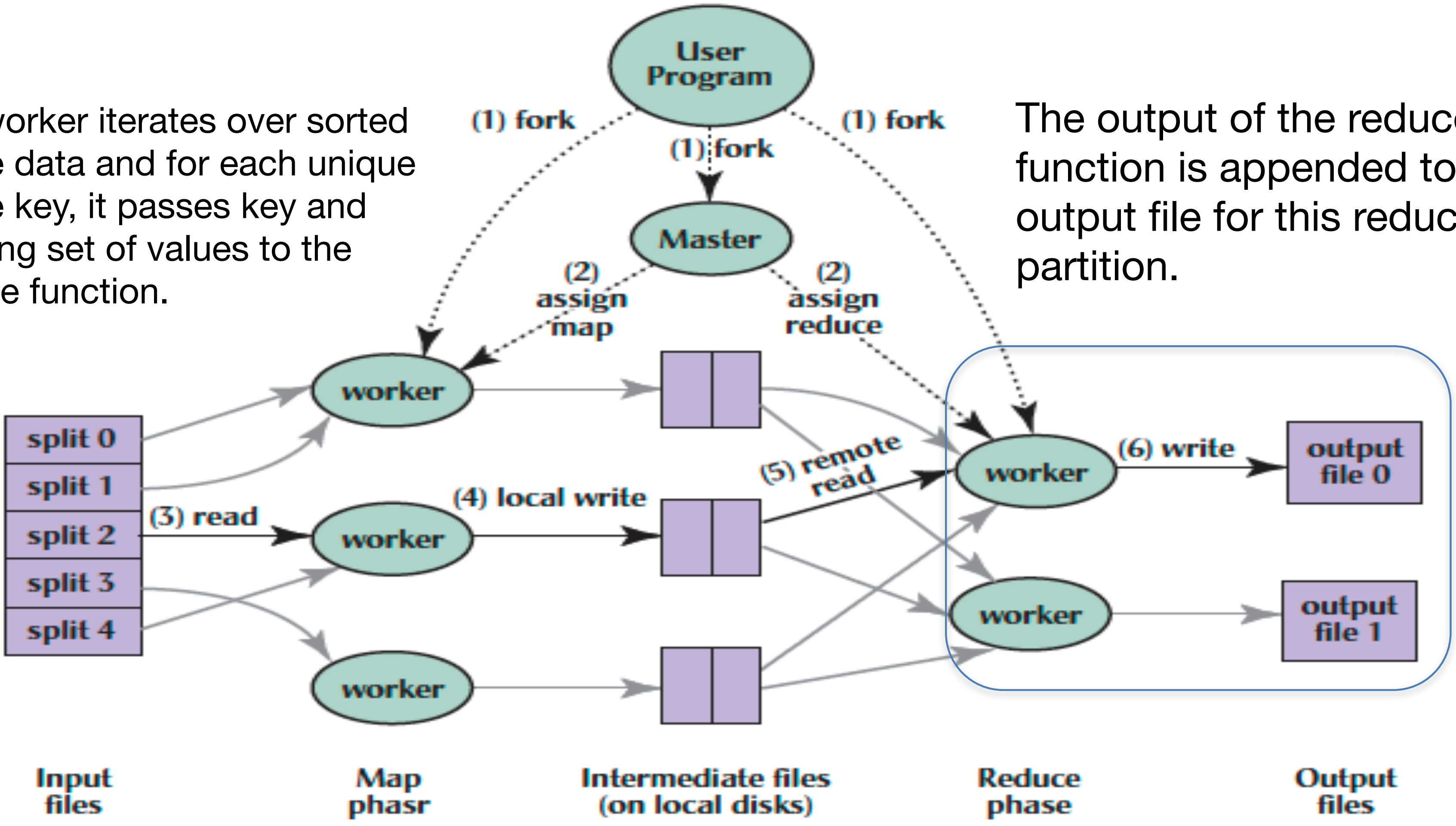
(The sorting is needed because typically many different keys map to the same reduce task)



MapReduce Processing

6. Reduce worker iterates over sorted intermediate data and for each unique intermediate key, it passes key and corresponding set of values to the user's reduce function.

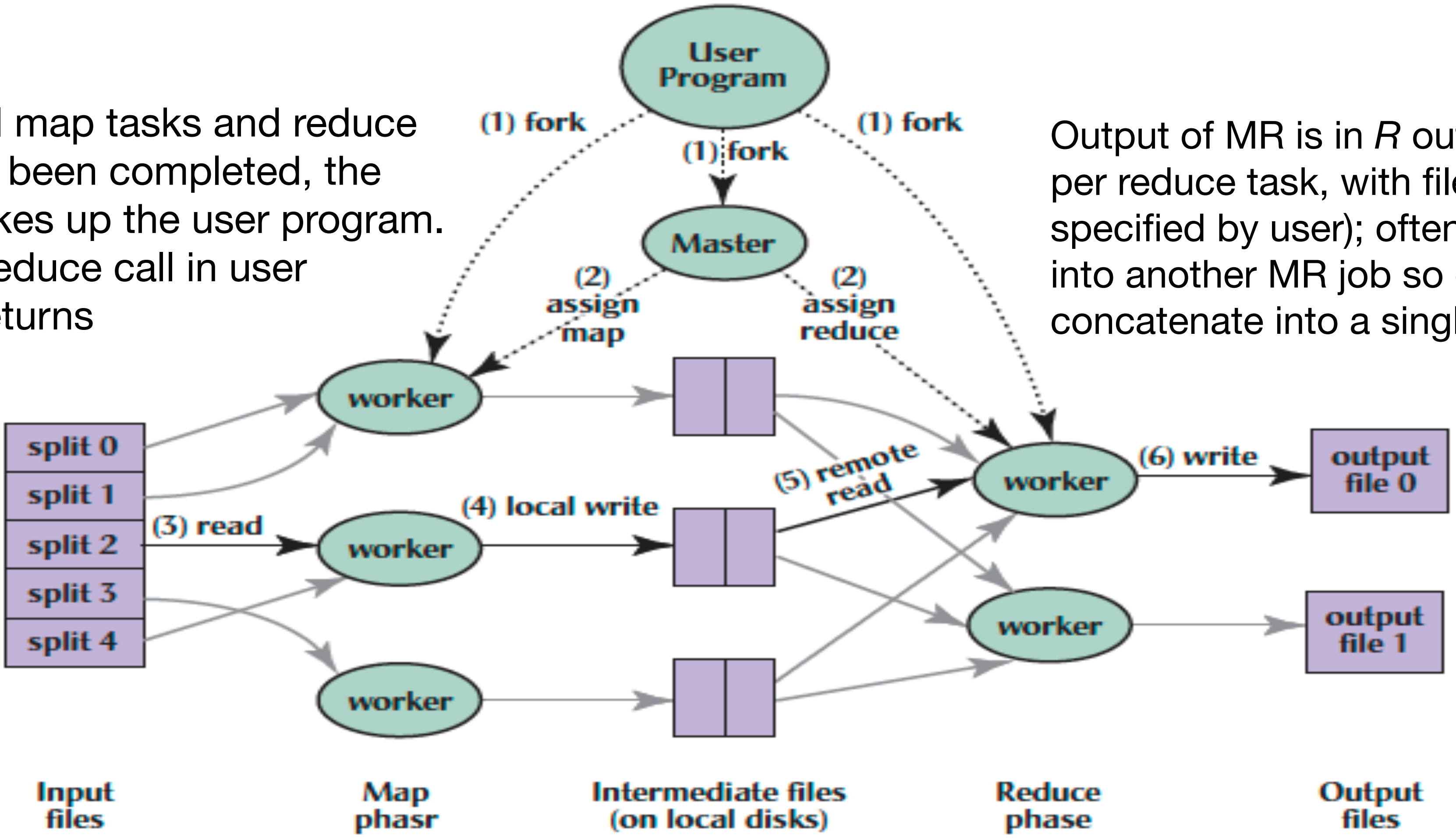
The output of the reduce function is appended to a final output file for this reduce partition.



MapReduce Processing

7. When all map tasks and reduce tasks have been completed, the master wakes up the user program. The MapReduce call in user program returns

Output of MR is in *R* output files (1 per reduce task, with file names specified by user); often passed into another MR job so don't concatenate into a single file



Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

Big Data Frameworks: Hadoop & Spark

- Apache Hadoop
  - Open-source MapReduce Framework
  - Hadoop Distributed File System (HDFS)
  - MapReduce Java APIs
- Apache Spark
  - Fast and general engine for large-scale data processing
  - Originally developed in the AMP lab at UC Berkeley
  - Running on top of HDFS
  - Provides Java, Scala, Python APIs for
    - Database
    - Machine learning
    - Graph algorithms

Apache Spark

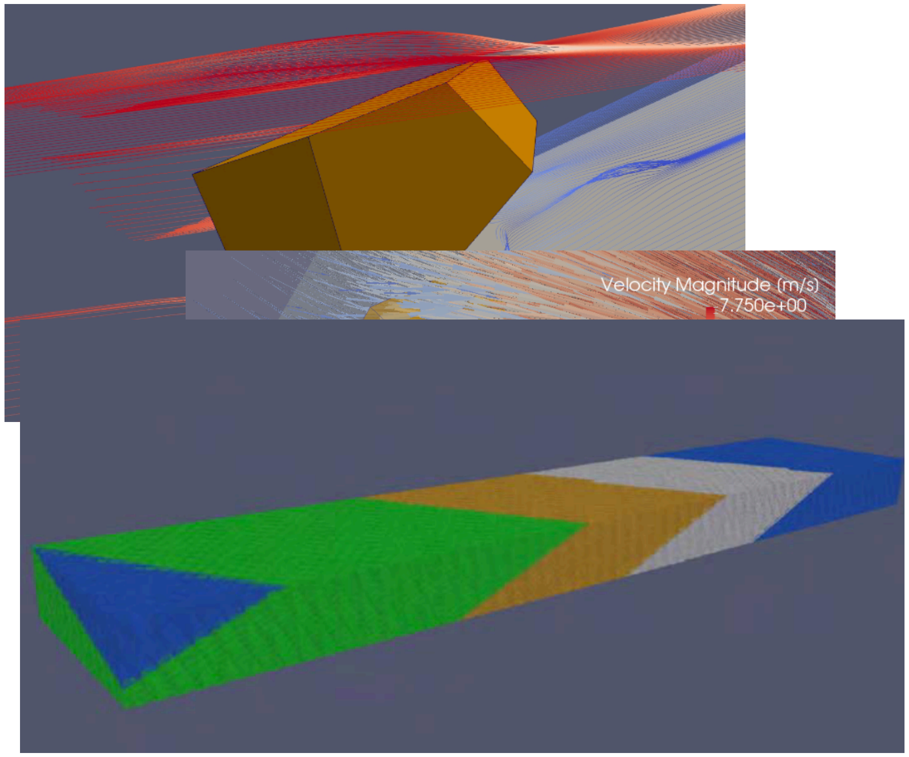
- Resilient Distributed Data Set (RDD): A collection of items partitioned across the members of a cluster
  - Can program against it just like an ordinary list, but operations are carried out in parallel on different machines
- Uses the same file system/infrastructure as Hadoop
  - Reuse existing systems, make it easier for users to transition
- Users can think about writing "ordinary" code to operate against RDDs rather than an explicit map/reduce structure
- Keep intermediate results in memory where possible
  - Issue with Hadoop: Write to disk after each map/reduce cycle, slow and inefficient when we want to compose many operations together (e.g., iterative method)

Word Count in Spark's Python API

```
file = sc.textFile("hdfs://…")
// Two kinds of operations:
// Actions: RDD → Value
// Transformations: RDD → RDD
// e.g. flatMap, Map, reduceByKey
file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a + b)
```

See http://spark.apache.org/examples.html

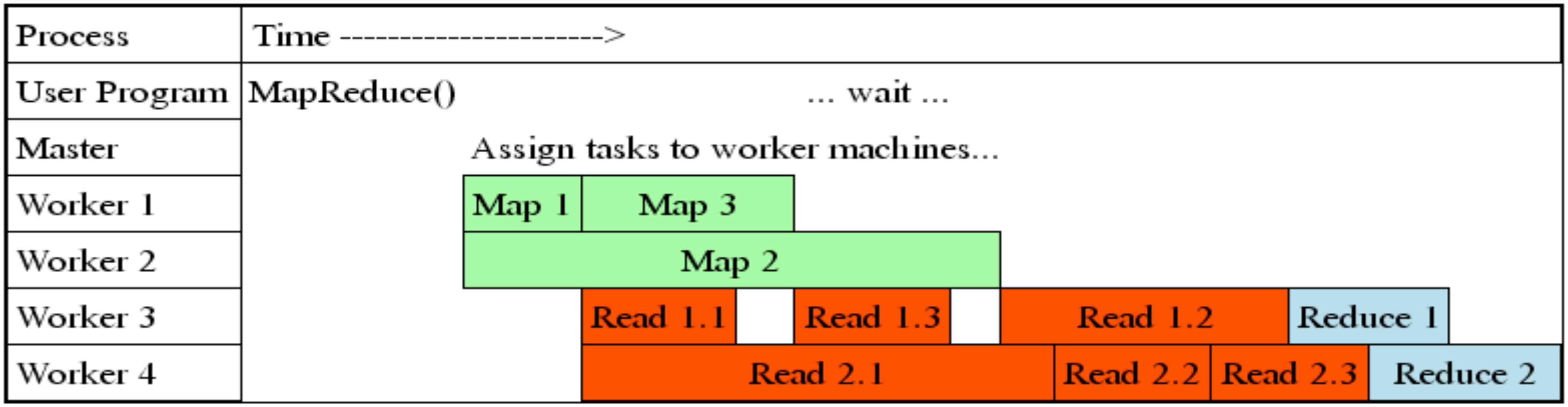
What about a *real* application of Spark?



- 50K Blocks in 15 min. -> 8 million Blocks in 15 min.
- How: Spatial partition of the problem

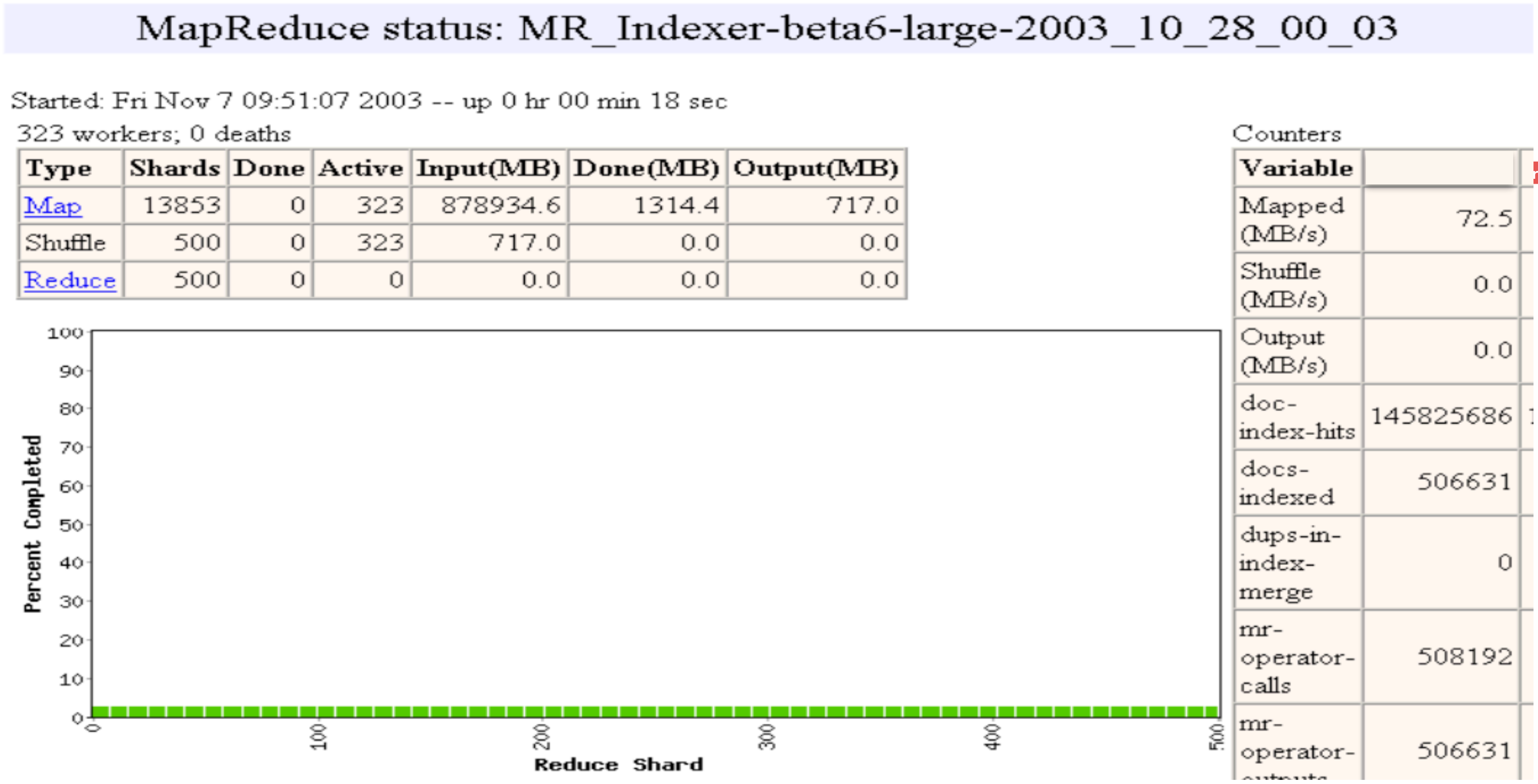
MapReduce Processing Time Line

- Master assigns map + reduce tasks to “worker” servers
- As soon as a map task finishes, worker server can be assigned a new map or reduce task
- Data shuffle begins as soon as a given Map finishes
- Reduce task begins as soon as all data shuffles finish
- To tolerate faults, reassign task if a worker server “dies”



A 2003 example...

- ~41 minutes total
  - ~29 minutes for Map tasks & Shuffle tasks
  - ~12 minutes for Reduce tasks
  - 1707 worker servers used
- Map (Green) tasks read 0.8 TB, write 0.5 TB
- Shuffle (Red) tasks read 0.5 TB, write 0.5 TB
- Reduce (Blue) tasks read 0.5 TB, write 0.5 TB



MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

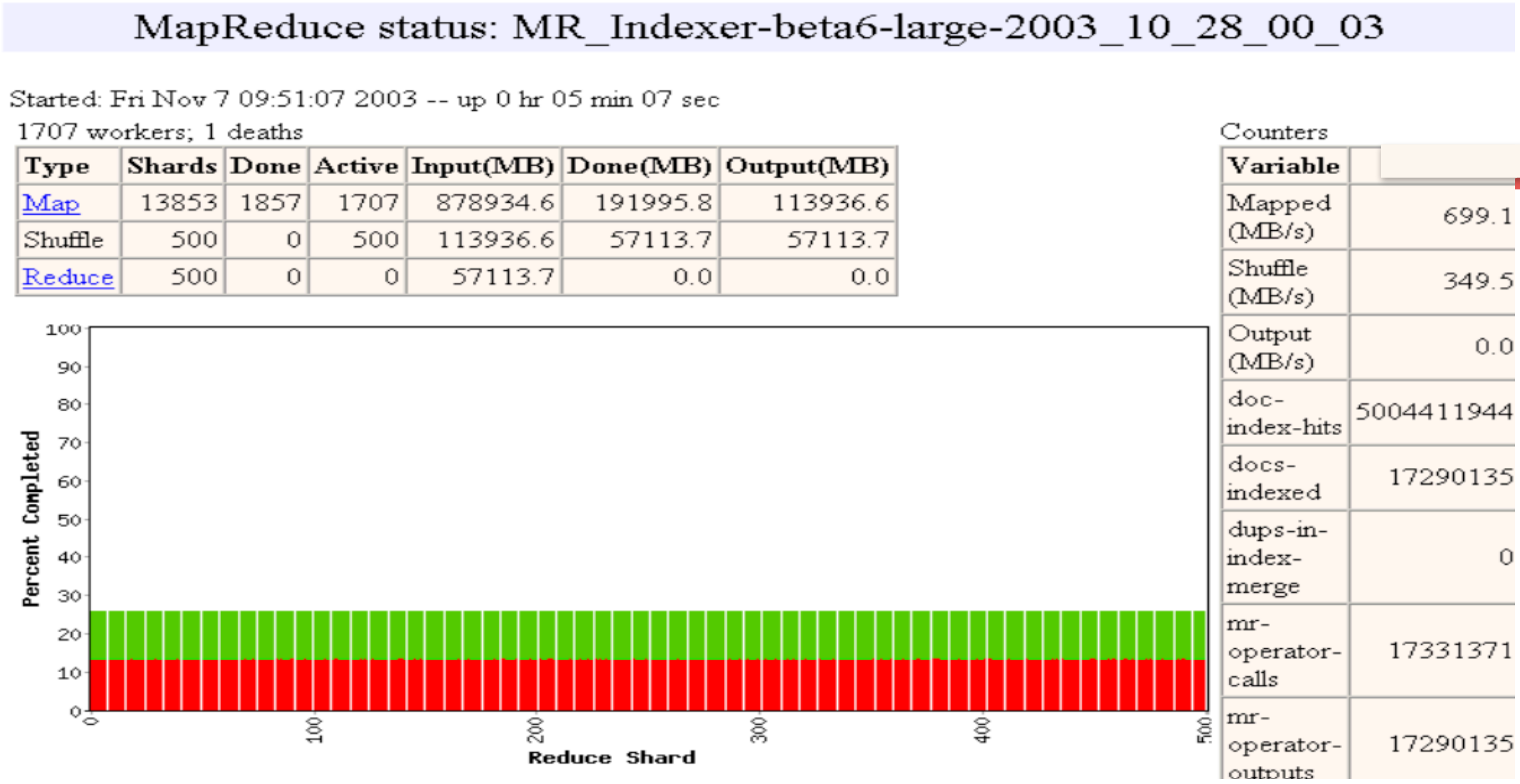
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

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	
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

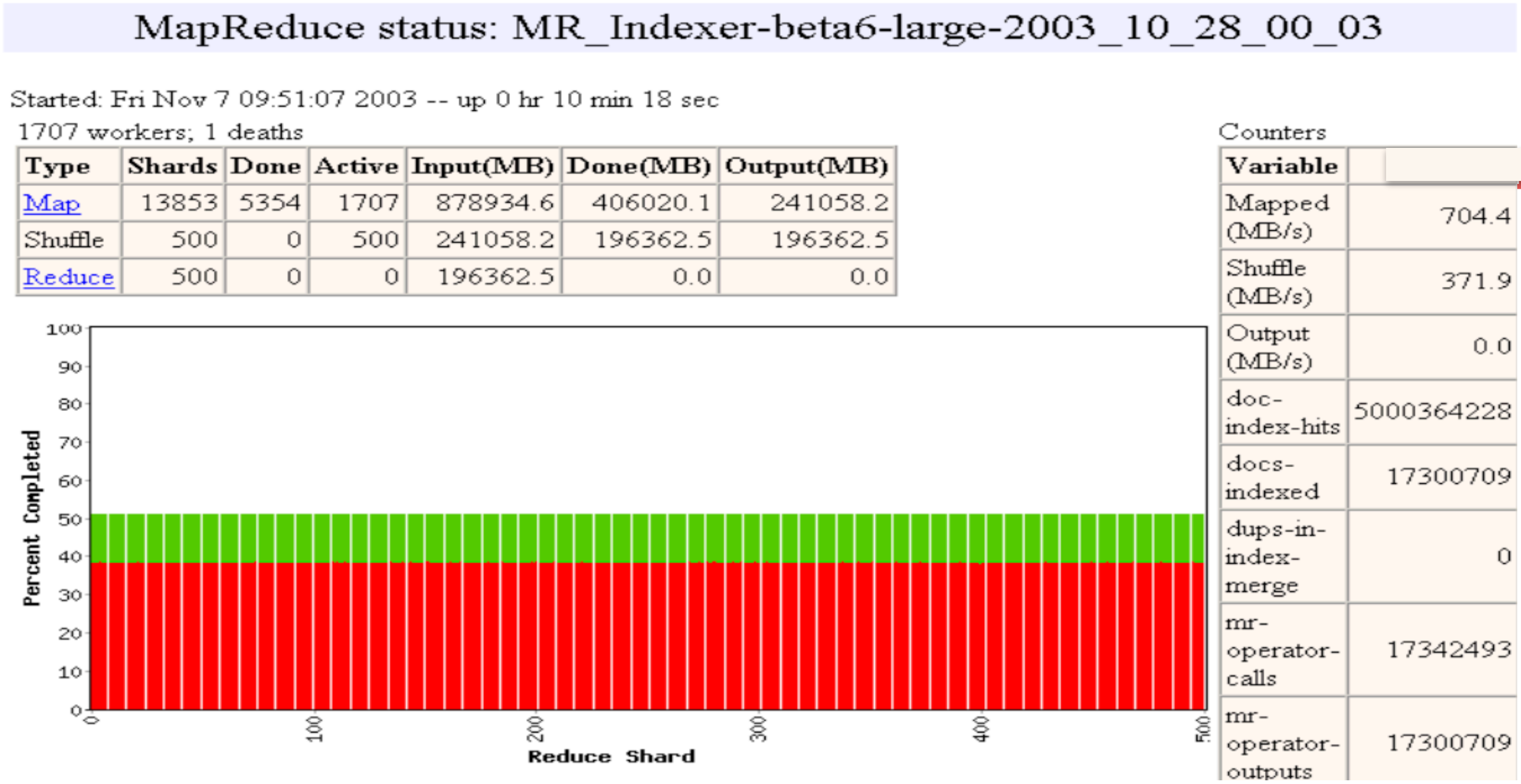
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

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	
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

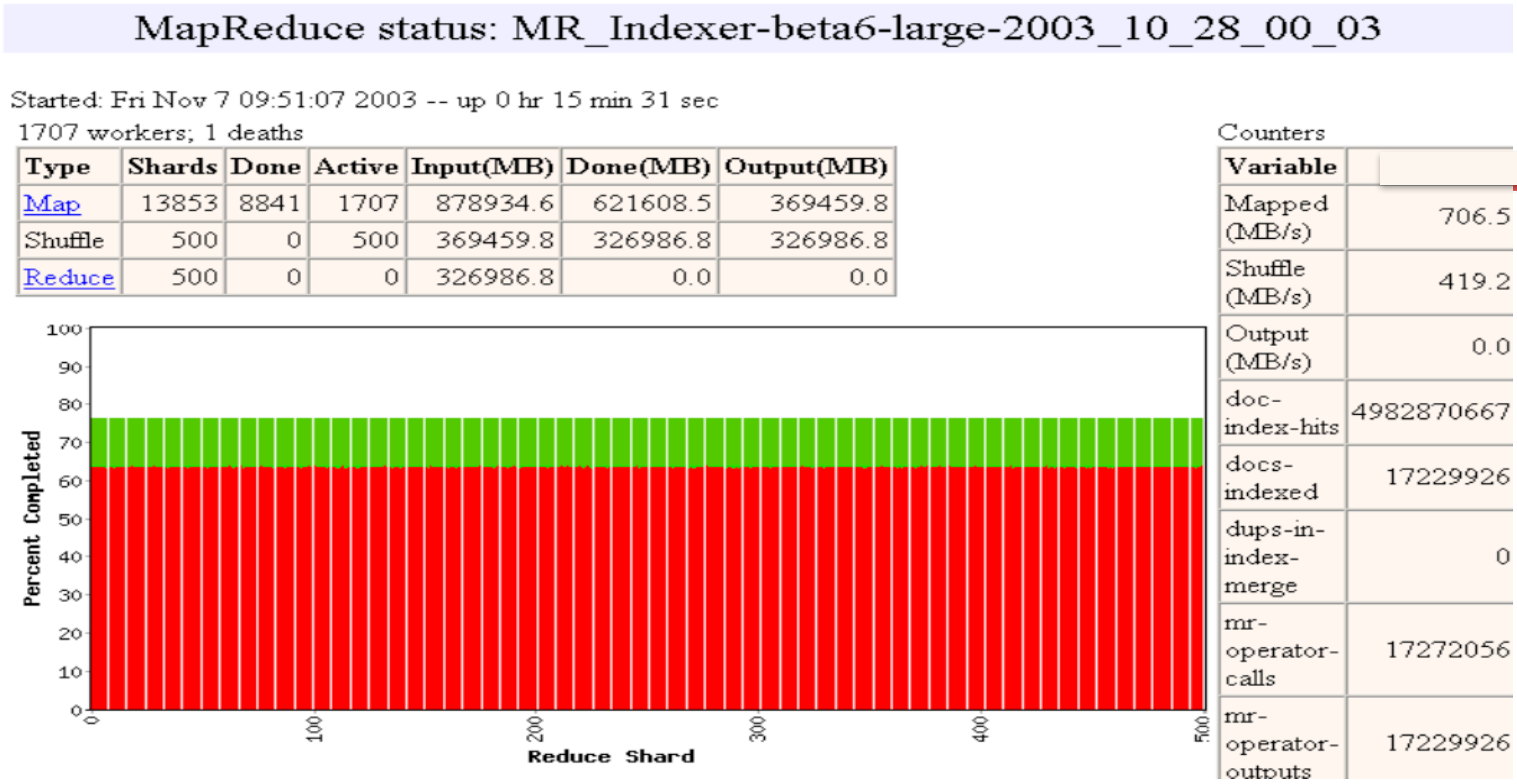
1707 workers; 1 deaths

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	
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

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

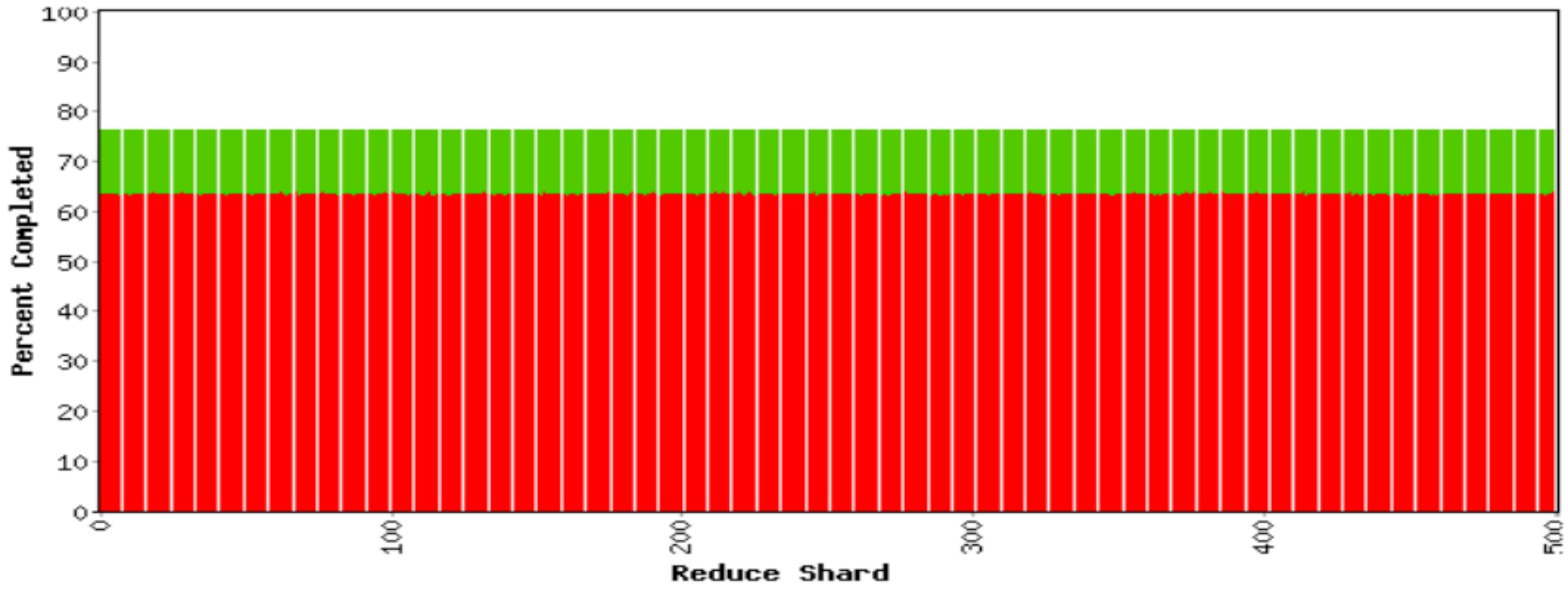


MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

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

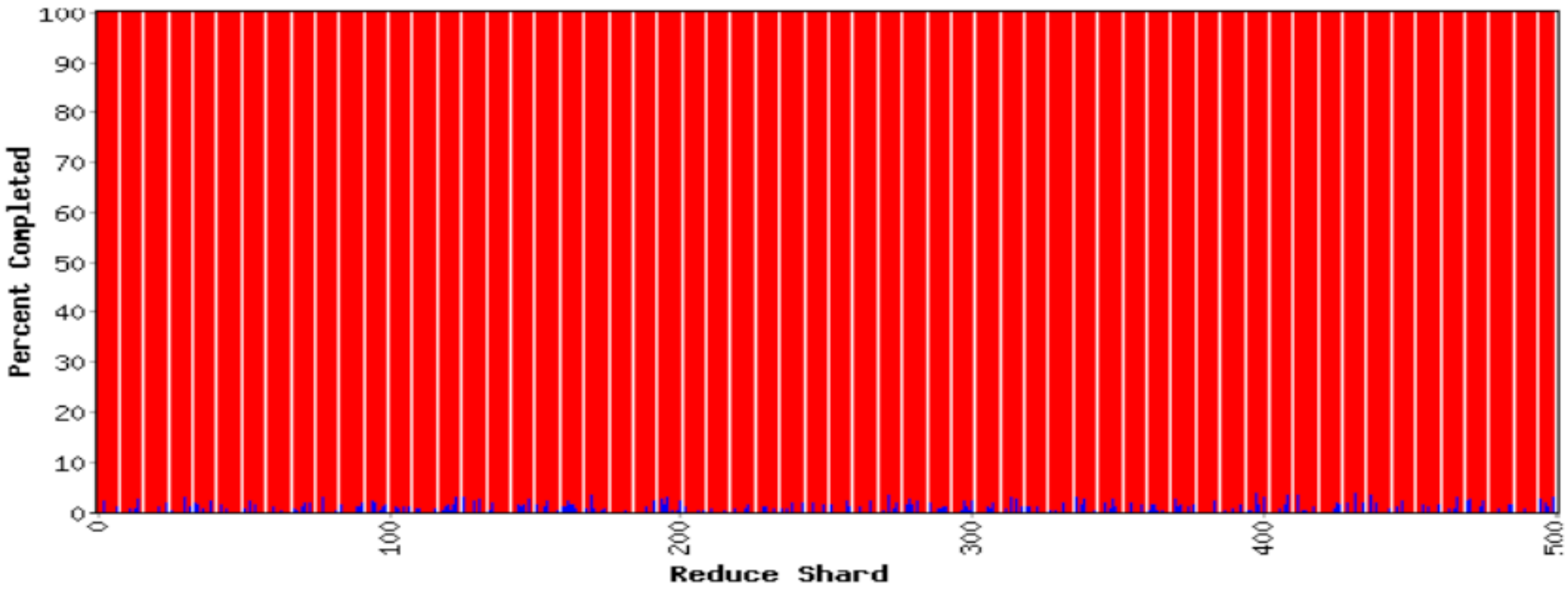
Variable	
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

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

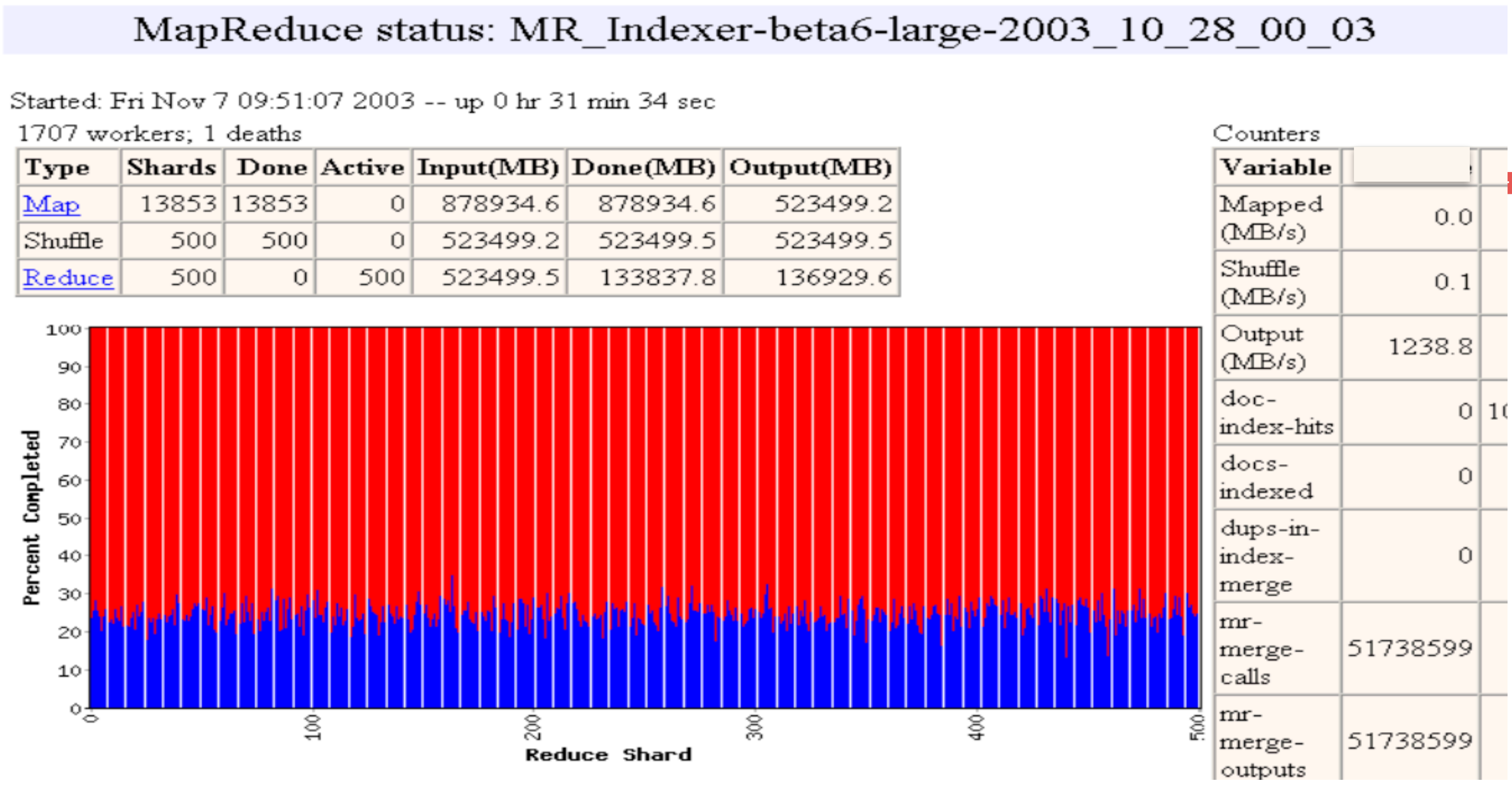
1707 workers; 1 deaths

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		
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		
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	

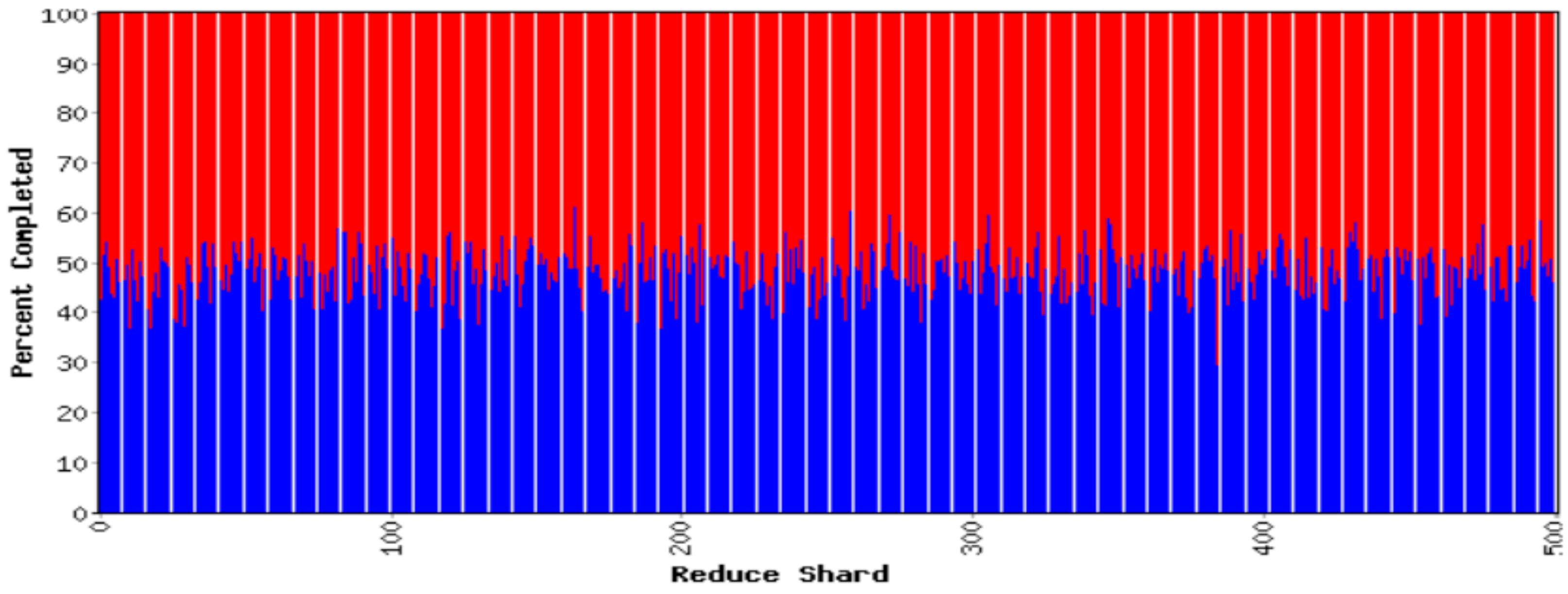
Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

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

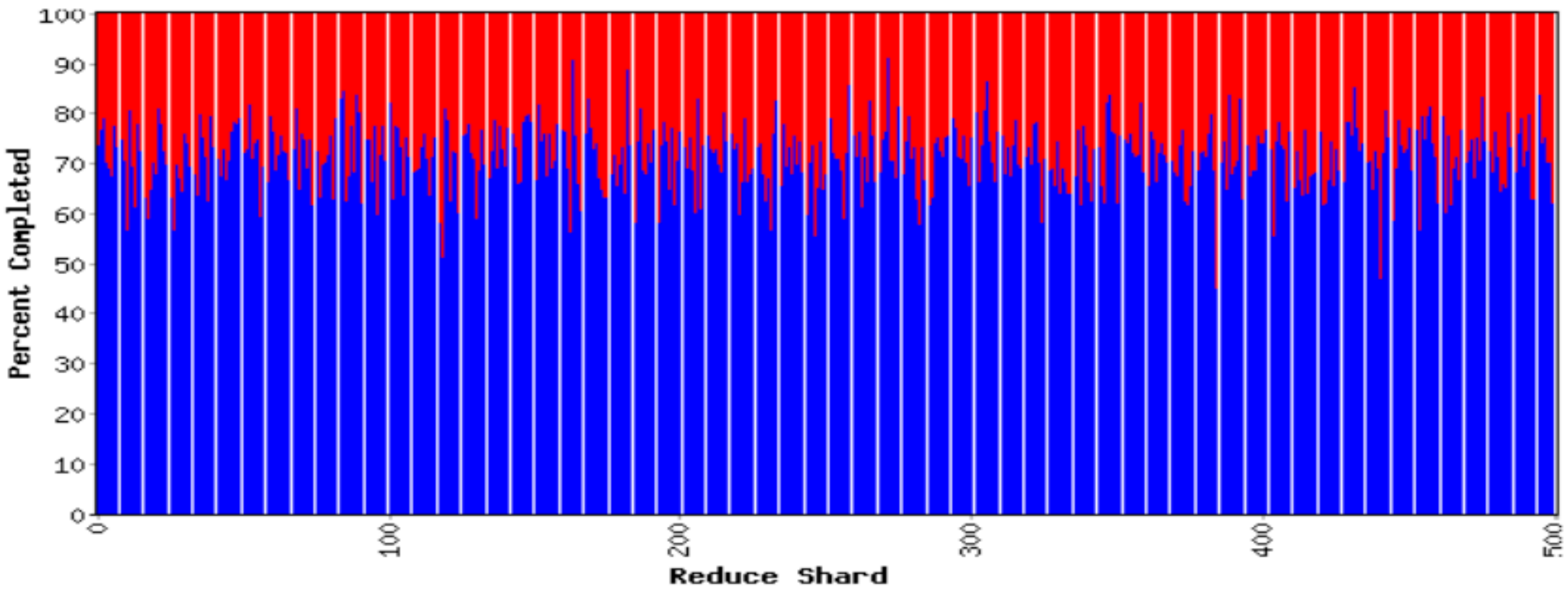
Variable		
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1225.1	
doc-index-hits	0	10
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51842100	
mr-merge-outputs	51842100	

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

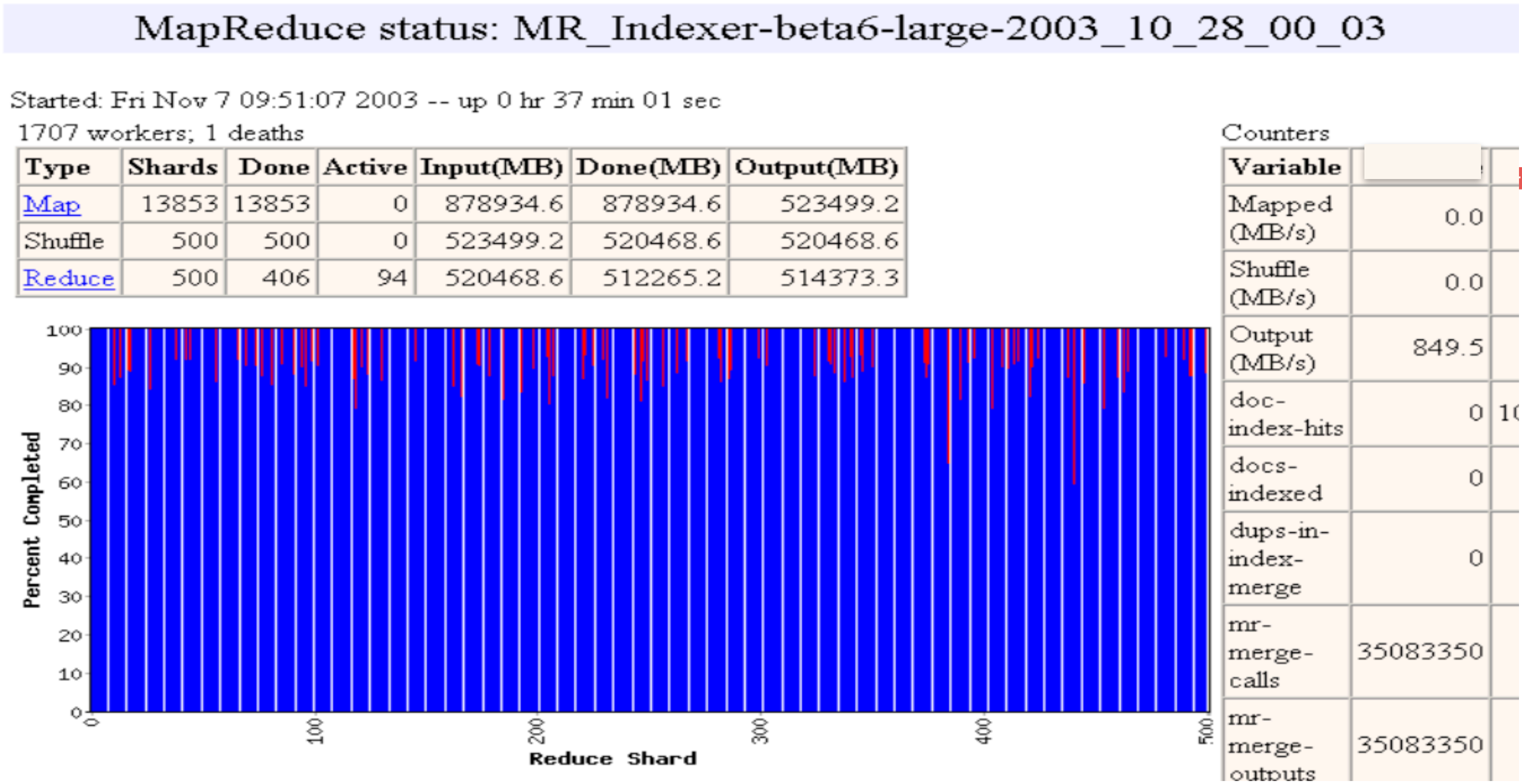
Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	390447.6	399457.2



Counters

Variable		
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1222.0	
doc-index-hits	0	10
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51640600	
mr-merge-outputs	51640600	

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

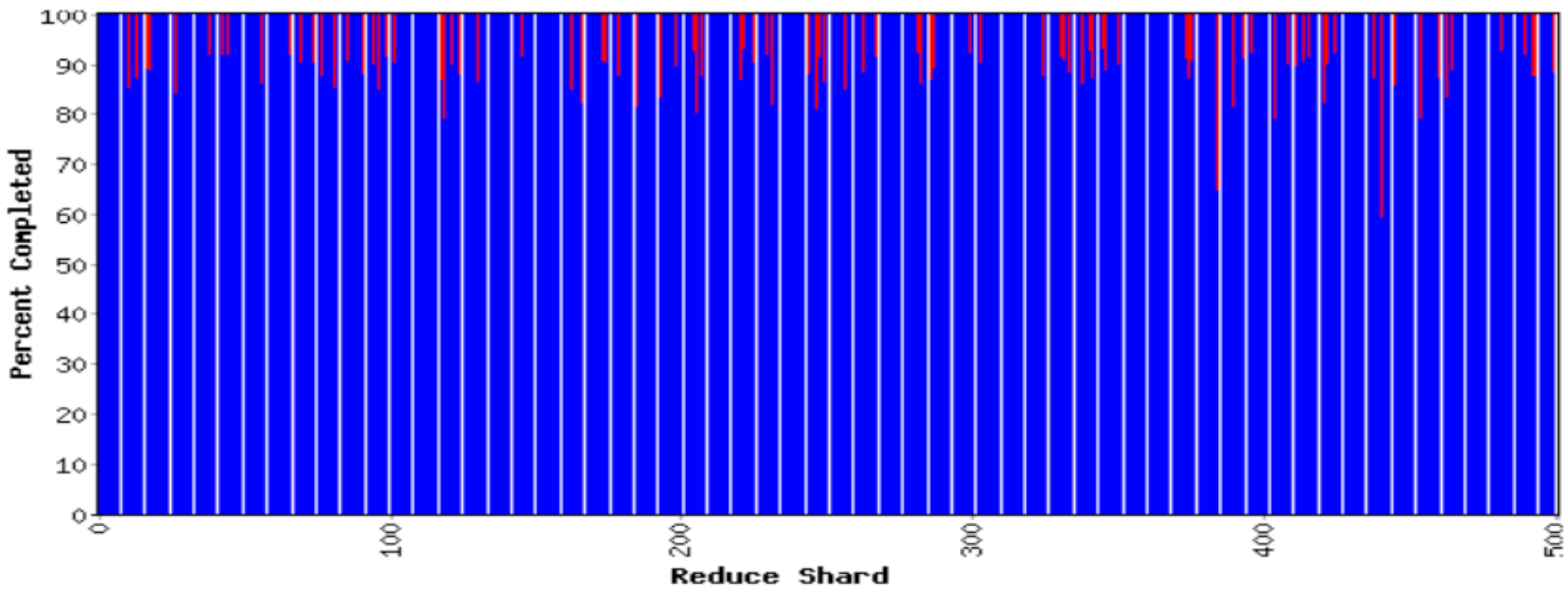


MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

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

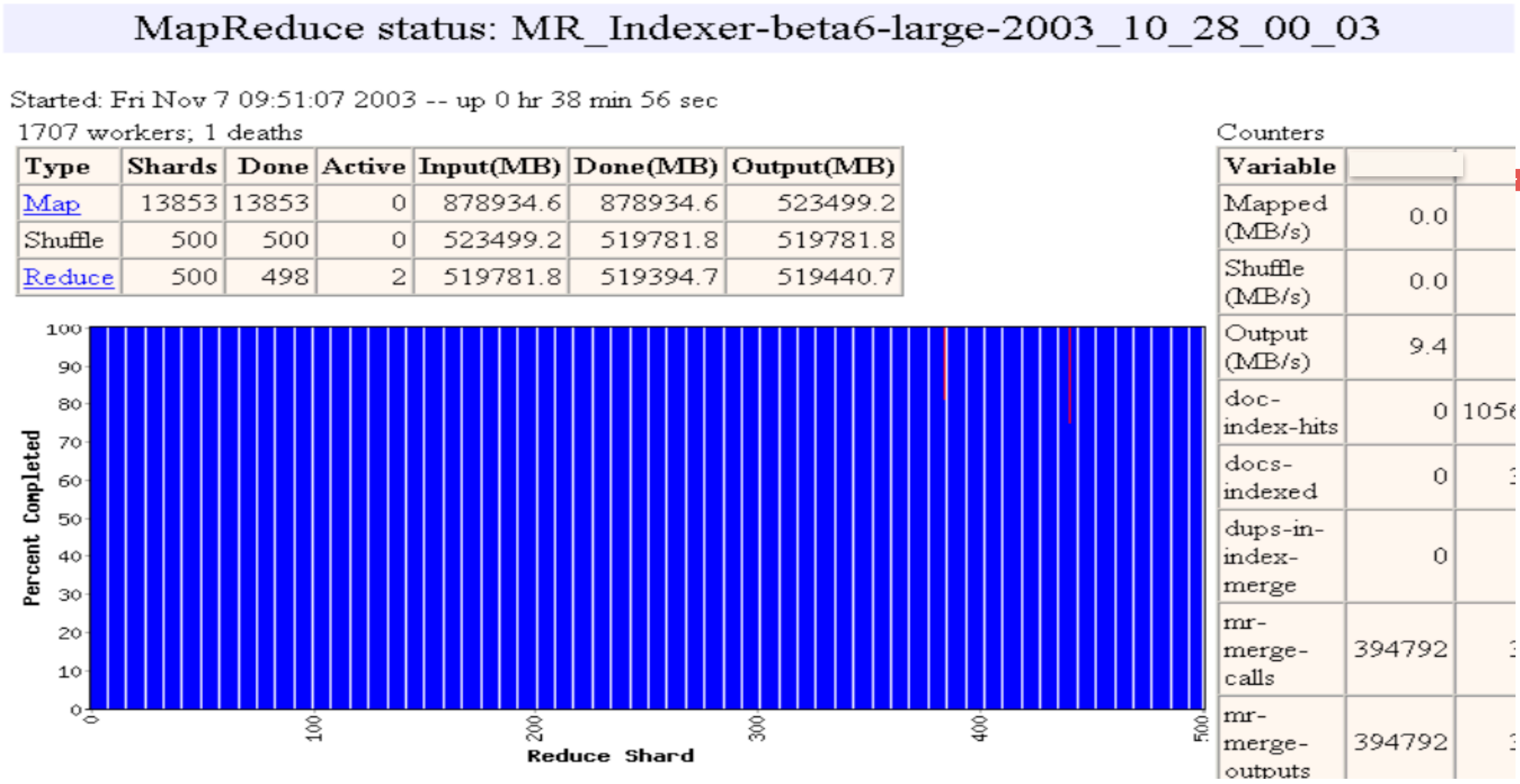
1707 workers; 1 deaths

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		
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	849.5	
doc-index-hits	0	10
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

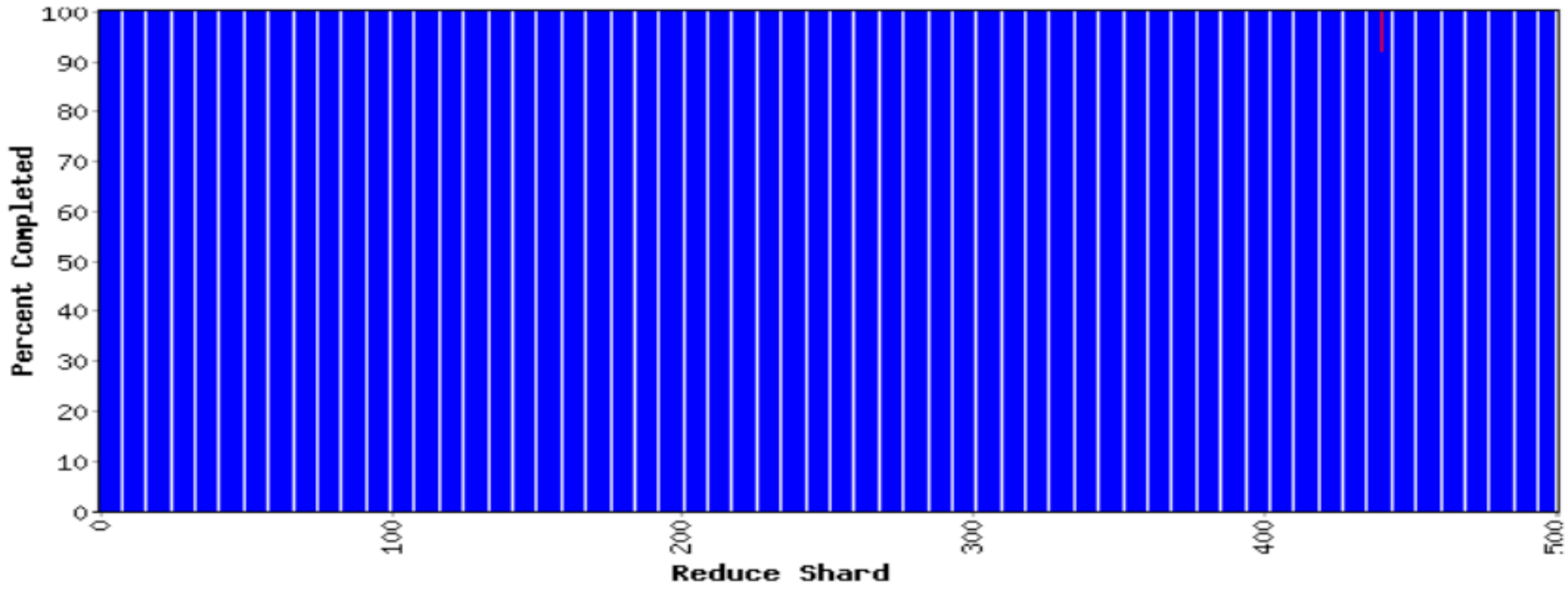
Variable		
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	1056
docs-indexed	0	3
dups-in-index-merge	0	
mr-merge-calls	394792	3
mr-merge-outputs	394792	3

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		
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	

Important Limitations

- This model only works for certain classes of problems
  - Need parallel compute over data and parallel reduction steps
  - **Critically**: Can divide a problem into many independent subproblems, minimal need for communication among workers when performing their computations
    - "Embarrassingly Parallel"
- Significant Overhead
  - Hadoop Distributed File System: 3x+ redundant storage
  - Lots of startup and control overhead:
    So unless you have many GiB/TiB of data, don't bother!
- For many cases, you are still better served sticking with a traditional database approach with big hardware behind it

Summary

- Warehouse-Scale Computers (WSCs)
  - New class of computers
  - Scalability, energy efficiency, high failure rate
- Cloud Computing
  - Benefits of WSC computing for third parties
  - “Elastic” pay as you go resource allocation
- Request-Level Parallelism
  - High request volume, each largely independent of other
  - Use replication for better request throughput, availability
- MapReduce Data Parallelism
  - Map: Divide large data set into pieces for independent parallel processing
  - Reduce: Combine and process intermediate results to obtain final result
  - Hadoop, Spark