

L02: Introduction to Hadoop, MapReduce, and Hadoop Streaming

ANLY 502: Massive Data Fundamentals

Simson Garfinkel & Marck Vaisman

January 23, 2017



GEORGETOWN UNIVERSITY

Outline for today's class

Questions (course, homework, lab, AWS, etc.)

Introduction to Hadoop, HDFS, Yarn and MapReduce (~45 min)

Lab # 1 (~30 min)

- Start EMR (Elastic MapReduce) Cluster
- Configure FoxyProxy
- Run some HDFS Commands
- Copy files from S3 to cluster

Introduction to Hadoop Streaming (30 min)

Lab # 2 (~30 min)

- Run example word count Hadoop Streaming job
- Navigate the various user interfaces

Recap...

Back in the early 2000s, companies were building bigger and bigger data centers.
They needed some way to scale computation.

Companies kept facing the same problems.

Price vs. Reliability:

- Cheap machines failed.
- Reliable machines were expensive.

Hardware vs. Software Diversity:

- “Data centers” were designed with computers having similar hardware, but different software configurations.
 - *Hard to keep the system going.*
 - *Hard to install, configure, administer and manager.*

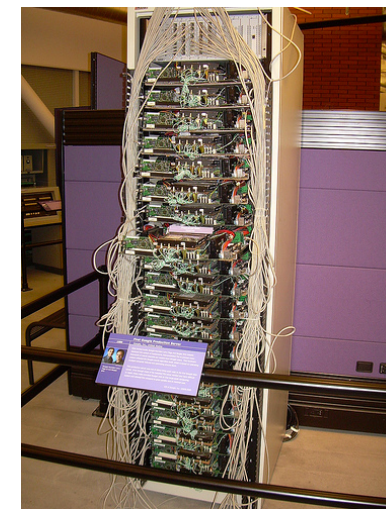
WWW
Server

Biz
Logic

DB
Server



**First Google Computer
Lego enclosure**

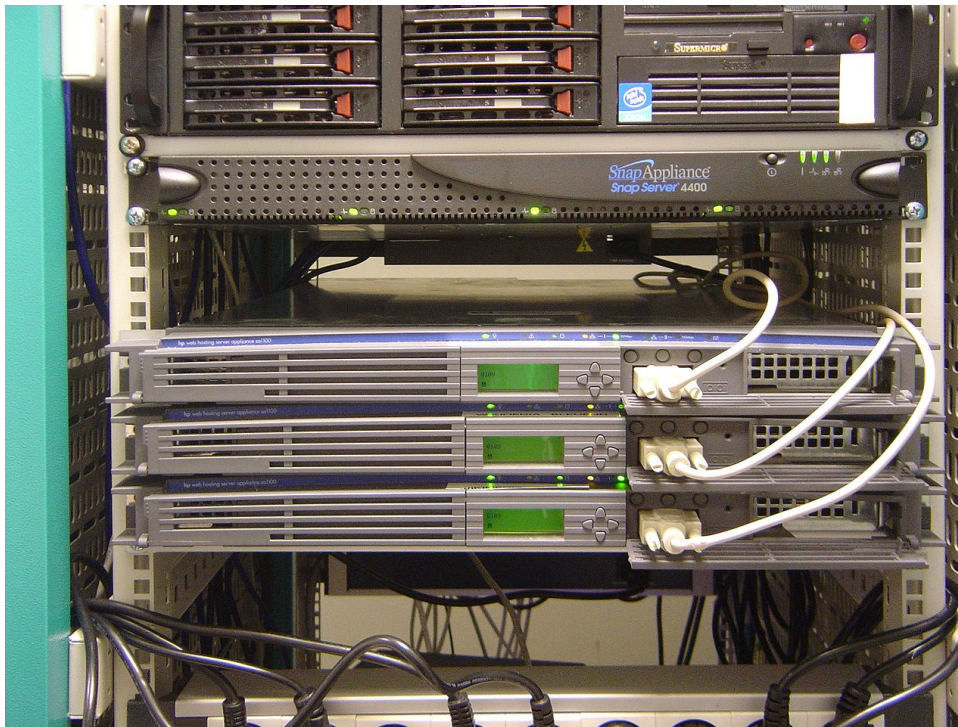


First Google Production Server

This is one approach to scaling...

A fast database server and lots of clients

Provides isolation between the front end and the database.



https://en.wikipedia.org/wiki/Data_center

SuperMicro server
6 high-speed SCSI drives
8 core processor?

HP Web Hosting
Server Appliance SA1100 (3)
2 x 100 Mbps ethernet
10 GB hard drive
533 MHz processor
128MB RAM

The database is a bottleneck.

An easier approach: identical servers.

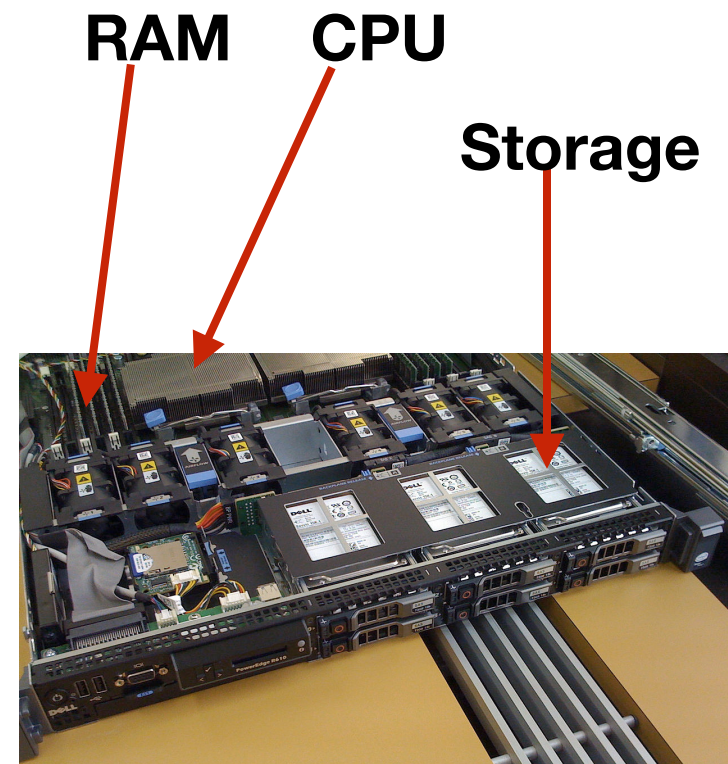
Every computer has same hardware configuration.

Distributed storage.

Distributed computation.



https://commons.wikimedia.org/wiki/File:Wikimedia_Servers-0001_43.jpg



https://en.wikipedia.org/wiki/Dell_PowerEdge



Introducing Hadoop
and MapReduce

In 2003 & 2004 Google Research published two seminal papers.

Google File System Paper

- How Google stored its information — at scale
- The Google File System
Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
19th ACM Symposium on Operating Systems Principles,
Lake George, NY, October, 2003.
<http://research.google.com/archive/gfs.html>

Google MapReduce

- How Google processed its information — at scale
- MapReduce: Simplified Data Processing on Large Clusters
Jeffrey Dean and Sanjay Ghemawat
OSDI'04: Sixth Symposium on Operating System Design and Implementation,
San Francisco, CA, December, 2004.
<http://research.google.com/archive/mapreduce.html>

These papers showed how Google had overcome the scaling problem.

Google File System (GFS) Requirements

Design assumptions:

- System built from many inexpensive components that often fail. These store DATA.
- A high-performance, high-reliability, system. The MASTER stores METADATA.
- Workload consists of two kinds of reads:
 - large, streaming reads. (typically 1MB of more)*
 - small random reads. (typically batched by performance-critical applications)*
- Workload consists of two kinds of writes:
 - large, streaming writes. (sequential, >>1MB)*
 - small writes at arbitrary locations (infrequent; need not be efficient)*
- Well-defined semantic for multiple clients writing to the same file
- High sustained bandwidth is more important than low latency.
 - Designed for bulk, batch processing. (building the index, not searching the index.)*

GFS Implementation

Files are divided into fixed-size (**64MB**) *chunks*.

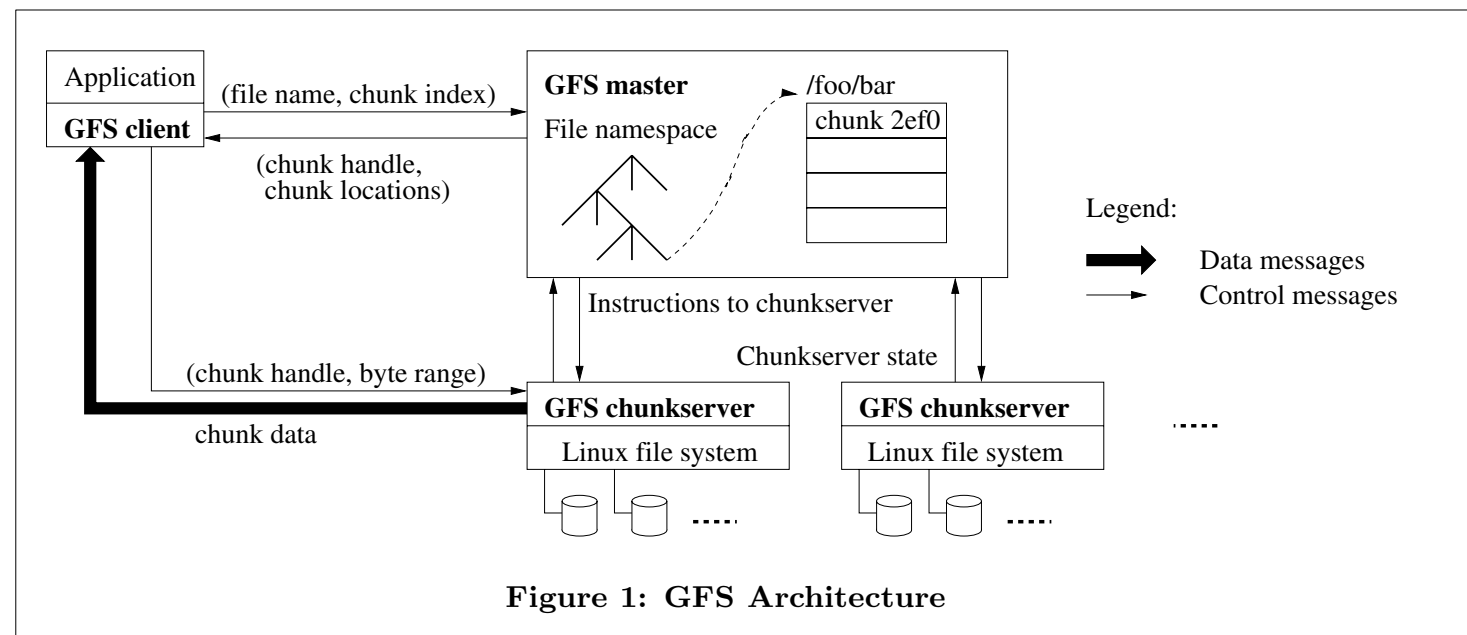
- Each chunk has a unique 64-bit *chunk handle*.
- Each chunk replicated on multiple GFS chunkservers.

Single master:

- Maps filenames to chunks
- Global directory of where each chunk is stored
- Metadata stored in RAM
- Checkpoints to hot backups
- Shadows for read-only access
- Replicates data on node fail

Clients:

- Send filename to master.
- Get chunk handles from master.
- Get chunks from chunkservers.



Google's MapReduce:

A programming paradigm based on functional programming.

Previous work on “grid” computing was based on the idea of splitting up jobs, performing work in parallel, and combining the work:



MapReduce is an *approach* and *infrastructure* for doing this at scale.

Provides:

- Automatic parallelization and distribution
- Speculative execution for slow jobs.
- Fault-tolerance
- I/O scheduling
- Integrated status and monitoring

MapReduce Limitations

Programmer:

- Everything is a Map or Reduce, but we don't think of problems that way
- No control over the *order* in which map() or reduce() runs.
- Mapper & Reducer must be stateless — can't depend on other map() or reduce() operations.
- No random access to the data
- Hard to implement algorithms that can't be easily partitioned

Performance:

- Data is not indexed
- Finding MIN() or MAX() requires scanning *all the data*.
- No memory-to-memory transfers
 - *Everything must be written back to the disk*
- Entire map() must finish before reduce() starts
- Memory limitations on HDFS “Name node”

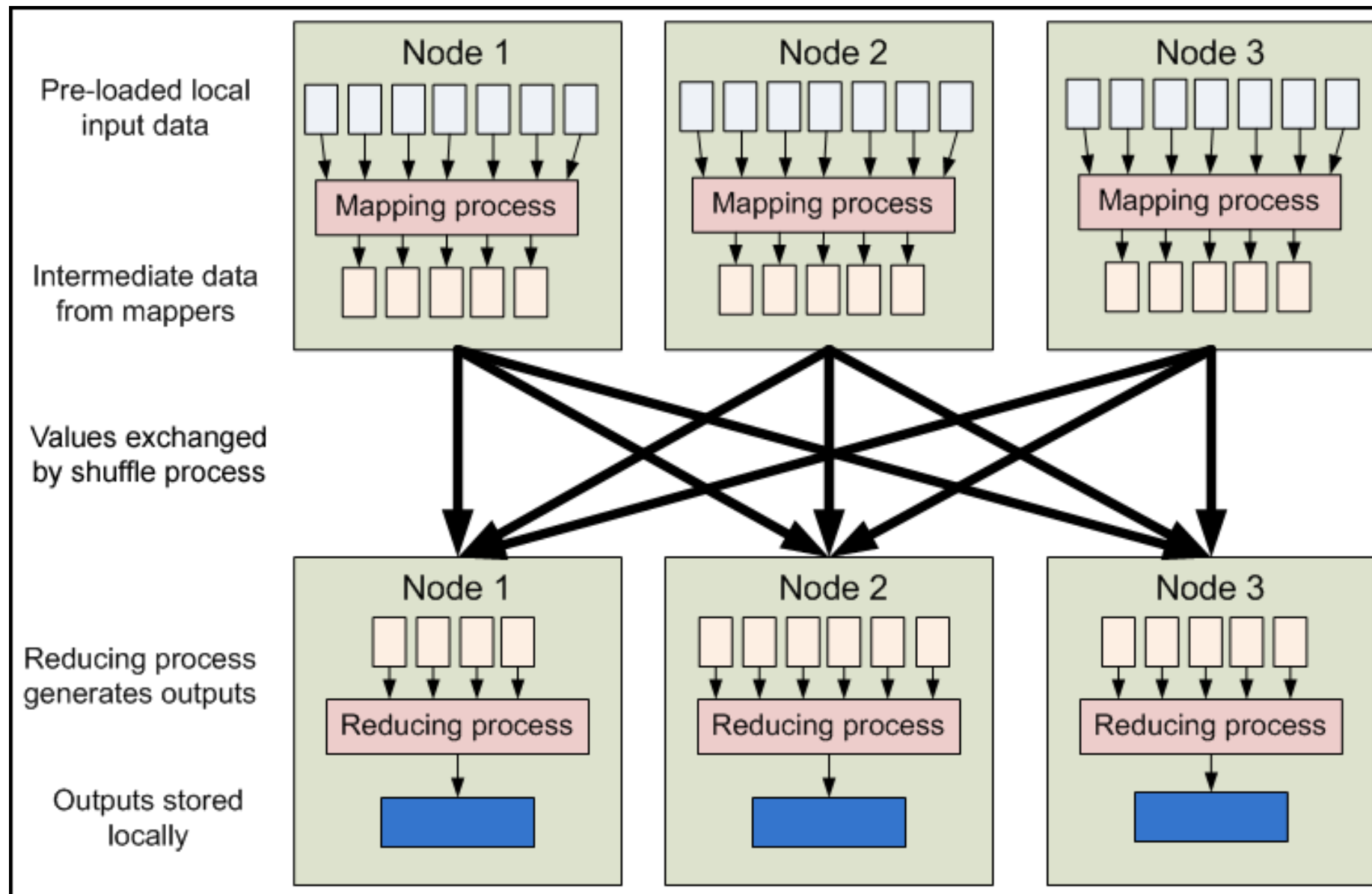
Operational:

- High overhead
- Batch processing
- Does not produce immediate answers

<https://www.quora.com/What-are-some-limitations-of-MapReduce>

<http://stackoverflow.com/questions/18585839/what-are-the-disadvantages-of-mapreduce>

Diagram from Yahoo! developer tutorial



<https://developer.yahoo.com/hadoop/tutorial/module1.html>

The MapReduce programming model is based on functional programming.

Input & Output is a set of key/value pairs.

Programmer specifies a mapper:

- `map (in_key, in_value) → list(out_key, intermediate_value)`
- `reduce (out_key, list(intermediate_values)) → list(out_value)`

Compare with Python:

```
>>> def square(x): return x*x
...
>>> a = range(0,10)
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> map(square, a)
[1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> def add(x,y): return x+y
...
>>> reduce(add, a)
45
```

Google's map & reduce operate on (name, value) pairs.

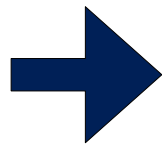
<https://docs.python.org/2/tutorial/datastructures.html>

Map/Reduce

The programmer writes a `map()` function:

```
map(input) -> key:value
```

The framework calls `map()` for every piece of input



```
map(input1) -> key1:v1  
map(input2) -> key2:v2  
map(input3) -> key3:v3
```

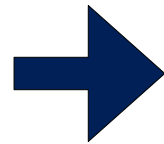
Map/Reduce

The programmer writes a map() function:

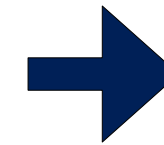
```
map(input) -> key:value
```

The framework sends the map() function to every computer with data:

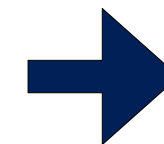
```
map(input1) -> key1:v1  
map(input2) -> key2:v2  
map(input3) -> key3:v3
```



```
map(input11) -> key1:v11  
map(input12) -> key2:v12  
map(input13) -> key3:v13
```



```
map(input21) -> key1:v21  
map(input22) -> key1:v22  
map(input23) -> key1:v23
```



**“embarrassingly
parallel”**

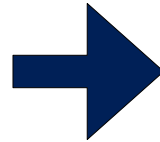
Map/Reduce

The programmer writes a **map()** function:

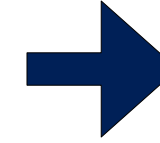
```
map(input) -> key:value
```

The framework sorts the output by key:

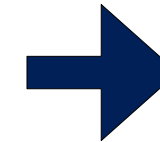
```
map(input1) -> key1:v1  
map(input2) -> key2:v2  
map(input3) -> key3:v3
```



```
map(input11) -> key1:v11  
map(input12) -> key2:v12  
map(input13) -> key3:v13
```



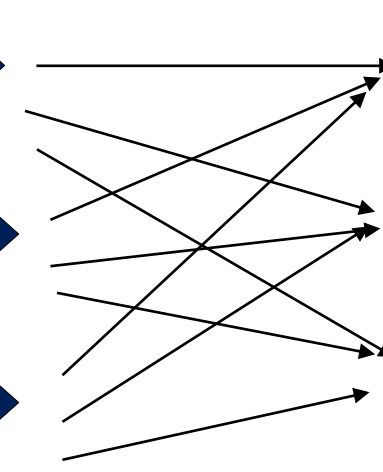
```
map(input21) -> key1:v21  
map(input22) -> key1:v22  
map(input23) -> key1:v23
```



key1: v1, v11, v21

key2: v2, v12, v22

key3: v3, v13, v23



**“embarrassingly
parallel”**

**group by key
(expensive)**

Map/Reduce

The programmer writes a **reduce()** function:

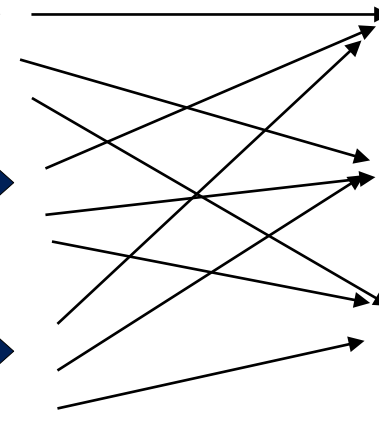
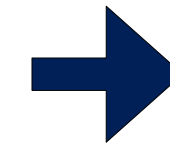
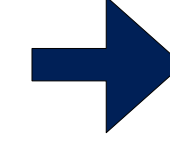
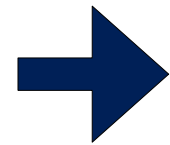
```
reduce(key, values) -> key:value
```

The framework sorts the output by key:

```
map(input1) -> key1:v1  
map(input2) -> key2:v2  
map(input3) -> key3:v3
```

```
map(input11) -> key1:v11  
map(input12) -> key2:v12  
map(input13) -> key3:v13
```

```
map(input21) -> key1:v21  
map(input22) -> key1:v22  
map(input23) -> key1:v23
```



key1: v1, v11, v21



final1

key2: v2, v12, v22



final2

key3: v3, v13, v23



final3

**“embarrassingly
parallel”**

**group by key
(expensive)**

reduce

“Word Count” is a common MapReduce demonstration program. This Word Count generates a word histogram.

The mapper:

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

The reducer:

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

The output:

“to be or not to be”

becomes:

to:1

be:1

or:1

not:1

to:1

be:1

The framework guarantees
that “reduce” is called with
all pairs of the same key.

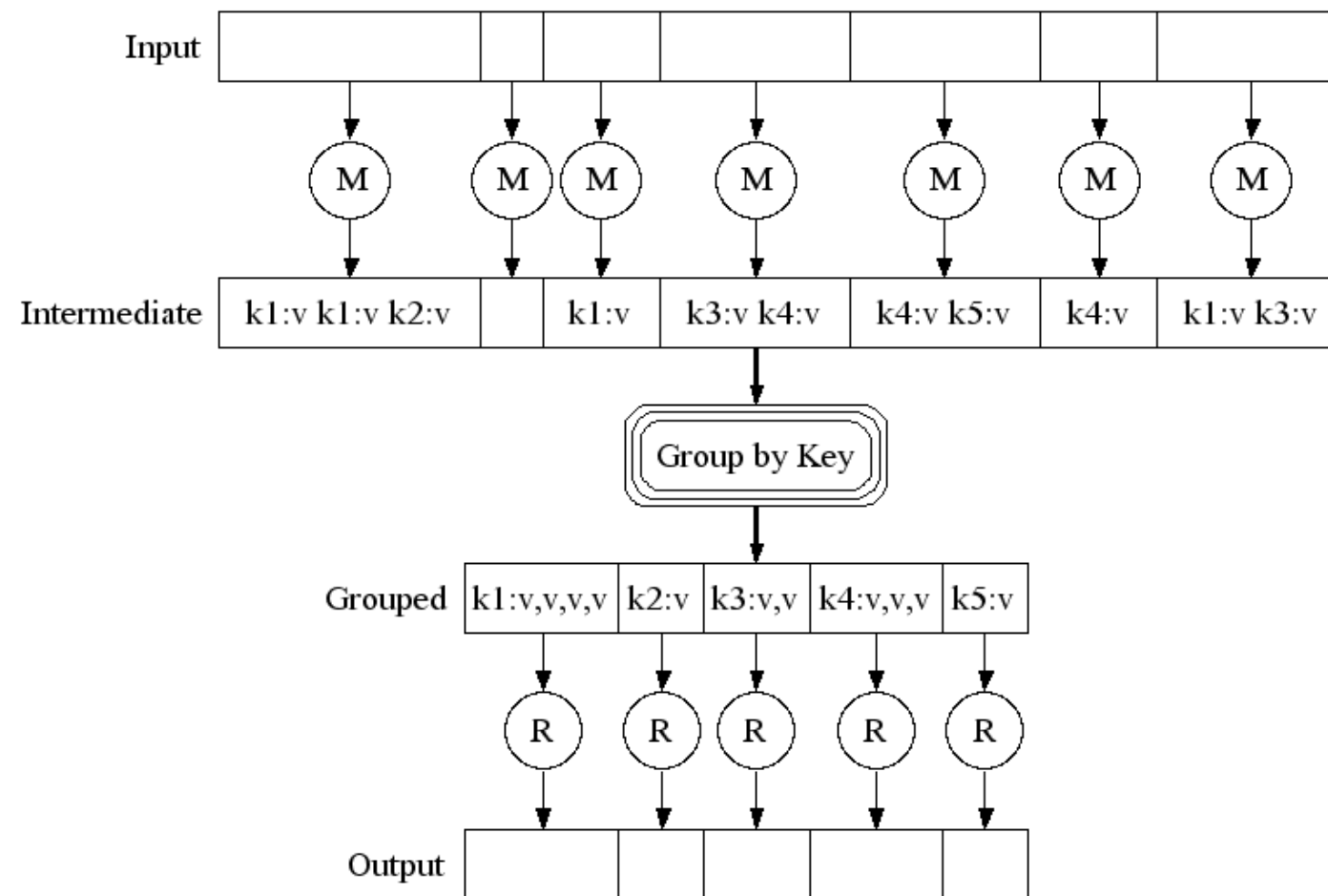
to:2

be:2

or:1

not:1

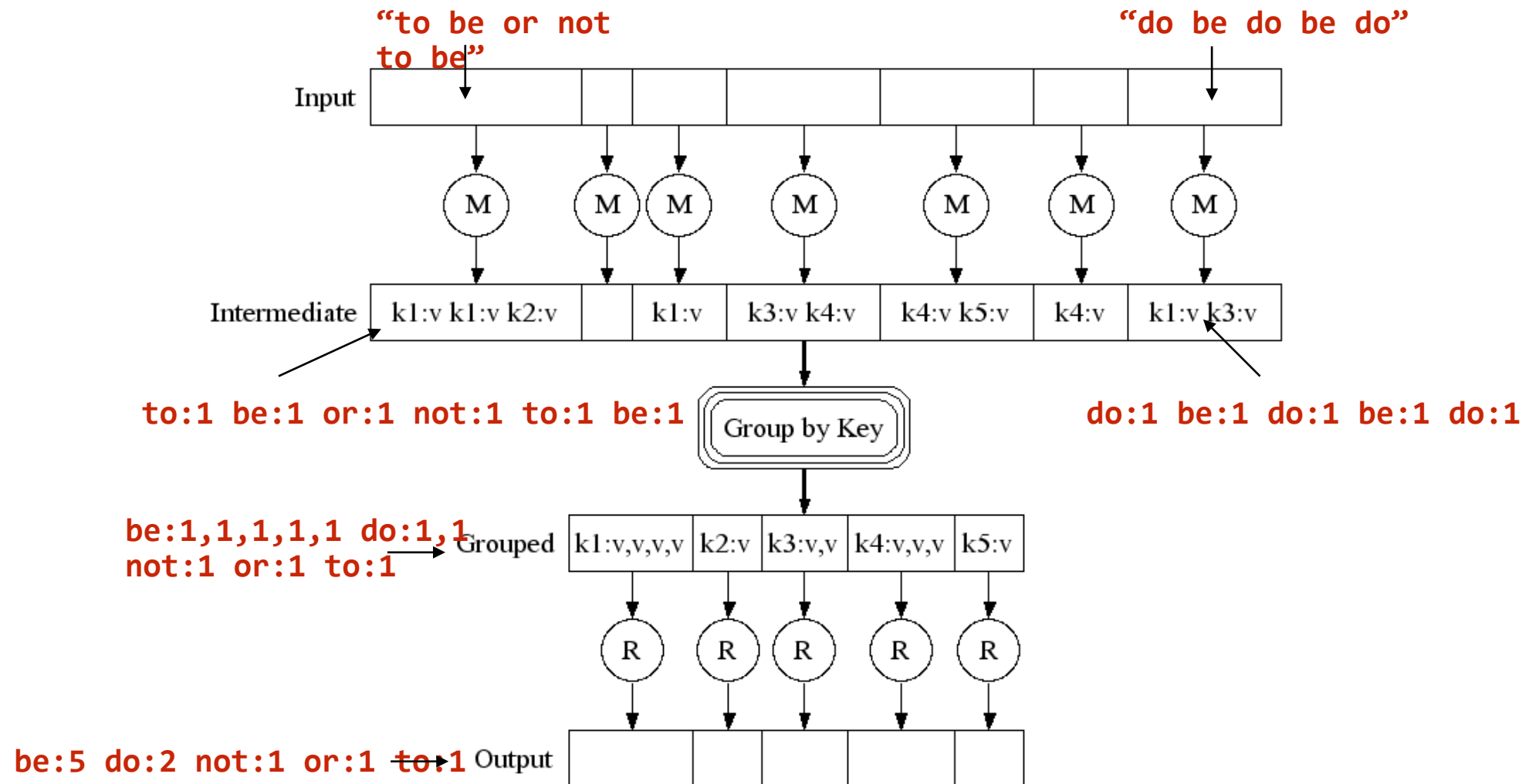
How this gets put together:



<http://research.google.com/archive/mapreduce-osdi04-slides/index-auto-0007.html>

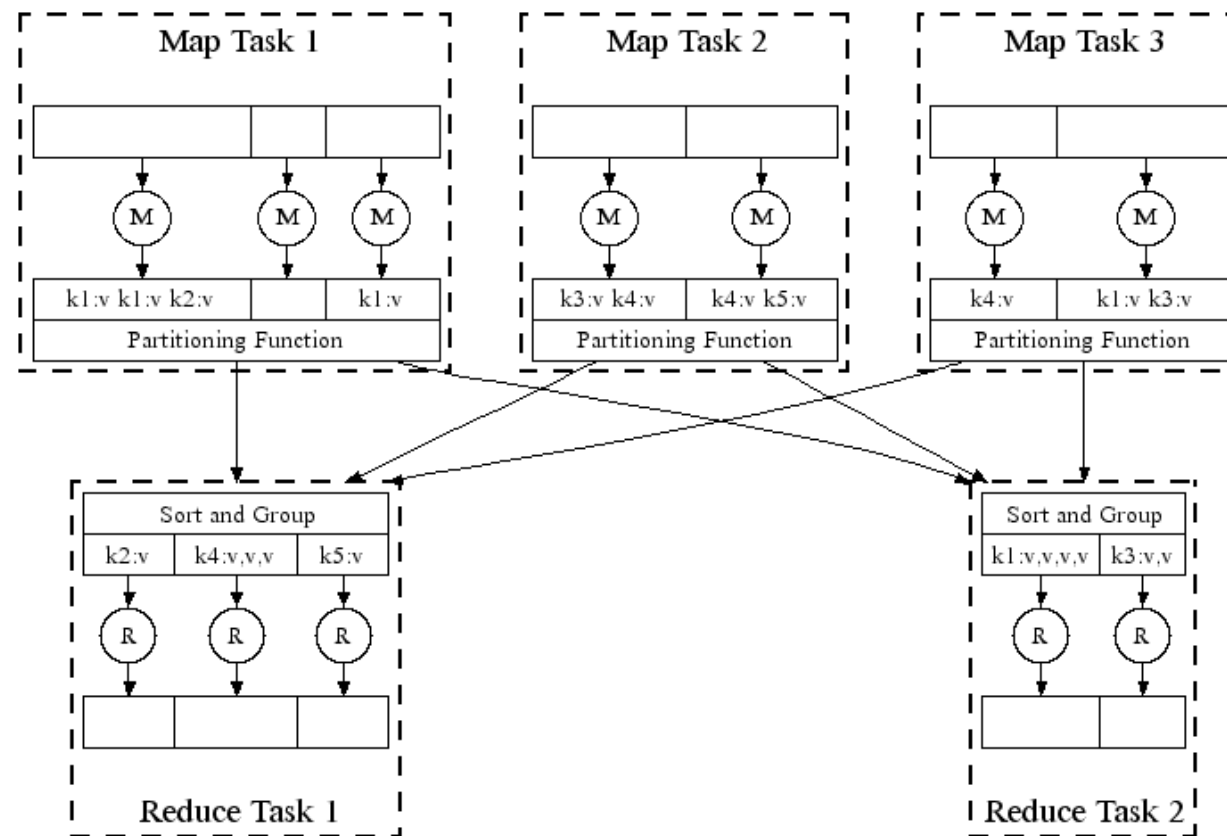
How this gets put together.

This time we'll use multiple inputs.



<http://research.google.com/archive/mapreduce-osdi04-slides/index-auto-0007.html>

Behind the scenes, MapReduce sorts and combines the data.



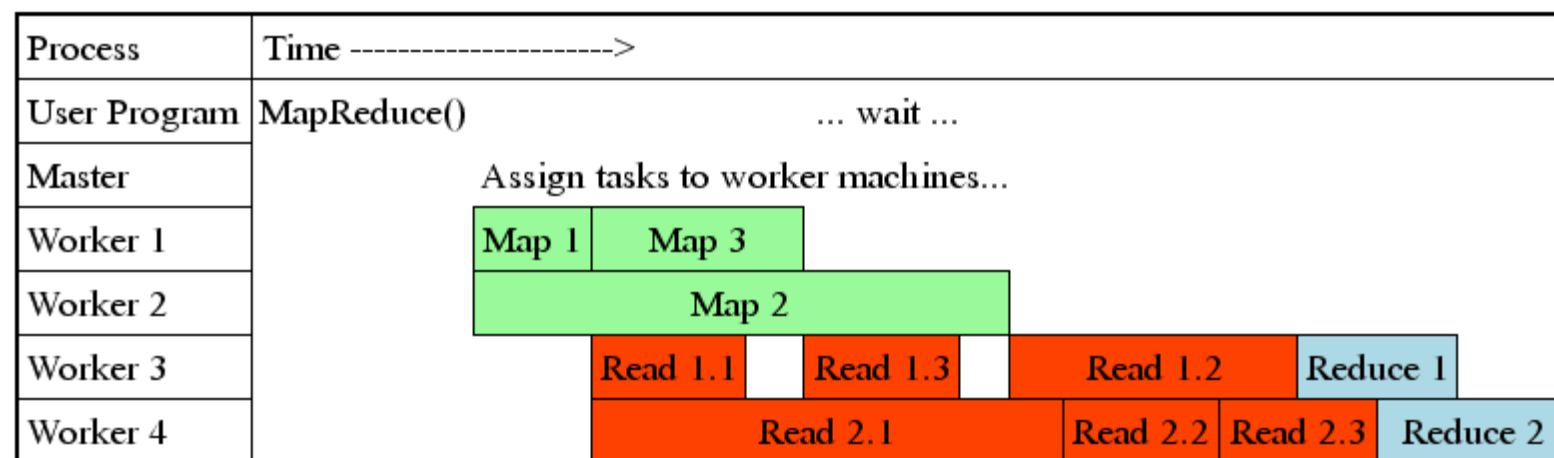
<http://research.google.com/archive/mapreduce-osdi04-slides/index-auto-0008.html>

MapReduce pipelines execution and provides fault recovery

Workers run both map & reduce tasks.

- Each task is scheduled when data are available.
- Failed tasks (or slow machines) are automatically rescheduled.
- If the same data causes two mappers to fail, the data is ignored.

“Often use 200,000 map/5000 reduce tasks with 2000 machines”



<http://research.google.com/archive/mapreduce-osdi04-slides/index-auto-0009.html>

MapReduce is a powerful programming paradigm.

The previous example used strings, but map & reduce can apply to any kind of data value.

Examples (from paper)

- Parallelized String search
 - *map emits a pair of string is present (line #, string)*
 - *Reduce is the “identity” function — copies input to output. (line #, string)*
- Count URL Access Frequency:
 - *map reads logfiles and outputs (URL, 1)*
 - *reduce adds up all of the URLs (URL, total count)*
- Reverse Web-Link Graph (what points to page P?)
 - *map outputs (target, source) for each link found on each web page.*
 - *reduce concatenates the sources: (target, list(source))*
e.g. (target, (source1, source2, source3))

MapReduce benefits

Fault tolerant: all of the inputs are pre-determined from the data.

- If a worker fails, that job can be run on another machine.
- The master writes periodic checkpoints. If it dies, it is restarted.
- “However, given that there is only a single master, it’s failure is unlikely; therefore our current implementation about the MapReduce computation if the master fails.”

Minimizes network bandwidth:

- Attempts schedule workers on the same network node as the data resides.
- Failing that, it tries to schedule the worker on the same network switch

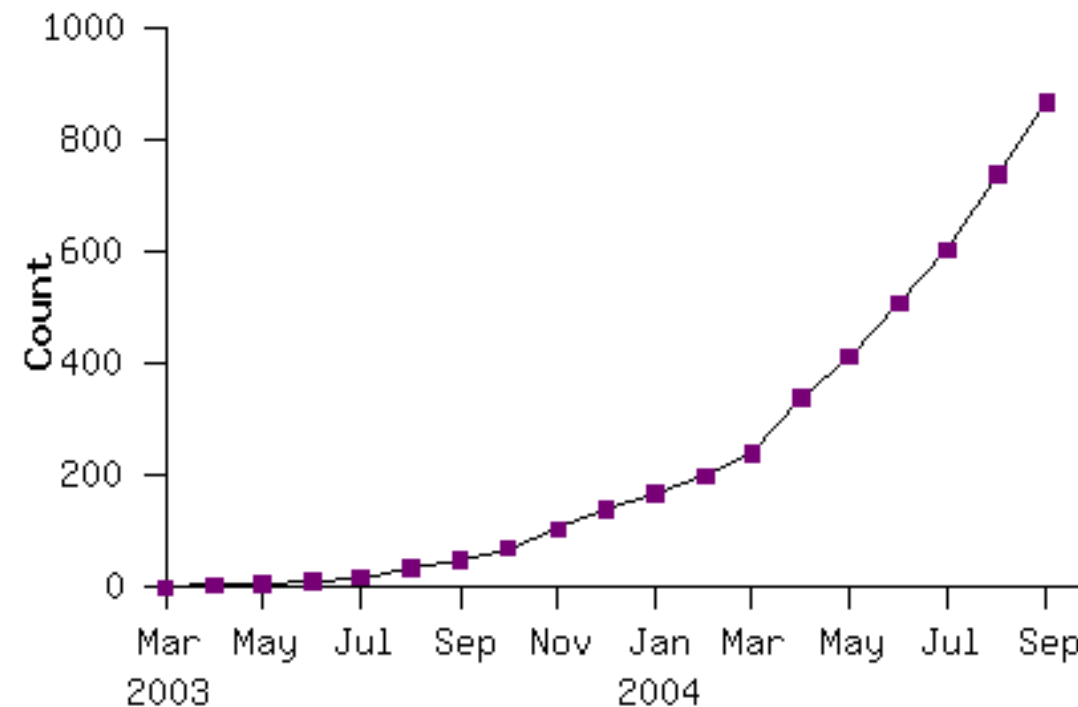
Easier to program!

- Map & Reduce functions are simple and easy to understand.
- Complexity is taken care of by infrastructure.
- Most tasks go faster when you add more machines.

MapReduce was hugely successful at Google.

Easy to modify existing tasks to run on MapReduce framework.

- MapReduce Programs in Google Source Tree:



Hadoop Origins — An open source version of Google's stack

Doug Cutting had been trying to build a search engine at the Internet archive.

- It could only run on certain kinds of machines.
- It required reliable computers.
- When it crashed, it needed to be manually restarted.

Cutting & Cafarella decided to build an open source version of the Google stack to handle the Internet Archive's search.

- In Java, so it would be portable.
—and because it's what they knew



Doug Cutting



Mike Cafarella

In 2006, Cutting moved to Yahoo.

- It was difficult scaling to larger # of nodes.
- Hadoop wasn't good enough to replace Yahoo's search, but it could be used for data analytics.

In 2011, Yahoo had 42,000 nodes and 100s of PBs of storage.

Yahoo spun out Hortonworks as a Hadoop-focused software company.

- <https://gigaom.com/2013/03/04/the-history-of-hadoop-from-4-nodes-to-the-future-of-data/>

Doug Cutting named Hadoop after his son's toy elephant

The New York Times

BUSINESS COMPUTING

Hadoop, a Free Software Program, Finds Uses Beyond Search

By ASHLEE VANCE MARCH 16, 2009



- <http://www.nytimes.com/2009/03/17/technology/business-computing/17cloud.html>

**Hadoop runs on individual computers in a data center.
These computers are called “nodes.”**

A typical small Hadoop system might have:

- 1 master node
- 1-10 Data Nodes
- 0-10 Compute Nodes



Master Node.

- Batch jobs submitted.
- Tracks progress of jobs.

Worker Nodes:



Compute Node.

- Runs Map/Reduce jobs



Data Node.

- Holds data
- (Can also run jobs)

Real-world Map Reduce.

MapReduce is run as a “batch” operation with a *job configuration*:

- Map function
- Reduce function
- Job parameters

The Hadoop *job client* submits the job (e.g. jar file) to the ResourceManager.

Hadoop Streaming lets jobs be run with any executable.

Hadoop Pipes is a SWIG C++ API for running from C++, python, etc.

<https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>

[https://github.com/chenmiao/Big_Data_Analytics_Web_Text/wiki/Hadoop-with-Cloudera-VM-\(the-Word-Count-Example\)](https://github.com/chenmiao/Big_Data_Analytics_Web_Text/wiki/Hadoop-with-Cloudera-VM-(the-Word-Count-Example))

Real Hadoop clusters can be huge.

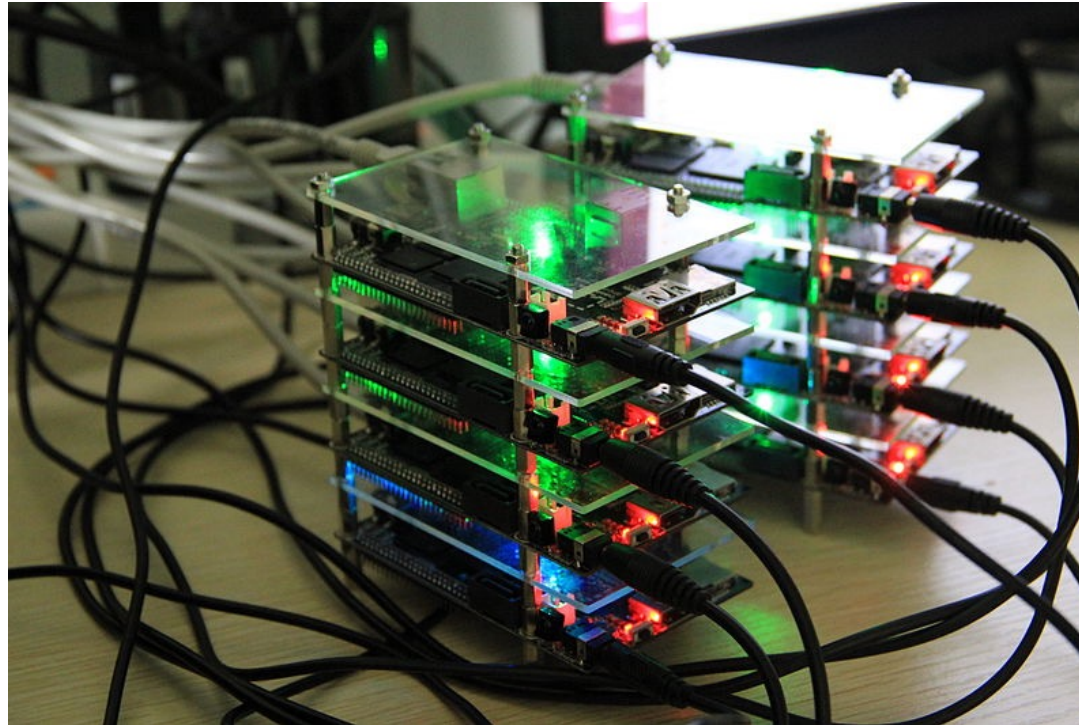


- Name Node — Keeps track of every file and where it is stored.

Hadoop cluster at Yahoo!

Hadoop doesn't *need* huge clusters

Hadoop running on 8 cubieboards:



<http://cubieboard.org/2013/08/01/hadoophigh-availability-distributed-object-oriented-platform-on-cubieboard/>

**But you would
never do this in
practice.**

Why not?

The power of Hadoop (and MapReduce) is that it:

- Provides a framework for having a distributing a workflow to multiple physical computers.
- Integrates management of computation and storage.

A Brief History of Hadoop

The Apache Nutch project

- Open source web crawler and search engine
- Created by Doug Cutting and Mike Cafarella

Searching and Indexing the Web is Expensive

- Estimated that a billion page index = \$500K in hardware
- Operating costs of ~\$30K/month



A Brief History of Hadoop (continued)

2003: *The Google File System* by Sanjay Ghemawat et.al.

Google File System

- stores data computation locally
- distributed file system
- management with a single master server
- low-cost, commodity storage nodes

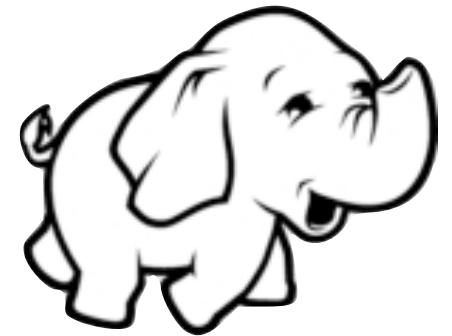


A Brief History of Hadoop (continued)

2004: *MapReduce: Simplified Data Processing on Large Clusters* by Dean & Ghemawat

- Parallelization: how to parallelize the computation
- Distribution: how to distribute the data
- Fault-tolerance: how to handle component failure

MapReduce is the computational framework that enables the parallel, distributed processing off the GFS cluster.



2005: Doug Cutting and Mike Cafarella reimplement MapReduce and GFS (HDFS) in open-source to power Nutch.

2008:

- Yahoo! announces that production search engine running on a 10,000 node Hadoop cluster
- Apache elevates Hadoop to a top-level project

The Motivation for Hadoop

- Traditional data analysis involve complex processing (regressions, etc.) upon small data sets, usually representative samples of a larger population of data.
- Computations of this type are dependent on the size and performance of a processor or the computer's main memory.
- To improve computational speed or the amount of data a computer is able to process, faster processors and more RAM is required.

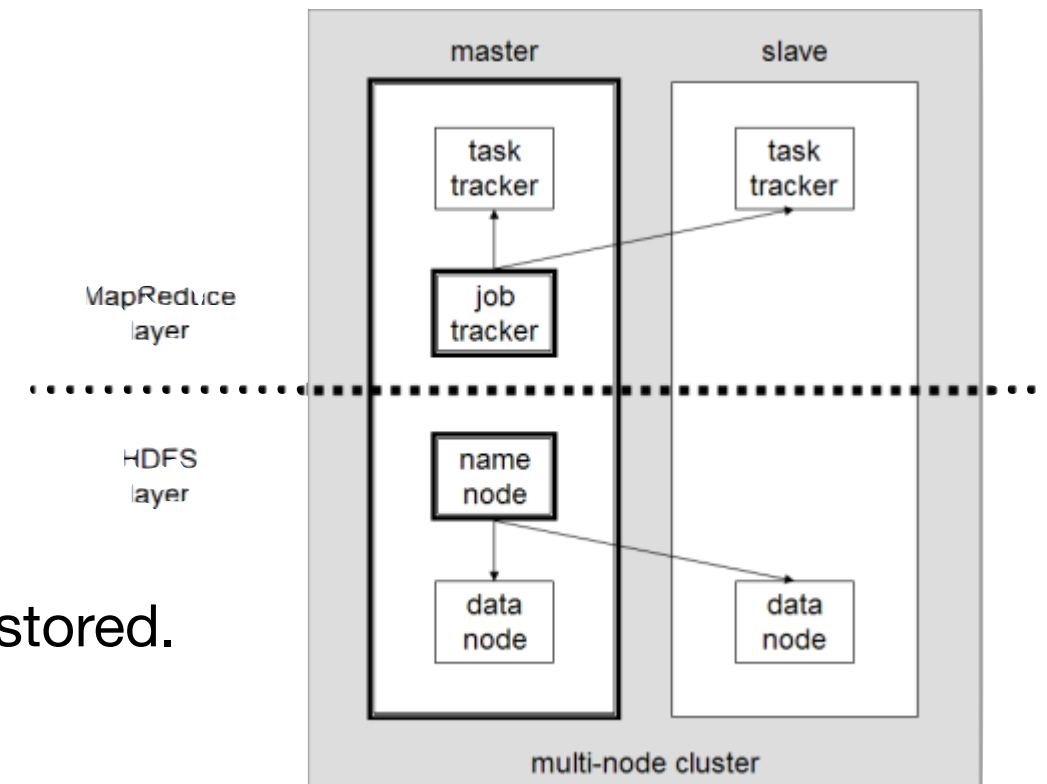
Hadoop has two main systems: MapReduce and HDFS

MapReduce — Performs computation

- Job Tracker — Master planner
- Task Tracker — Runs each task

HDFS — Stores the data

- Data Node — Stores the blocks for each file.
- Name Node — Keeps track of every file and where it is stored.
— *Controls the Data Nodes.*



Data must be stored where MapReduce can reach it

- Hadoop MapReduce: HDFS or Amazon S3
- mrjob: in local file system, HDFS or S3

Remember the basic map reduce idea:

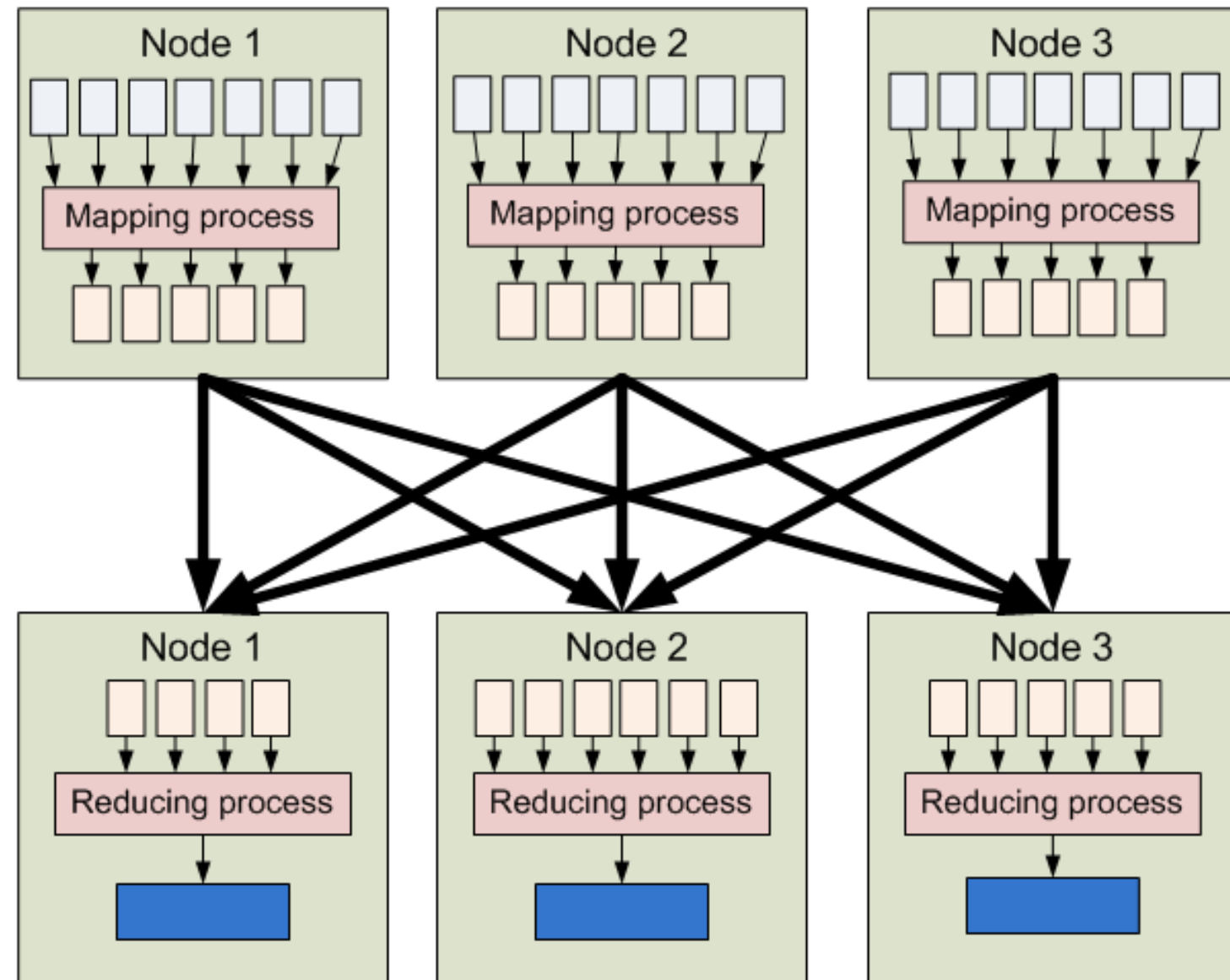
1. Data are distributed on nodes.
2. Data are split
3. Data sent to mapping processes
3. Map data are “shuffled” (sorted)
4. Data are reduced.

You must write:

- mapper
- reducer

You may also write:

- combiner — at Node before shuffle
- partitioner — describes how data are split



<https://developer.yahoo.com/hadoop/tutorial/module1.html>



Hadoop 2 on Yarn

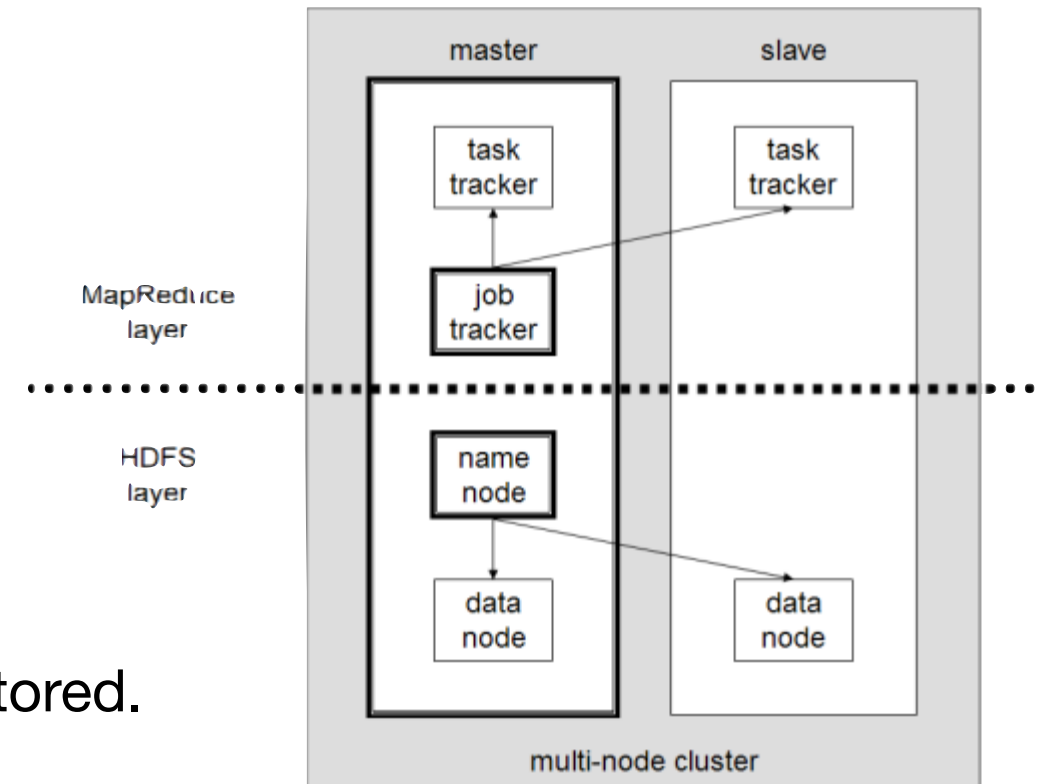
Hadoop has two main systems: MapReduce and HDFS

MapReduce — Performs computation

- Job Tracker — Master planner
- Task Tracker — Runs each task

HDFS — Stores the data

- Data Node — Stores the blocks for each file.
- Name Node — Keeps track of every file and where it is stored.
 - *Controls the Data Nodes.*



Data must be stored where MapReduce can reach it

- Hadoop MapReduce: HDFS or Amazon S3
- mrjob: in local file system, HDFS or S3

Hadoop 2 on YARN

YARN is the resource management and computation framework that is new as of Hadoop 2, which was released late in 2013.

YARN implements a MapReduce application, therefore for the most part, discussing MapReduce as the distributed computational framework of Hadoop is correct, however YARN expands on the distributed computational ability that is available to Hadoop.

Hadoop 2 also led to some version confusion as there were, at one point, parallel versions of Hadoop that are now simply called Hadoop 1 and Hadoop 2.



Hadoop Releases

Apache Hadoop Releases

hadoop.apache.org/releases.html

Apps VA SOUPS 2017 Japan book H PX M YouTube 15 wikis apps \$

Apache > Hadoop >



Top Wiki

Search with Apache Solr Search

Last Published: 10/11/2016 18:50:42

About

- Welcome
- Releases
- Download
- Release Notes
- Release Versioning
- Mailing Lists
- Issue Tracking
- Who We Are?
- Who Uses Hadoop?
- Buy Stuff
- Sponsorship
- Thanks
- Privacy Policy
- Bylaws
- Committer criteria
- License
- Documentation
- Related Projects

Apache Hadoop Releases

Download

Hadoop is released as source code tarballs with corresponding binary tarballs for convenience. The downloads are distributed via mirror sites and should be checked for tampering using GPG or SHA-256.

Version	Release Date	Tarball	GPG	SHA-256
3.0.0-alpha1	03 September, 2016	source	signature	checksum file
		binary	signature	checksum file
2.7.3	25 August, 2016	source	signature	227785DC 6E3E6EF8..
		binary	signature	D489DF38 08244B90..
2.6.5	08 October, 2016	source	signature	3A843F18 73D9951A..
		binary	signature	001AD18D 4B6D0FE5..
2.5.2	19 Nov, 2014	source	signature	139EF872 09C5637E..
		binary	signature	0BDB4850 A3825208..

To verify Hadoop releases using GPG:

1. Download the release `hadoop-X.Y.Z-src.tar.gz` from a [mirror site](#).
2. Download the signature file `hadoop-X.Y.Z-src.tar.gz.asc` from [Apache](#).
3. Download the [Hadoop KEYS](#) file.
4. `gpg --import KEYS`
5. `gpg --verify hadoop-X.Y.Z-src.tar.gz.asc`

To perform a quick check using SHA-256:

1. Download the release `hadoop-X.Y.Z-src.tar.gz` from a [mirror site](#).
2. Download the checksum `hadoop-X.Y.Z-src.tar.gz.md5` from [Apache](#).
3. `shasum -a 256 hadoop-X.Y.Z-src.tar.gz`

All previous releases of Hadoop are available from the [Apache release archive](#) site.

Many third parties distribute products that include Apache Hadoop and related tools. Some of these are listed on the [Distributions wiki page](#).

Release Notes

08 October, 2016: Release 2.6.5 available

Client applications submit jobs to the Job tracker.

The JobTracker talks to the NameNode to determine the location of the data

The JobTracker locates TaskTracker nodes with available slots at or near the data

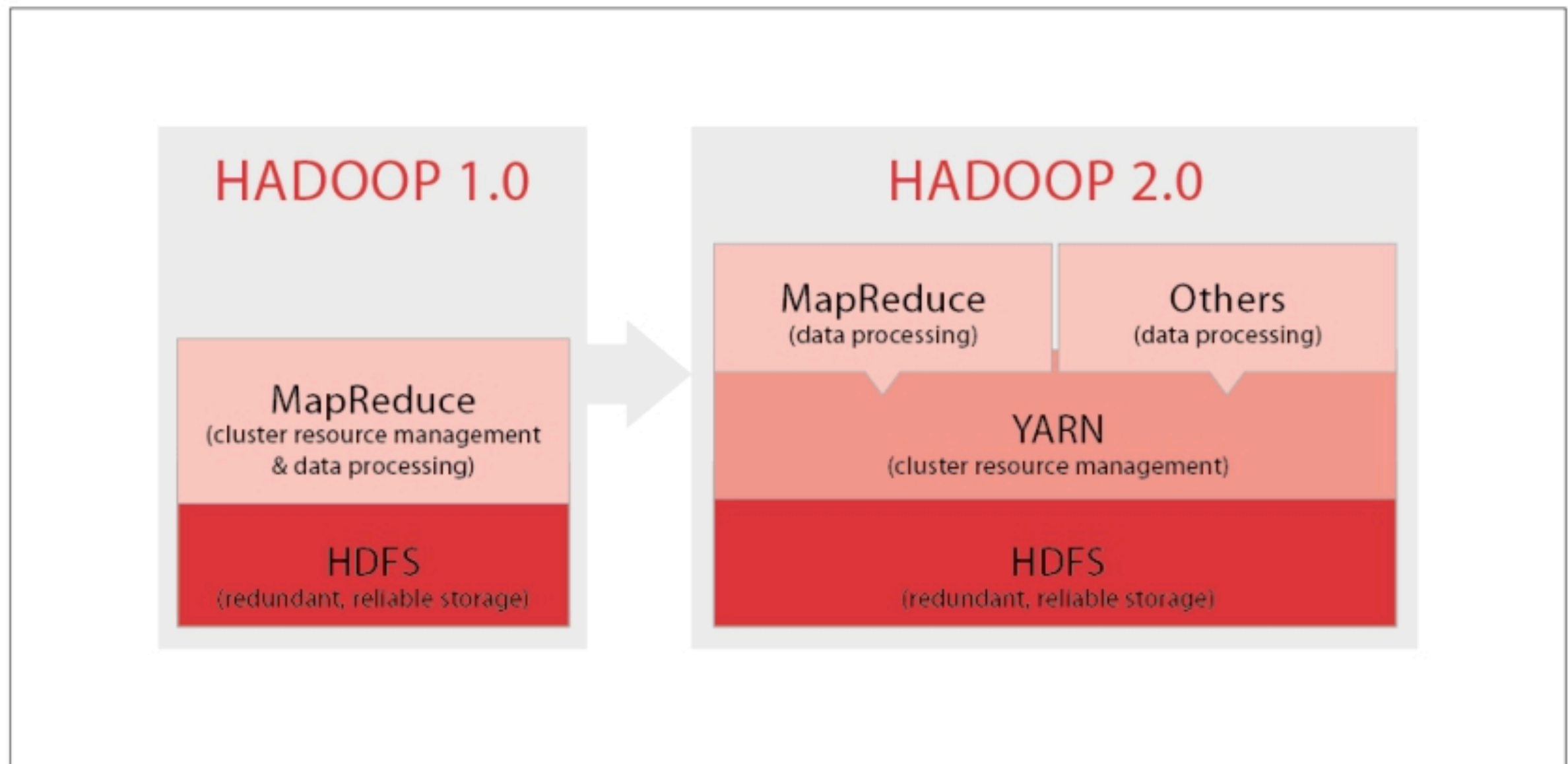
The JobTracker submits the work to the chosen TaskTracker nodes.

The TaskTracker nodes are monitored. If they do not submit heartbeat signals often enough, they are deemed to have failed and the work is scheduled on a different TaskTracker.

A TaskTracker will notify the JobTracker when a task fails. The JobTracker may resubmit the job elsewhere, it may mark that specific record as something to avoid, and it may even blacklist the TaskTracker as unreliable.

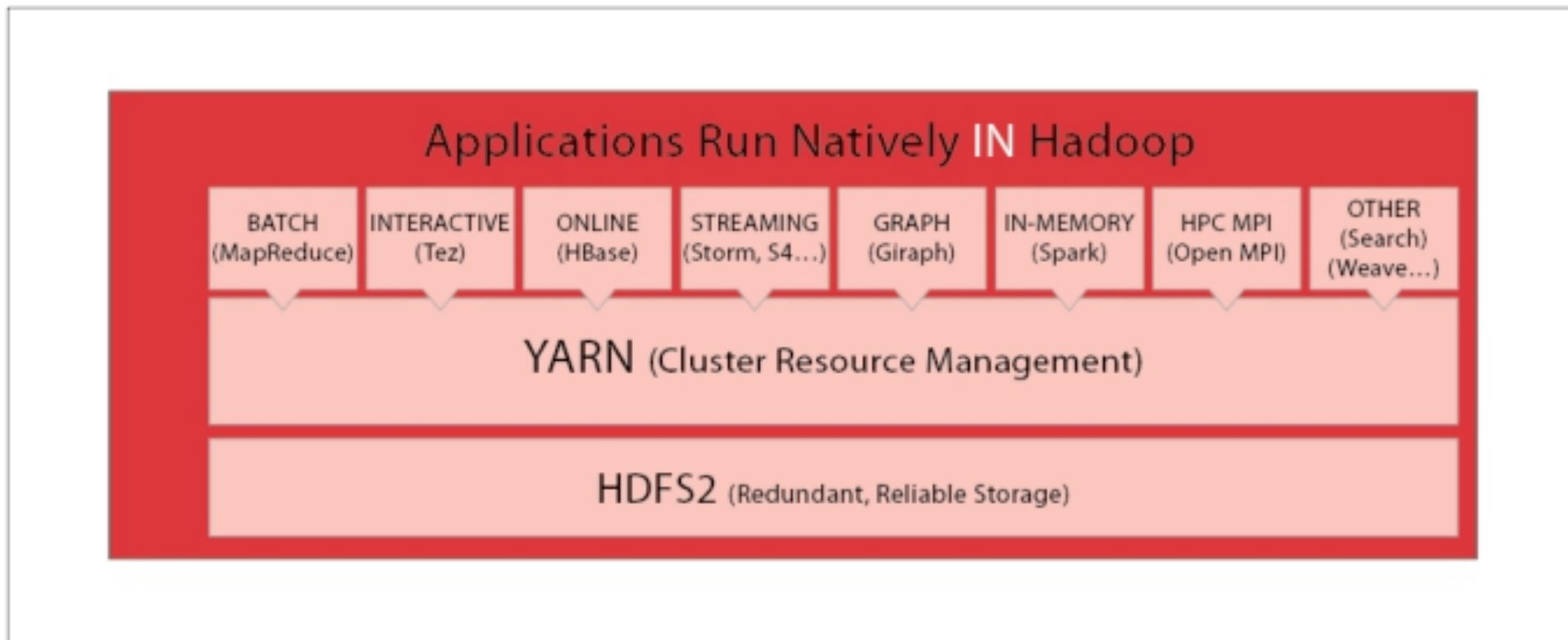
When the work is completed, the JobTracker updates its status.

Client applications can poll the JobTracker for information.



Hadoop 2 on YARN

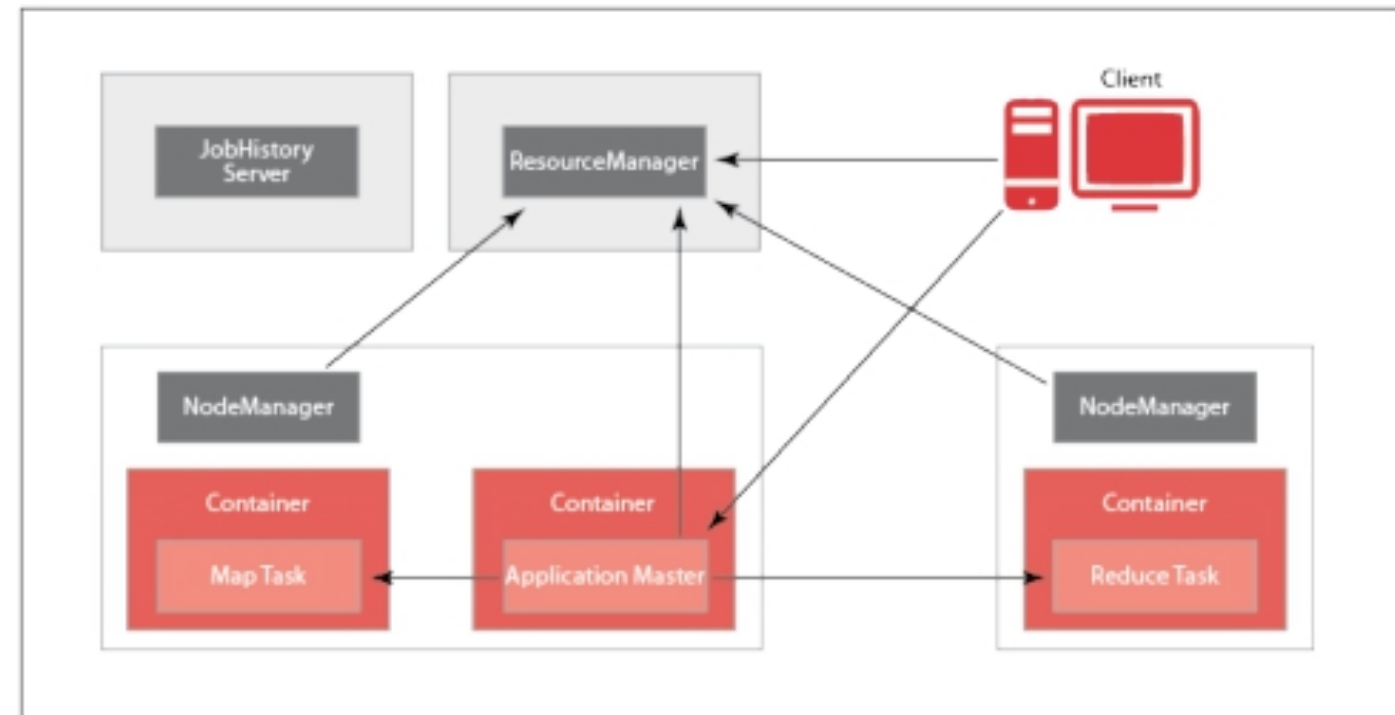
YARN supports multiple processing models in addition to MapReduce.
All share common resource management service.
No need to ship out data!



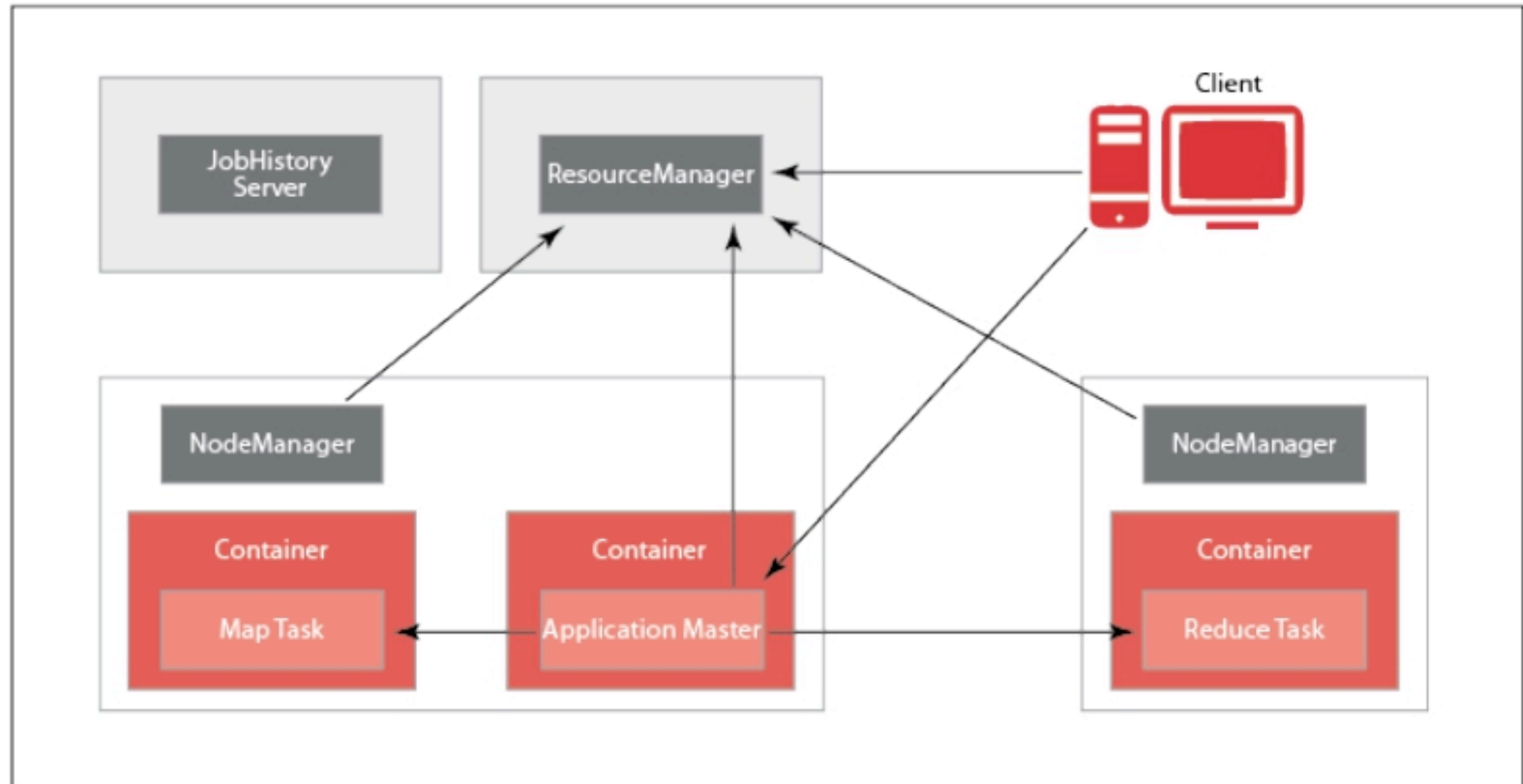
Resource Manager (RM) - serves as the central agent for managing and allocating cluster resources

Node Manager (NM) - per node agent that manages and enforces node resources

Application Master (AM) - per application manager that manages lifecycle and task scheduling



Hadoop 2 on YARN



Hadoop 2 on YARN

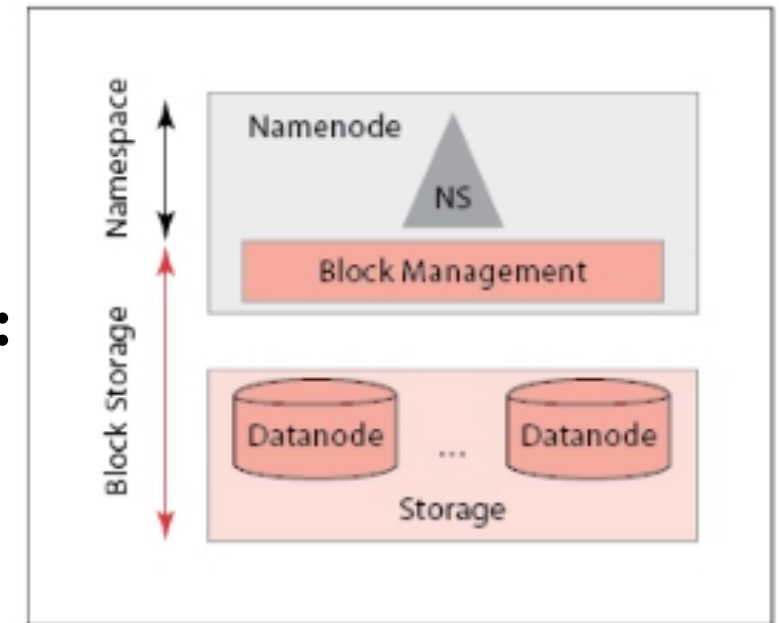
Hadoop 1: single namespace for the entire cluster managed by a single NameNode.

Hadoop 2 HDFS Federation adds support for multiple NameNodes/namespaces and horizontal scalability of the name service.

Each namespace is independently operating in the same cluster.

Standby NameNode can be configured to automatically take over if the active NameNode fails.

Hadoop 1:



Hadoop 2:



Hadoop 2 on YARN

Although slightly more complex, Hadoop 2 on YARN allows for some very important features:

Multi-tenancy: Other computation frameworks can be run on the Hadoop cluster, simultaneously.

Single job queue for multiple job submissions and standardized resource management

Improved availability and prevention of job failure

(NameNode redundancy, separation of resource, job, and task management)

HDFS Federation: Multiple namespace support for better isolation in a multi-user and multi-tenant environment

MapReduce in Detail

Although slightly more complex - this application framework allows for some very important features:

- Multiple MapReduce jobs may be run simultaneously
- Other computation frameworks for Graph computation
- and SQL-like queries can be run without MapReduce
- implementations
- Job failure is prevented by using speculative execution
- and redundant tasks for slow running processes
- Processes run on local data to reduce network traffic.

At this point, we will discuss the specifics of implementing MapReduce in a distributed fashion, which is more or less the same for both Hadoop 1 and 2.

HDFS is a distributed filesystem that is based on the GFS paper from 2003.

- Provides redundant storage of extremely large datasets
- Can be linearly expanded to increase storage space
- It's a software layer on top of a native filesystem like ext4
 - swimming pool analogy*

Files are split into HDFS block size chunks (default 64MB) and replicated across the cluster (default of 3x).

- At 3X, a 1TB file needs 3TB of storage

HDFS performs best with a modest number of large files

- millions of 100MB files is better than billions of 1MB files

The number of splits of a file determines the number of map jobs

- There is a 1 to 1 relationship between a task and a block of data

File permissions are handled similarly to POSIX filesystems



What is Amazon's Elastic MapReduce Service?

It is a product/service from Amazon Web Services that allows users to start a set of VMs configured as a cluster running Hadoop and other tools.

It is a managed service, and it automatically installs the software, which has been optimized to run on Amazon's VMs, and configures the cluster.

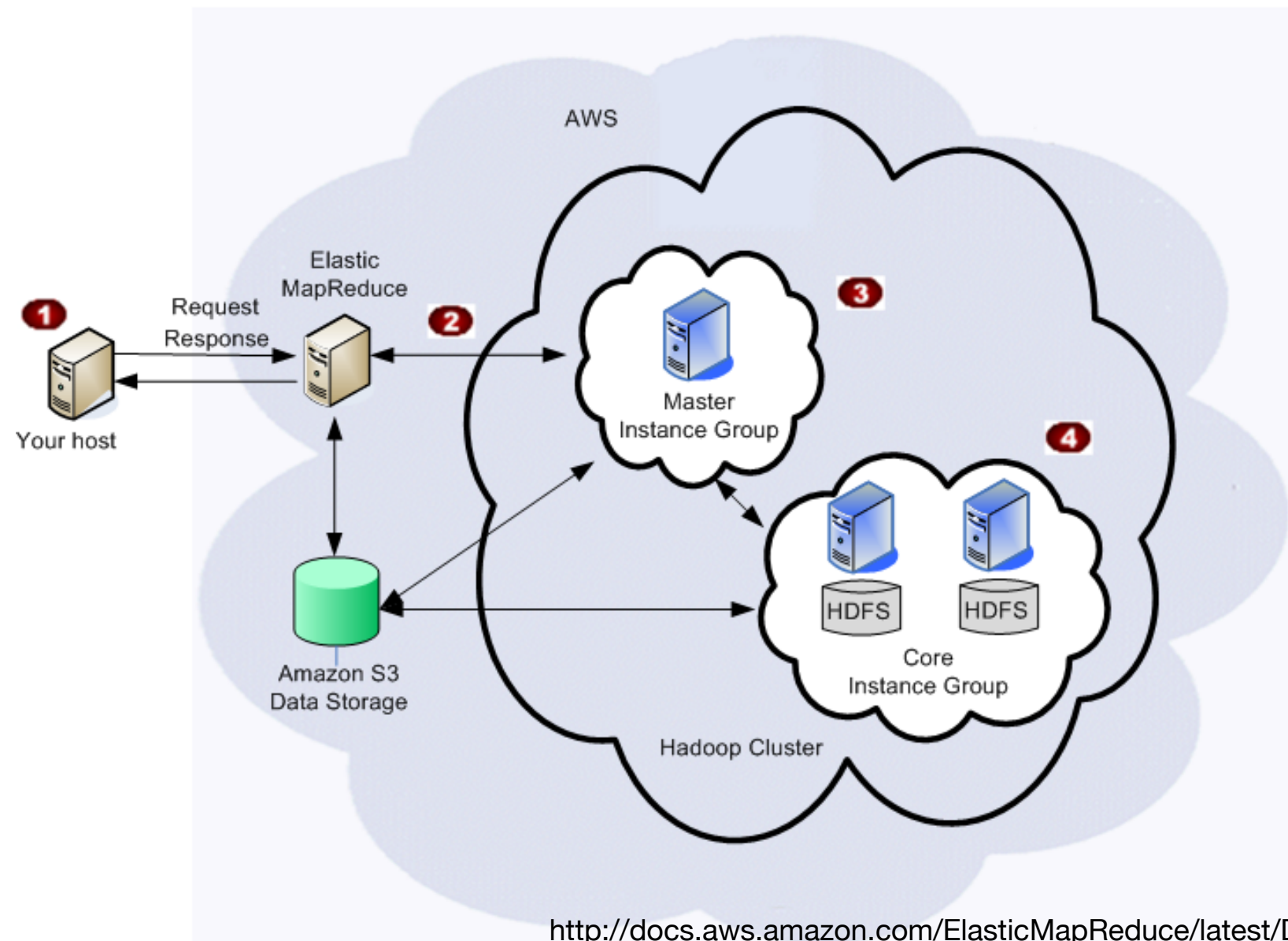
- Installs and configures Hadoop and additional tools as needed
- It's elastic (scalable)
- Connects your cluster to your S3 bucket

Three types of nodes:

- Master: acts as gateway node, runs NameNode and other services, stores data in HDFS
- Core: runs TaskNode and stores data in HDFS
- Task: runs TaskNode only

Data can be read/written to both the cluster's HDFS and/or S3

Elastic Map Reduce: Amazon's Managed Hadoop Cluster



Elastic Map Reduce: Amazon's managed Hadoop cluster.

Hourly rate for every instance hour you use.

- 10-node cluster for 10 hours = 100-node cluster for 1 hour
- Higher prices for more powerful instances.
- Instance prices \$0.11/hour to \$0.27/hour
- EMR price is *in addition* to EC2 price.

US East is generally cheapest.

- Even cheaper with spot instances.
- Hadoop & Spark don't run on "Previous Generation" small & medium instances.

	Amazon EC2 Price	Amazon Elastic MapReduce Price
General Purpose - Current Generation		
m3.xlarge	\$0.266 per Hour	\$0.070 per Hour
m3.2xlarge	\$0.532 per Hour	\$0.140 per Hour
General Purpose - Previous Generation		
m1.small	\$0.044 per Hour	\$0.011 per Hour
m1.medium	\$0.087 per Hour	\$0.022 per Hour
m1.large	\$0.175 per Hour	\$0.044 per Hour
m1.xlarge	\$0.350 per Hour	\$0.088 per Hour
Compute Optimized - Current Generation		
c3.xlarge	\$0.210 per Hour	\$0.053 per Hour
c3.2xlarge	\$0.420 per Hour	\$0.105 per Hour
c3.4xlarge	\$0.840 per Hour	\$0.210 per Hour
c3.8xlarge	\$1.680 per Hour	\$0.270 per Hour

Options when creating EMR clusters

Note:

- Logging — S3 bucket
- Vendors:
 - *Amazon & MapR*
- Instances:
 - *Instance Type*
 - *# of Instances*
- Security
 - *EC2 key pair.*

AWS Elastic MapReduce M: x

https://console.aws.amazon.com/elasticmapreduce/home?region=us-east-1#quick-create:

Apps VA AWS Services Edit

Simson L. Garfinkel N. Virginia Support

Elastic MapReduce Create Cluster EMR Help

Create Cluster - Quick Options [Go to advanced options](#)

General Configuration

Cluster name

☒ Logging [?](#)

S3 folder

Launch mode ☒ Cluster [?](#) ☐ Step execution [?](#)

Software configuration

Vendor ☒ Amazon ☐ MapR

Release [?](#)

Applications ☒ All Applications: Ganglia 3.6.0, Hadoop 2.6.0, Hive 1.0.0, Hue 3.7.1, Mahout 0.11.0, Pig 0.14.0, and Spark 1.5.2

☐ Core Hadoop: Hadoop 2.6.0 with Ganglia 3.6.0, Hive 1.0.0, and Pig 0.14.0

☐ Presto-Sandbox: Presto 0.125 with Hadoop 2.6.0 HDFS and Hive 1.0.0 Metastore

☐ Spark: Spark 1.5.2 on Hadoop 2.6.0 YARN with Ganglia 3.6.0

Hardware configuration

Instance type

Number of instances (1 master and 2 core nodes)

Security and access

EC2 key pair [?](#) [Learn how to create an EC2 key pair.](#)

Permissions ☒ Default ☐ Custom

Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.

EMR role [EMR_DefaultRole](#) [?](#)

EC2 instance profile [EMR_EC2_DefaultRole](#) [?](#)

But for this course,
please use the
“Advanced Options”

AWS Elastic MapReduce M: x

https://console.aws.amazon.com/elasticmapreduce/home?region=us-east-1#create-cluster:

AWS Services Edit

Simson L. Garfinkel N. Virginia Support

Elastic MapReduce Create Cluster EMR Help

Create Cluster - Advanced Options [Go to quick options](#)

Step 1: Software and Steps

Step 2: Hardware

Step 3: General Cluster Settings

Step 4: Security

Software Configuration

Vendor ☒ Amazon ☐ MapR

Release

<input checked="" type="checkbox"/> Hadoop 2.6.0	<input checked="" type="checkbox"/> Hive 1.0.0	<input type="checkbox"/> Mahout 0.11.0
<input type="checkbox"/> Zeppelin-Sandbox 0.5.5	<input checked="" type="checkbox"/> Hue 3.7.1	<input type="checkbox"/> Spark 1.5.2
<input type="checkbox"/> Ganglia 3.6.0	<input type="checkbox"/> Presto-Sandbox 0.125	<input type="checkbox"/> Oozie-Sandbox 4.2.0
<input checked="" type="checkbox"/> Pig 0.14.0		

Edit software settings (optional) ⓘ

☒ Enter configuration ☐ Load JSON from S3

`classification=config-file-name,properties=[myKey1=myValue1,myKey2=myValue2]`

Add steps (optional) ⓘ

Step type [Configure](#)

☐ Auto-terminate cluster after the last step is completed

[Cancel](#) [Next](#)

[Feedback](#) [English](#)

© 2008 - 2015, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) [Terms of Use](#)

Core: Compute & HDFS nodes

If you shrink “data to the re

only

redistribute your

Step 1: Software and Steps

Step 2: Hardware

Step 3: General Cluster Settings

Step 4: Security

Hardware Configuration

Network: Launch into EC2-Classical

EC2 availability zone: No preference

Type	Name	EC2 instance type
Master	Master instance group - 1	m3.xlarge
Core	Core instance group - 2	m3.xlarge
Task	Task instance group - 3	m3.xlarge

Add task instance group

Instance types list:

- General Purpose
 - m3.xlarge
 - m3.2xlarge
- Compute Optimized
 - c3.xlarge
 - c3.2xlarge
 - c3.4xlarge
 - c3.8xlarge
- GPU Instances
 - g2.2xlarge
- Memory Optimized
 - r3.xlarge
 - r3.2xlarge
 - r3.4xlarge
 - r3.8xlarge
- Storage Optimized
 - i2.xlarge
 - i2.2xlarge
 - i2.4xlarge
 - i2.8xlarge
 - hs1.8xlarge
 - d2.xlarge
 - d2.2xlarge
 - d2.4xlarge
 - d2.8xlarge
- General Purpose (Previous Generation)
 - m1.medium
 - m1.large

Buttons: Cancel, Previous, Next

AWS Elastic MapReduce M: x

https://console.aws.amazon.com/elasticmapreduce/home?region=us-east-1#create-cluster:ABP ⓘ b 1

AWS Services Edit

Simson L. Garfinkel N. Virginia Support

Simson

Elastic MapReduce Create Cluster

EMR Help

Create Cluster - Advanced OptionsGo to quick options

Step 1: Software and Steps

Step 2: Hardware

Step 3: General Cluster Settings

Step 4: Security

General Options

Cluster nameMy cluster

☒ Logging ⓘ
S3 folder s3://aws-logs-376778049323-us-east-1/elasticmapreduce/

☒ Debugging ⓘ

☒ Termination protection ⓘ

Tags ⓘ

Key	Value (optional)
Add a key to create a tag	

Additional Options

☐ EMRFS consistent view ⓘ

Bootstrap Actions

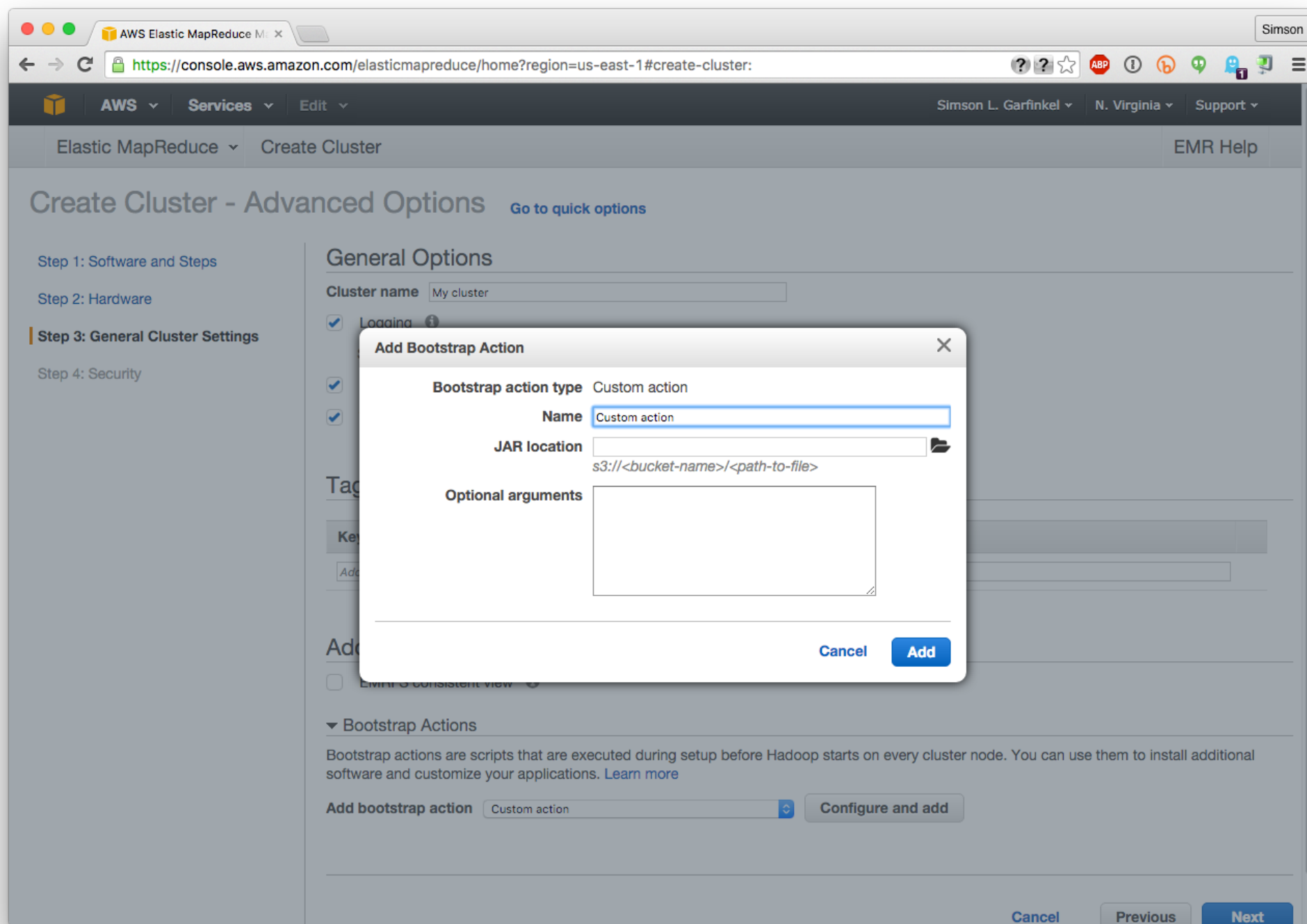
Cancel

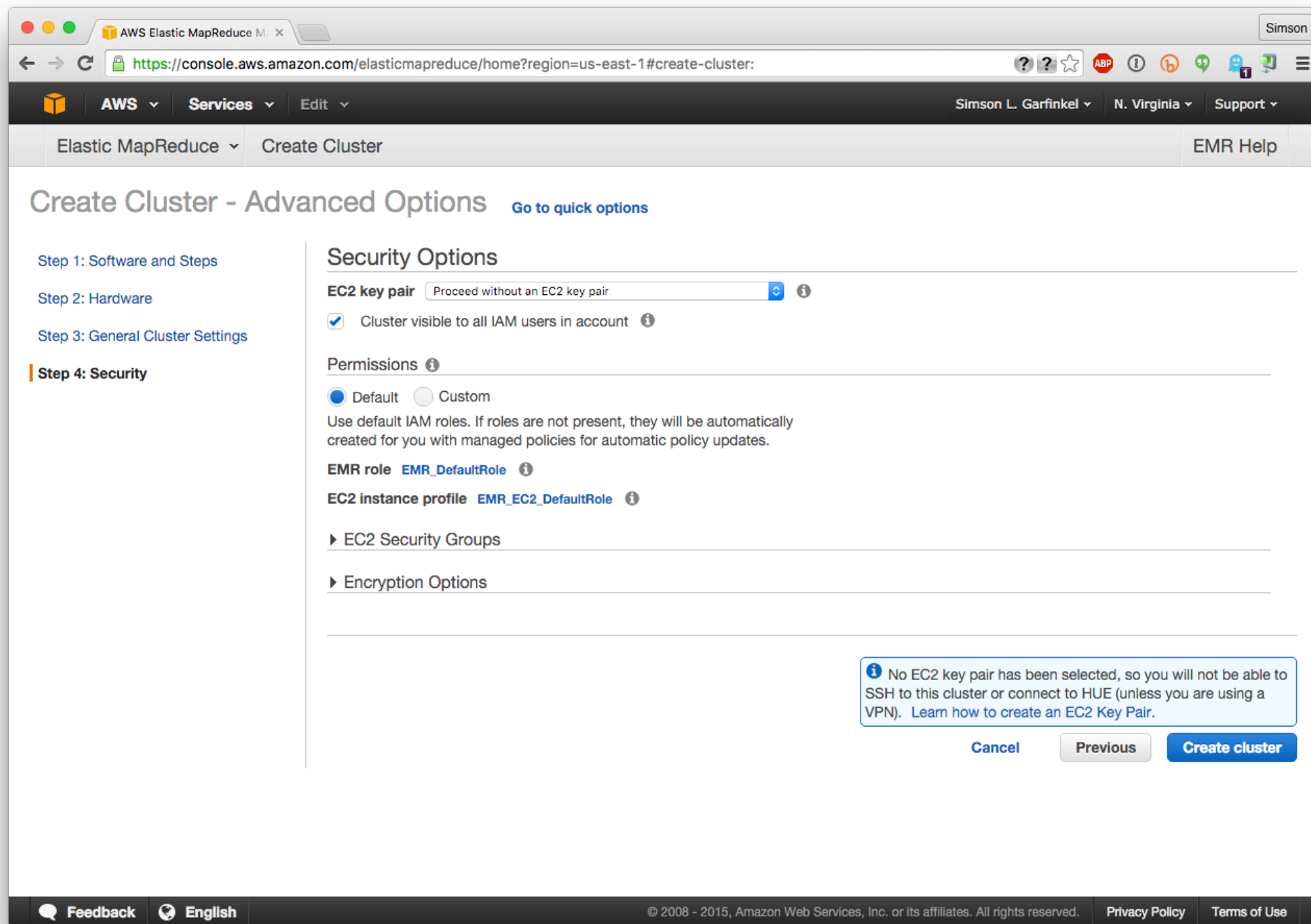
Previous

Next

FeedbackEnglish

© 2008 - 2015, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy PolicyTerms of Use





Connect with SSH tunneling and FoxyProxy

The screenshot shows the AWS Elastic MapReduce Management Console. A modal window titled "Enable Web Connection" is open, providing instructions on how to access the EMR master node's web interfaces. The window is divided into two main sections: "Step 1: Open an SSH Tunnel to the Amazon EMR Master Node" and "Step 2: Configure a proxy management tool".

Step 1: Open an SSH Tunnel to the Amazon EMR Master Node - [Learn more](#)

Windows | Mac / Linux

1. Open a terminal window. On Mac OS X, choose Applications > Utilities > Terminal. On other Linux distributions, terminal is typically found at Applications > Accessories > Terminal.
2. To establish an SSH tunnel with the master node using dynamic port forwarding, type the following command. Replace ~/muchapem with the location and filename of the private key file (.pem) used to launch the cluster.

```
ssh -i ~/muchapem -ND 8157 hadoop@ec2-54-163-63-17.compute-1.amazonaws.com
```

Note: Port 8157 used in the command is a randomly selected, unused local port.

3. Type yes to dismiss the security warning.

Step 2: Configure a proxy management tool - [Learn more](#)

Chrome | Firefox

1. Download and install the standard version of FoxyProxy from: <http://foxyproxy.mozdev.org/downloads.html>
2. Restart Chrome after installing FoxyProxy.
3. Using a text editor create a file named foxyproxy-settings.xml containing the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<foxyproxy>
  <proxies>
    <proxy name="emr-socks-proxy" id="2322596116" notes="" fromSubscription="false" enabled="true"
mode="manual" selectedTabIndex="2" lastresort="false" animatedIcons="true" includeInCycle="true" color="#0055E5"
proxyDNS="true" noInternalIPs="false" autoconfMode="pac" clearCacheBeforeUse="false" disableCache="false"
clearCookiesBeforeUse="false" rejectCookies="false">
      <matches>
        <match enabled="true" name="*ec2*.amazonaws.com*" pattern="*ec2*.amazonaws.com*" isRegex="false"
isBlackList="false" isMultiLine="false" caseSensitive="false" fromSubscription="false" />
        <match enabled="true" name="*ec2*.compute*" pattern="*ec2*.compute*" isRegex="false"
isBlackList="false" isMultiLine="false" caseSensitive="false" fromSubscription="false" />
        <match enabled="true" name="10.*" pattern="http://10.*" isRegex="false" isBlackList="false"
isMultiLine="false" caseSensitive="false" fromSubscription="false" />
        <match enabled="true" name="*10*.amazonaws.com*" pattern="*10*.amazonaws.com*" isRegex="false"
isBlackList="false" isMultiLine="false" caseSensitive="false" fromSubscription="false" />
        <match enabled="true" name="*10*.compute*" pattern="*10*.compute*" isRegex="false"
isBlackList="false" isMultiLine="false" caseSensitive="false" fromSubscription="false" />
        <match enabled="true" name="*.compute.internal*" pattern="*.compute.internal*" isRegex="false"
isBlackList="false" isMultiLine="false" caseSensitive="false" fromSubscription="false" />
        <match enabled="true" name="*.ec2.internal*" pattern="*.ec2.internal*" isRegex="false" />
      </matches>
    </proxy>
  </proxies>
</foxyproxy>
```

Close

Clusters remain after termination so you can “clone” them.

Be sure to terminate in the EMR, not EC2 panel.

Remember: you lose your disk and HDFS when you terminate!

- Store things in S3 and git!

Elastic MapReduce ▾		Cluster List					EMR Help
Create cluster		View details		Clone	Terminate		
Filter:		All clusters ▾	Filter clusters ...	4 clusters (all loaded)		C	
		Name	ID	Status	Creation time (UTC-5) ▾	Elapsed time	Normalized instance hours
<input type="checkbox"/>	▶	My cluster	j-2K7FT5K87H41E	Terminated User request	2015-11-09 21:43 (UTC-5)	27 minutes	12
<input type="checkbox"/>	▶	My cluster	j-1CVFQRI4UZCWA	Terminated User request	2015-11-09 21:28 (UTC-5)	14 minutes	24
<input checked="" type="checkbox"/>	▶	Cluster2	j-2HP7YEOU8UB6X	Terminated with errors Bootstrap failure	2015-11-09 21:01 (UTC-5)	18 minutes	0
<input checked="" type="checkbox"/>	▶	My cluster	j-XTXAL12XZHV7	Terminated with errors Instance failure	2015-10-31 22:39 (UTC-5)	43 minutes	8

Lab # 1

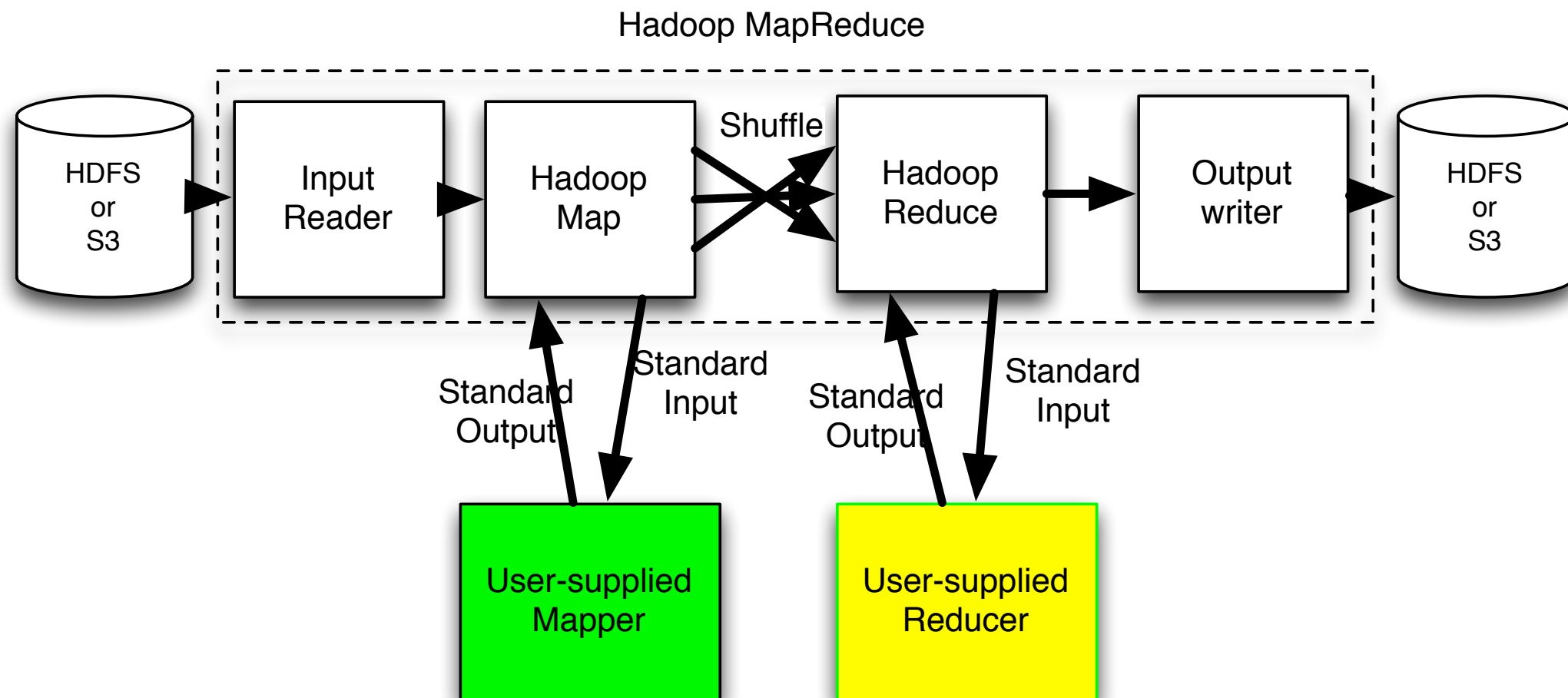
- Start an Amazon EMR Cluster
 - *1 Master, 2 Core*
- Ssh into the Master Node
 - *Make sure you use `ssh -A` so you can clone your git repository*
 - *Make sure that the security groups are setup correctly (open port 22) so you can ssh*
 - *`ssh -A hadoop@cluster-ip-address`*
- Install git
 - *`sudo yum install -y git`*
- Go to AWS Console, select your cluster
 - *Follow instructions related to FoxyProxy - install, create `foxyproxy-settings.xml` file and configure FP for your browser*
- List files in cluster's HDFS (will be empty)
 - *`hdfs dfs -ls`*
 - *`hadoop fs -ls`*
- List files in course's S3 repository
 - *`hdfs dfs -ls s3://gu-anly502/`*
- Explore the HDFS Filesystem Commands:
 - *<https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/FileSystemShell.html>*

Hadoop Streaming

Hadoop Streaming

Hadoop streaming — reads from stdin & writes to stdout.

- Allows using Hadoop MapReduce with any language.
- Performance penalty — all I/O has to go over pipes.



What is stdin, stdout?

[https://en.wikipedia.org/wiki/Standard_streams#Standard_input_\(stdin\)](https://en.wikipedia.org/wiki/Standard_streams#Standard_input_(stdin))

Use Python programs as a mapper:

```
#!/usr/bin/env python

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print("%s\t%s" % (word, 1))
```

Use Python program as reducer:

```
#!/usr/bin/env python

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print("%s\t%s" % (current_word, current_count))
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print("%s\t%s" % (current_word, current_count))
```

You can independently test the mapper and reducer:

Mapper:

```
[myprompt]$ echo "one two two three three three" | ./streaming_mapper.py
one1
two1
two1
three      1
three      1
three      1
[myprompt]$
```

Mapper and Reducer:

```
[myprompt]$ cat infile.txt
one two two three three three
[myprompt]$ cat infile.txt | ./streaming_mapper.py | sort | ./streaming_reducer.py
one1
three      3
two2
[myprompt]$
```

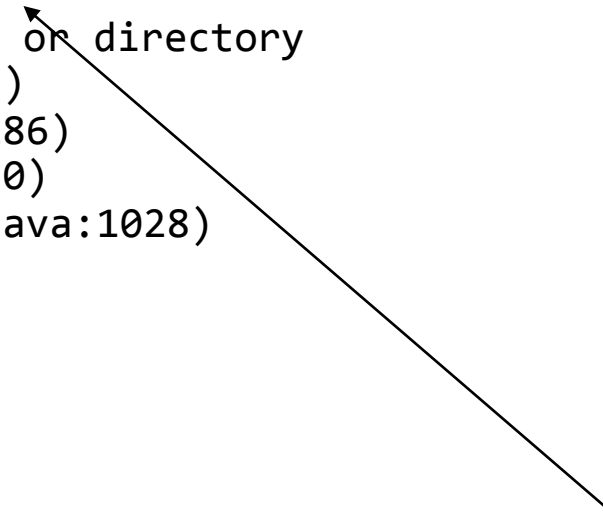
Hadoop Streaming Example

Simple Hadoop Streaming command on Amazon EMR:

```
$ hadoop jar /usr/lib/hadoop/hadoop-streaming-2.7.3-amzn-1.jar -files q2_mapper.py,q2_reducer.py -input  
s3://gu-anly502/A2/sonnet18.txt -output output-2017-01-22T00-06-42 -mapper q2_mapper.py -reducer  
q2_reducer.py
```


if you see this:

```
Caused by: java.lang.reflect.InvocationTargetException
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:606)
    at org.apache.hadoop.util.ReflectionUtils.setJobConf(ReflectionUtils.java:106)
    ... 17 more
Caused by: java.lang.RuntimeException: configuration exception
    at org.apache.hadoop.streaming.PipeMapRed.configure(PipeMapRed.java:221)
    at org.apache.hadoop.streaming.PipeMapper.configure(PipeMapper.java:66)
    ... 22 more
Caused by: java.io.IOException: Cannot run program "streaming_mapper.py": error=2, No such file or
directory
    at java.lang.ProcessBuilder.start(ProcessBuilder.java:1047)
    at org.apache.hadoop.streaming.PipeMapRed.configure(PipeMapRed.java:208)
    ... 23 more
Caused by: java.io.IOException: error=2, No such file or directory
    at java.lang.UNIXProcess.forkAndExec(Native Method)
    at java.lang.UNIXProcess.<init>(UNIXProcess.java:186)
    at java.lang.ProcessImpl.start(ProcessImpl.java:130)
    at java.lang.ProcessBuilder.start(ProcessBuilder.java:1028)
    ... 24 more
```



Look for errors that you can understand

Hadoop Streaming Couldn't find the executable

Caused by: java.io.IOException: Cannot run program "streaming_mapper.py": error=2, No such file or directory

Streaming tries to run the programs from the your home directory!

- streaming_mapper.py and streaming_reducer.py were in the *current* directory
- You need to specify the complete path name.
- ``pwd`` evaluates to the current directory

Revised Simple Hadoop Streaming command on CDH Quickstart VM:

```
$ hadoop jar /usr/lib/hadoop/hadoop-streaming-2.7.3-amzn-1.jar \  
-input data -output data2 \  
-mapper `pwd`/streaming_mapper.py -reducer `pwd`/streaming_reducer.py
```

You can specify more complex things

Sample command from tutorial:

```
$ hadoop jar /usr/lib/hadoop/hadoop-streaming-2.7.3-amzn-1.jar \  
-D mapreduce.map.output.key.field.separator=. \  
-D mapreduce.partition.keypartitioner.options=-k1,2 \  
-D mapreduce.fieldsel.data.field.separator=. \  
-D mapreduce.fieldsel.map.output.key.value.fields.spec=6,5,1-3:0- \  
-D mapreduce.fieldsel.reduce.output.key.value.fields.spec=0-2:5- \  
-D mapreduce.map.output.key.class=org.apache.hadoop.io.Text \  
-D mapreduce.job.reduces=12 \  
-input myInputDirs \  
-output myOutputDir \  
-mapper org.apache.hadoop.mapred.lib.FieldSelectionMapReduce \  
-reducer org.apache.hadoop.mapred.lib.FieldSelectionMapReduce \  
-partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner
```

Specifies:

- mapper, reducer & partitioner*
- Different field separators*
- Output formats.*
- etc.*

Good tutorials on Hadoop Streaming: streaming

- <http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/>

Hadoop Gotchas

- Output directory (S3 or HDFS) must not exist before running a Hadoop job
- Keep track of which filesystem you are accessing
 - *Linux filesystem*
 - *HDFS*
 - *S3*
- Hadoop/Java errors are cryptic. Many times you need to look at the stdout/stderr logs

Lab # 2

- ssh into cluster using port forwarding to be able to use the EMR Cluster Hadoop Web UI. You need two terminals. Also, enable the FoxyProxy settings
 - ssh -A hadoop@my-ip-address
 - ssh -ND 8157 hadoop@my-ip-address
- git clone the class Bitbucket repo or your fork (which you must update before you clone)
- Change directory to A2
- "Get" text file from S3 to A2 directory
 - hadoop fs -get <s3://gu-anly502/gutenberg/Meyers.txt>
- Explore this text file using head, tail, less, more, etc.
- "Test" the hadoop streaming framework
 - cat Meyers.txt | ./q2_mapper.py
 - cat Meyers.txt | ./q2_mapper.py | sort | ./q2_reducer.py
- Run Hadoop Streaming job on this file, reading from S3, and writing to S3
 - hadoop jar /usr/lib/hadoop/hadoop-streaming-2.7.3-amzn-1.jar -files q2_mapper.py,q2_reducer.py -input s3://gu-anly502/gutenberg/Meyers.txt -output s3://[mybucket]/my-first-streaming-job -mapper q2_mapper.py -reducer q2_reducer.py
- "Put" Meyers.txt into EMR Cluster HDFS and run Hadoop Streaming Job, reading and writing from HDFS
 - hadoop fs -mkdir my-source-data-on-hdfs
 - hadoop fs -put Meyers.txt my-source-data-on-hdfs/
 - hadoop jar /usr/lib/hadoop/hadoop-streaming-2.7.3-amzn-1.jar -files q2_mapper.py,q2_reducer.py -input my-source-data-on-hdfs/Meyers.txt -output my-first-output-on-hdfs -mapper q2_mapper.py -reducer q2_reducer.py
- ***DON'T FORGET TO STOP YOUR CLUSTER AT THE END OF CLASS***

Coming Up

A02 - Due Friday 2/3

Q2- Due Friday 1/27

L03 - Mon 1/30: MapReduce Patte

Parking Lot