

CC5212-1

PROCESAMIENTO MASIVO DE DATOS

OTOÑO 2016

Lecture 4: DFS & MapReduce I

Aidan Hogan

aidhog@gmail.com

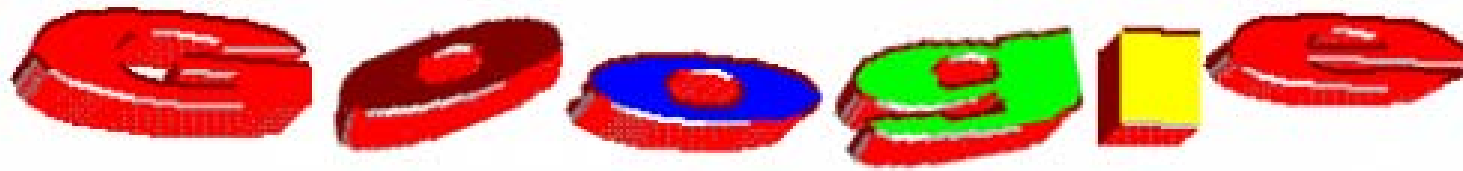
Fundamentals of Distributed Systems

external sorts replication consistency
consensus protocols cap theorem
availability two phase commit
fault tolerance
distributed hash table partitions
client server synchronous
paxos java rmi
distributed systems
peer to peer asynchronous fallacies
three phase commit
transparency three tier architecture

byzantine failure cloud grid scalability cluster

MASSIVE DATA PROCESSING (THE GOOGLE WAY ...)

Inside Google circa 1997/98



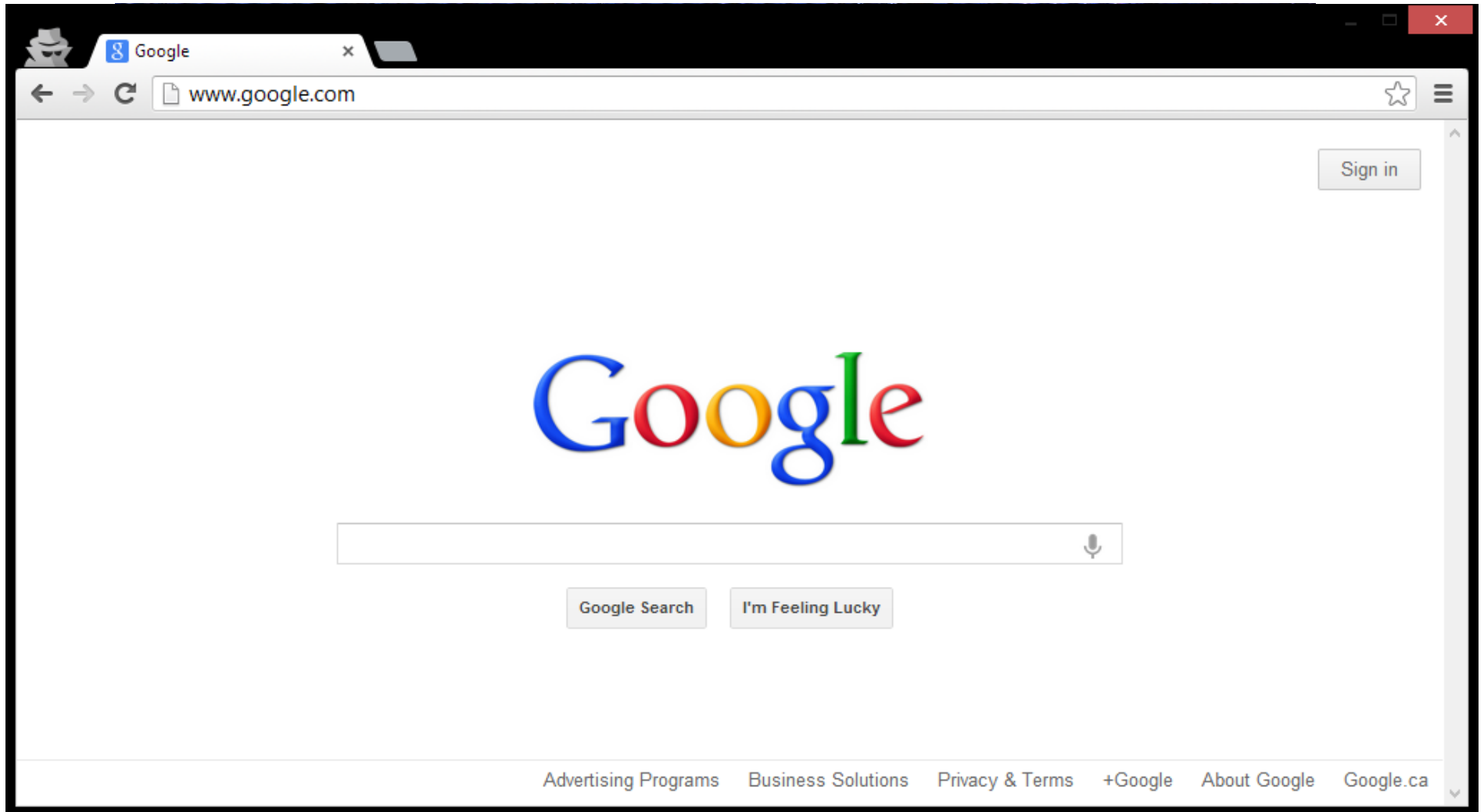
Search Stanford

10 results ▼ clustering on ▼ Search

Search The Web

10 results ▼ clustering on ▼ Search

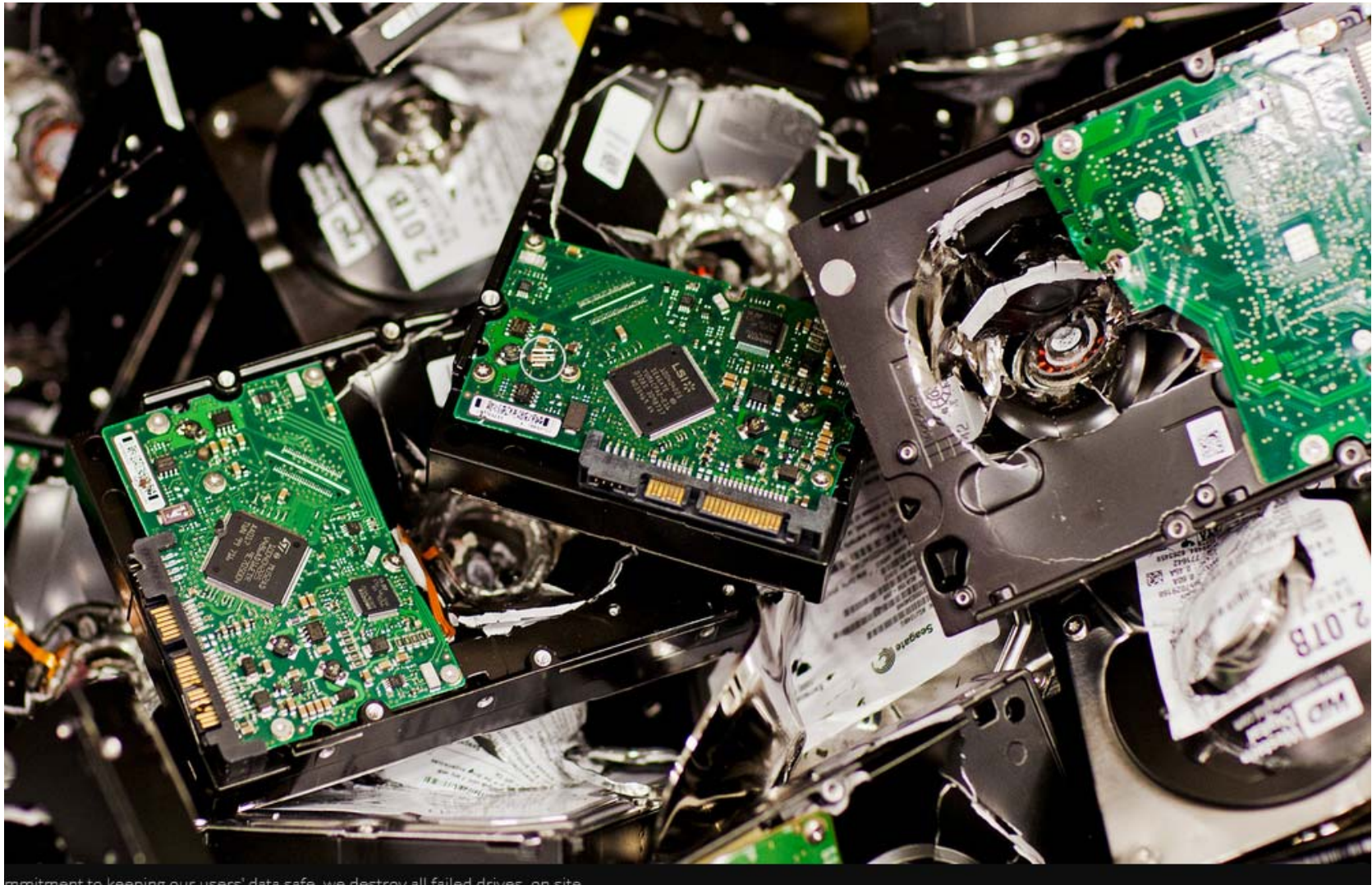
Inside Google circa 2015



Google's Cooling System



Google's Recycling Initiative

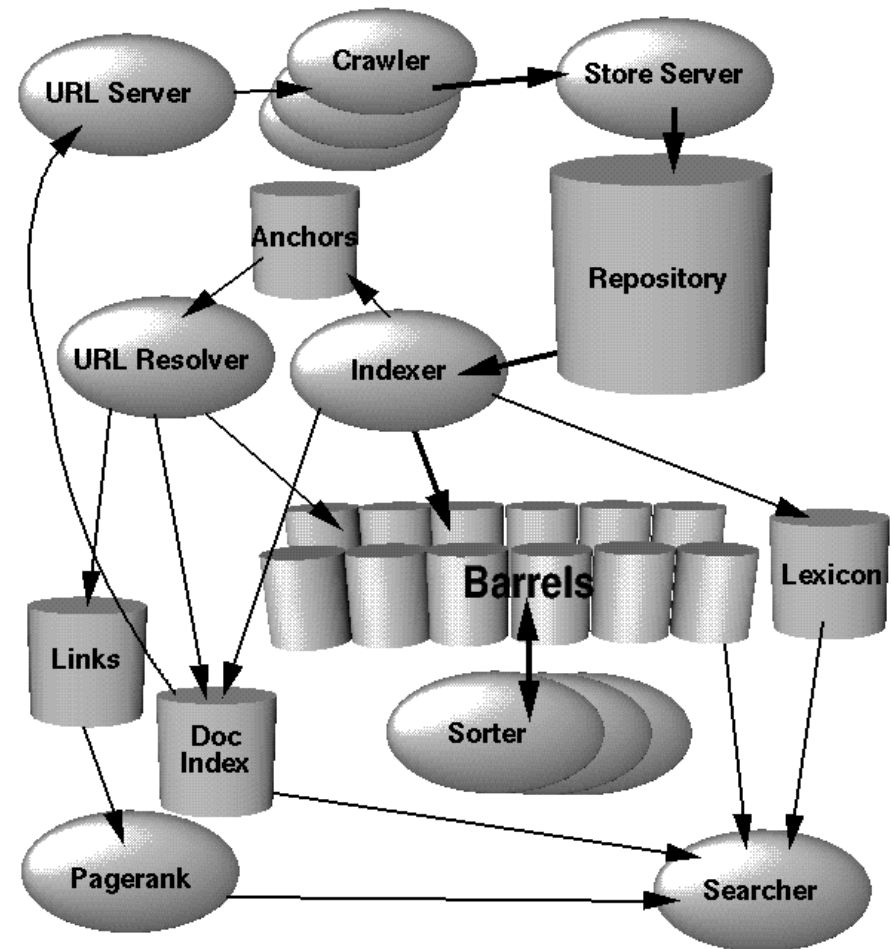


commitment to keeping our users' data safe, we destroy all failed drives, on site.

Google Architecture (ca. 1998)

Information Retrieval

- Crawling
- Inverted indexing
- Word-counts
- Link-counts
- greps/sorts
- PageRank
- Updates
- ...



Google Engineering

- Massive amounts of data
- Each task needs communication protocols
- Each task needs fault tolerance
- Multiple tasks running concurrently

Ad hoc solutions would repeat the same code

Google Engineering

- **Google File System**
 - Store data across multiple machines
 - Transparent Distributed File System
 - Replication / Self-healing
- **MapReduce**
 - Programming abstraction for distributed tasks
 - Handles fault tolerance
 - Supports many “simple” distributed tasks!
- **BigTable, Pregel, Percolator, Dremel ...**

Google *Re-Engineering*

Google

Google File System (GFS)



Google

MapReduce



Google

BigTable



APACHE
HBASE

GOOGLE FILE SYSTEM (GFS)

What is a File-System?

- Breaks files into chunks (or clusters)
- Remembers the sequence of clusters
- Records directory/file structure
- Tracks file meta-data
 - File size
 - Last access
 - Permissions
 - Locks

What is a Distributed File-System

- Same thing, but distributed

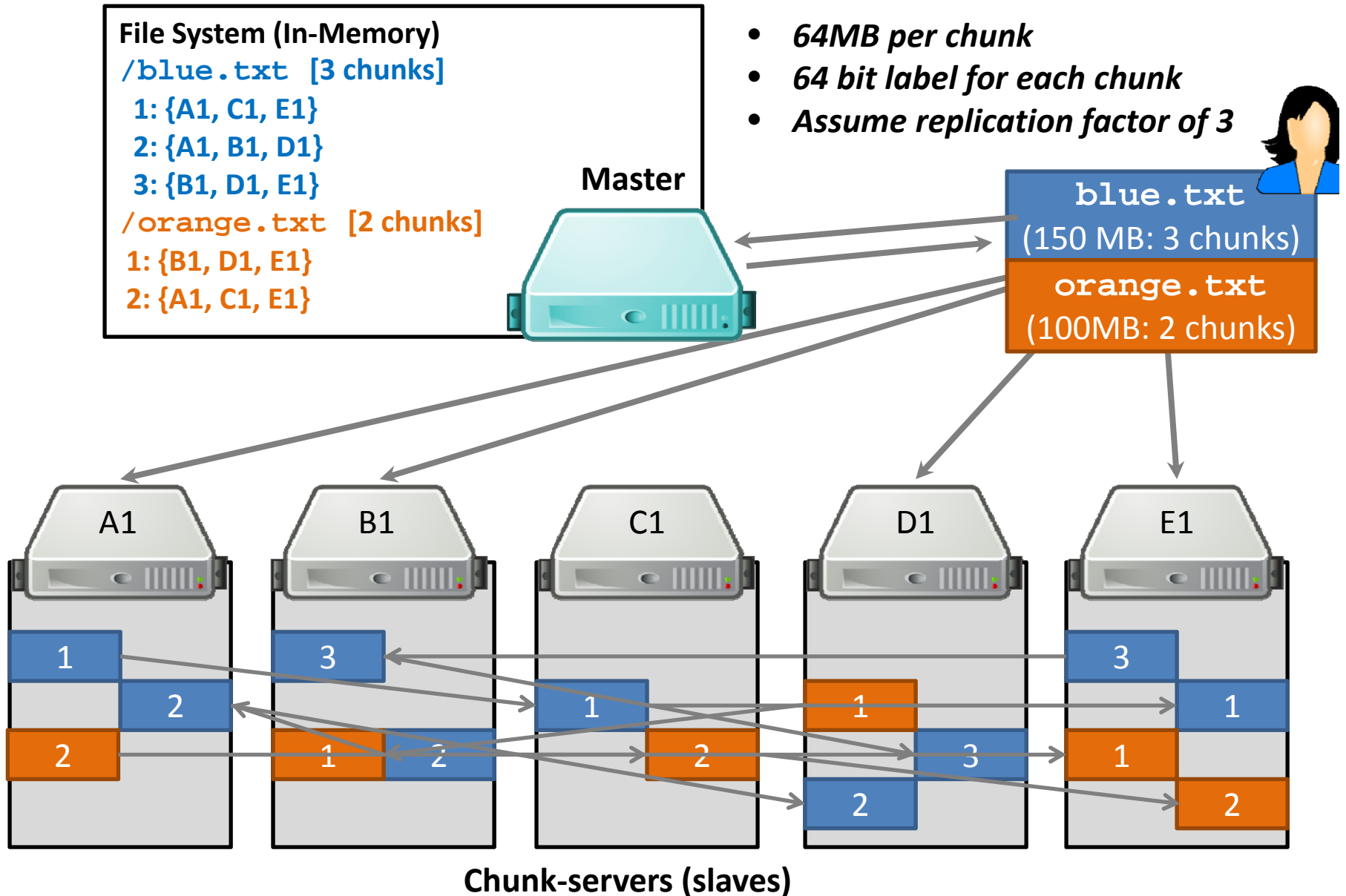
What would **transparency** / **flexibility** / **reliability** / **performance** / **scalability** mean for a distributed file system?

- **Transparency:** Like a normal file-system
- **Flexibility:** Can mount new machines
- **Reliability:** Has replication
- **Performance:** Fast storage/retrieval
- **Scalability:** Can store a lot of data / support a lot of machines

Google File System (GFS)

- Files are huge
- Files often read or appended
 - Writes in the middle of a file not (really) supported
- Concurrency important
- Failures are frequent
- Streaming important

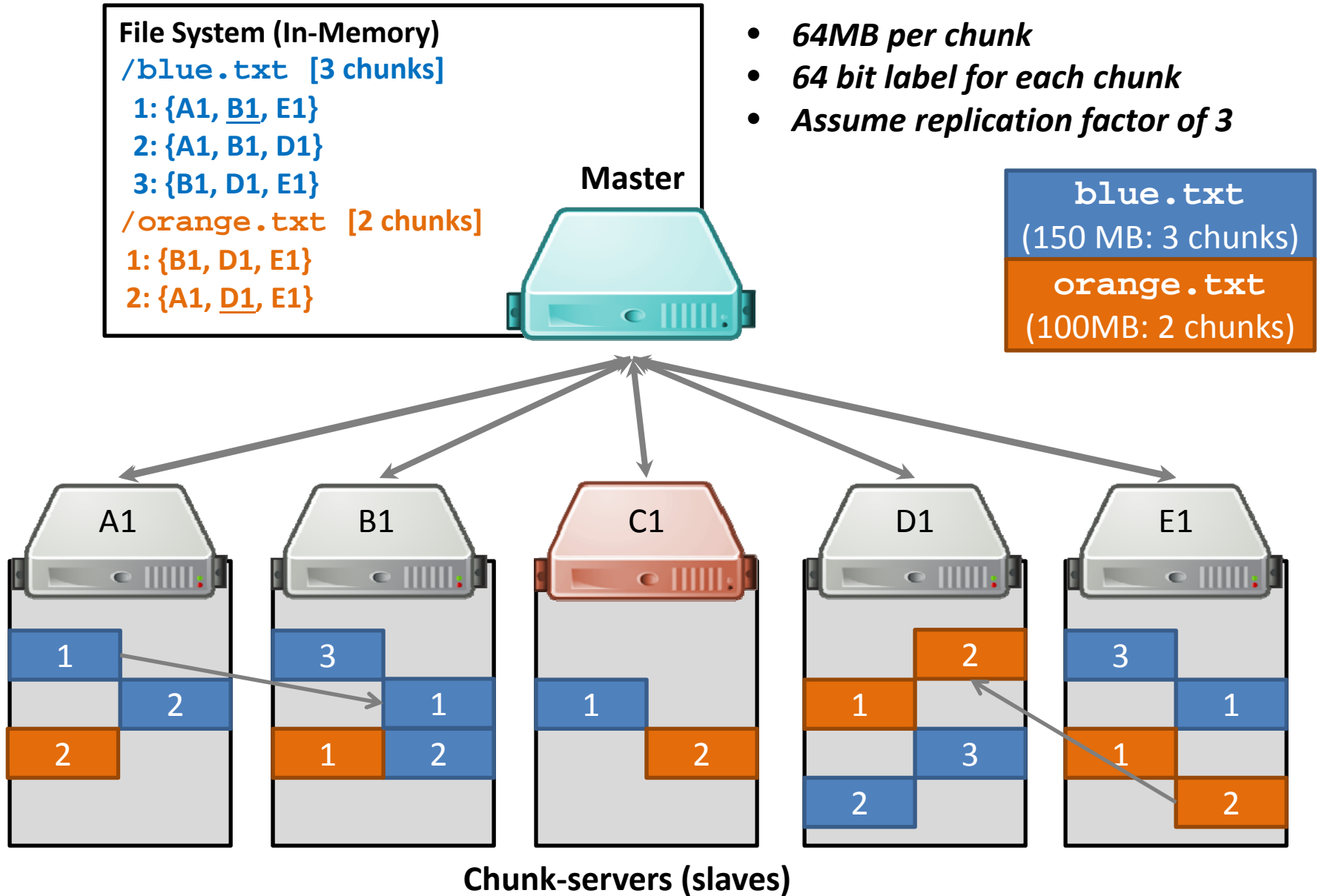
GFS: Pipelined Writes



GFS: Pipelined Writes (In Words)

1. Client asks Master to write a file
2. Master returns a primary chunkserver and secondary chunkservers
3. Client writes to primary chunkserver and tells it the secondary chunkservers
4. Primary chunkserver passes data onto secondary chunkserver, which passes on ...
5. When finished, message comes back through the pipeline that all chunkservers have written
 - Otherwise client sends again

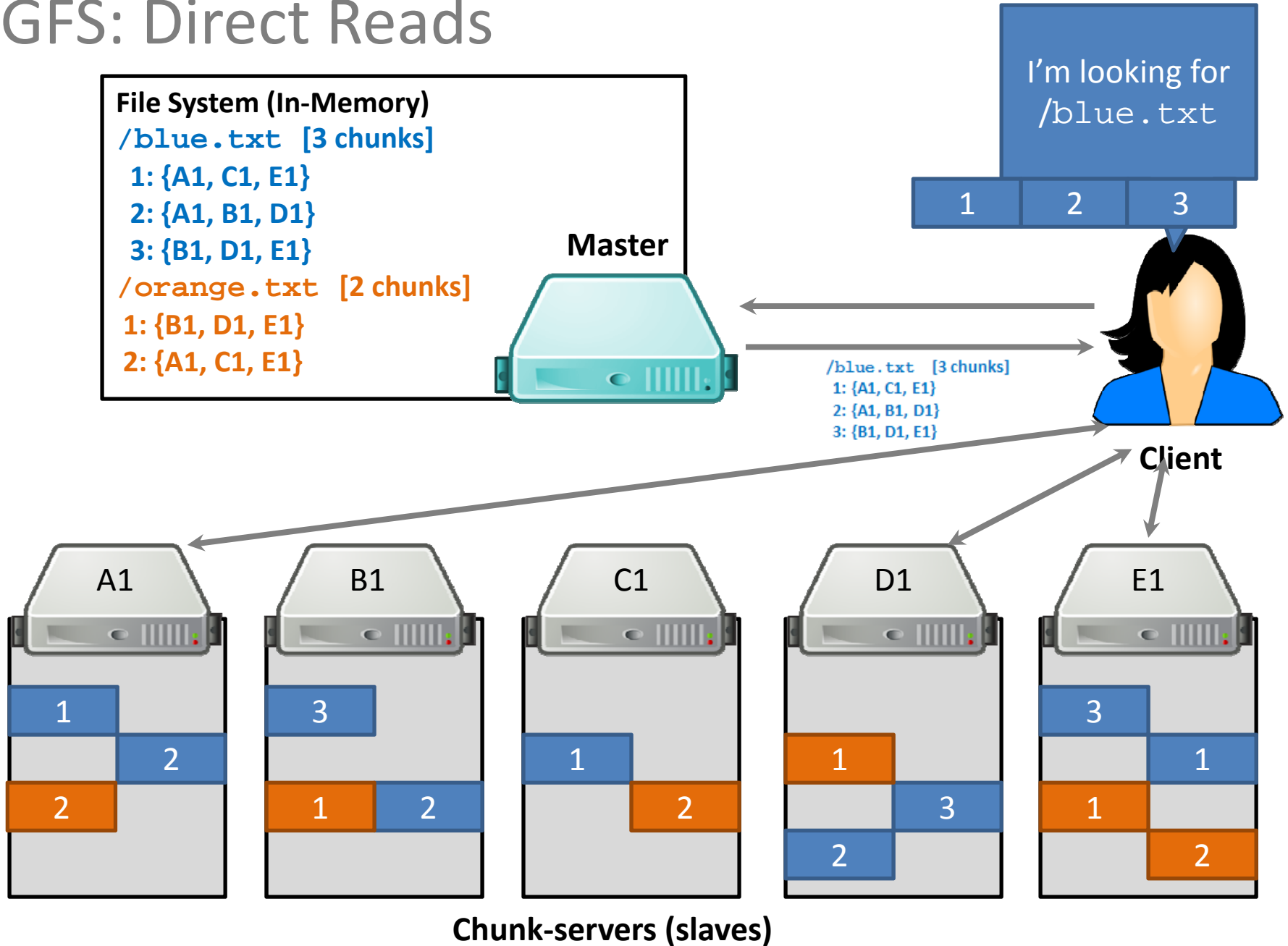
GFS: Fault Tolerance



GFS: Fault Tolerance (In Words)

- Master sends regular “Heartbeat” pings
- If a chunkserver doesn’t respond
 1. Master finds out what chunks it had
 2. Master assigns new chunkserver for each chunk
 3. Master tells new chunkserver to copy from a specific existing chunkserver
- Chunks are prioritised by number of remaining replicas, then by demand

GFS: Direct Reads



GFS: Direct Reads (In Words)

1. Client asks Master for file
2. Master returns location of a chunk
 - Returns a ranked list of replicas
3. Client reads chunk directly from chunkserver
4. Client asks Master for next chunk

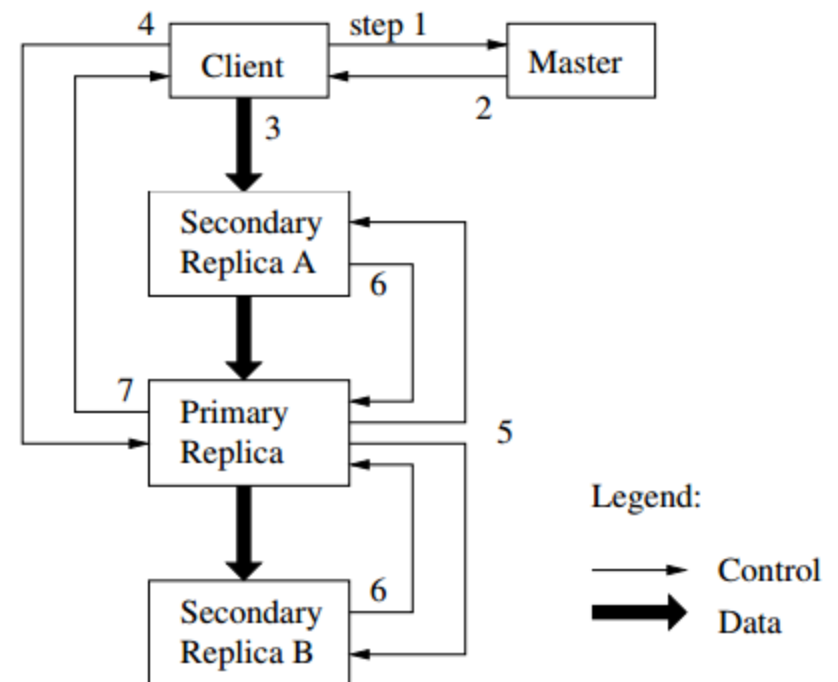
Software makes transparent for client!

GFS: Modification Consistency

Masters assign leases to one replica: a “primary replica”

Client wants to change a file:

1. Client asks Master for the replicas (incl. primary)
2. Master returns replica info to the client
3. Client sends change data
4. Client asks primary to execute the changes
5. Primary asks secondaries to change
6. Secondaries acknowledge to primary
7. Primary acknowledges to client

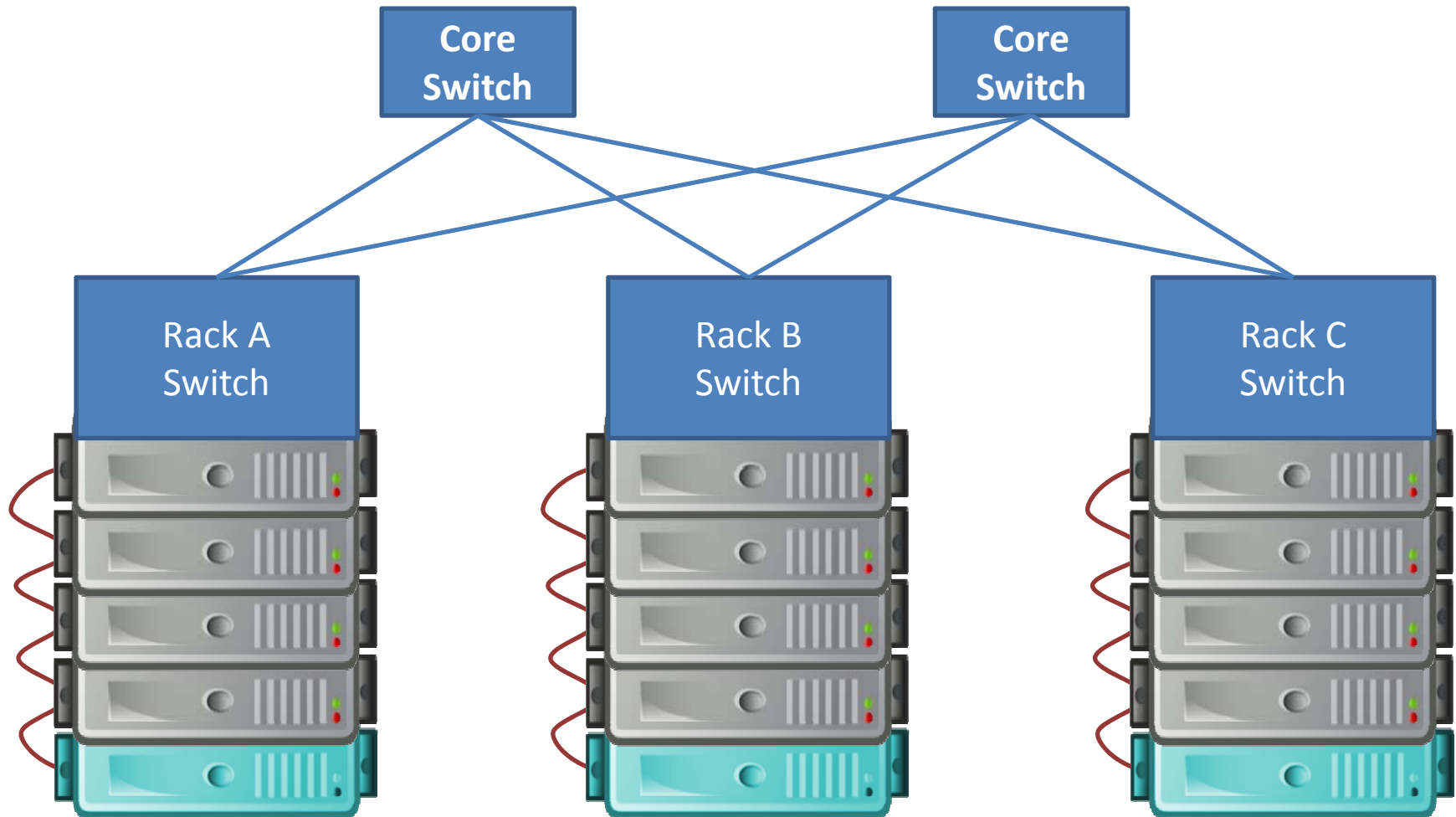


Data & Control Decoupled

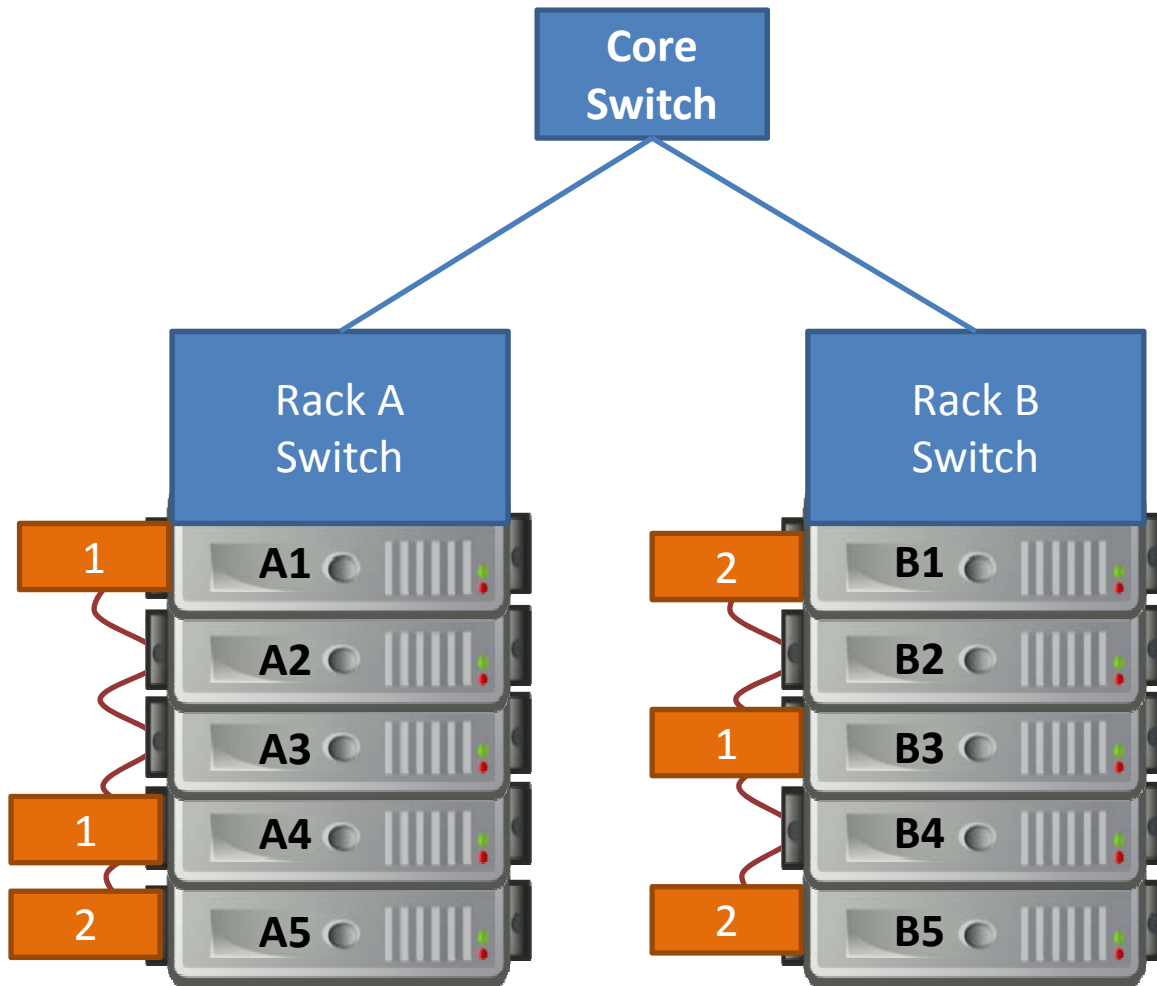
GFS: Rack Awareness



GFS: Rack Awareness



GFS: Rack Awareness



Files:

`/orange.txt`

1: {A1, A4, B3}

2: {A5, B1, B5}

Racks:

A: {A1, A2, A3, A4, A5}

B: {B1, B2, B3, B4, B5}

GFS: Rack Awareness (In Words)

- Make sure replicas not on same rack
 - In case rack switch fails!
- But communication can be slower:
 - Within rack: pass one switch (rack switch)
 - Across racks: pass three switches (two racks and a core)
- (Typically) pick two racks with low traffic
 - Two nodes in same rack, one node in another rack
 - (Assuming 3x replication)
- Only necessary if more than one rack! 😊

GFS: Other Operations

Rebalancing: Spread storage out evenly

Deletion:

- Just rename the file with hidden file name
 - To recover, rename back to original version
 - Otherwise, three days later will be wiped

Monitoring Stale Replicas: Dead slave reappears with old data: master keeps version info and will recycle old chunks

GFS: Weaknesses?

What do you see as the core weaknesses of the Google File System?

- **Master node single point of failure**
 - Use hardware replication
 - Logs and checkpoints!
- **Master node is a bottleneck**
 - Use more powerful machine
 - Minimise master node traffic
- **Master-node metadata kept in memory**
 - Each chunk needs 64 bytes
 - Chunk data can be queried from each slave

GFS: White-Paper

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 envi-

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

HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

Google *Re-Engineering*

The Google logo, consisting of the word "Google" in its characteristic multi-colored font.

Google File System (GFS)

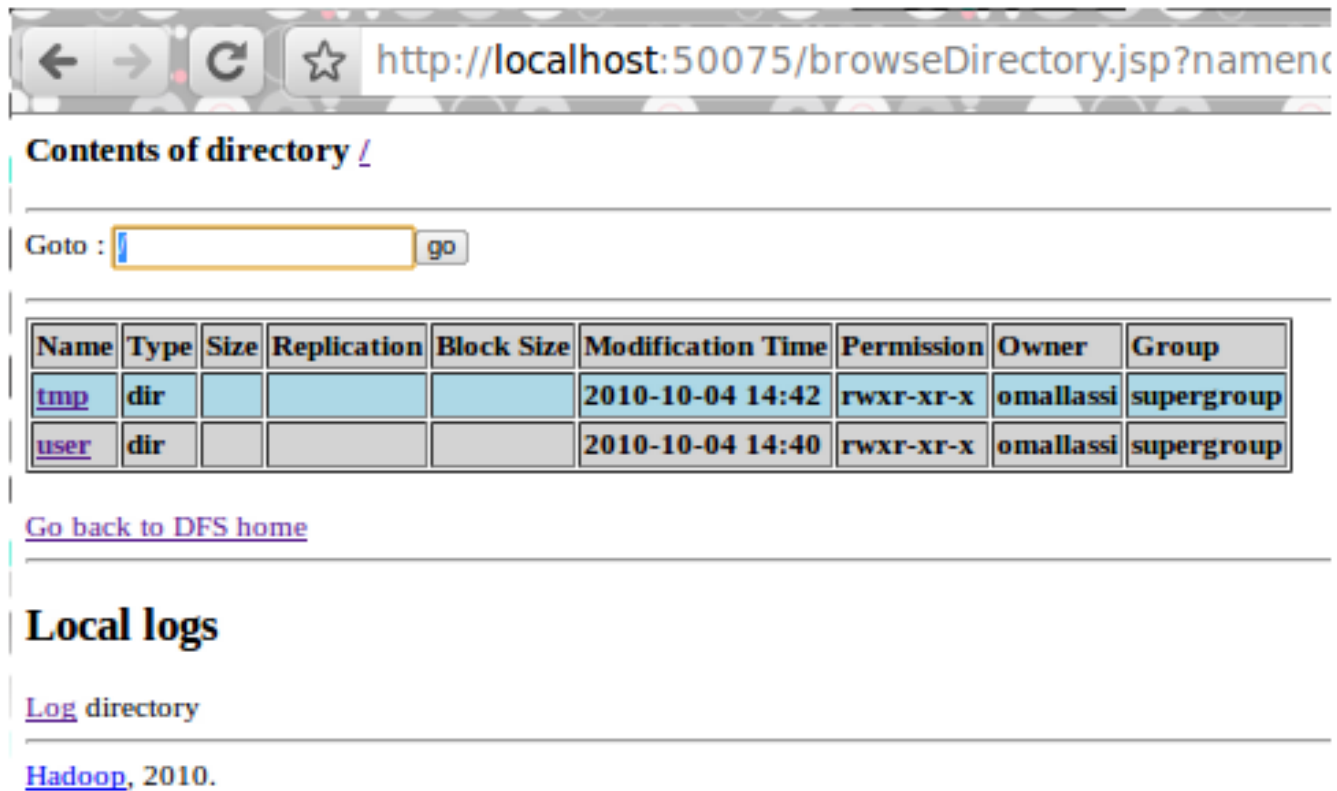


HDFS

- HDFS-to-GFS
 - Data-node = Chunkserver/Slave
 - Name-node = Master
- HDFS does not support modifications
- Otherwise pretty much the same except ...
 - GFS is proprietary (hidden in Google)
 - HDFS is open source (Apache!)



HDFS Interfaces



The screenshot shows a web browser window with the address bar displaying `http://localhost:50075/browseDirectory.jsp?namenode=...`. The page title is "Contents of directory /". Below the title is a "Goto :" field with a text input box and a "go" button. The main content is a table with 9 columns: Name, Type, Size, Replication, Block Size, Modification Time, Permission, Owner, and Group. The table lists two directories: "tmp" and "user", both with type "dir", size 0, replication 1, block size 128, modification time 2010-10-04 14:42 and 2010-10-04 14:40 respectively, permission "rwxr-xr-x", owner "omallassi", and group "supergroup". Below the table is a link "Go back to DFS home". The page also has a section "Local logs" with a link "Log directory". At the bottom, there is a footer "Hadoop, 2010."

Contents of directory /

Goto : go

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
tmp	dir	0	1	128	2010-10-04 14:42	rwxr-xr-x	omallassi	supergroup
user	dir	0	1	128	2010-10-04 14:40	rwxr-xr-x	omallassi	supergroup

[Go back to DFS home](#)

Local logs

[Log](#) directory

[Hadoop](#), 2010.

HDFS Interfaces

```
Hadoop Command Line
c:\Hadoop\hadoop-1.1.0-SNAPSHOT\bin>hadoop fs -lsr /
drwxrwxrwx - mohammad supergroup 0 2012-10-25 15:55 /dir1
-rw-r--r-- 1 mohammad supergroup 99271 2012-10-25 15:55 /dir1/sample.log
drwxr-xr-x - mohammad supergroup 0 2012-10-25 15:57 /hdfs
drwxr-xr-x - mohammad supergroup 0 2012-10-25 15:57 /hdfs/tmp
drwxr-xr-x - mohammad supergroup 0 2012-10-25 15:57 /hdfs/tmp/mapred
drwxr-xr-x - mohammad supergroup 0 2012-10-25 15:57 /hdfs/tmp/mapred/staging
drwxr-xr-x - mohammad supergroup 0 2012-10-25 15:57 /hdfs/tmp/mapred/staging/mohammad
drwx----- - mohammad supergroup 0 2012-10-25 15:58 /hdfs/tmp/mapred/staging/mohammad/.staging
drwxr-xr-x - mohammad supergroup 0 2012-10-25 15:57 /mapout
drwxr-xr-x - mohammad supergroup 0 2012-10-25 15:58 /mapout/mr1
-rw-r--r-- 1 mohammad supergroup 0 2012-10-25 15:58 /mapout/mr1/_SUCCESS
drwxr-xr-x - mohammad supergroup 0 2012-10-25 15:57 /mapout/mr1/_logs
drwxr-xr-x - mohammad supergroup 0 2012-10-25 15:57 /mapout/mr1/_logs/history
-rw-r--r-- 1 mohammad supergroup 16685 2012-10-25 15:57 /mapout/mr1/_logs/history/job_201210251258_0001_1351160842525_mohammad_tutorial1.jar
-rw-r--r-- 1 mohammad supergroup 21940 2012-10-25 15:57 /mapout/mr1/_logs/history/job_201210251258_0001_conf.xml
-rw-r--r-- 1 mohammad supergroup 52 2012-10-25 15:57 /mapout/mr1/part-00000
drwxr-xr-x - hadoop supergroup 0 2012-10-25 17:01 /mapred
drwxr-xr-x - hadoop supergroup 0 2012-10-25 03:55 /mapred/history
drwxr-xr-x - hadoop supergroup 0 2012-10-25 03:55 /mapred/history/done
drwxr-xr-x - hadoop supergroup 0 2012-10-25 15:58 /mapred/history/done/version-1
drwxr-xr-x - hadoop supergroup 0 2012-10-25 15:58 /mapred/history/done/version-1/localhost_1351150120448_
drwxr-xr-x - hadoop supergroup 0 2012-10-25 15:58 /mapred/history/done/version-1/localhost_1351150120448_/2012
drwxr-xr-x - hadoop supergroup 0 2012-10-25 15:58 /mapred/history/done/version-1/localhost_1351150120448_/2012/10
drwxr-xr-x - hadoop supergroup 0 2012-10-25 15:58 /mapred/history/done/version-1/localhost_1351150120448_/2012/10/25
drwxr-xr-x - hadoop supergroup 0 2012-10-25 15:58 /mapred/history/done/version-1/localhost_1351150120448_/2012/10/25/000
000
-rw-r--r-- 1 hadoop supergroup 16685 2012-10-25 15:58 /mapred/history/done/version-1/localhost_1351150120448_/2012/10/25/000
000/job_201210251258_0001_1351160842525_mohammad_tutorial1.jar
-rw-r--r-- 1 hadoop supergroup 21940 2012-10-25 15:58 /mapred/history/done/version-1/localhost_1351150120448_/2012/10/25/000
000/job_201210251258_0001_conf.xml
drwx----- - hadoop supergroup 0 2012-10-25 17:01 /mapred/system
-rw----- 1 hadoop supergroup 4 2012-10-25 17:01 /mapred/system/jobtracker.info

c:\Hadoop\hadoop-1.1.0-SNAPSHOT\bin>
```

GOOGLE'S MAP-REDUCE

Google *Re-Engineering*

Google

Google File System (GFS)



Google

MapReduce



MapReduce in Google

- Divide & Conquer:

1. Word count

How could we do a distributed top- k word count?

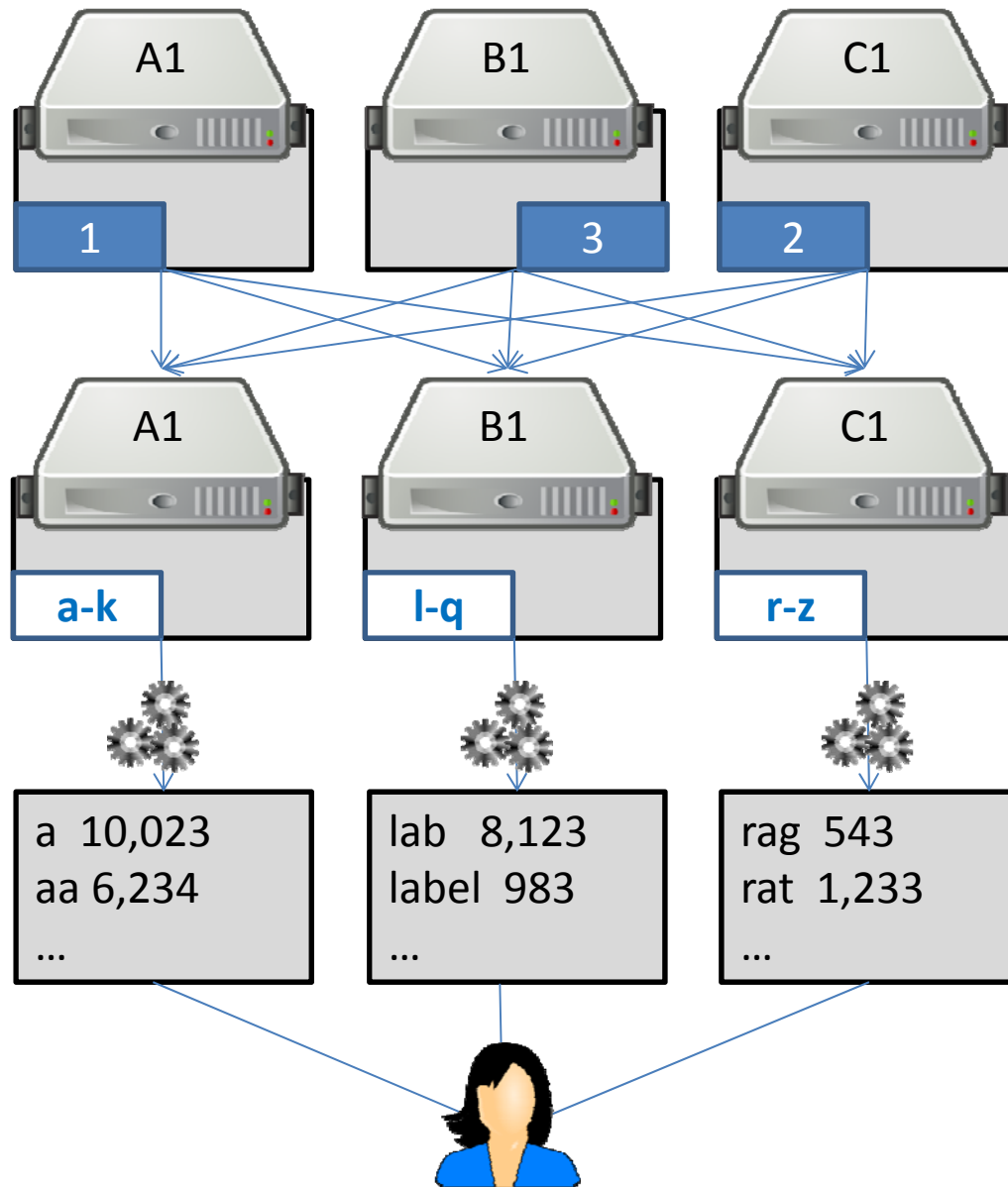
2. Total searches per user

3. PageRank

4. Inverted-indexing

5. ...

MapReduce: Word Count



Better partitioning
method?

Input

Distr. File Sys.

Map()

(Partition/Shuffle)

(Distr. Sort)

Reduce()

Output

MapReduce (in more detail)

1. **Input:** Read from the cluster (e.g., a DFS)

- Chunks raw data for mappers
- Maps raw data to initial $(key_{in}, value_{in})$ pairs

What might **Input** do in the word-count case?

2. **Map:** For each $(key_{in}, value_{in})$ pair, generate zero-to-many $(key_{map}, value_{map})$ pairs

- $key_{in} / value_{in}$ can be diff. type to $key_{map} / value_{map}$

What might **Map** do in the word-count case?

MapReduce (in more detail)

- 3. Partition:** Assign sets of key_{map} values to reducer machines

How might **Partition** work in the word-count case?

- 4. Shuffle:** Data are moved from mappers to reducers (e.g., using DFS)
- 5. Comparison/Sort:** Each reducer sorts the data by key using a comparison function
 - *Sort is taken care of by the framework*

MapReduce

6. **Reduce**: Takes a bag of $(key_{map}, value_{map})$ pairs with the same key_{map} value, and produces zero-to-many outputs for each bag
- Typically zero-or-one outputs

How might **Reduce** work in the word-count case?

7. **Output**: Merge-sorts the results from the reducers / writes to stable storage

MapReduce: Word Count PseudoCode

```
function map(String name, String document):  
    // name: document name  
    // document: document contents  
    for each word w in document:  
        emit (w, 1)  
  
function reduce(String word, Iterator partialCounts):  
    // word: a word  
    // partialCounts: a list of aggregated partial counts  
    sum = 0  
    for each pc in partialCounts:  
        sum += ParseInt(pc)  
    emit (word, sum)
```

MapReduce: Scholar Example

Google scholar

[My Citations](#)

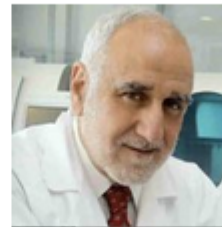
Authors

1-10 [Next >](#)



Ricardo Baeza-Yates

Yahoo! Research - Univ. Pompeu Fabra - Univ. de Chile, Spain & Chile
Verified email at upf.edu
Cited by 29988



Ricardo Uauy

Institute of Nutrition INTA U Chile and Pediatrics Department Catholic University
Verified email at inta.cl
Cited by 25089



JUAN CARLOS CASTILLA ZENOBI

Pontificia Universidad Católica de Chile
Cited by 18903



Gonzalo Navarro

University of Chile
Verified email at dcc.uchile.cl
Cited by 13443



Susana Eyheramendy

Professor of Statistics, Pontificia Universidad Católica de Chile
Verified email at mat.puc.cl
Cited by 9776



Pablo A. Marquet

Professor of Ecology, Pontificia Universidad Católica de Chile
Verified email at bio.puc.cl
Cited by 7812



José Joaquín Brunner,
Universidad Diego Portales

Jose Joaquin Brunner

Profesor investigador Universidad Diego Portales, Chile
Verified email at cpce.cl
Cited by 6603



Ortuzar J de D

Professor of Transport Engineering, Pontificia Universidad Católica de Chile
Verified email at ing.puc.cl
Cited by 5638

MapReduce: Scholar Example

Assume that in Google Scholar we have inputs like:

paper_A citedBy paper_B

How can we use MapReduce to count the total incoming citations per paper?

MapReduce as a Dist. Sys.

- **Transparency:** Abstracts physical machines
- **Flexibility:** Can mount new machines; can run a variety of types of jobs
- **Reliability:** Tasks are monitored by a master node using a heart-beat; dead jobs restart
- **Performance:** Depends on the application code but exploits parallelism!
- **Scalability:** Depends on the application code but can serve as the basis for massive data processing!

MapReduce: Benefits for Programmers

- **Takes care of low-level implementation:**
 - Easy to handle inputs and output
 - No need to handle network communication
 - No need to write sorts or joins
- **Abstracts machines (transparency)**
 - Fault tolerance (through heart-beats)
 - Abstracts physical locations
 - Add / remove machines
 - Load balancing

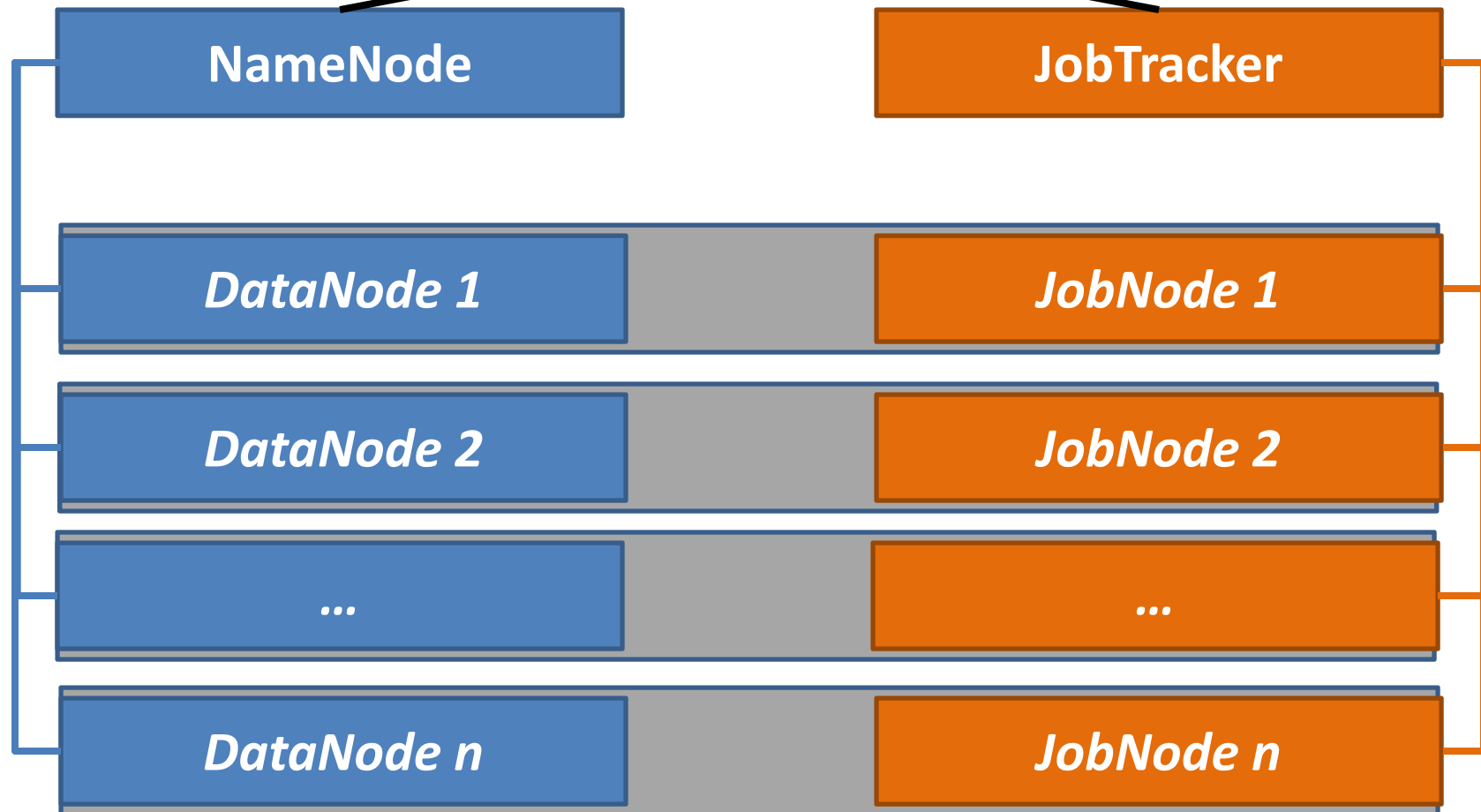
MapReduce: Benefits for Programmers

Time for more important things ...

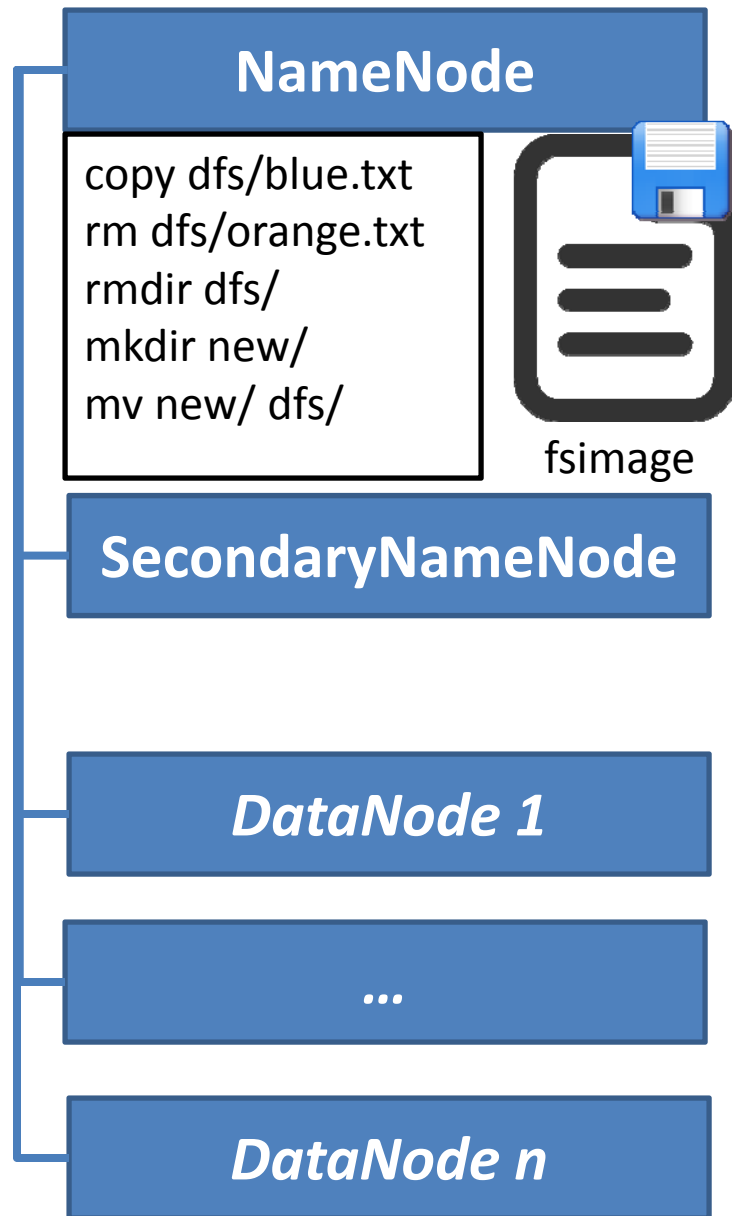


HADOOP OVERVIEW

Hadoop Architecture



HDFS: Traditional / SPOF



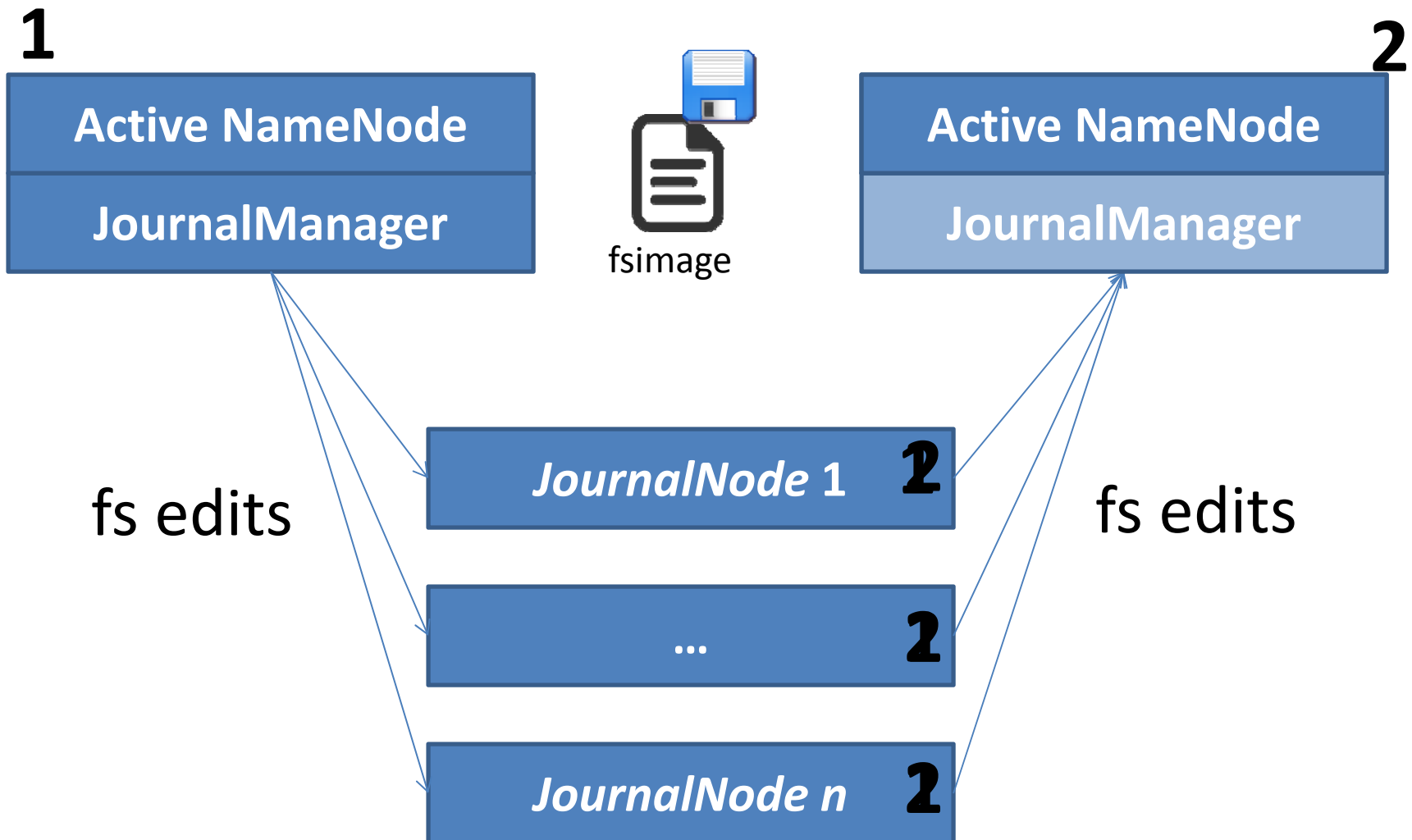
1. NameNode appends edits to log file
2. SecondaryNameNode copies log file and image, makes checkpoint, copies image back
3. NameNode loads image on start-up and makes remaining edits

SecondaryNameNode not a backup NameNode

What is the secondary name-node?

- Name-node quickly logs all file-system actions in a sequential (but messy) way
- Secondary name-node keeps the main `fsimage` file up-to-date based on logs
- When the primary name-node boots back up, it loads the `fsimage` file and applies the remaining log to it
- Hence secondary name-node helps make boot-ups faster, helps keep file system image up-to-date and takes load away from primary

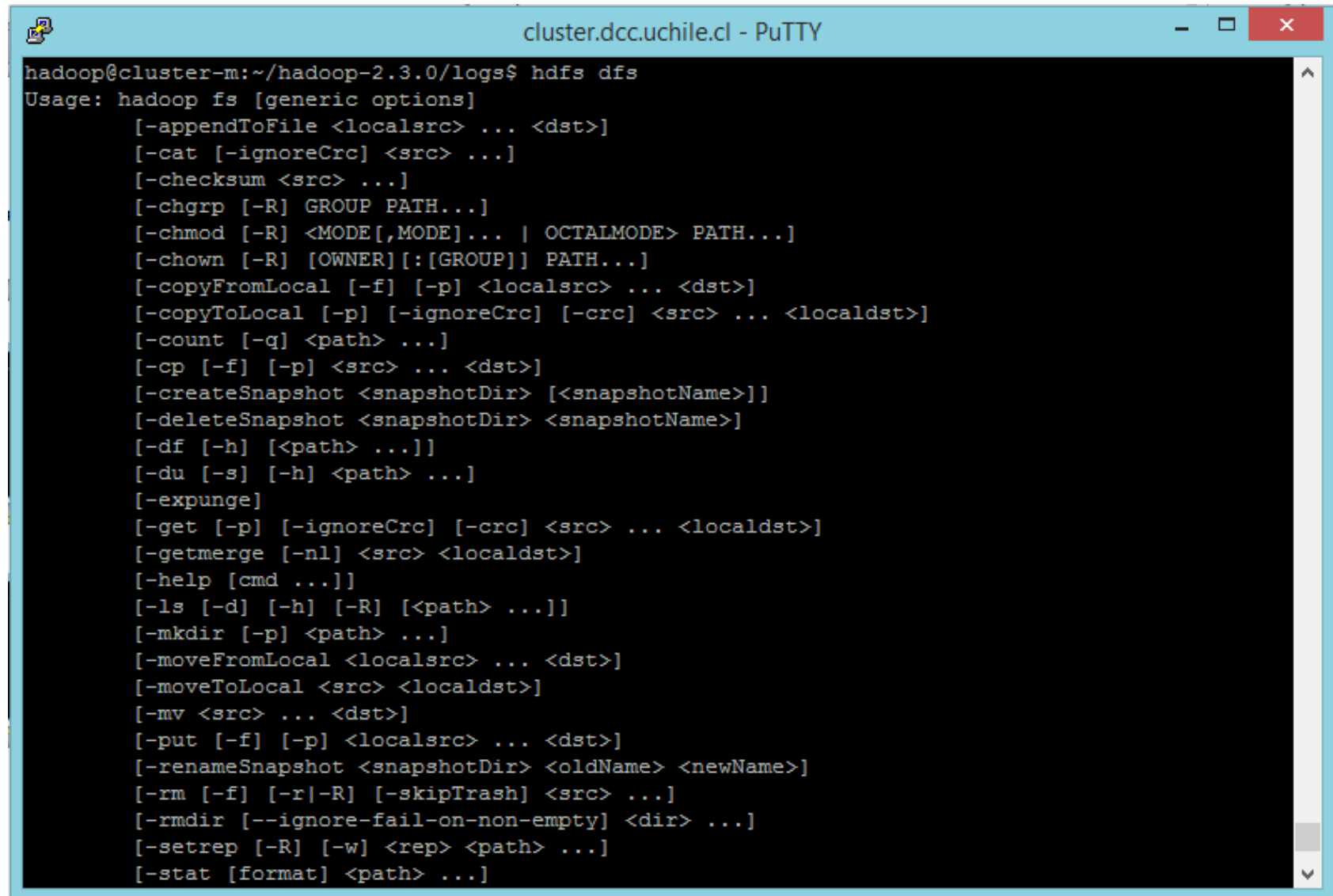
Hadoop: High Availability



PROGRAMMING WITH HADOOP

1. Input/Output (cmd)

> hdfs dfs



```
cluster.dcc.uchile.cl - PuTTY
hadoop@cluster-m:~/hadoop-2.3.0/logs$ hdfs dfs
Usage: hadoop fs [generic options]
    [-appendToFile <localsrc> ... <dst>]
    [-cat [-ignoreCrc] <src> ...]
    [-checksum <src> ...]
    [-chgrp [-R] GROUP PATH...]
    [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
    [-chown [-R] [OWNER][:[GROUP]] PATH...]
    [-copyFromLocal [-f] [-p] <localsrc> ... <dst>]
    [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
    [-count [-q] <path> ...]
    [-cp [-f] [-p] <src> ... <dst>]
    [-createSnapshot <snapshotDir> [<snapshotName>]]
    [-deleteSnapshot <snapshotDir> <snapshotName>]
    [-df [-h] [<path> ...]]
    [-du [-s] [-h] <path> ...]
    [-expunge]
    [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
    [-getmerge [-nl] <src> <localdst>]
    [-help [cmd ...]]
    [-ls [-d] [-h] [-R] [<path> ...]]
    [-mkdir [-p] <path> ...]
    [-moveFromLocal <localsrc> ... <dst>]
    [-moveToLocal <src> <localdst>]
    [-mv <src> ... <dst>]
    [-put [-f] [-p] <localsrc> ... <dst>]
    [-renameSnapshot <snapshotDir> <oldName> <newName>]
    [-rm [-f] [-r|-R] [-skipTrash] <src> ...]
    [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
    [-setrep [-R] [-w] <rep> <path> ...]
    [-stat [format] <path> ...]
```

1. Input/Output (Java)

```
public class HDFSHelloWorld {  
  
    public static final String theFilename = "hello.txt";  
    public static final String message = "Hello, world!\n";  
  
    public static void main (String [] args) throws IOException {  
  
        Configuration conf = new Configuration();  
        FileSystem fs = FileSystem.get(conf);  
  
        Path filenamePath = new Path(theFilename);  
  
        try {  
            if (fs.exists(filenamePath)) {  
                // remove the file first  
                fs.delete(filenamePath, false);  
            }  
  
            FSDDataOutputStream out = fs.create(filenamePath);  
            out.writeUTF(message);  
            out.close();  
  
            FSDDataInputStream in = fs.open(filenamePath);  
            String messageIn = in.readUTF();  
            System.out.print(messageIn);  
            in.close();  
        } catch (IOException ioe) {  
            System.err.println("IOException during operation: " + ioe.toString());  
            System.exit(1);  
        }  
    }  
}
```

Creates a file system for default configuration

Check if the file exists; if so delete

Create file and write a message

Open and read back

1. Input (Java)

InputFormat:	Description:	Key:	Value:
TextInputFormat	Default format; reads lines of text files	The byte offset of the line	The line contents
KeyValueInputFormat	Parses lines into key, val pairs	Everything up to the first tab character	The remainder of the line
SequenceFileInputFormat	A Hadoop-specific high-performance binary format	user-defined	user-defined

2. Map

Mapper<InputKeyType,
InputValueType,
MapKeyType,
MapValueType>

```
public static class CitationCountMapper extends Mapper<Object, Text, Text, IntWritable>{
```

```
    private final IntWritable one = new IntWritable(1);  
    private Text paperTitle = new Text();
```

```
    /**  
     * @throws InterruptedException  
     */  
    @Override
```

```
    public void map(Object key, Text value, Context output)  
        throws IOException, InterruptedException {
```

```
        String line = value.toString();  
        String[] paperCitedByPaper = line.split(SPLIT_REGEX);  
        paperTitle.set(paperCitedByPaper[0]);  
        output.write(paperTitle, one);
```

```
    }
```

```
}
```

(input) key: file offset.
(input) value: line of the file.
context: handles output and
logging.

Emit output

(Writable *for values*)

```
package ejemplo;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class WritableCitation implements Writable {
    public String citingPaper;
    public String citingVenue;
    public int mentions;

    public WritableCitation(String citingPaper, String citingVenue, int mentions) {
        this.citingPaper = citingPaper;
        this.citingVenue = citingVenue;
        this.mentions = mentions;
    }

    public void write(DataOutput out) throws IOException {
        out.writeUTF(citingPaper);
        out.writeUTF(citingVenue);
        out.writeInt(mentions);
    }

    public void readFields(DataInput in) throws IOException {
        citingPaper = in.readUTF();
        citingVenue = in.readUTF();
        mentions = in.readInt();
    }

    public String toString() {
        return citingPaper + "\t" + citingVenue + "\t" + mentions;
    }
}
```

Same order

(not needed in the running example)

(WritableComparable *for keys/values*)

```
public class WritableComparableCitation implements WritableComparable<WritableComparableCitation> {
```

```
    public String citingPaper;  
    public String citingVenue;  
    public int mentions;
```

New Interface

```
    public WritableComparableCitation(String citingPaper, String citingVenue, int mentions) {}  
    public void write(DataOutput out) throws IOException {}  
    public void readFields(DataInput in) throws IOException {}  
    public String toString() {}
```

Same as before

```
    public int compareTo(WritableComparableCitation other) {  
        int comp = citingPaper.compareTo(other.citingPaper);  
        if(comp==0){  
            comp = citingVenue.compareTo(other.citingVenue);  
            if(comp == 0){  
                comp = Integer.compare(mentions, other.mentions);  
            }  
        }  
        return comp;  
    }  
}
```

Needed to sort keys

```
    public boolean equals(Object o) {  
        if(o==null) return false;  
        if(o==this) return true;  
        if (!(o instanceof WritableComparableCitation)) return false;  
        WritableComparableCitation wcp = (WritableComparableCitation)o;  
        return citingPaper.equals(wcp.citingPaper) && this.citingVenue.equals(wcp.citingVenue)  
            && this.mentions == wcp.mentions;  
    }  
}
```

Needed for default
partition function

```
    public int hashCode() {  
        return citingPaper.hashCode() ^ citingVenue.hashCode() ^ mentions;  
    }  
}
```

(not needed in the
running example)

3. Partition

PartitionerInterface

```
package ejemplo;

import org.apache.hadoop.mapred.JobConf;

public class PartitionCites<E> implements Partitioner<WritableComparableCitation, E> {

    @Override
    public int getPartition(WritableComparableCitation key, E val, int machines) {
        return Math.abs(key.hashCode() % machines);
    }

    @Override
    public void configure(JobConf arg0) {
    }

}
```

(This happens to be the default
partition method!)

(not needed in the
running example)

4. Shuffle



5. Sort/Comparision

```
public class WritableComparableCitation implements WritableComparable<WritableComparableCitation> {  
    public String citingPaper;  
    public String citingVenue;  
    public int mentions;  
  
    public WritableComparableCitation(String citingPaper, String citingVenue, int mentions) {}  
    public void write(DataOutput out) throws IOException {}  
    public void readFields(DataInput in) throws IOException {}  
    public String toString() {}  
  
    public int compareTo(WritableComparableCitation other) {  
        int comp = citingPaper.compareTo(other.citingPaper);  
        if(comp==0){  
            comp = citingVenue.compareTo(other.citingVenue);  
            if(comp == 0){  
                comp = Integer.compare(mentions, other.mentions);  
            }  
        }  
        return comp;  
    }  
  
    public boolean equals(Object o) {  
        if(o==null) return false;  
        if(o==this) return true;  
        if (!(o instanceof WritableComparableCitation)) return false;  
        WritableComparableCitation wcp = (WritableComparableCitation)o;  
        return citingPaper.equals(wcp.citingPaper) && this.citingVenue.equals(wcp.citingVenue)  
            && this.mentions == wcp.mentions;  
    }  
  
    public int hashCode() {  
        return citingPaper.hashCode() ^ citingVenue.hashCode() ^ mentions;  
    }  
}
```



Methods in
WritableComparator

(not needed in the
running example)

6. Reduce

Reducer<MapKey, MapValue,
OutputKey, OutputValue>

```
public static class CitationCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
```

```
/**  
 * @throws InterruptedException  
 */
```

```
@Override
```

```
public void reduce(Text key, Iterable<IntWritable> values,  
    Context output) throws IOException, InterruptedException {
```

```
    int sum = 0;
```

```
    for(IntWritable value: values) {  
        sum += value.get();
```

```
    }
```

```
    output.getCounter("citations", key.toString().substring(0, 1)).increment(1);
```

```
    output.write(key, new IntWritable(sum));
```

```
}
```

```
}
```

key: as emitted from
map

values: iterator over
all values for that key
context for output

Write to output

7. Output / Input (Java)

```
public class HDFSHelloWorld {  
  
    public static final String theFilename = "hello.txt";  
    public static final String message = "Hello, world!\n";  
  
    public static void main (String [] args) throws IOException {  
  
        Configuration conf = new Configuration();  
        FileSystem fs = FileSystem.get(conf);  
  
        Path filenamePath = new Path(theFilename);  
  
        try {  
            if (fs.exists(filenamePath)) {  
                // remove the file first  
                fs.delete(filenamePath, false);  
            }  
  
            FSDataOutputStream out = fs.create(filenamePath);  
            out.writeUTF(message);  
            out.close();  
  
            FSDataInputStream in = fs.open(filenamePath);  
            String messageIn = in.readUTF();  
            System.out.print(messageIn);  
            in.close();  
        } catch (IOException ioe) {  
            System.err.println("IOException during operation: " + ioe.toString());  
            System.exit(1);  
        }  
    }  
}
```

Creates a file system for default configuration

Check if the file exists; if so delete

Create file and write a message

Open and read back

7. Output (Java)

OutputFormat:	Description
TextOutputFormat	Default; writes lines in "key \t value" form
SequenceFileOutputFormat	Writes binary files suitable for reading into subsequent MapReduce jobs
NullOutputFormat	Disregards its inputs

Control Flow

```
public static void main(String[] args) throws Exception {  
    Configuration conf = new Configuration();  
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();  
    if (otherArgs.length != 2) {  
        System.err.println("Usage: CitationCount <in> <out>");  
        System.exit(2);  
    }  
    String inputLocation = otherArgs[0];  
    String outputLocation = otherArgs[1];  
  
    Job job = Job.getInstance(new Configuration());  
  
    FileInputFormat.setInputPaths(job, new Path(inputLocation));  
    FileOutputFormat.setOutputPath(job, new Path(outputLocation));  
  
    job.setOutputKeyClass(Text.class);  
    job.setOutputValueClass(IntWritable.class);  
    job.setMapOutputKeyClass(Text.class);  
    job.setMapOutputValueClass(IntWritable.class);  
  
    job.setMapperClass(CitationCountMapper.class);  
    job.setCombinerClass(CitationCountReducer.class);  
    job.setReducerClass(CitationCountReducer.class);  
  
    job.setJarByClass(CitationCount.class);  
    job.waitForCompletion(true);  
}
```

Create a JobClient, a JobConf and pass it the main class

Set input and output paths

Set the type of map and output keys and values in the configuration

Set the mapper class

Set the reducer class (and optionally “combiner”)

Run and wait for job to complete.

More in Hadoop: Combiner

- Map-side “mini-reduction”
- Keeps a fixed-size buffer in memory
- Reduce within that buffer
 - e.g., count words in buffer
 - Lessens bandwidth needs
- In Hadoop: can simply use Reducer class 😊

More in Hadoop: Counters

```
public static class CitationCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {  
  
    /**  
     * @throws InterruptedException  
     */  
    @Override  
    public void reduce(Text key, Iterable<IntWritable> values,  
        Context output) throws IOException, InterruptedException {  
        int sum = 0;  
        for(IntWritable value: values) {  
            sum += value.get();  
        }  
        output.getCounter("citations", key.toString().substring(0, 1)).increment(1);  
        output.write(key, new IntWritable(sum));  
    }  
}
```

Context has a group of maps
of counters

More in Hadoop: Chaining Jobs

- Sometimes we need to chain jobs
- In Hadoop, can pass a set of Jobs to the client
- `x.addDependingJob(y)`

More in Hadoop: Distributed Cache

- Some tasks need “global knowledge”
 - For example, a white-list of conference venues and journals that should be considered in the citation count
 - Typically small
- Use a distributed cache:
 - Makes data available locally to all nodes

IF WE HAVE TIME ...

MapReduce: Scholar Example

Assume that in Google Scholar we have two inputs like:

```
paperA citedBy paperB  
paperA citedBy paperD  
paperC citedBy paperD  
paperD citedBy paperB
```

```
paperA author1 author2 author3  
paperA author2 author1  
paperC author4 author2 author5  
paperD author6
```

How can we use MapReduce to count the total incoming citations per author?

RECAP

Distributed File Systems

- **Google File System (GFS)**
 - **Master and Chunkslaves**
 - Replicated pipelined writes
 - Direct reads
 - Minimising master traffic
 - Fault-tolerance: self-healing
 - Rack awareness
 - Consistency and modifications
- **Hadoop Distributed File System**
 - **NameNode and DataNodes**

MapReduce

- 1. Input**
- 2. Map**
- 3. Partition**
- 4. Shuffle**
- 5. Comparison/Sort**
- 6. Reduce**
- 7. Output**

MapReduce/GFS Revision

- GFS: distributed file system
 - Implemented as HDFS
- MapReduce: distributed processing framework
 - Implemented as Hadoop

Hadoop

- `FileSystem`
- `Mapper<InputKey, InputValue, MapKey, MapValue>`
- `OutputCollector<OutputKey, OutputValue>`
- `Writable, WritableComparable<Key>`
- `Partitioner<KeyType, ValueType>`
- `Reducer<MapKey, MapValue, OutputKey, OutputValue>`
- `JobClient/JobConf`

...