MapReduce

CompSci 516

Junghoon Kang

Announcement

- HW3 has been posted
  (due on March 23rd after the spring break).
- Key components in HW3 are:
  Apache Spark, Scala, and Unix variants.

BIG
DATA
INFORMATION
STORAGE
LARGE
PETABYTES
SEARCH
ANALYSIS
RESEARCH
INTERNET
SETS
MILLION
SOFTWARE
MANAGEMENT
TECHNOLOGIES
GOVERNMENT
VOLUME
PROCESS
SAN
NAS
DEVELOPMENT
TECHNOLOGY
SCIENCE
FUTURE
NEW
BILLION
PARALLEL
INITIATIVE
ECONOMIC
TERABYTES
EXABYTES
LEARNING
WORLD
BASED
EVERY
SIZE
DATABASES
PROCESSING
ANALYTICS
SIMULATIONS
VISUALIZATION
FRAMEWORK
BUSINESS
BETWEEN
EFFECTIVE
STATISTICS
DEFINITION
USING
MASSIVE
FACTOR
MANAGE
ONLINE
DAY
WORLDWIDE
SYSTEMS
MAKING
INTEGRATION
SOCIAL
YEARS
RESULTS
DRIVE
REPLICATION
HANDLE
SOURCES
CHALLENGES
LIMITS
COMPLEX
HAND
CAPACITY
ONE
ANNOUNCED
PAST
INCLUDE
NETWORKS
HYPOTHESIS
END
MOST
NEARLY
MAPREDUCE
CRITIQUES
STATE
CRITIQUE
PEOPLE
TRAFFIC
USE
CURRENTLY
APPROACH
MANY
OUT
PRIVACY
INTERNATIONAL
IMPLEMENTATION
BECOMES
SECOND
NATIONAL
INTELLIGENCE
FLOW
RELATED
MARKET
TIME
VARIETY
SENSOR
WOULD
PROJECT
ANNUAL
NOW
DISTRIBUTED
RELEVANT
USED
PER
APPROACHES
PROCESSES
PARADIGM
RECORDS
COST
FUNDING
INSIGHT
TOOLS
CENTER
ARCHITECTURE
DETERMINE
WELL
SET
DEPARTMENT
QUERIES
HIGH
USERS
HIGHER
ALGORITHMS
COMPANIES
AMOUNT
RATE
REAL
STRUCTURE
WORK
APPLICATIONS
INFRASTRUCTURE
TERMS
YEAR
PRIVATE
CAPABILITIES
SUCCESSFUL
NEEDED
USES
LESS
HUGE
VARIOUS
TIMES
COMPARED
SHARED

Big Data

$$\begin{tikzcd}[column sep=small]
 & \text{Big Data} \arrow[dl] \arrow[dr] & \\
\text{it cannot be stored in one machine} \arrow[d] & & \text{it cannot be processed in one machine} \arrow[d] \\
\text{store the data sets on multiple machines} \arrow[d] & & \text{parallelize computation on multiple machines} \arrow[d] \\
\text{Google File System} & & \boxed{\text{MapReduce}}
\end{tikzcd}$$

Today!

But before we go into
MapReduce,
let me briefly talk about
Google File System

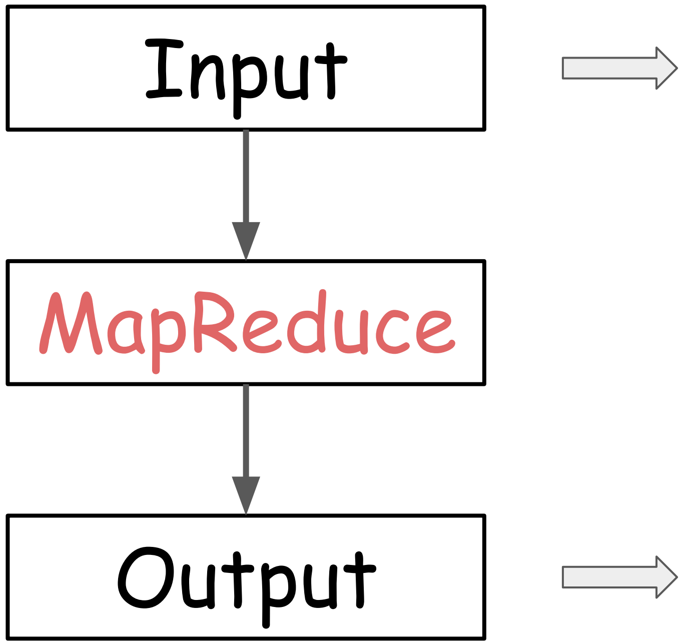
Google File System

- It is a distributed file system.
- When I say GFS in this slides, it means Google File System.
- For more information, please read GFS paper from SOSP 2003.

MapReduce

- MapReduce refers to Google MapReduce.
- Google published MapReduce paper in OSDI 2004, a year after the GFS paper.

Where does Google use MapReduce?



- crawled documents
- web request logs

- inverted indices
- graph structure of web documents
- summaries of the number of pages crawled per host
- the set of most frequent queries in a day

What is MapReduce?

It is a programming model

that processes large data by:

apply a function to each logical record in the input (map)

categorize and combine the intermediate results
into summary values (reduce)

Google's MapReduce is inspired
by
map and reduce functions
in functional programming
languages.

For example, in Scala functional programming language,

```
scala> val lst = List(1,2,3,4,5)

scala> lst.map( x => x + 1 )

res0: List[Int] = List(2,3,4,5,6)
```

For example, in Scala functional programming language,

```
scala> val lst = List(1,2,3,4,5)

scala> lst.reduce( (a, b) => a + b )

res0: Int = 15
```

Ok, it makes sense
in one machine.

Then, how does Google
extend the functional idea
to multiple machines
in order to
process large data?

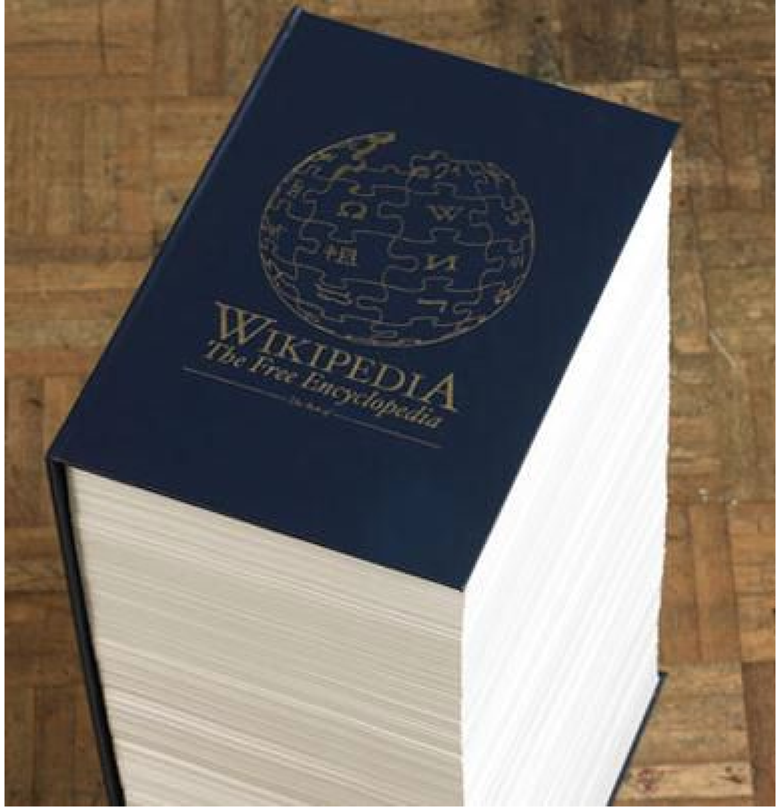
Understanding MapReduce

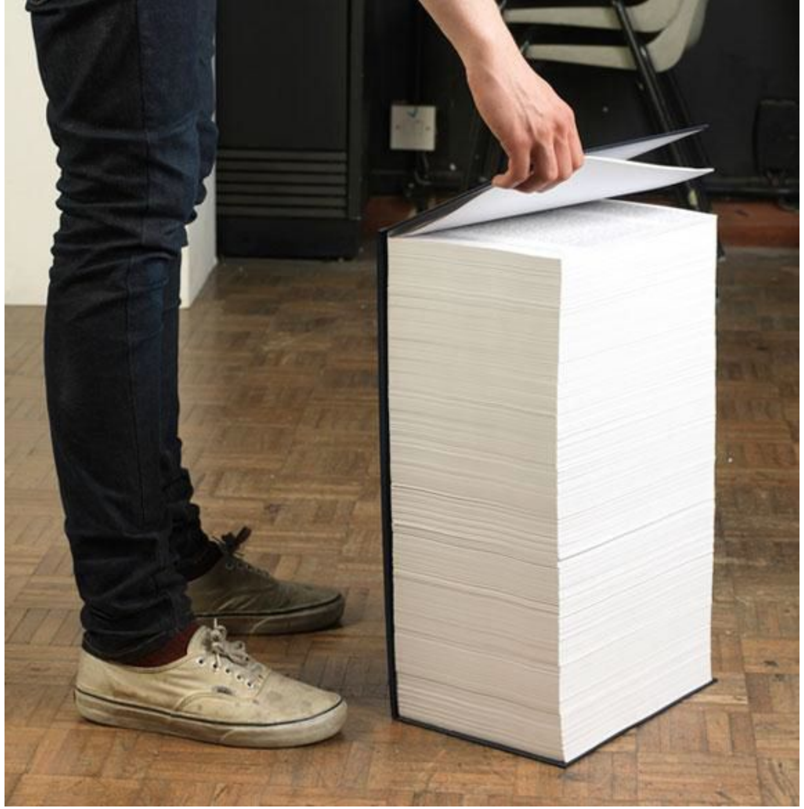
(by example)

I am a class president

An English teacher asks you:

“Could you count the number of occurrences of each word in this book?”

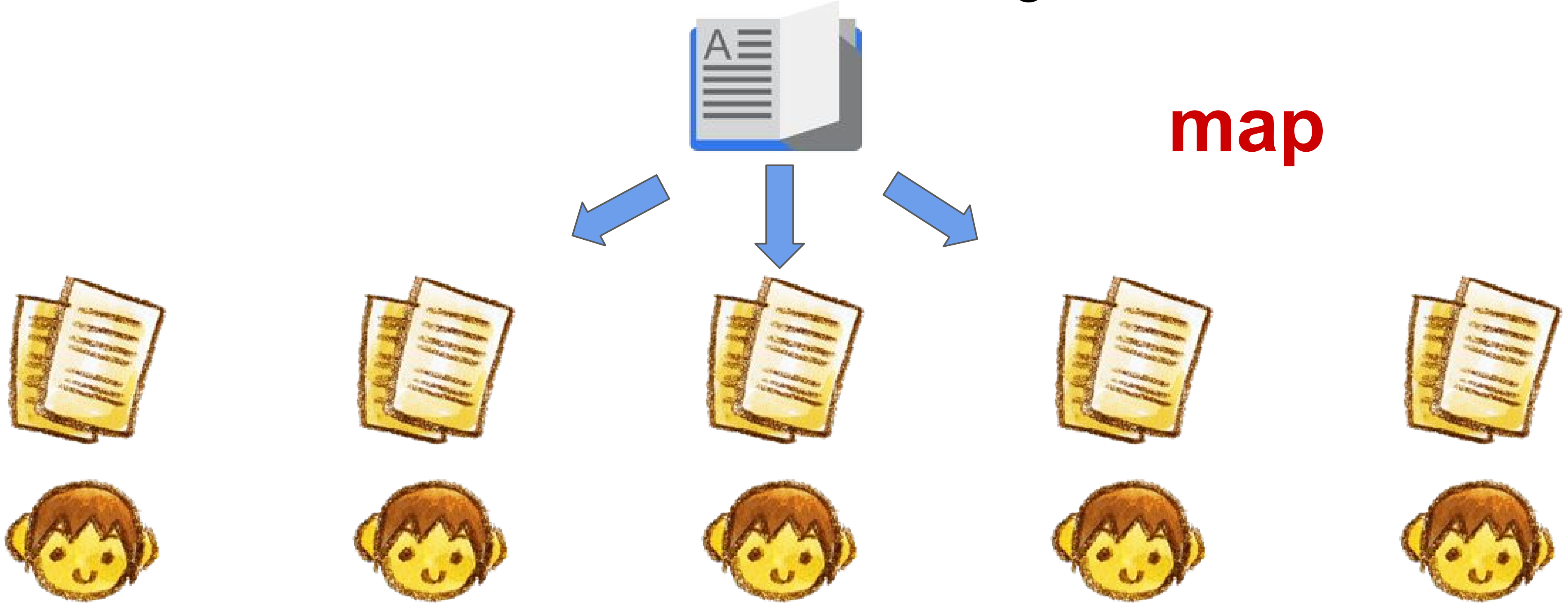




Um... Ok...

Let’s divide the workload among classmates.

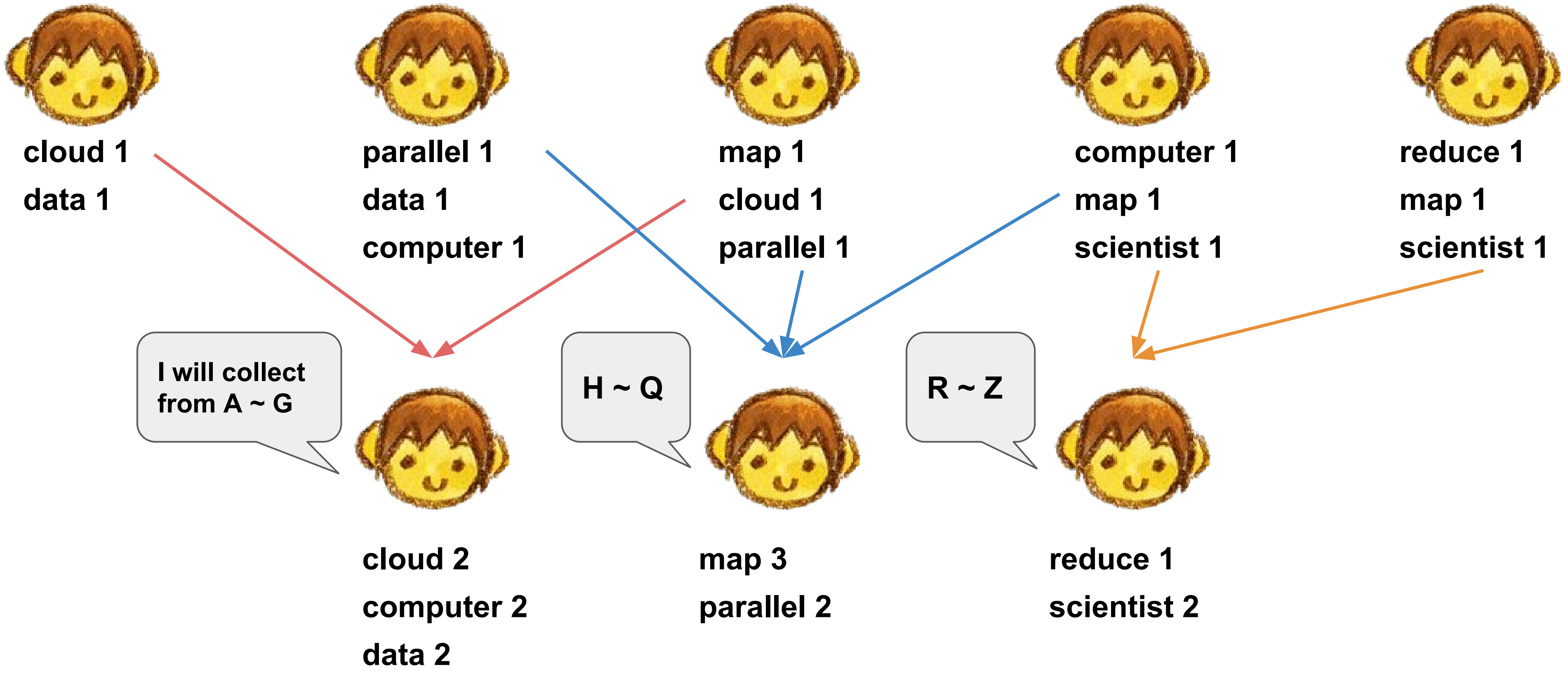
map



cloud 1	**parallel 1**	**map 1**	**computer 1**	**reduce 1**
data 1	**data 1**	**cloud 1**	**map 1**	**map 1**
	computer 1	**parallel 1**	**scientist 1**	**scientist 1**

And let few combine the intermediate results.

reduce



Why did MapReduce become so popular?

Is it because Google uses it?

Distributed Computation Before MapReduce

Things to consider:

- how to divide the workload among multiple machines?
- how to distribute data and program to other machines?
- how to schedule tasks?
- what happens if a task fails while running?
- … and … and ...

Distributed Computation After MapReduce

Things to consider:

- how to write Map function?
- how to write Reduce function?

MapReduce lowered the knowledge barrier in distributed computation.

Developers needed
before MapReduce

Developers needed
after MapReduce

Given the brief intro to MapReduce,

let's begin our journey to real implementation details in MapReduce !

Key Players in MapReduce

One Master

- coordinates many workers.
- assigns a task* to each worker.

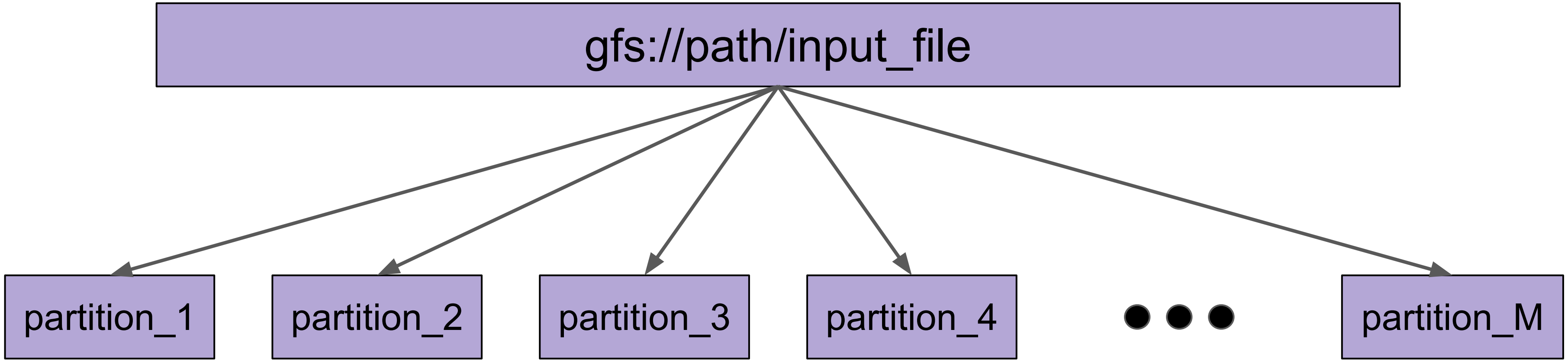
  (* task = partition of data + computation)

Multiple Workers

- Follow whatever the Master asks to do.

Execution Overview

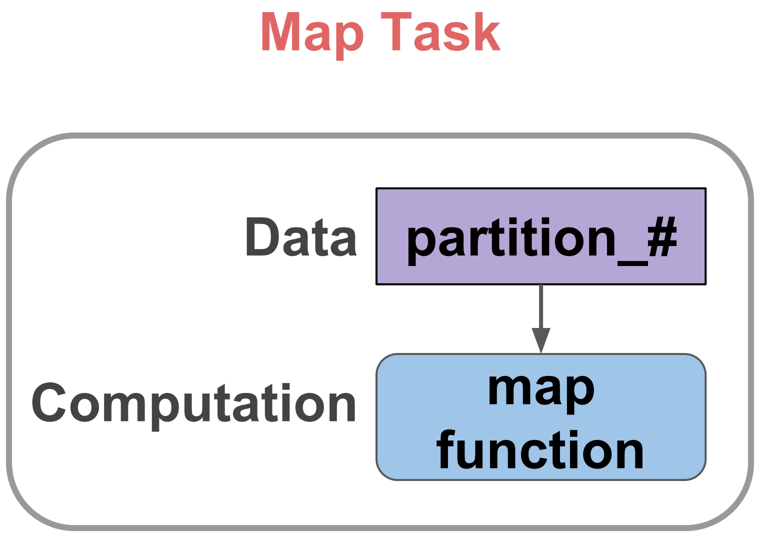
1. The MapReduce library in the user program first splits the input file into **M** pieces.

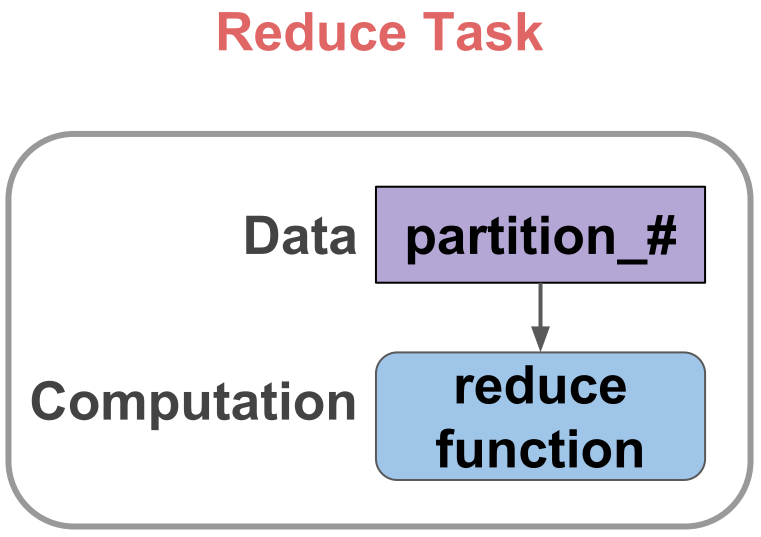


2. The MapReduce library in the user program then starts up many copies of the program on a cluster of machines: one master and multiple workers .

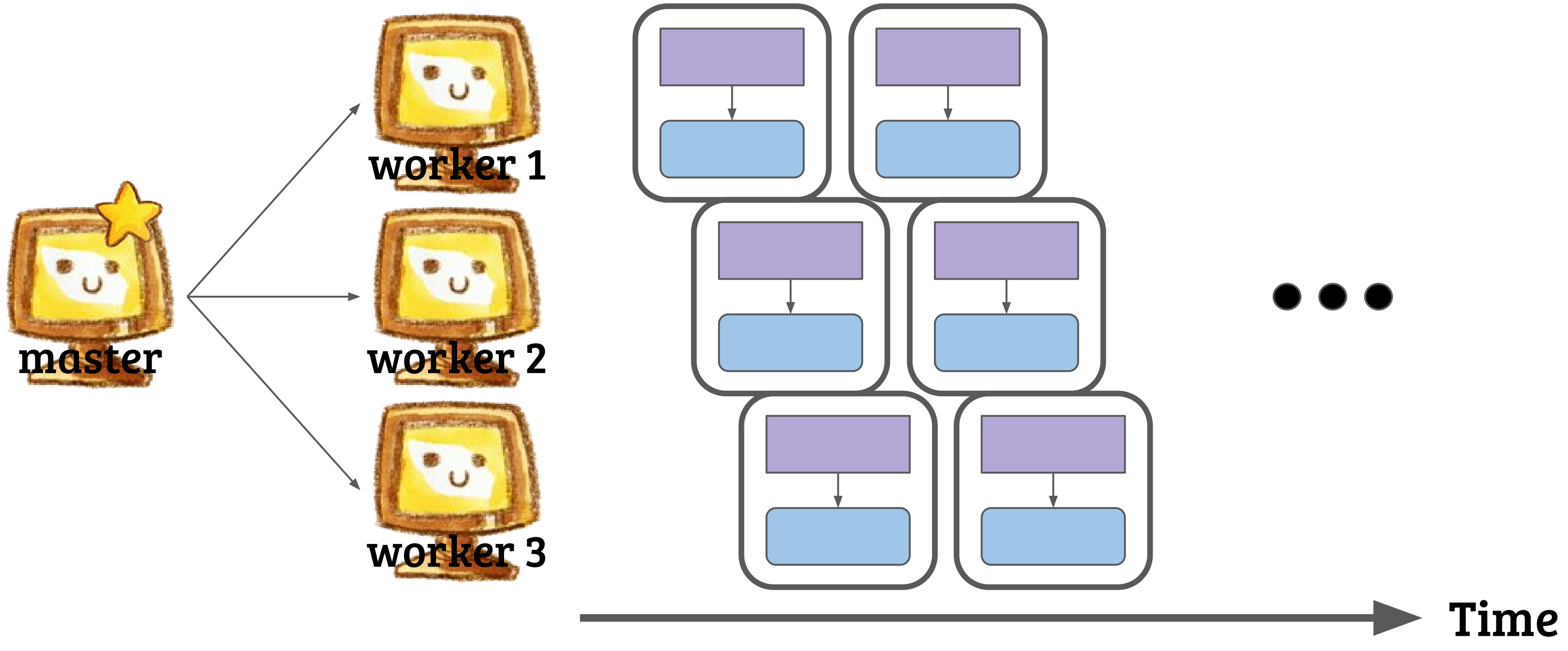
There are **M** map tasks and **R** reduce tasks to assign.

(The figures below depicts task = data + computation)





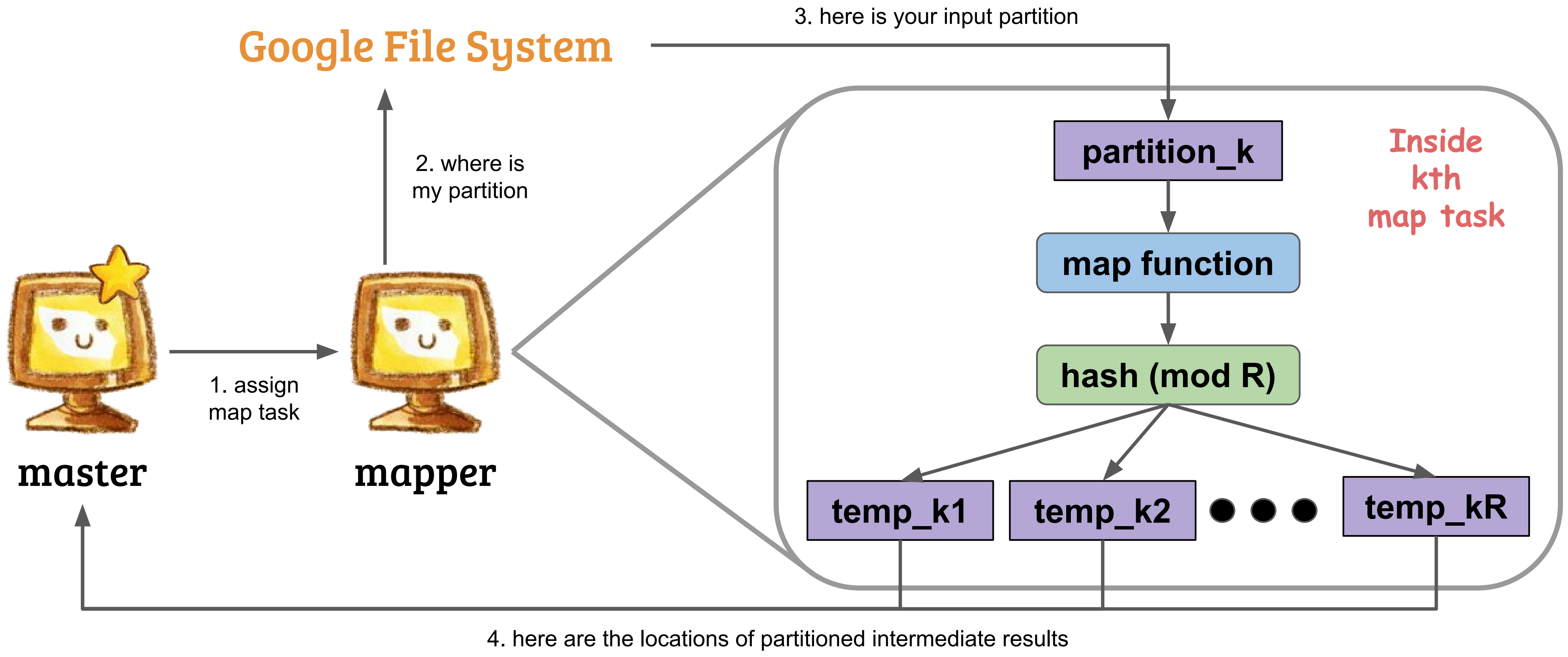
3. The master picks idle workers and assigns each one a map task.



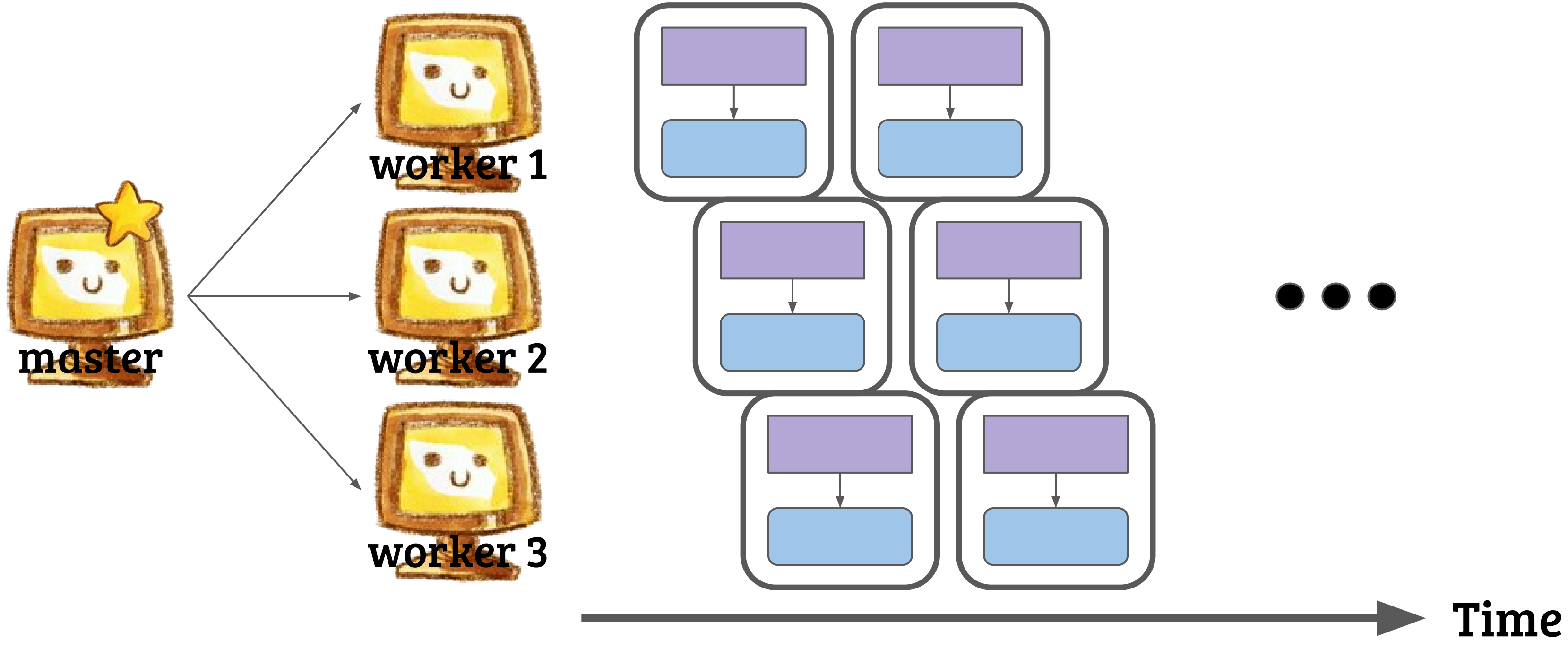
4. Map Phase (each mapper node)

1) Read in a corresponding input partition.
2) Apply the user-defined map function to each key/value pair in the partition.
3) Partition the result produced by the map function into **R** regions using the partitioning function.
4) Write the result into its local disk (not GFS).
5) Notify the master with the locations of each partitioned intermediate result.

Map Phase


3. here is your input partition
Google File System
2. where is my partition
partition_k
Inside kth map task
map function
hash (mod R)
1. assign map task
master
mapper
temp_k1
temp_k2
● ● ●
temp_kR
4. here are the locations of partitioned intermediate results

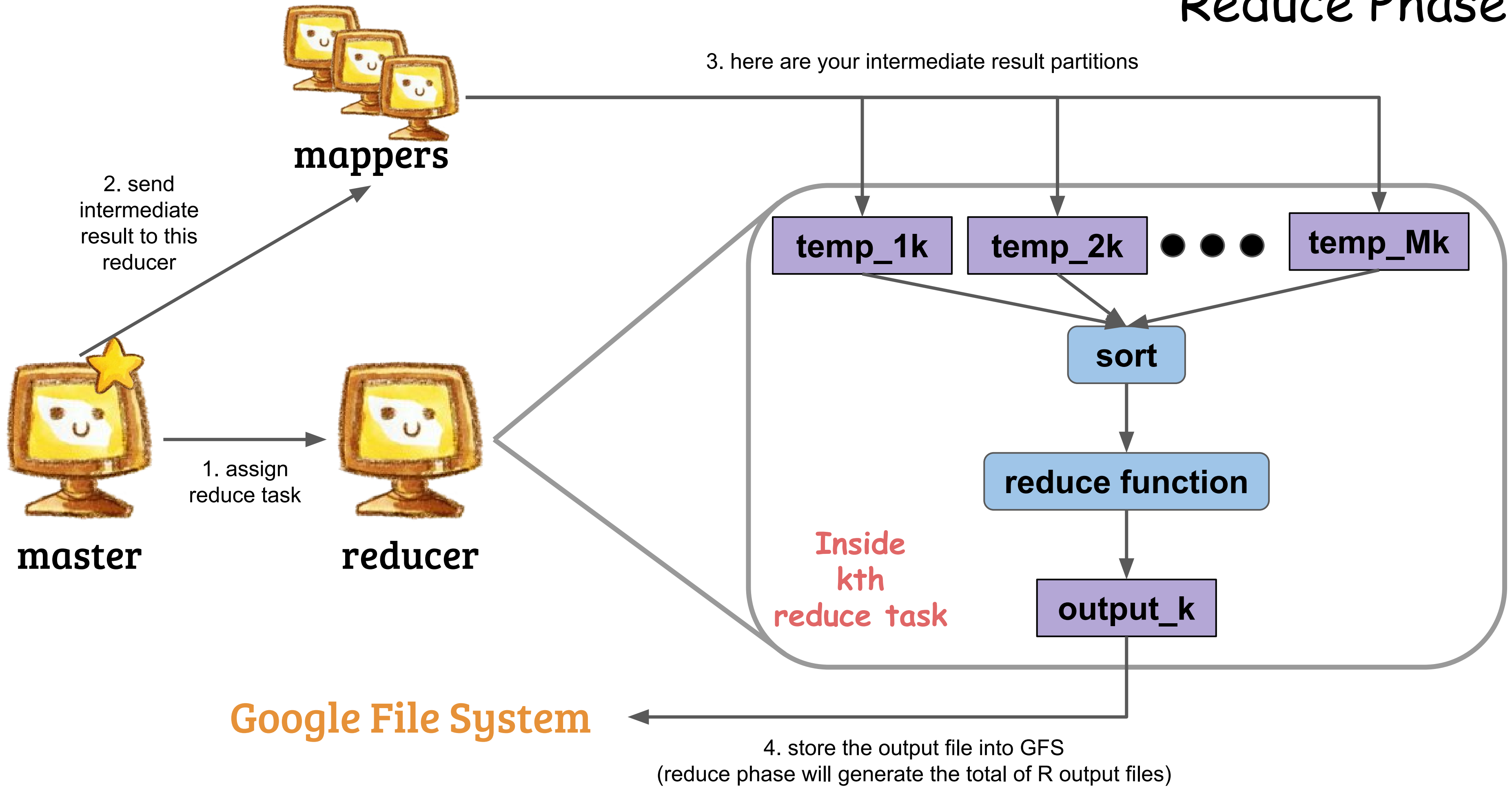
5. After all the map tasks are done, the master picks idle workers and assigns each one a reduce task.



6. Reduce Phase (each reducer node)

1) Read in all the corresponding intermediate result partitions from mapper nodes.
2) Sort the intermediate results by the intermediate keys.
3) Apply the user-defined reduce function on each intermediate key and the corresponding set of intermediate values.
4) Create one output file.

Reduce Phase


3. here are your intermediate result partitions
mappers
2. send intermediate result to this reducer
temp_1k
temp_2k
● ● ●
temp_Mk
sort
reduce function
Inside kth reduce task
output_k
master
1. assign reduce task
reducer
Google File System
4. store the output file into GFS
(reduce phase will generate the total of R output files)

Fault Tolerance

Although the probability of a machine failure is low,
the probability of a machine failing among thousands of machines is common.

How does MapReduce handle machine failures?

Worker Failure

- The master sends heartbeat to each worker node.
- If a worker node fails, the master reschedules the tasks handled by the worker.

Master Failure

- The whole MapReduce job gets restarted through a different master.

Locality

- The input data is managed by GFS.
- Choose the cluster of MapReduce machines such that those machines contain the input data on their local disk.
- We can conserve network bandwidth.

Task Granularity

- It is preferable to have the number of tasks to be multiples of worker nodes.
- Smaller the partition size, faster failover and better granularity in load balance.
  But it incurs more overhead. Need a balance.

Backup Tasks

- In order to cope with a straggler, the master schedules backup executions of the remaining *in-progress* tasks.

MapReduce Pros and Cons

- MapReduce is good for off-line batch jobs on large data sets.
- MapReduce is not good for iterative jobs due to high I/O overhead as each iteration needs to read/write data from/to GFS.
- MapReduce is bad for jobs on small datasets and jobs that require low-latency response.

Apache Hadoop

- Apache Hadoop has an open-source version of GFS and MapReduce.
- GFS -> HDFS (Hadoop File System)
  Google MapReduce -> Hadoop MapReduce
- You can download the software and implement your own MapReduce applications.

In HW3,

- We will be using Apache Spark, which is also a compute engine that can process large sets of data.
- Then, write Spark programs (like MapReduce program) -- PageRank algorithm!
- Lastly, run them on your local machine.
  Each core in your machine will behave as a node.

In HW4,

- We will be using Apache Spark on Amazon Web Services (AWS).
- We will instantiate multiple virtual machines (AWS EC2), and run Spark applications on the cluster of machines.

References

- MapReduce: Simplified Data Processing on Large Cluster - Jeffrey Dean, et al. - 2004
- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing - Matei Zaharia, et al. - 2012
- http://www.slideshare.net/yongho/2011-h3