Introduction to MapReduce

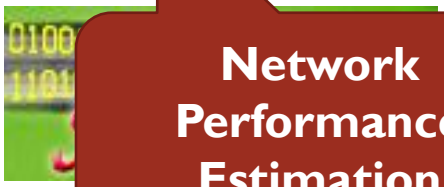
April 19, 2012

Jinoh Kim, Ph.D.
Computer Science Department
Lock Haven University of Pennsylvania

Areas

Datacenter Energy Management

Exa-scale Computing



Network Performance Estimation

Healthcare System Performance Optimization

Network Security Network Mgmt.

Outline

- Background
- MapReduce functions
- MapReduce system aspects
- Summary

Some Technical Terms

- Distributed computing
- Parallel computing (parallelism)
- Failure-prone
- Data replication

Distributed Computing?

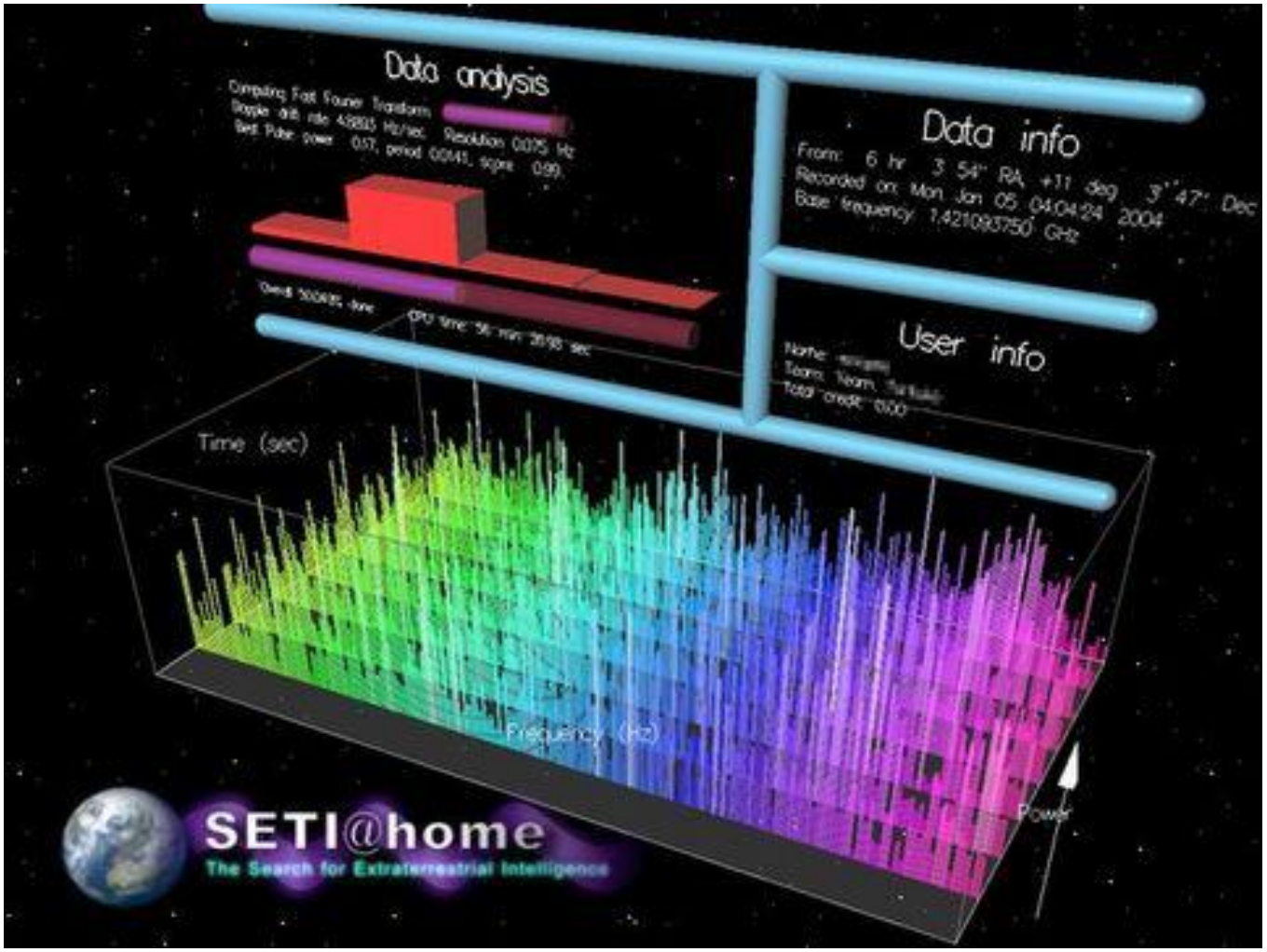
- Utilizes multiple computers over a network
- Examples?
  - SETI@home, web crawling (search engine), parallel rendering (computer graphics), and a lot!

- Distributed computing systems provide a platform to execute distributed computing jobs
- Examples?
  - Datacenter clusters: 100s~10000s computers
  - PlanetLab: 1,000+ computers
  - BOINC (for @home projects): 500,000 participants

SETI@home

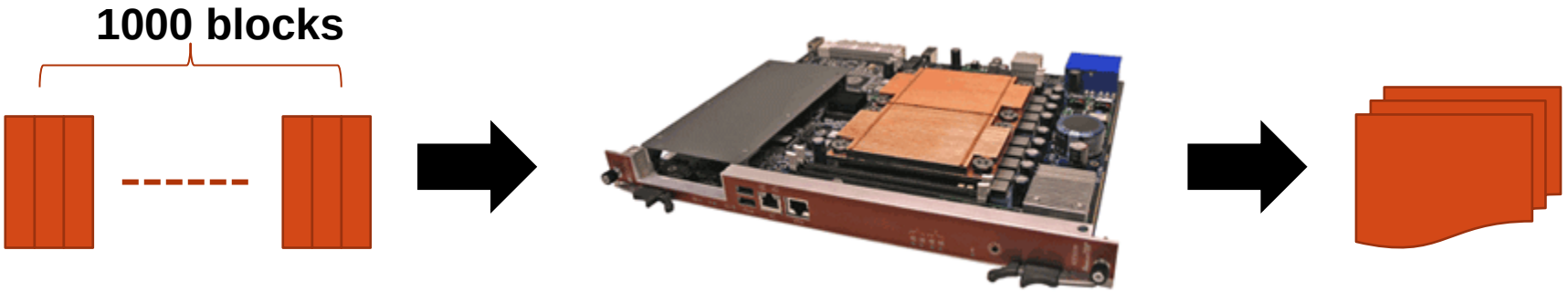
- Search for Extraterrestrial Intelligence
  - http://setiathome.berkeley.edu/index.php
- Volunteer-based distributed computing
  - Uses donated computers whenever they are not used (i.e., idle)
  - You can donate your computer!
  - Provides a fancy screen saver
- Computing power: 600 Tera FLOPS
  - Cray Jaguar supercomputer: 1800 Tera FLOPS

SETI@home Screen Saver



Parallel Computing?

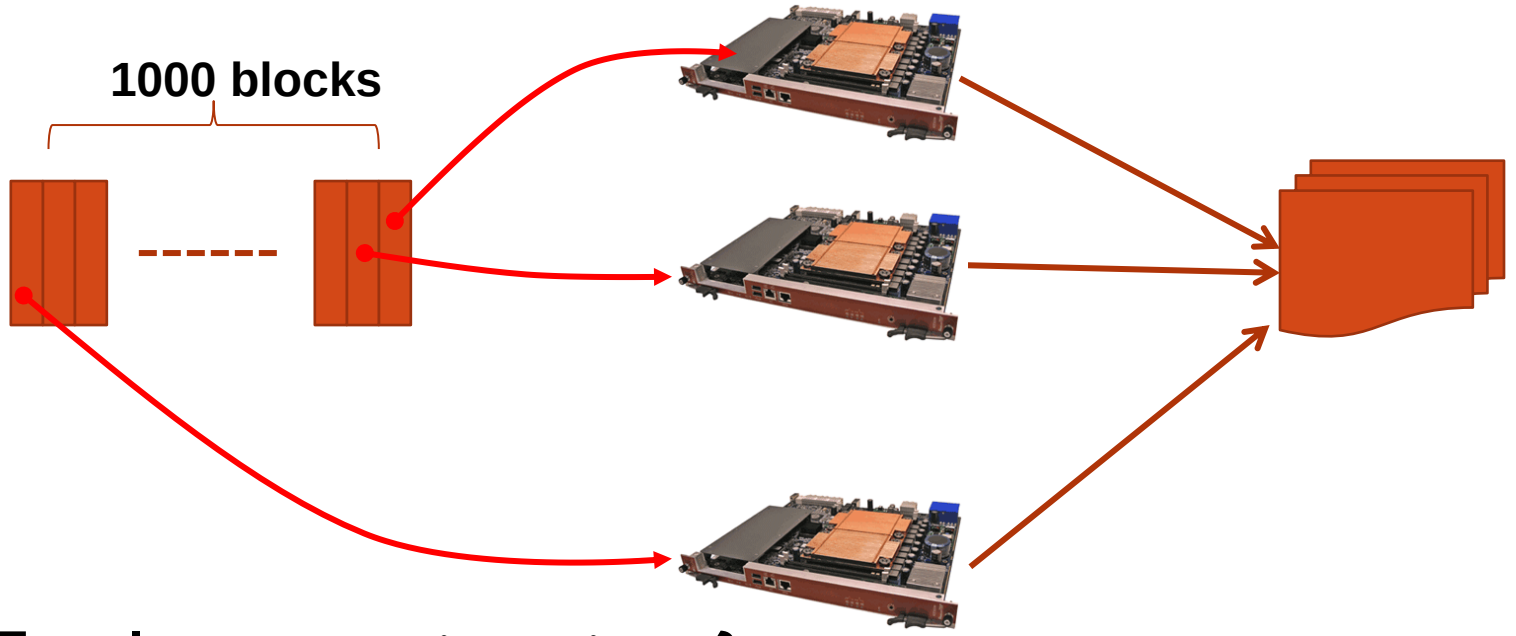
- Suppose a job with 1,000 data blocks
  - E.g., Rendering 1,000 individual movie frames
- Average processing time per block in a node = 5 seconds



- Total processing time?
  = 1000 * 5 seconds = 1.4 hours

Using Parallelism

- Use 1,000 nodes in parallel (1 block/node)
  - E.g., Rendering individual frames can be separate



- Total processing time?

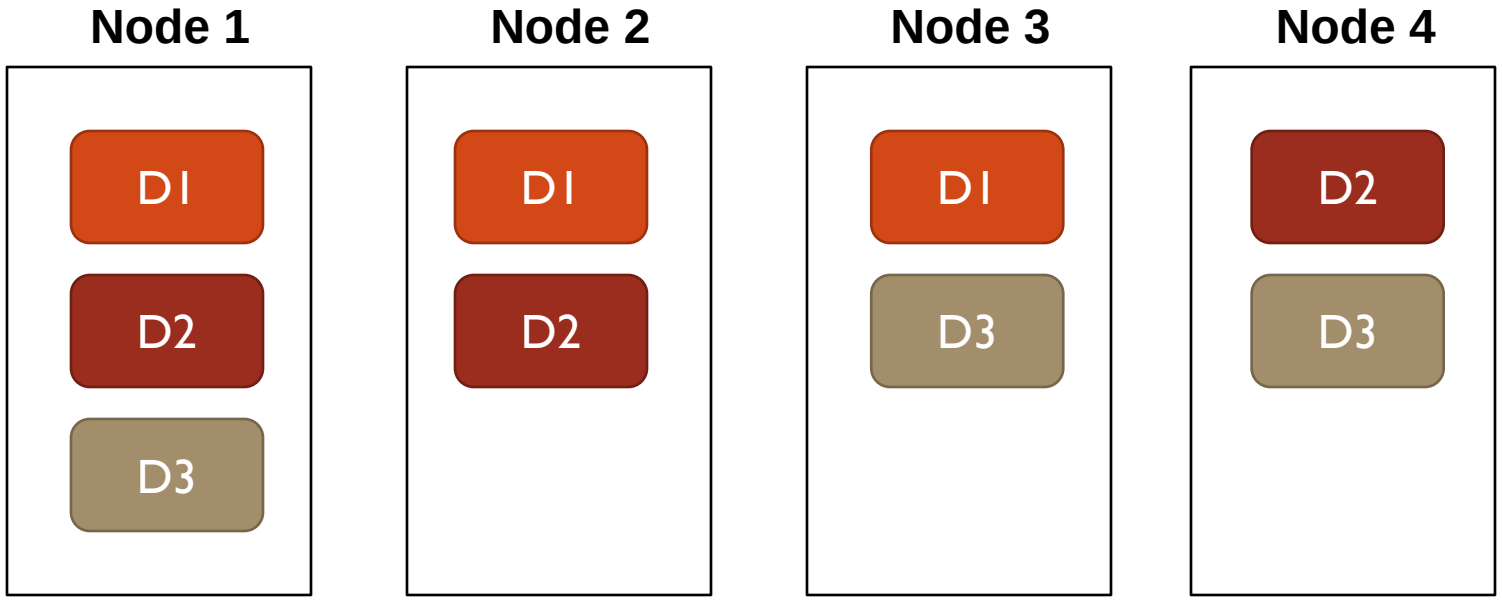
  = 5 seconds (ideally!)

Failure-prone?

- Many clusters use commodity PCs rather than high-end servers
- So, failures are a part of everyday life
- A PlanetLab report (HotMetrics 2008)
  - # failures per node in three months
    - Hardware failure: 2.1
    - Software failure: 41
  - Failure happens every 4 days for a node
  - Think about 1,000 nodes in a cluster

Data Replication?

- Place data to multiple machines redundantly



- Why replicating data?
  - Data availability under failures, load balancing
  - More storage required, consistency, management overheads

MapReduce Functions

Motivation

- Want to process lots of data (> 1TB)
  - 1TB = 16,384 data blocks as usual
- Want to parallelize across thousands of CPUs
- Want to make this easy

- MapReduce provides a sort of functions for ***large-scale data*** processing

Who uses MapReduce?

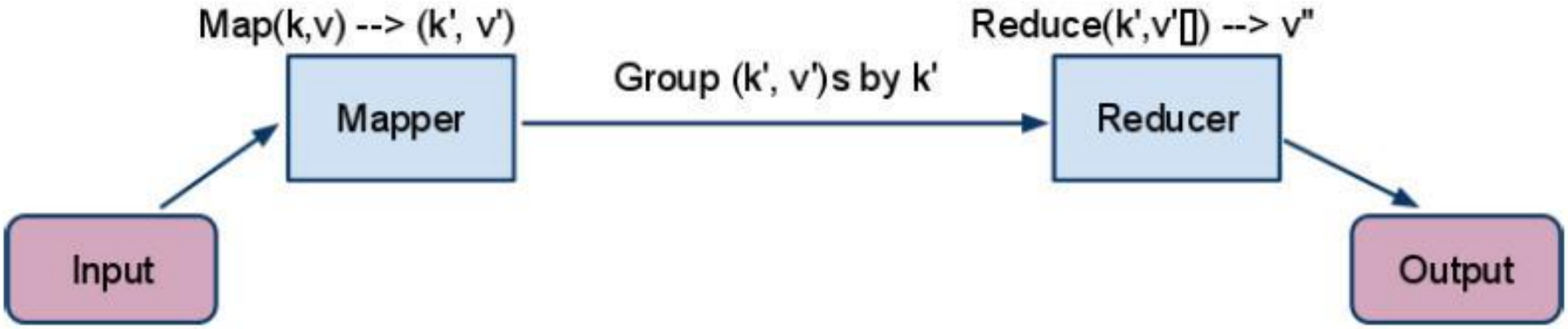
- NY Times
  - Convert 4TB of raw images TIFF data into PDFs
  - $240 for Amazon MapReduce cloud service
- Google (sort, grep, indexing, etc)

Stats for Month	Aug.'04	Mar.'06	Sep.'07
Number of jobs	29,000	171,000	2,217,000
Avg. completion time (secs)	634	874	395
Machine years used	217	2,002	11,081
Map input data (TB)	3,288	52,254	403,152
Map output data (TB)	758	6,743	34,774
reduce output data (TB)	193	2,970	14,018
Avg. machines per job	157	268	394
Unique implementations			
Mapper	395	1958	4083
Reducer	269	1208	2418

Source: MapReduce Tutorial, SIGMETRICS 2009

Programming Model

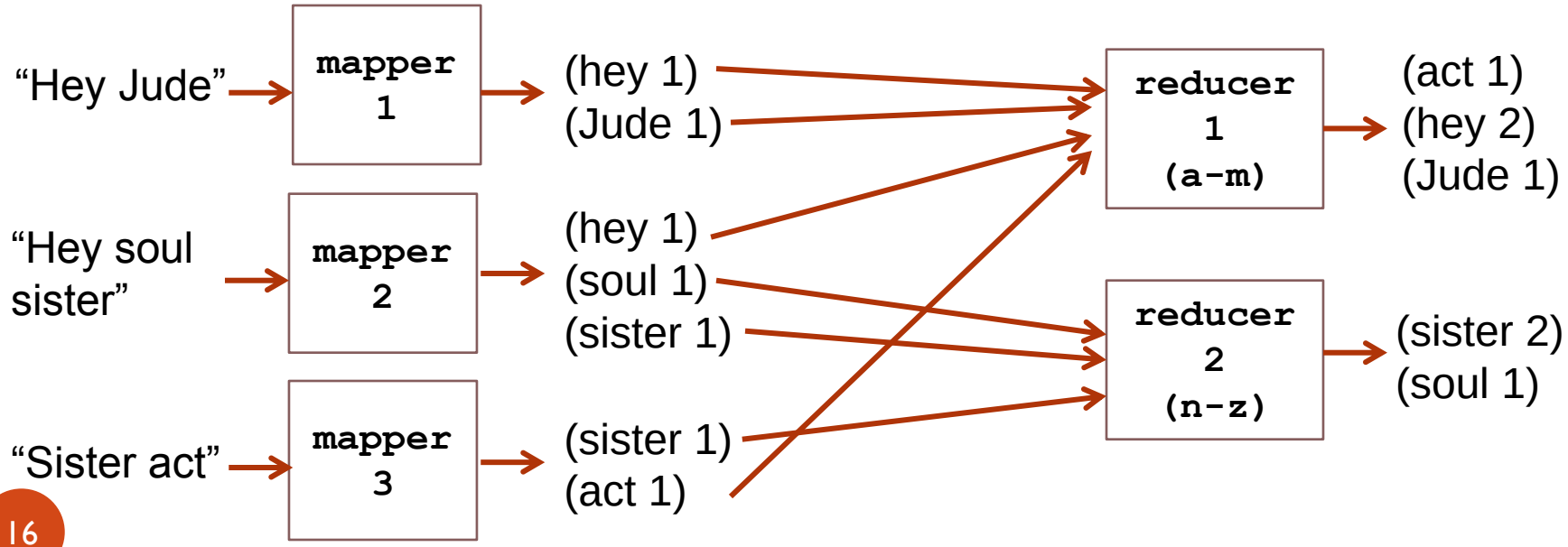
- Borrows from functional programming
- Users implement interface of two functions
  - `map(key,value) -> (key',value')`
  - `reduce(key',value' list) -> value''`



Source: MapReduce Tutorial, SIGMETRICS 2009

Example: Word Count

- map(key=document name, val=contents):
  - For each word *w* in contents, emit (*w*, "1")
- reduce(key=word, values=count)
  - Sum all "1"s in values list and emit result (word, sum)



Word Count Pseudo-code

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```


(From Jeffrey Dean and Sanjay Ghemawat, “MapReduce: Simplied Data Processing on Large Clusters,” OSDI’04)

MapReduce Systems

A (MapReduce) Cluster

- Thousands of commodity PCs
- High bandwidth network links and switches
- Google MapReduce, Hadoop Map-Reduce

A *Rack*
=
nodes +
switch +
…

(from http://www.scbi.uma.es/portal/)

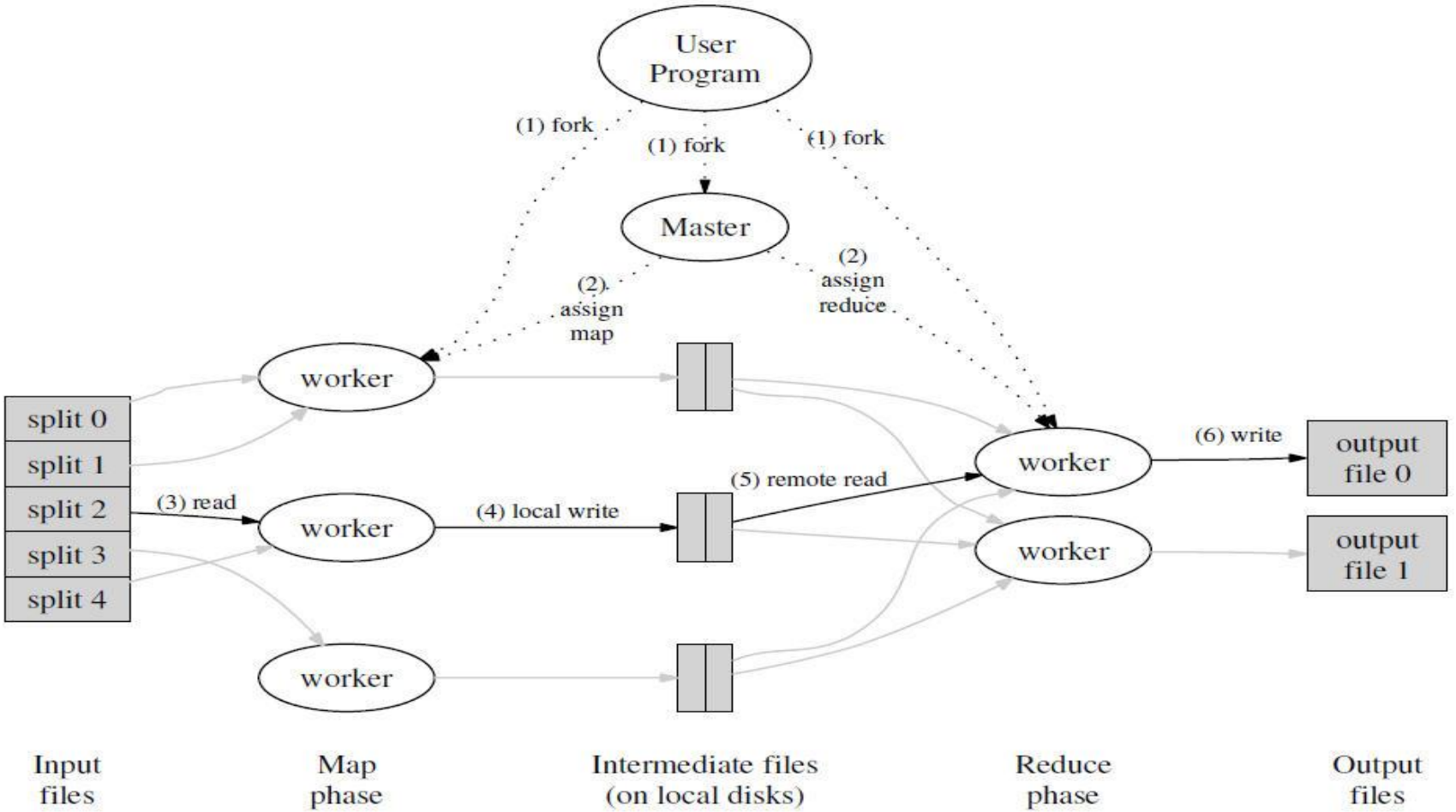
(Picture from picmg.org)

A *Node*
= CPU, memory,
disk, network card,
…

A MapReduce System

- Automatic parallelization and distribution
- Fault-tolerant
- Provides status and monitoring tools
- Clean abstraction for programmers

Execution Model



(From Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters," OSDI'04)

MapReduce Data Layout

- MapReduce places three copies for each data
- Often uses ***rack-awareness*** for replication
  1) No two replicas of a data block are stored on any single node
  2) A data block can be found on at least two racks
- Why?
  1) Don't need that a node contains two copies
  2) A whole rack can be failed (e.g., power failure)

Locality

- Master program allocates tasks based on ***location*** of data
- Precedence rule for map task allocation
  1) On same machine as physical file data
  2) On a machine in the same rack
  3) Anywhere else
- Why?
  - Network is a bottleneck of performance
  - Networking cost:
    - same machine (0) < a machine in the same rack < other

Fault Tolerance

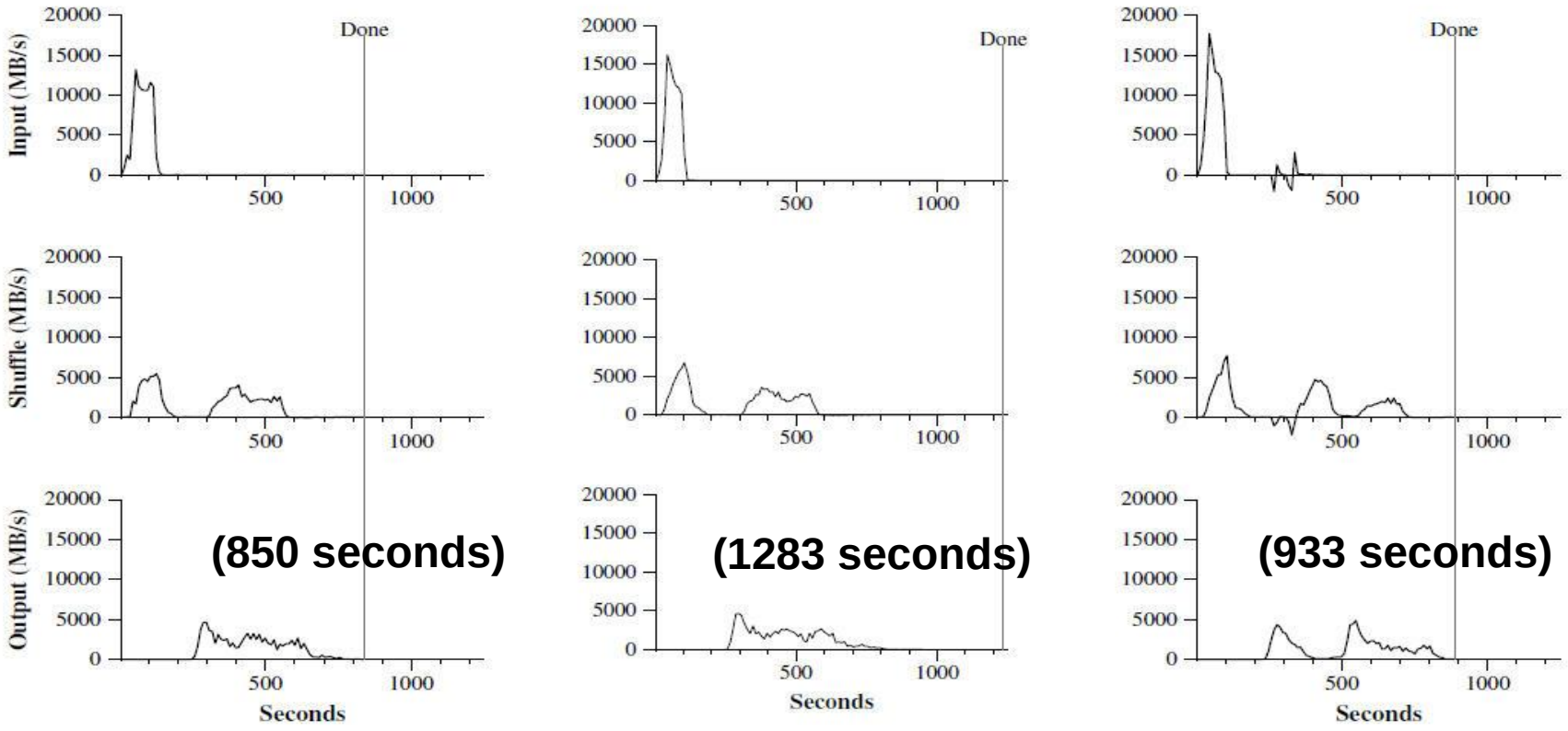
- Master detects worker failures
  - Re-executes completed and in-progress map() tasks
  - Why re-executing even completed map tasks?
    - Intermediate results are stored in map() tasks
  - Re-executes in-progress reduce() tasks

Optimization for Group Performance

- No reduce can start until map is complete:
  - A single slow disk controller can rate-limit the whole process
- Group performance is more important than individual performance
- Master redundantly executes "slow-moving" map tasks (i.e., execute backup tasks)
  - uses results of first copy to finish

Performance Results

- 1764 workers, 15000 mappers, 4000 reducers for sort



(From Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters," OSDI'04)

Summary

- MapReduce provides clean abstraction for programmers for large-scale data processing
- Also provides an implementation of system for:
  - Automatic parallelization and distribution
  - Fault-tolerant
  - Status and monitoring tools
- Fun to use: focus on problem, let library deal with messy details

Take-away

- Try Hadoop!
  - Download and install the software
  - http://hadoop.apache.org/
  - Or do google for "Hadoop"

References

1. Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplifed Data Processing on Large Clusters," OSDI'04
2. Jerry Zhao and Jelena Pjesivac-Grbovic, MapReduce: The Programming Model and Practice, Tutorial in SIGMETRICS 2009
3. Hadoop, http://hadoop.apache.org
4. Eric Tzeng, ttp://inst.cs.berkeley.edu/~cs61a/su11/lectures/Su11_CS61a_Lec09_print.pdf
5. SETI@home, http://setiathome.berkeley.edu/index.php
6. PlanetLab, http://www.planet-lab.org