Cloud Computing and Big Data

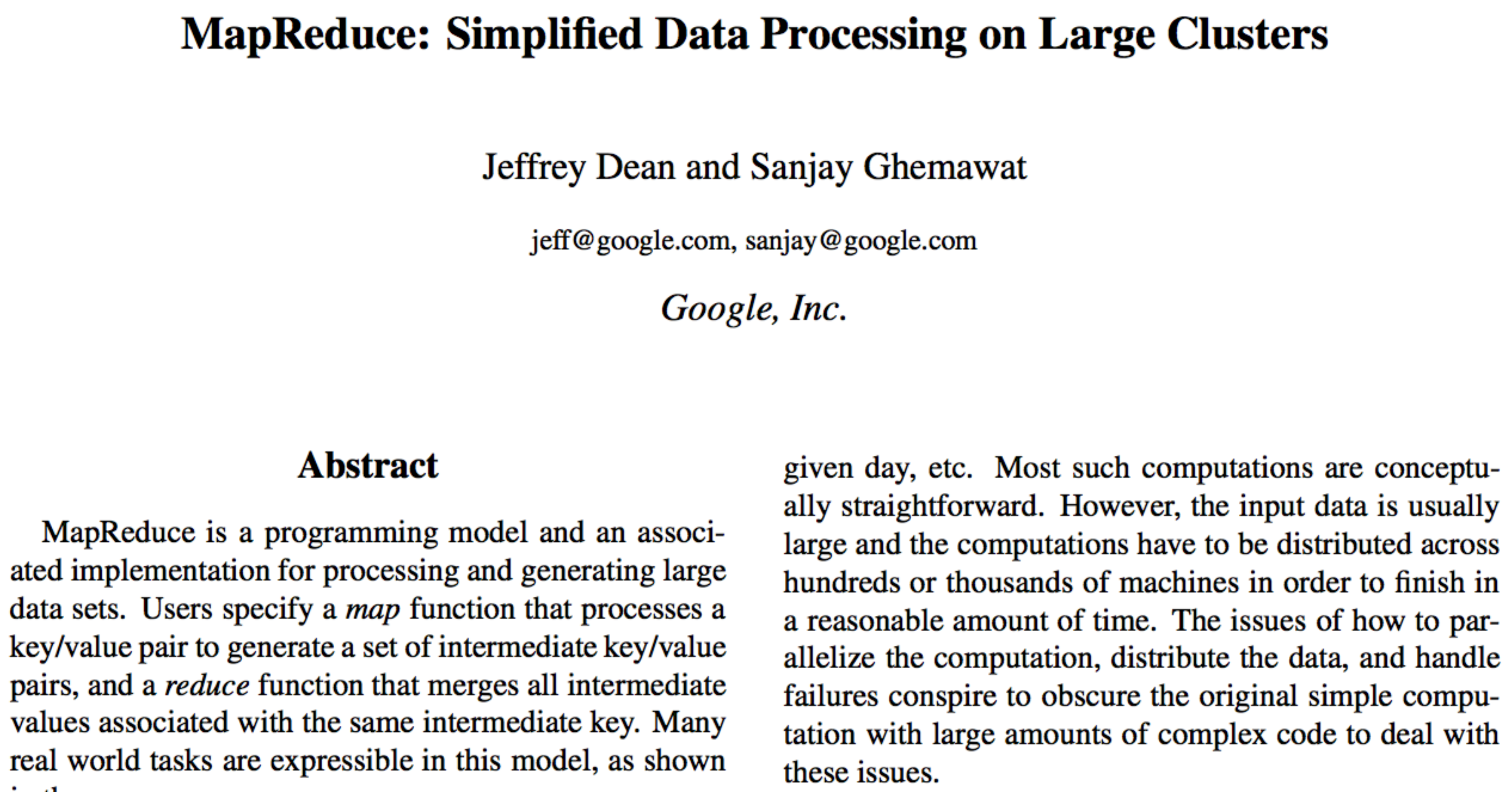
Hadoop and Map-Reduce

Oxford University
Software Engineering Programme
Nov 2015

Contents

- Understanding Map Reduce
  - Functional programming patterns applied for scalability
- Hadoop
  - Map-reduce in Hadoop
    - Python
    - Java
  - HDFS
  - Yarn
  - Pig and Hive
- Further reading

Original 2008 Google Paper



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

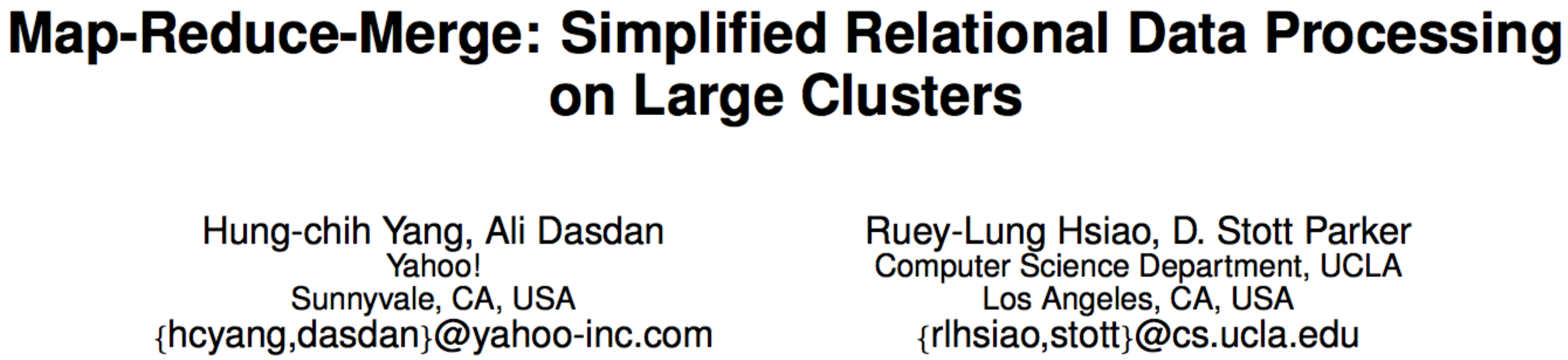
Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

Yahoo 2007



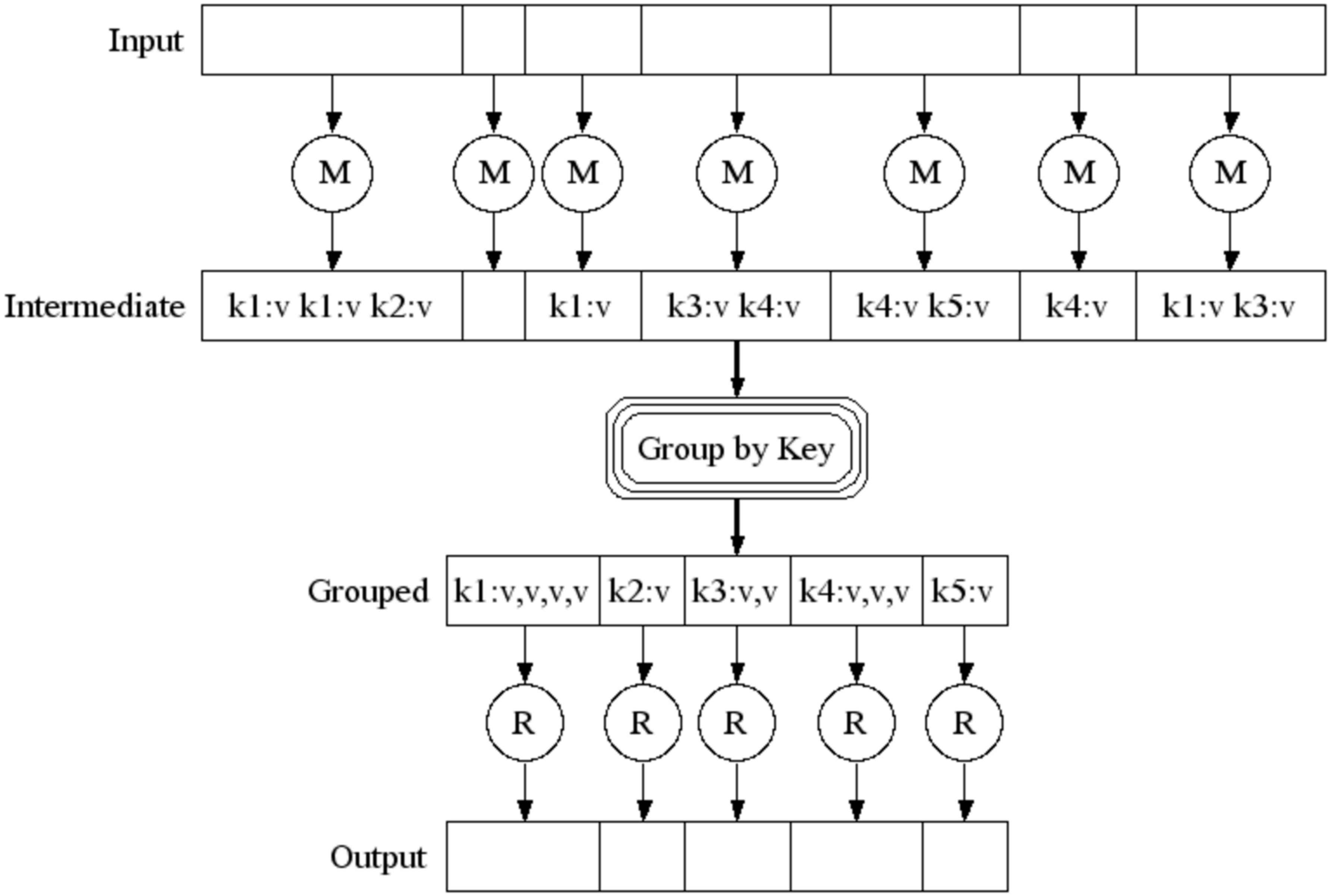
Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters

Hung-chih Yang, Ali Dasdan
Yahoo!
Sunnyvale, CA, USA
{hcyang,dasdan}@yahoo-inc.com

Ruey-Lung Hsiao, D. Stott Parker
Computer Science Department, UCLA
Los Angeles, CA, USA
{rlhsiao,stott}@cs.ucla.edu

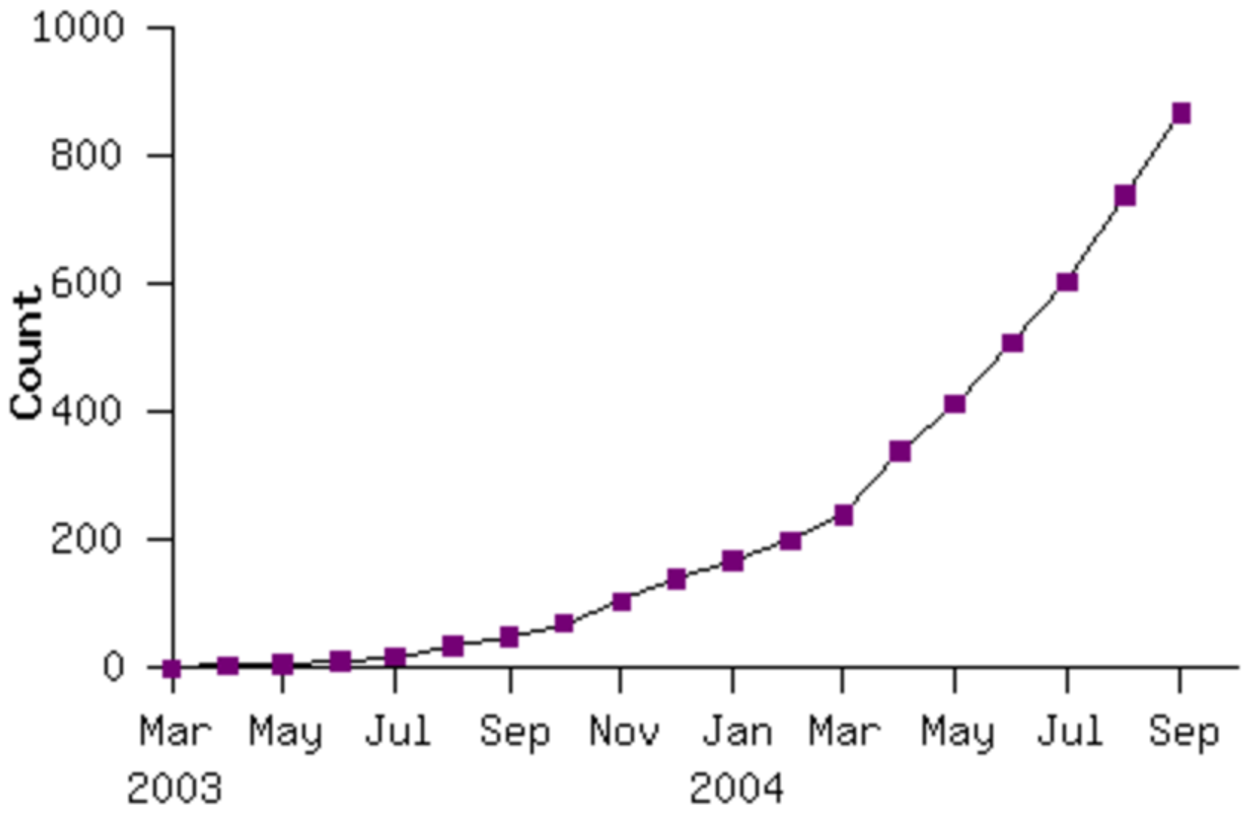
Programming Model

- Input & Output: each a set of key/value pairs
  - map (in_key, in_value) -> list(out_key, intermediate_value)
    - Processes input key/value pair
    - Produces set of intermediate pairs
  - reduce (out_key, list(intermediate_value)) -> list(out_value)
    - Combines all intermediate values for a particular key
    - Produces a set of merged output values (usually just one)
- Inspired by similar primitives in LISP and other languages


Input
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
Output

Google's early use of MR

Map Reduce programs in their code repository



Map Reduce example #1 Java

```
// some input
Integer array[] = {1,2,3,4,5};
// make a Stream of it
Stream<Integer> stream =
   Arrays.stream(array);
// map then reduce
Integer map_reduced =
   stream
      .map(x -> x*x)
      .reduce(0, (a, b)->a+b);
// should print 55
System.out.println(map_reduced);
```

Notice how

- We could have squared each number in the stream ***at the same time***
- Adding them up needed all the results available

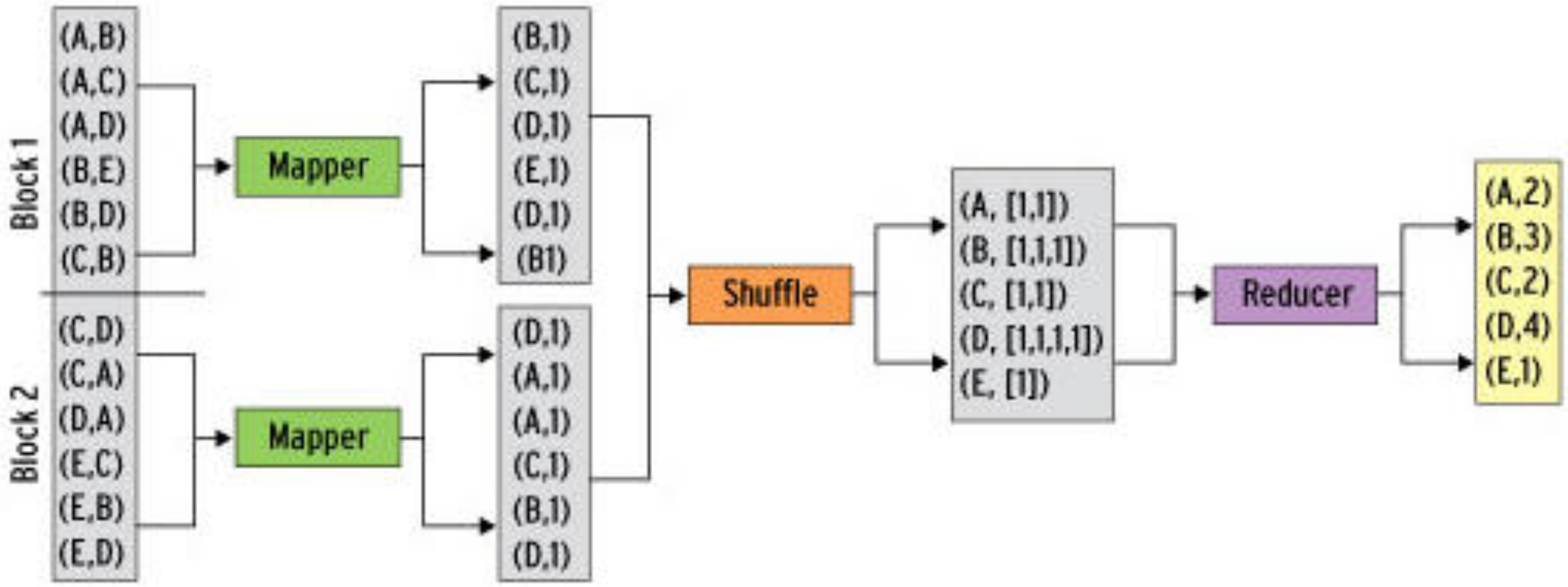
Map Reduce example #2 in words

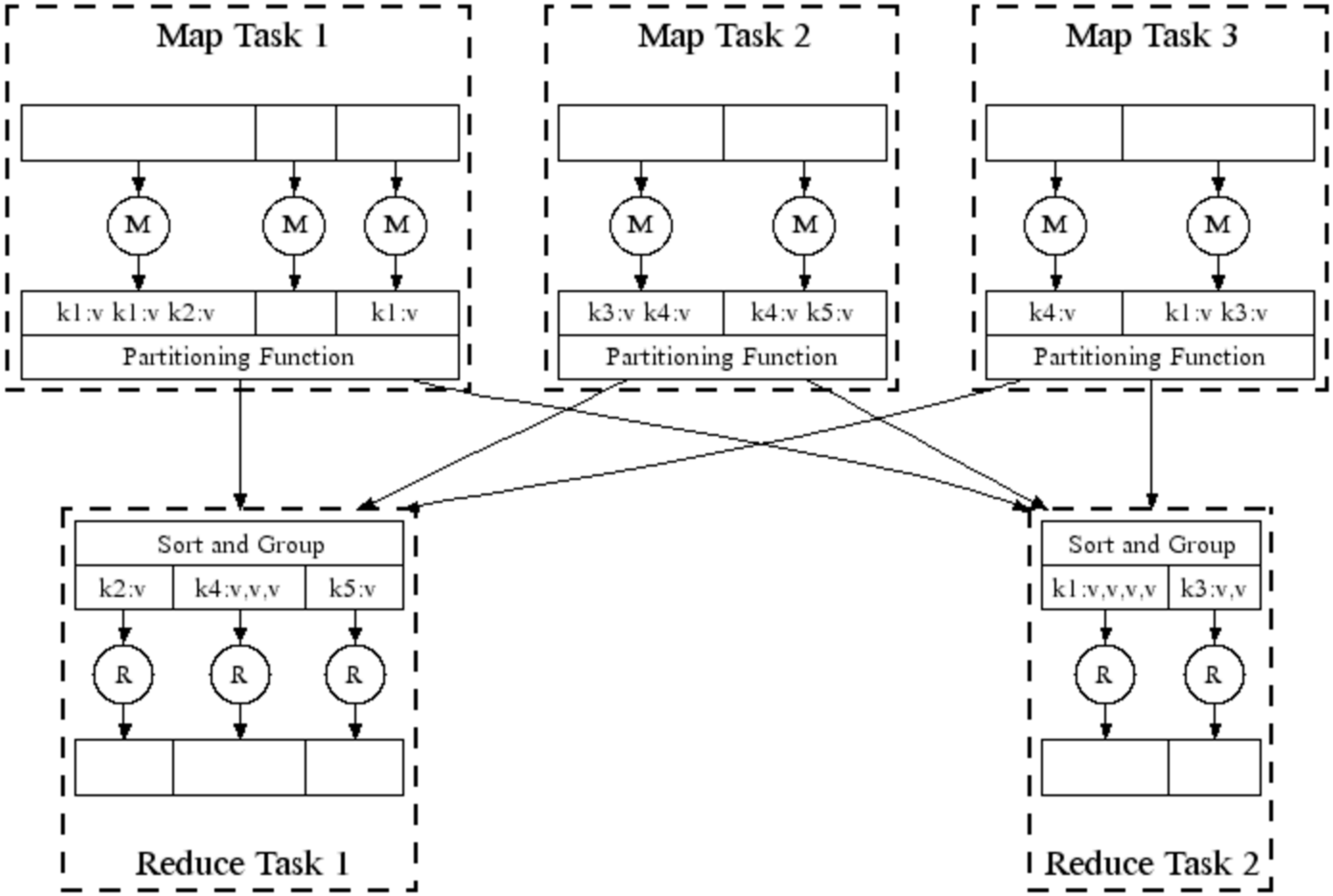
- Do a word count on 1000 books:
  - First count each book (**map** wc onto book)
  - Then **reduce** the outputs to a global wordcount across all books

Efficiency

- Reduce phase:
  - We can theoretically process each word in parallel
- How?
  - Shuffle / Sort the results from the map phase by key (word)
  - Partition by keys
  - Parallelize the reduce phase

Map/Shuffle/Reduce




Map Task 1
Map Task 2
Map Task 3
M
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Partitioning Function
Sort and Group
k2:v
k4:v,v,v
k5:v
k1:v,v,v,v
k3:v,v
R
Reduce Task 1
Reduce Task 2

Map Reduce in Real Life

- Analysing web logs
  - Summarise by user / cookie
  - Then aggregate to identify who did what
- Analysing twitter data
  - Who retweeted
  - Who was retweeted the most
- Almost all big data problems can be re-factored into Map Reduce
  - Some more efficiently than others

Tuning

- Fault tolerance
  - Simply re-execute work that fails
- Performance:
  - Partitioning the data
  - Moving the work to near the data

Apache Hadoop

- The most famous and popular Map Reduce framework
  - Open Source
    - Written in Java, but supports other languages
  - Runs Map Reduce workloads across a cloud or cluster of machines
  - Supports a distributed filesystem to store data for these jobs
  - Provides reliability when servers in the cluster fail

Components of Hadoop

Map Reduce or Other Workloads

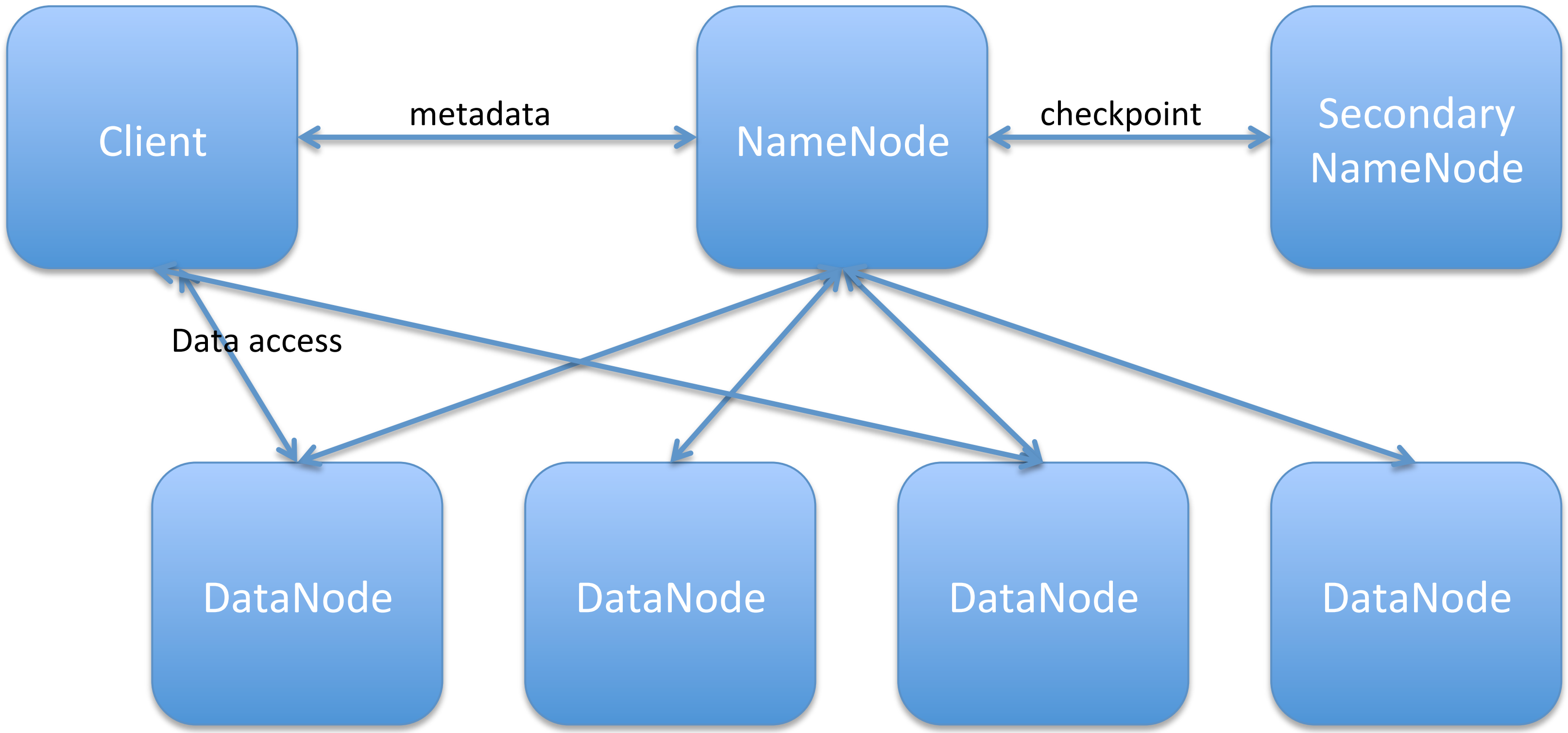
Java, Scala, Python, Apache Pig, Apache Hive, etc

YARN (Yet Another Resource Negotiator)
Cluster Resource Management

Hadoop Distributed File System (HDFS)

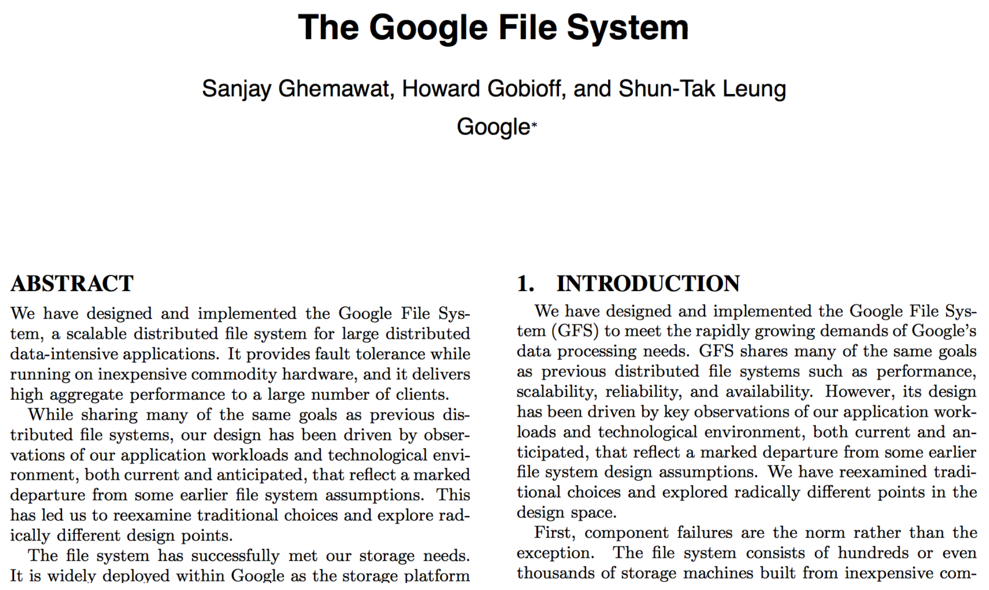
Redundant Reliable Distributed File System

HDFS in a nutshell



HDFS inspiration

- Google File System
  - Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03). ACM, New York, NY, USA, 29-43.



The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive com-

HDFS overview

- Good for streaming access to large files, reliability, scale
- Not good for random access, small files
- Blocks of data 64Mb in size (configurable)
- Each block can be replicated across multiple data nodes for High Availability (HA)

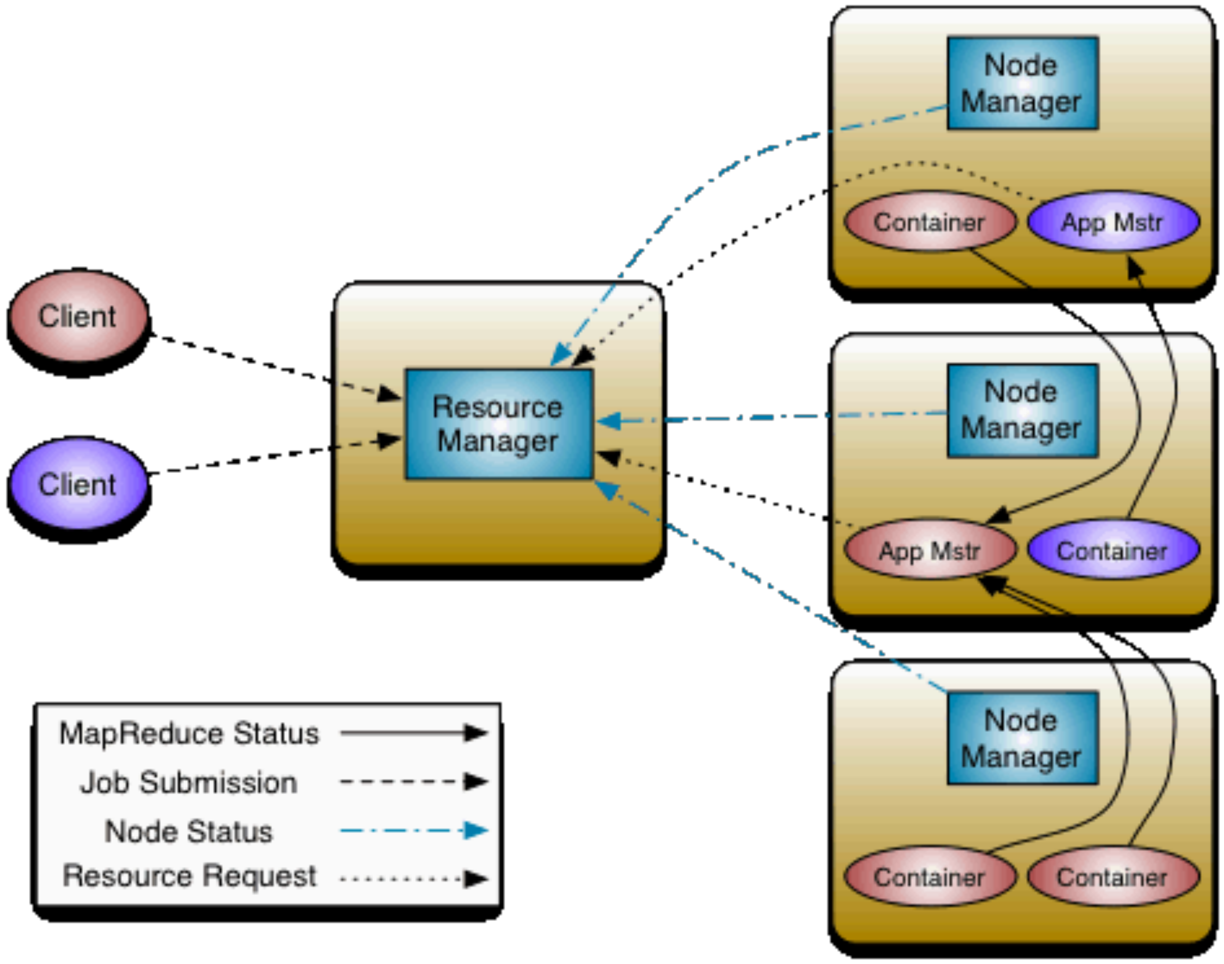
HDFS commands

- start-dfs.sh
- stop-dfs.sh
- hadoop fs <command>
  - e.g. hadoop fs cat /user/hduser/file
  - hadoop fs mkdir –p /user/hduser/
  - hadoop fs put localfile /user/hduser/remotefile
  - hadoop fs get /user/hduser/remotefile localfile

What is YARN?

- **YARN is the system that runs your code on multiple nodes**
- Hadoop 2.0 replacement for the cluster manager
  - Basically a model to distribute and manage workloads
  - Not just MapReduce but supports other workloads

YARN architecture



Map Reduce in Hadoop

- Map reduce in Hadoop actually consists of multiple steps
  - Mapper
    - Works on a single file, line by line
  - Combiner
    - Like a reducer, but still on a single system taking the output of the mapper
  - Reducer
    - Takes the outputs of multiple mapper/ combiners

The general flow

- On the whole, we expect to produce key-value <K,V> pairs from any mapper or reducer
  - In some cases we may produce <K,<V1,V2,..>>
- The results are stored to file and then read from file

Mappers

- Take input files and produce <K,V> pairs
- Each mapper gets a complete file
- Each mapper runs on a single system

Combiners

- The combiner function must take the <K,V> output of the mapper
- Produce the same format <K,V>
- Must be associative and commutative
- *Runs on the same node as the mapper*
- May actually be the reducer function if the reducer follows the rules above

Reducers

- The reducers get the <K,V> pairs output from the mappers/combiners
- The output is first sorted/partitioned
- The reducers produce the final expected output, usually in the form of <K,V> pairs

Map Reduce in Python
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)
```

Map Reduce in Python
A reducer

```
#!/usr/bin/env python
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:
        continue
    # this IF-switch only works because Hadoop sorts map output by key
    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)
```

Executing this on a set of files

```
yarn
jar
/usr/..../lib/hadoop-streaming-2.7.1.jar
-input /user/hduser/books
-output /user/hduser/output
-mapper ./mapper.py
-reducer ./reducer.py
```

Efficiency

- Please note this is not the most efficient way of coding Python
  - But the simplest and easiest to explain

Map Reduce in Java

```
 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Map Reduce in Java cont

```
$ yarn jar wc.jar
    WordCount
    /user/joe/wordcount/input
    /user/joe/wordcount/output
```

Summary

- Understanding the Map Reduce Model
- How is it implemented in Hadoop
- HDFS
- Yarn
- Running jobs

Questions?