MapReduce: Simplified Data Processing on Large Clusters

J. Dean, S. Ghemawat
Google, Inc

Presented by: Luna Xu

MapReduce -- Key Contribution

- A programming model for processing large data sets
  - *Map* and *reduce* operations on key/value pairs
- An interface addresses details:
  - Parallelization
  - Fault-tolerance
  - Data distribution
  - Load balancing
- An implementation of the interface
  - Achieve high performance on large clusters of commodity PCs

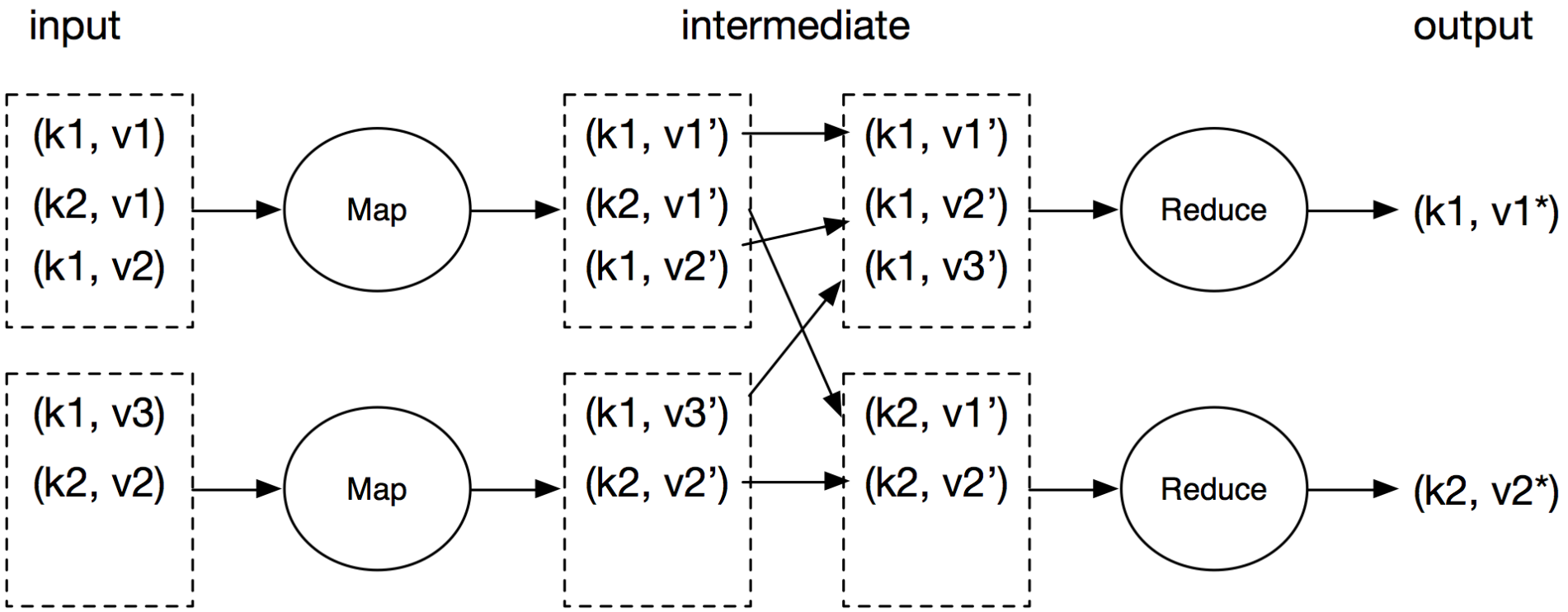
Overview

- Programming Model
- Execution Overview
- Fault Tolerance
- Locality
- Improvements
- Evaluation
- Hadoop

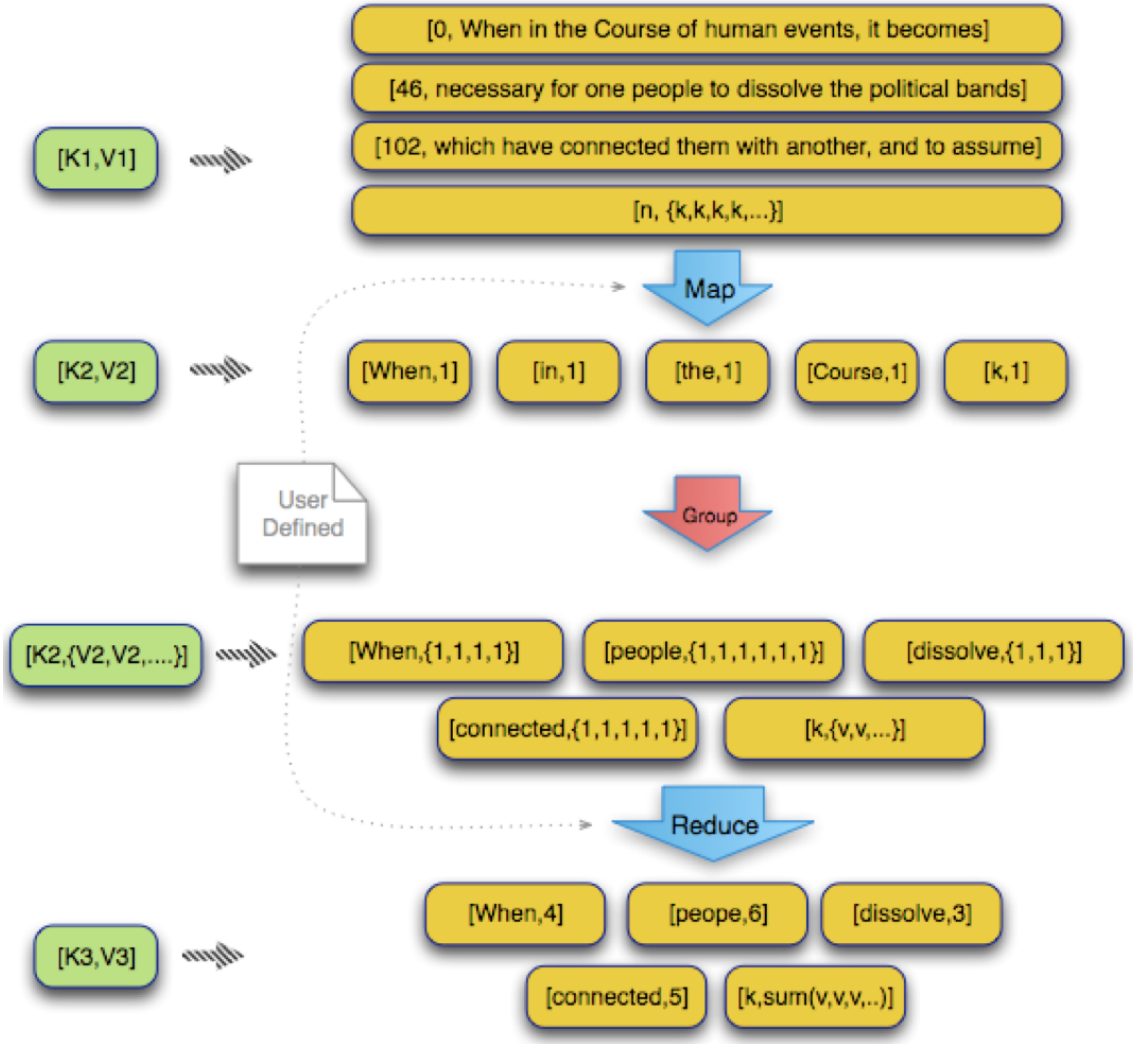
Overview

- Programming Model
- Execution Overview
- Fault Tolerance
- Locality
- Improvements
- Evaluation
- Hadoop

Programming Model



Word Count


[0, When in the Course of human events, it becomes]
[46, necessary for one people to dissolve the political bands]
[102, which have connected them with another, and to assume]
[n, {k,k,k,k,...}]
[K1,V1]
Map
[K2,V2]
[When,1]
[in,1]
[the,1]
[Course,1]
[k,1]
User Defined
Group
[K2,{V2,V2,....}]
[When,{1,1,1,1}]
[people,{1,1,1,1,1,1}]
[dissolve,{1,1,1}]
[connected,{1,1,1,1,1}]
[k,{v,v,...}]
Reduce
[When,4]
[peope,6]
[dissolve,3]
[K3,V3]
[connected,5]
[k,sum(v,v,v,..)]

Word Count (cont.)

```
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));
```

Examples

- Word count
- Distributed Sort
- Page rank
- Indexing
- K-means clustering
- Bayesian classification
- ….

Overview

- Programming Model
- Execution Overview (Parallelization)
- Fault Tolerance
- Locality
- Improvements
- Evaluation
- Hadoop

Framework

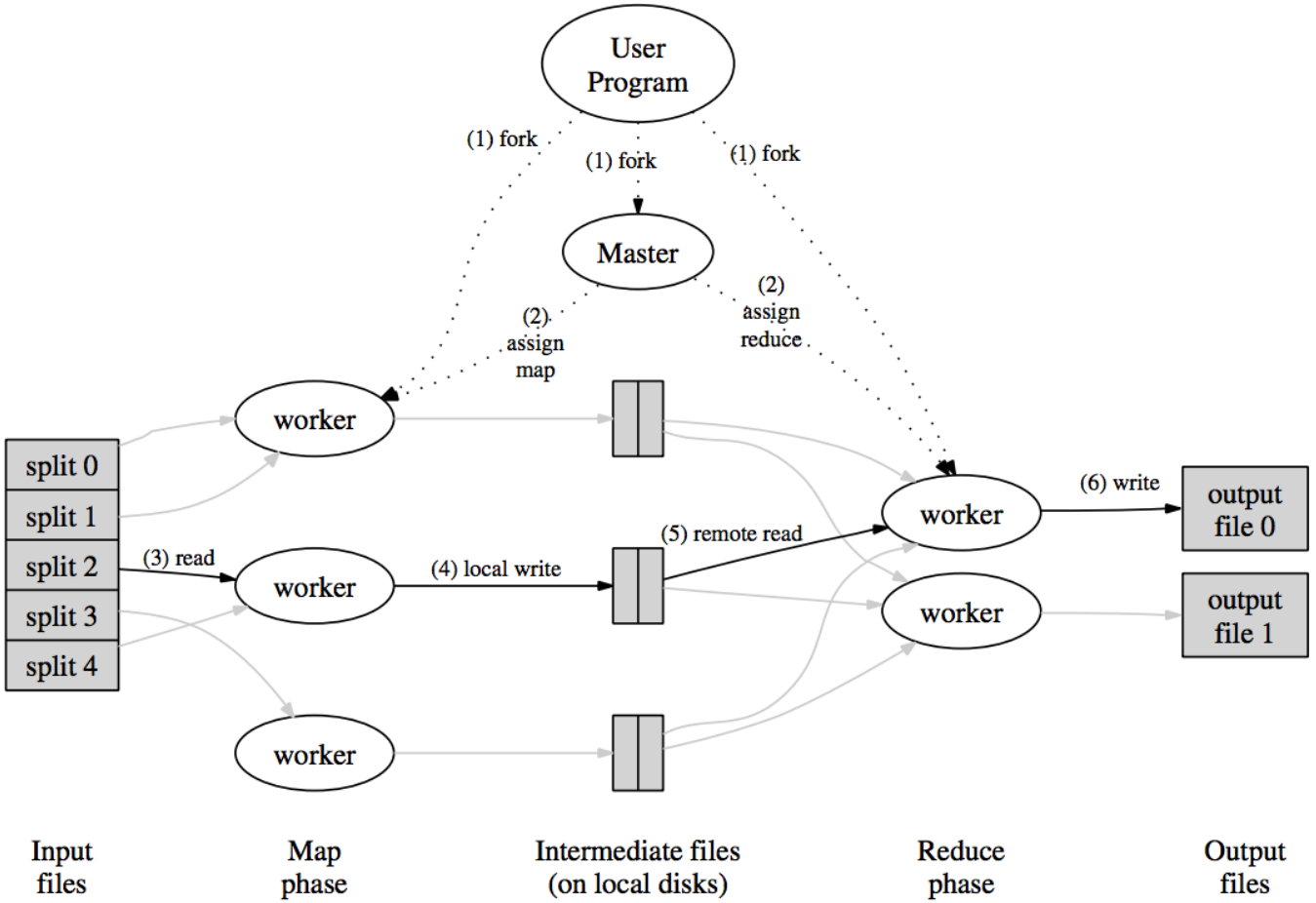
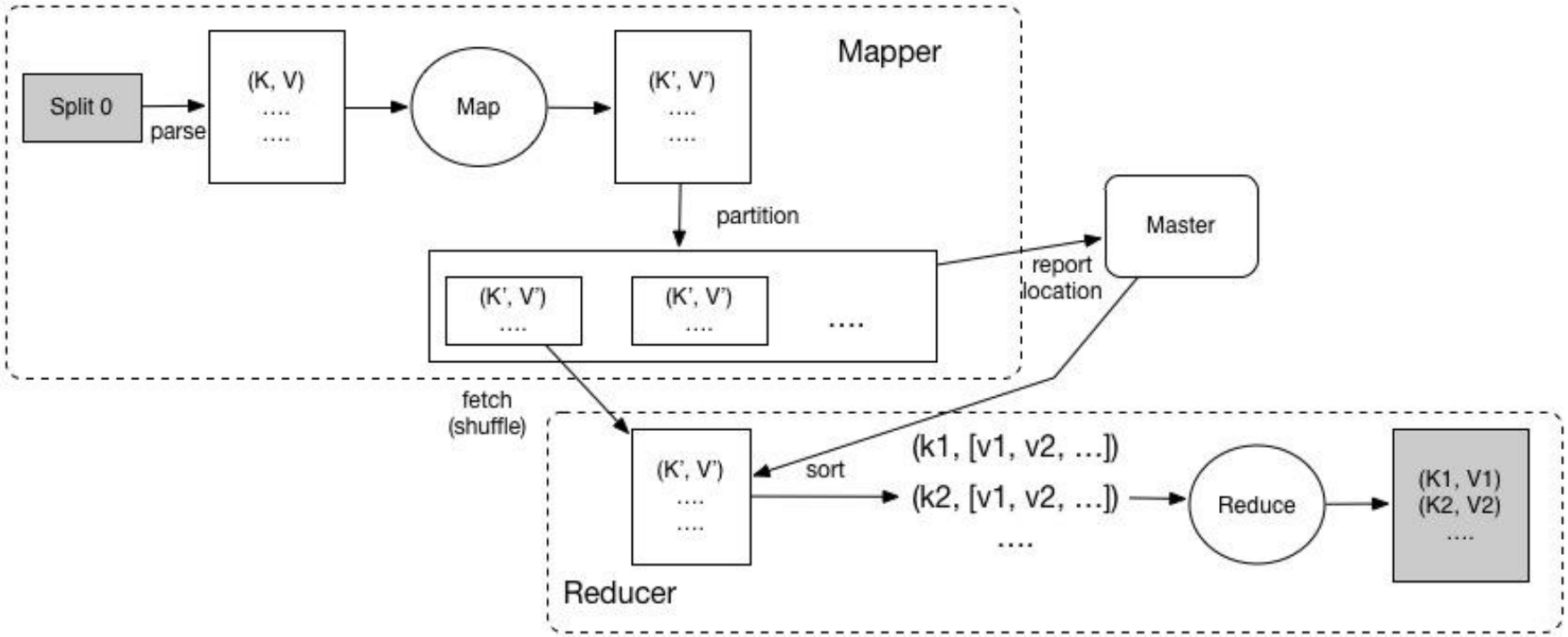


Figure 1: Execution overview

M: no. of mappers R: no. of reducers

Execution Flow



Overview

- Programming Model
- Execution Overview
- Fault Tolerance
- Locality
- Improvements
- Evaluation
- Hadoop

Worker Failure

- Define: heartbeat, timeout
- Handle:
  - Map tasks/in-progress reduce tasks reset to *idle* for re-scheduling
  - Map tasks are re-executed
  - Notifications are sent to all reduce tasks to redirect the file location

Flexible and Resilient to large-scale worker failures.

Master Failure

- Checkpointing of the maintained data structures
- Restart from the checkpointed state
- Abort whole computation

In Hadoop:

- Master (JobTracker) high availability configuration along with Zookeeper

Overview

- Programming Model
- Execution Overview
- Fault Tolerance
- Locality (Data Distribution)
- Improvements
- Evaluation
- Hadoop

Locality

- Bring codes to where data locates
- Avoid network traffic
- Optimal when M is chosen by the size of a data chunk

Overview

- Programming Model
- Execution Overview
- Fault Tolerance
- Locality (Data Distribution)
- Improvements
- Evaluation
- Hadoop

Backup Tasks

- Problem: “straggler” tasks lengthens the total job time
- Solution: master schedules backup executions of the remaining in progress tasks

Skipping Bad Records

- Problem: Bugs in user code that cause functions to crash deterministically on certain records
- Solution: If more than one failure is seen by master, it skips the record in the next re-execution

Combiner Function

- Problem: mapper generates the same intermediate key when reducers are commutative and associative, same data will be sent multiple times through network
- Solution: user can specify *Combiner* function to merge data inside a mapper before sent

Overview

- Programming Model
- Execution Overview
- Fault Tolerance
- Locality (Data Distribution)
- Improvements
- Evaluation
- Hadoop

Experiment Settings

- Applications:
  - Grep on one terabyte of data
    - Chunk size = 64MB; M=15,000; R=1
  - Sort on one terabyte of data
    - Chunk size = 64MB; M=15,000; R=4,000
- Cluster:
  - 1800 machines: 2GHz Intel Xeon, 4GB memory, two 160GB IDE disks, 1 gigabit Ethernet link

Grep

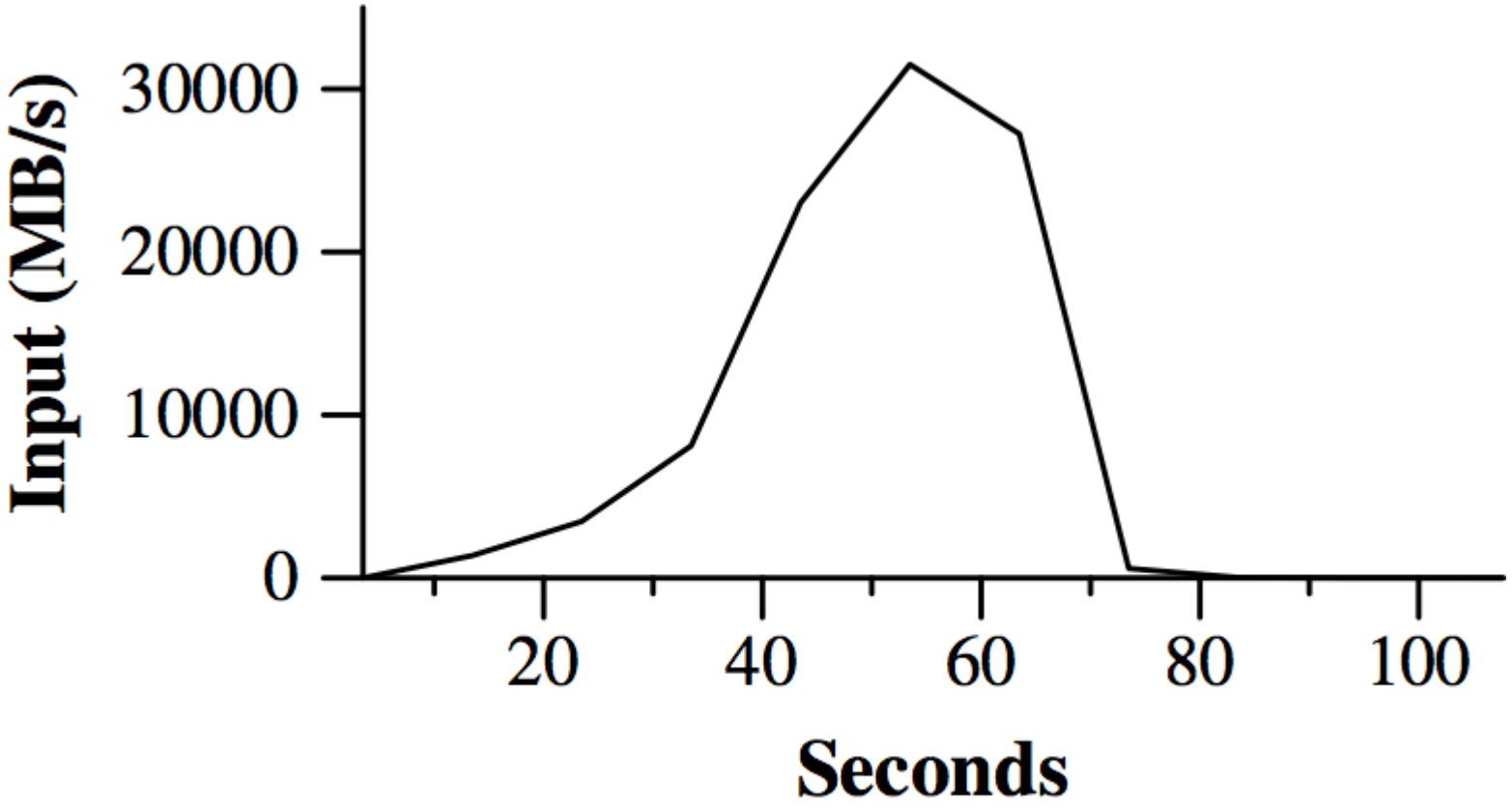
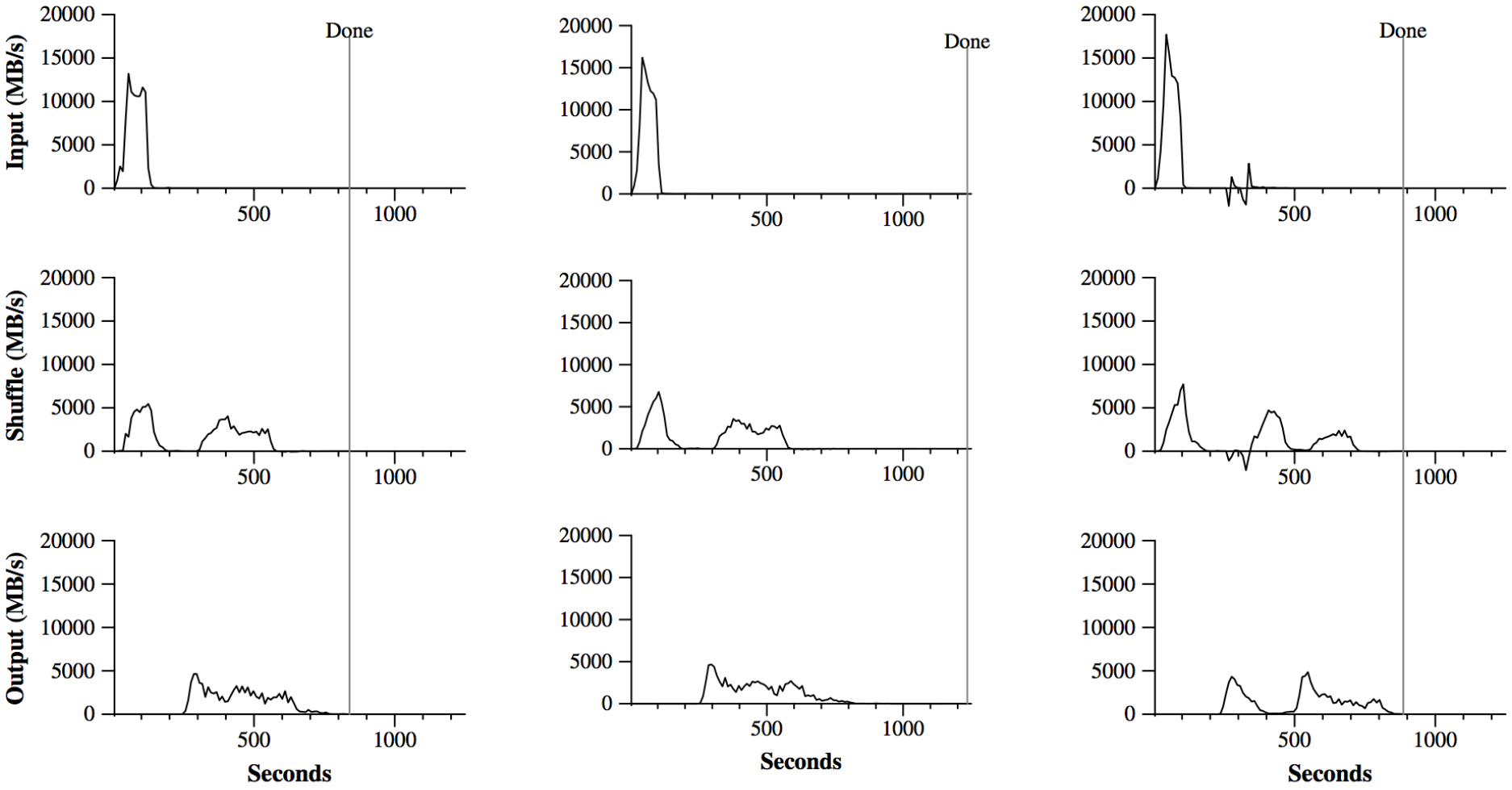


Figure 2: Data transfer rate over time

Sort



(a) Normal execution

(b) No backup tasks

(c) 200 tasks killed

Figure 3: Data transfer rates over time for different executions of the sort program

Experience in Google

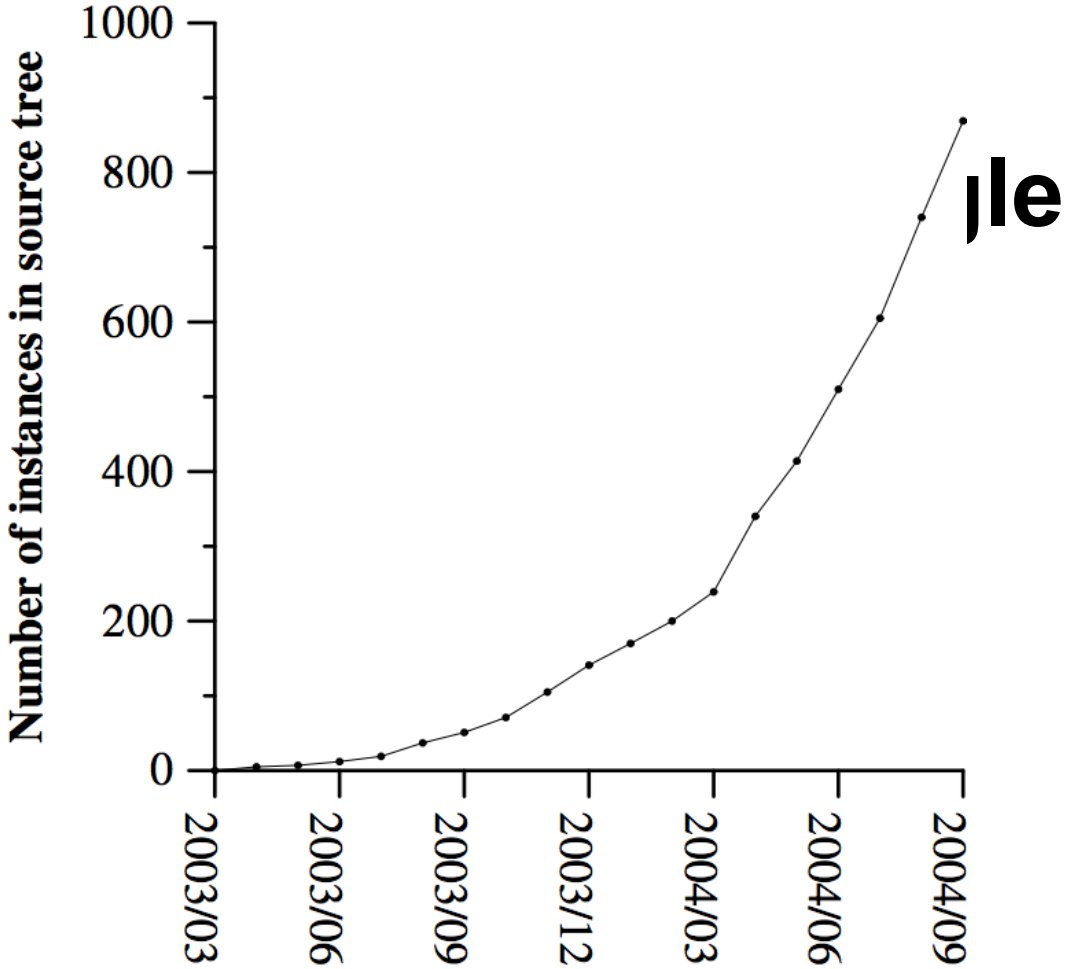


Figure 4: MapReduce instances over time

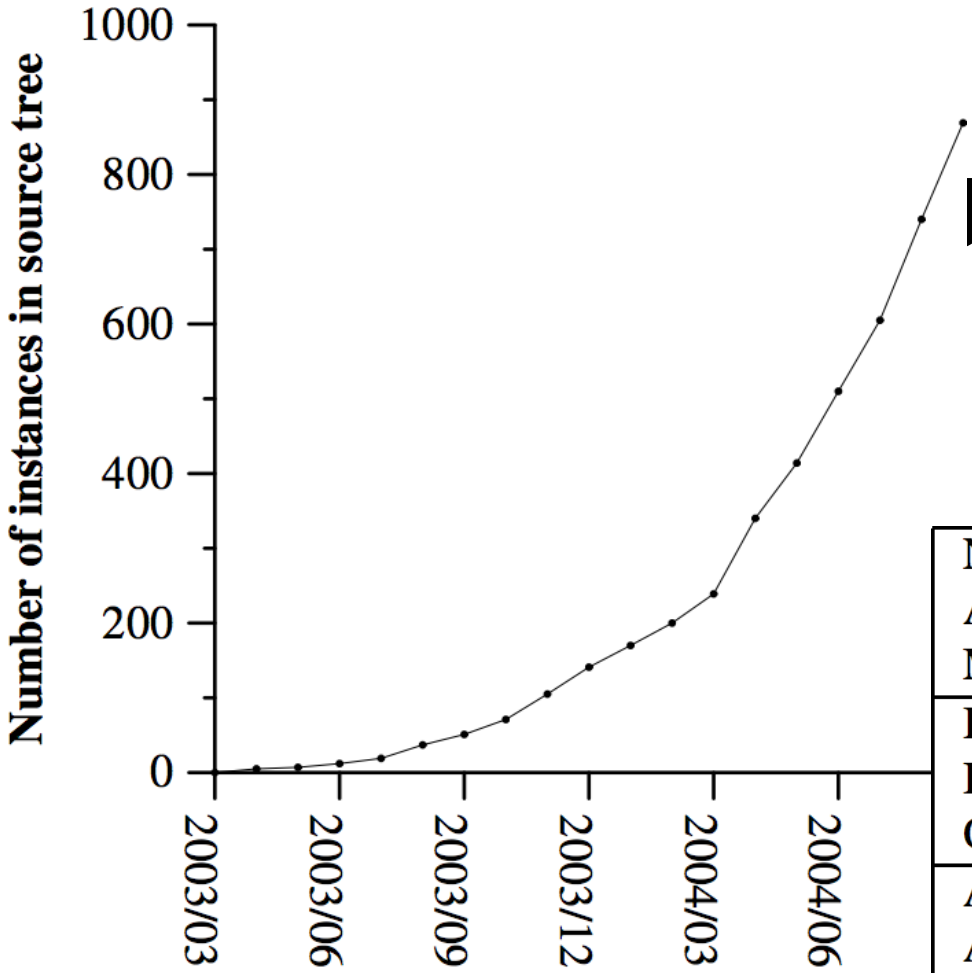


Figure 4: MapReduce instances over tim

Number of jobs	29,423
Average job completion time	634 secs
Machine days used	79,186 days
Input data read	3,288 TB
Intermediate data produced	758 TB
Output data written	193 TB
Average worker machines per job	157
Average worker deaths per job	1.2
Average map tasks per job	3,351
Average reduce tasks per job	55
Unique *map* implementations	395
Unique *reduce* implementations	269
Unique *map/reduce* combinations	426

Table 1: MapReduce jobs run in August 2004

Overview

- Programming Model
- Execution Overview
- Fault Tolerance
- Locality (Data Distribution)
- Improvements
- Evaluation
- Hadoop

What is Hadoop

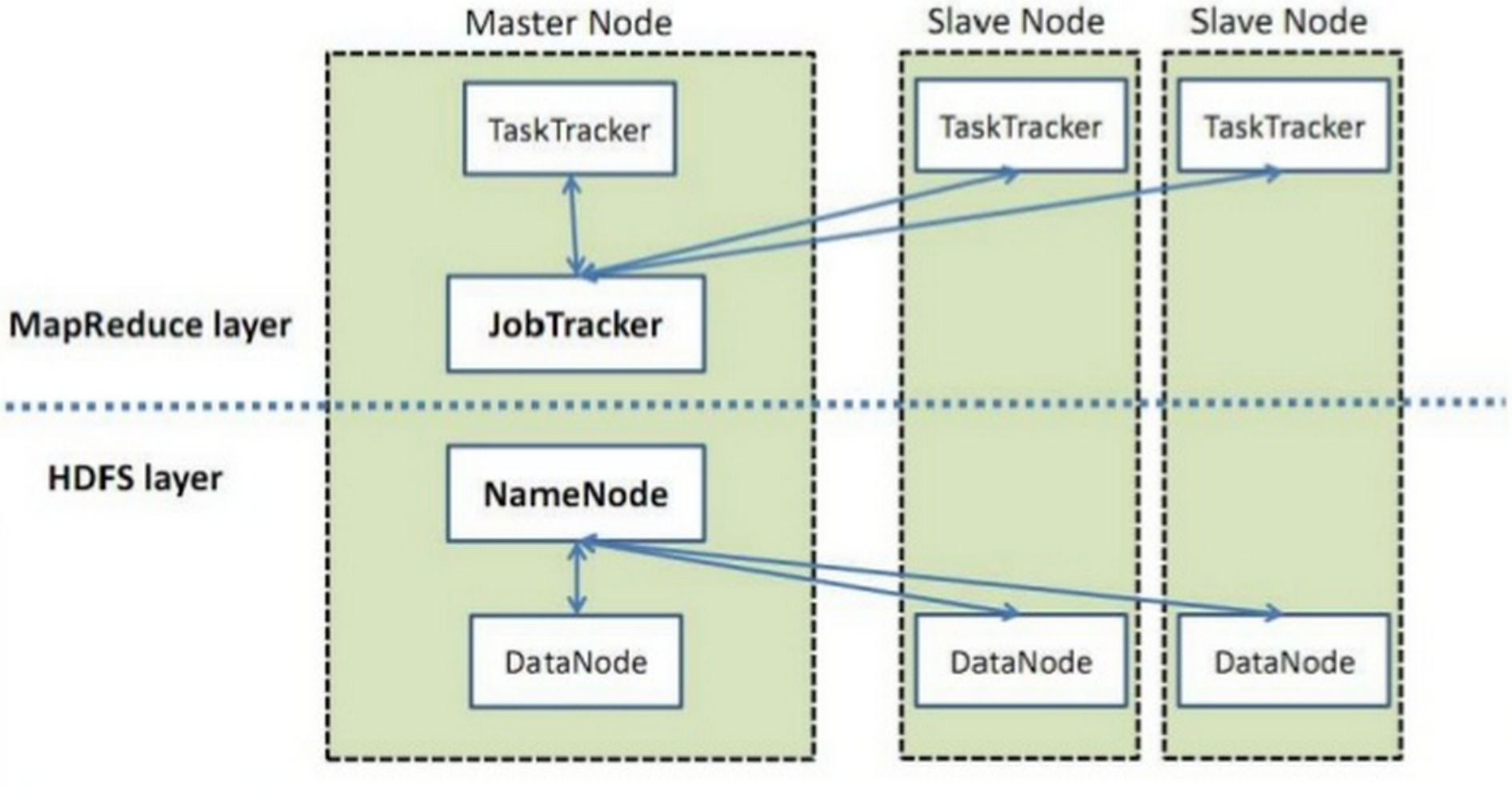
- Inspired by Google File System (GFS) and MapReduce
- Scale up from single servers to thousands of machines
- Widely deployed in real world systems
- Apache project -- Open source
- JAVA
- Yahoo!

Popular Framework

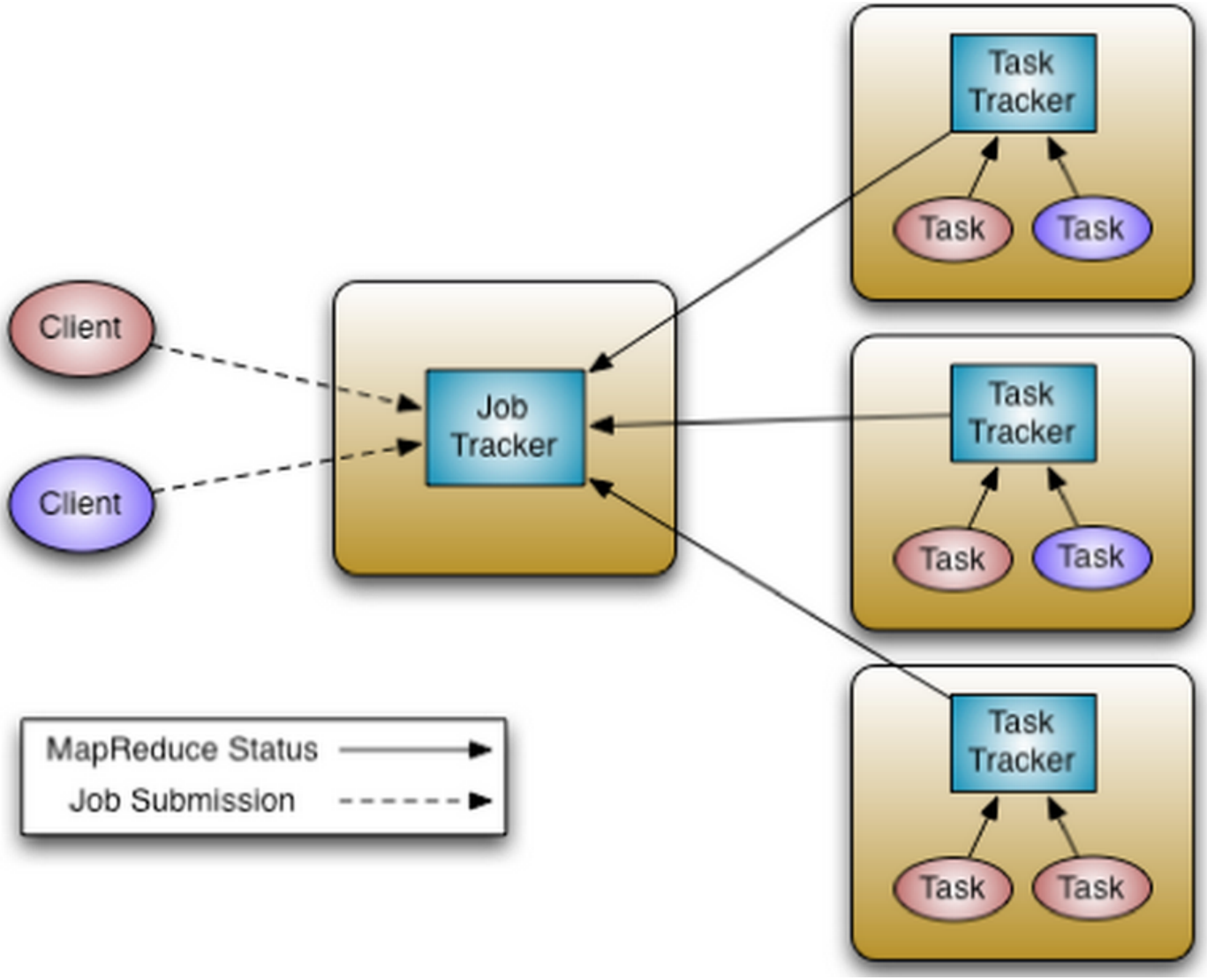
Hadoop Components

- Hadoop Distributed File System (HDFS)
  - Single namespace for entire cluster
  - Almost same as GFS
  - 3 default replicas
- Hadoop MapReduce

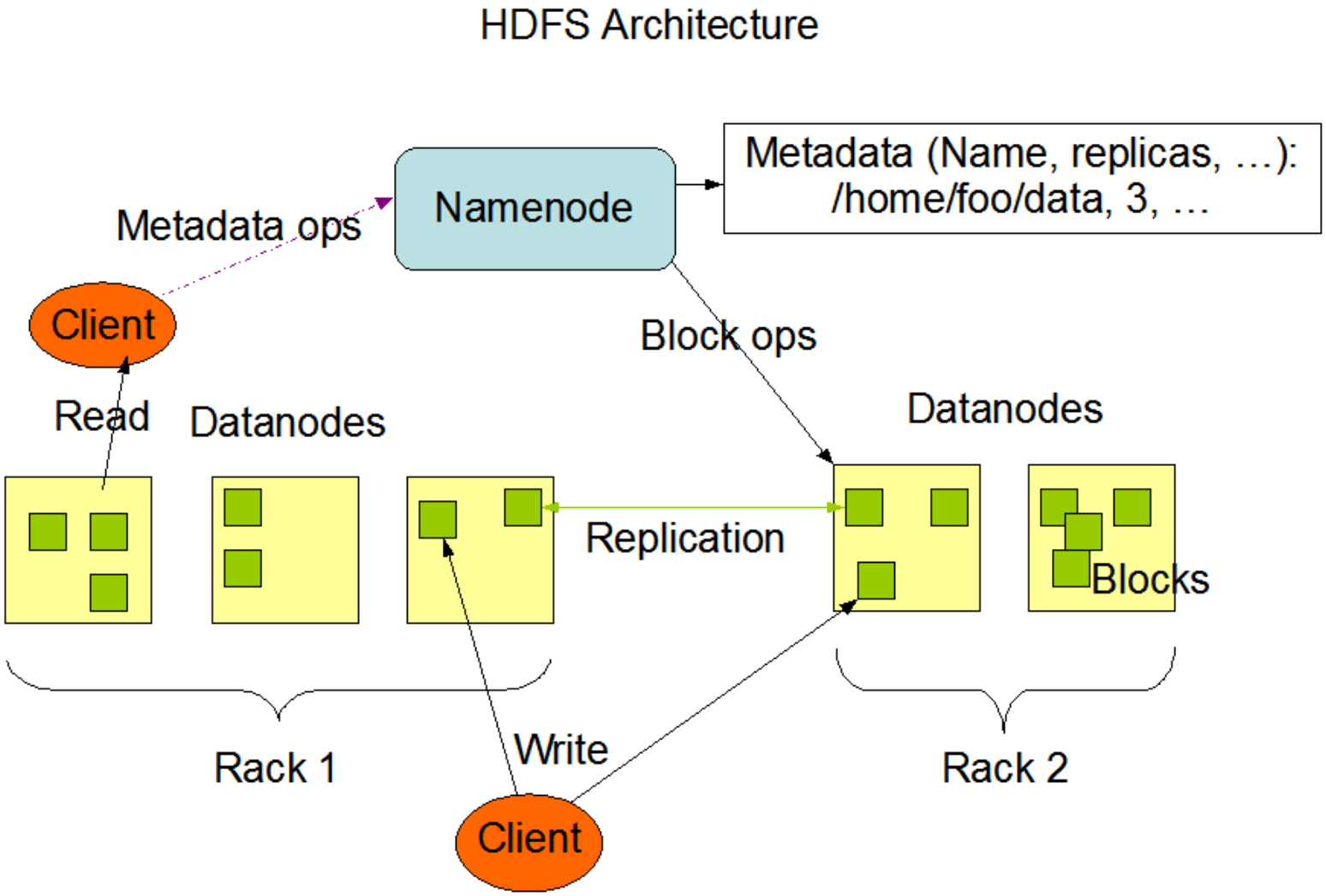
Big Picture


Master Node
Slave Node
Slave Node
TaskTracker
TaskTracker
TaskTracker
JobTracker
MapReduce layer
HDFS layer
NameNode
DataNode
DataNode
DataNode

Hadoop MapReduce



HDFS


HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

Hadoop Ecosystem

- Ambari™
- Avro™
- Cassandra™
- Chukwa™
- HBase™
- Hive™
- Mahout™
- Pig™
- Spark™
- Tez™
- ZooKeeper™

Thank You

Q & A

References

- Hadoop: http://hadoop.apache.org/
- Storm: http://hortonworks.com/hadoop/storm/
- Google Data Flow: http://googlecloudplatform.blogspot.com/2014/06/sneak-peek-google-cloud-dataflow-a-cloud-native-data-processing-service.html