Hadoop in Action

Justin Quan

March 15, 2011

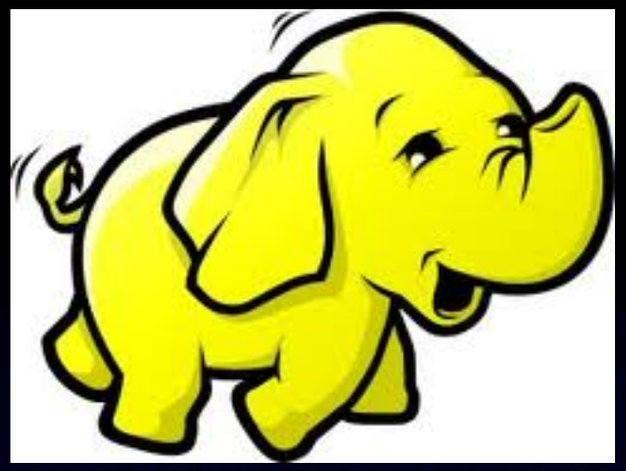
Poll

What's to come

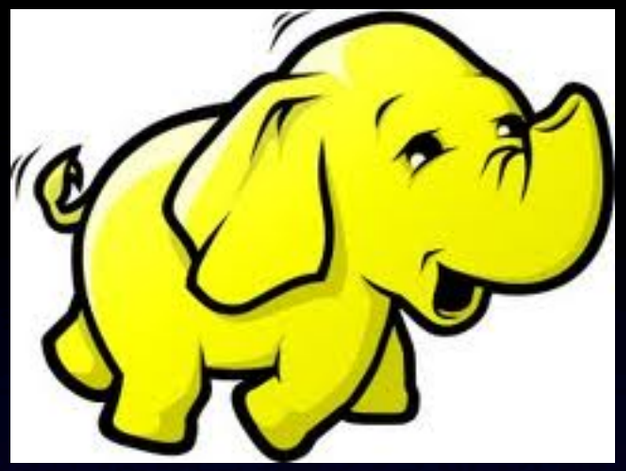
- Overview of Hadoop for the uninitiated
- How does Hadoop work?
- How do I use Hadoop?
- How do I get started?
- Final Thoughts

Key Take Aways

- Hadoop is a widely used open source framework for process large datasets with multiple machines
- Hadoop is a tool simple enough to add to your back pocket



What is Hadoop?



What is Hadoop?

"Hadoop is one way of using an enormous cluster of computers to store an enormous amount of data and then operate on that data in parallel."

-Keith Wiley

History of Hadoop

- Open source implementation of the Google File System (GFS) and MapReduce framework (2004)
- Suited for processing large data sets (page rank)
- Doug Cutting joins Yahoo in 2006 and spearheads the open source implementation

Motivation for Hadoop

- Processing large datasets requires lots of cpu, I/O, bandwidth, memory
- Large scale means failures
- Adding fault tolerance to your app is hard
- Hadoop to the rescue! Let application developers worry about writing applications

What does Hadoop Provide?

- Decouples distributed systems fault tolerance from application logic
- Scalable storage (just add nodes)
- System that can tolerate machine failure
- Distributes your data processing code to take advantage of idle CPUs and data locality

Hadoop Limitations

- Data privacy
- small datasets
- realtime processing

How does Hadoop work?

How does Hadoop work?

How does it store data?
How does it process data?

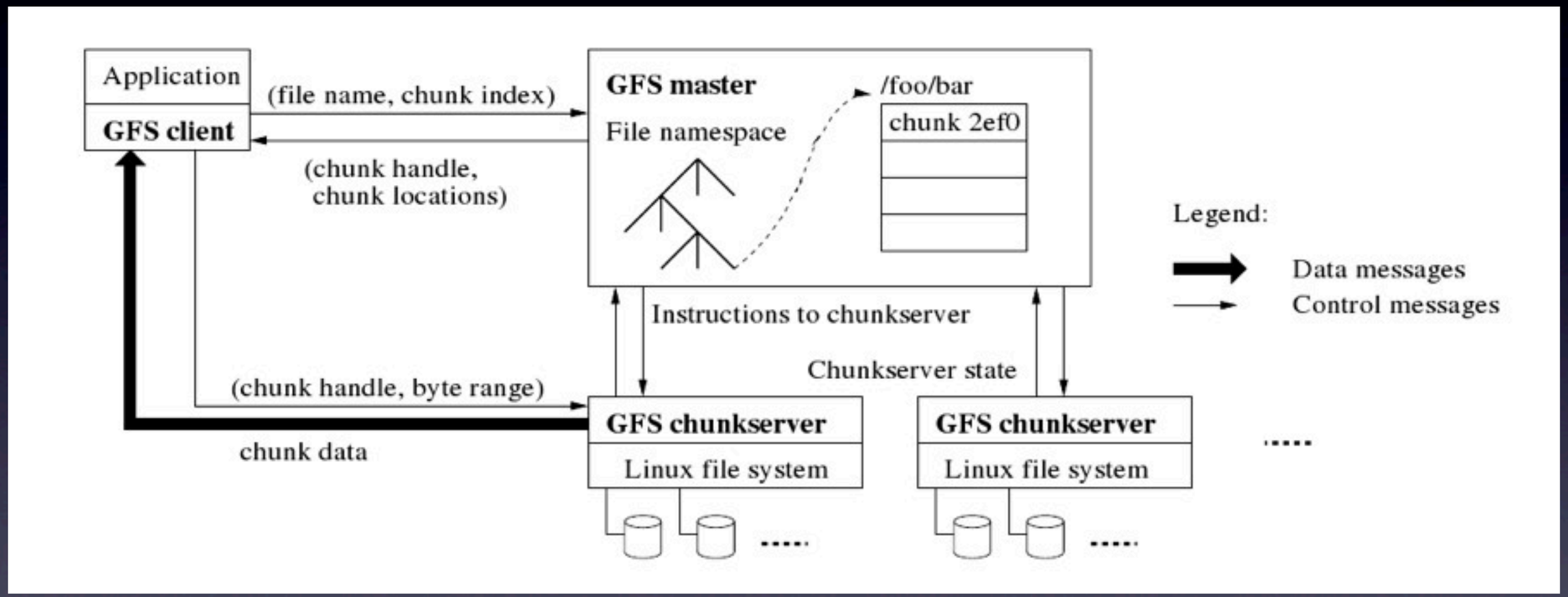
HDFS: Overview

- Shared namespace for the entire cluster (/user/justin/todo.txt)
- Write once. Append ok.
- unix like access: ls, df, du, mv, cp, rm, cat, chmod, chown, etc...
- use -put -get to move between local filesystem and hdfs

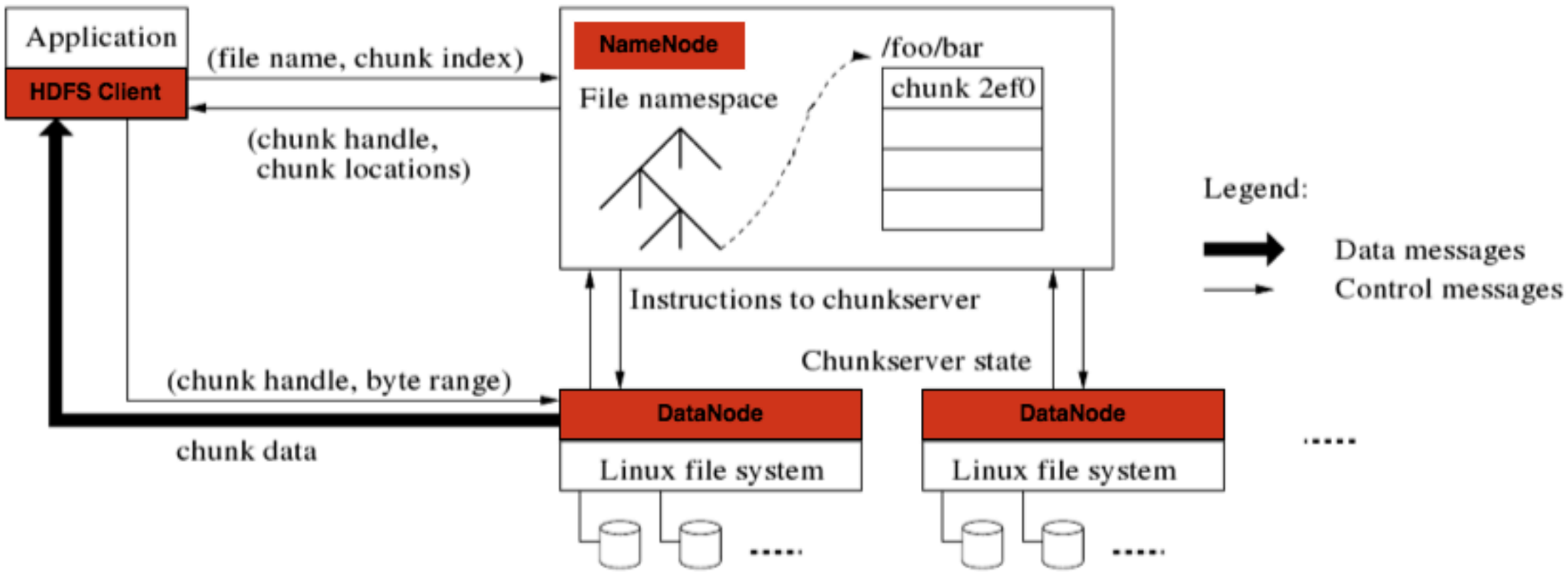
HDFS: Underneath

- Files are split up in large chunks (64Mb+)
- Replication for durability. Default=3 copies
- Single 'NameNode' tracks filenames, permissions, block locations, cluster config, transaction log
- Many 'DataNodes', each stores blocks on the local filesystem

HDFS: How it works



HDFS: How it works



Data Processing

- Map Reduce Paradigm
- Borrows constructs similar to those found in functional programming languages
- Map and Reduce gets called on a list of key value pairs of unknown length

Map Reduce

- Two-stage processing
- Mappers produce intermediate results
- Reduce aggregates and consolidates results

Map

`Map(K,V) -> (`K_i`,`V_i`) list`

- Each invocation is fed a key value pair
- Each invocation returns 0 or more key value pairs

Map Examples

Wordcount App:

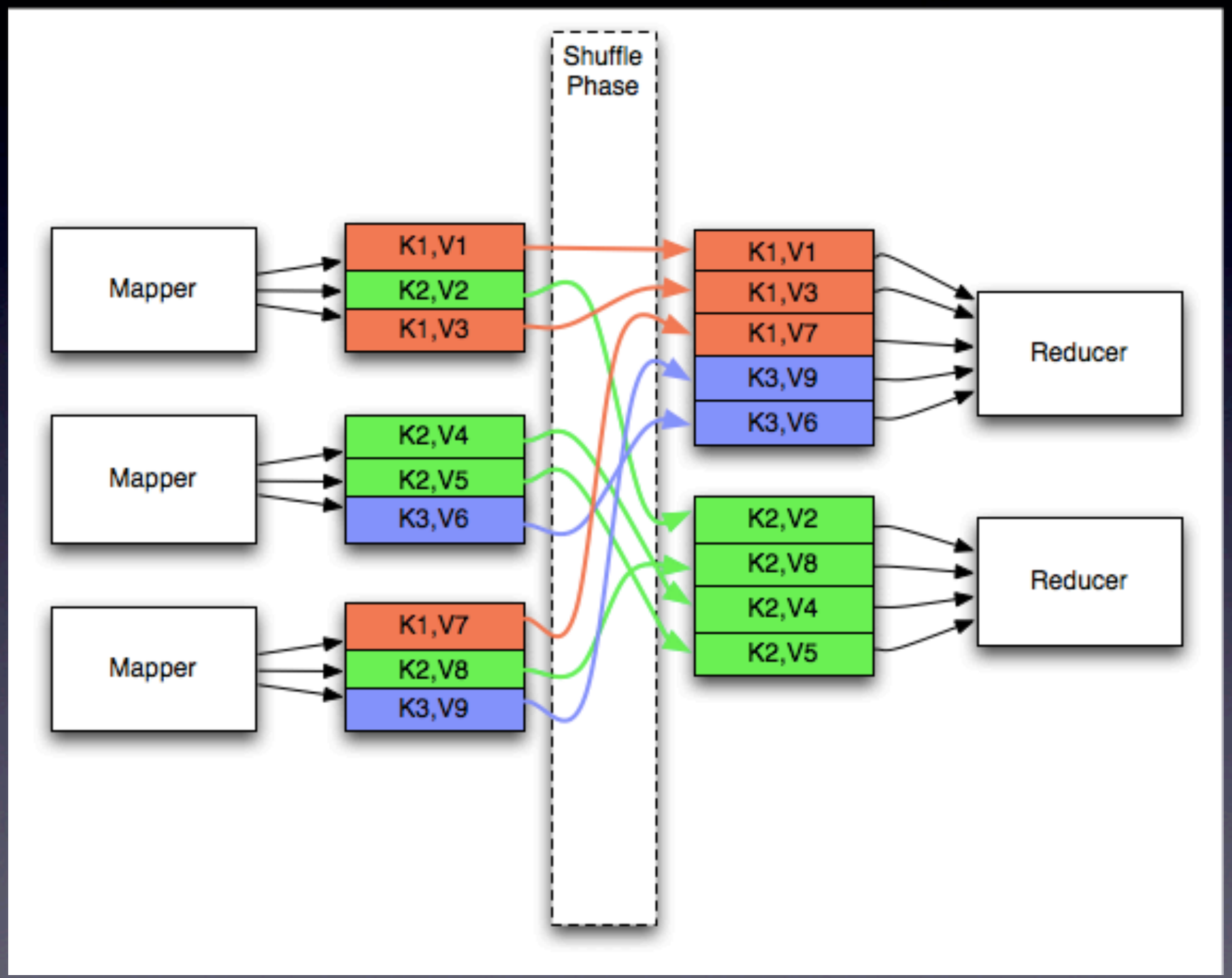
K is the file line number, V is the line of text

```
def map(k,v):
  wordList = v.split(" ")
  foreach word in wordList:
    output(word, 1)
  end
end

...

map(421, "around and around")
output(around, 1)
output(and, 1)
output(around, 1)
```

Shuffle Phase

- Done automatically
- Each Mapper sorts output by key
- Hashes key to send data to appropriate reducer

Shuffle Phase



Reduce

`Reduce(K,`V_i` list) -> (`K_j`,`V_j`) list`

- Each invocation is fed a key and list of values
- Each invocation returns 0 or more key value pairs

Reduce Examples

Wordcount App:

```
def reduce(k,v):
  count = 0
  foreach value in v:
    count += value
  end
  output(k, count)
end

...

reduce(and, [1])
output(and, 1)
reduce(around, [1, 1])
output(around, 2)
```

Combiner

- An optionally run reducer for Mapper output
- Goal is to reduce size of mapper output
- No guarantee to run, so don't depend on it!

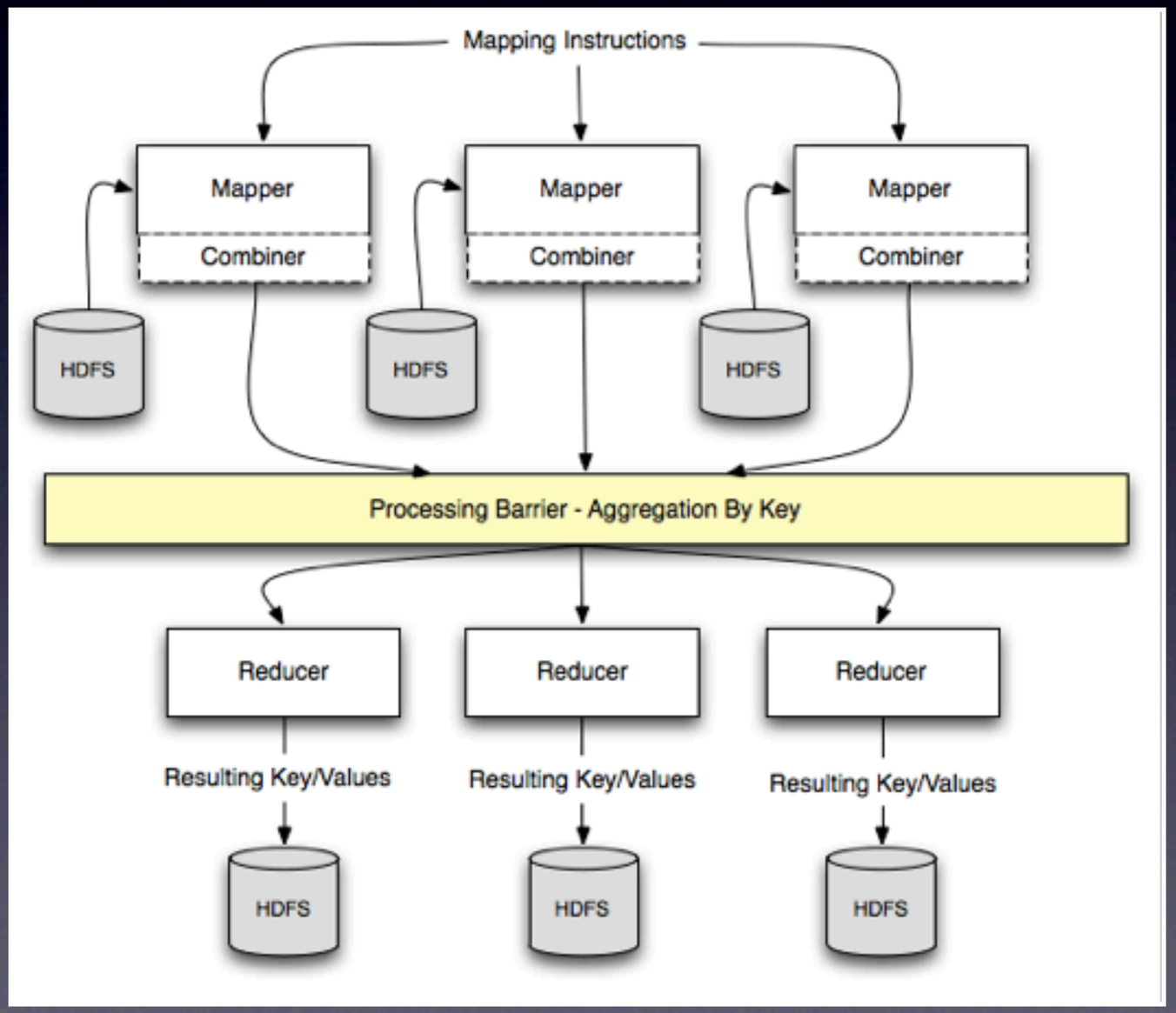
Combiner Example

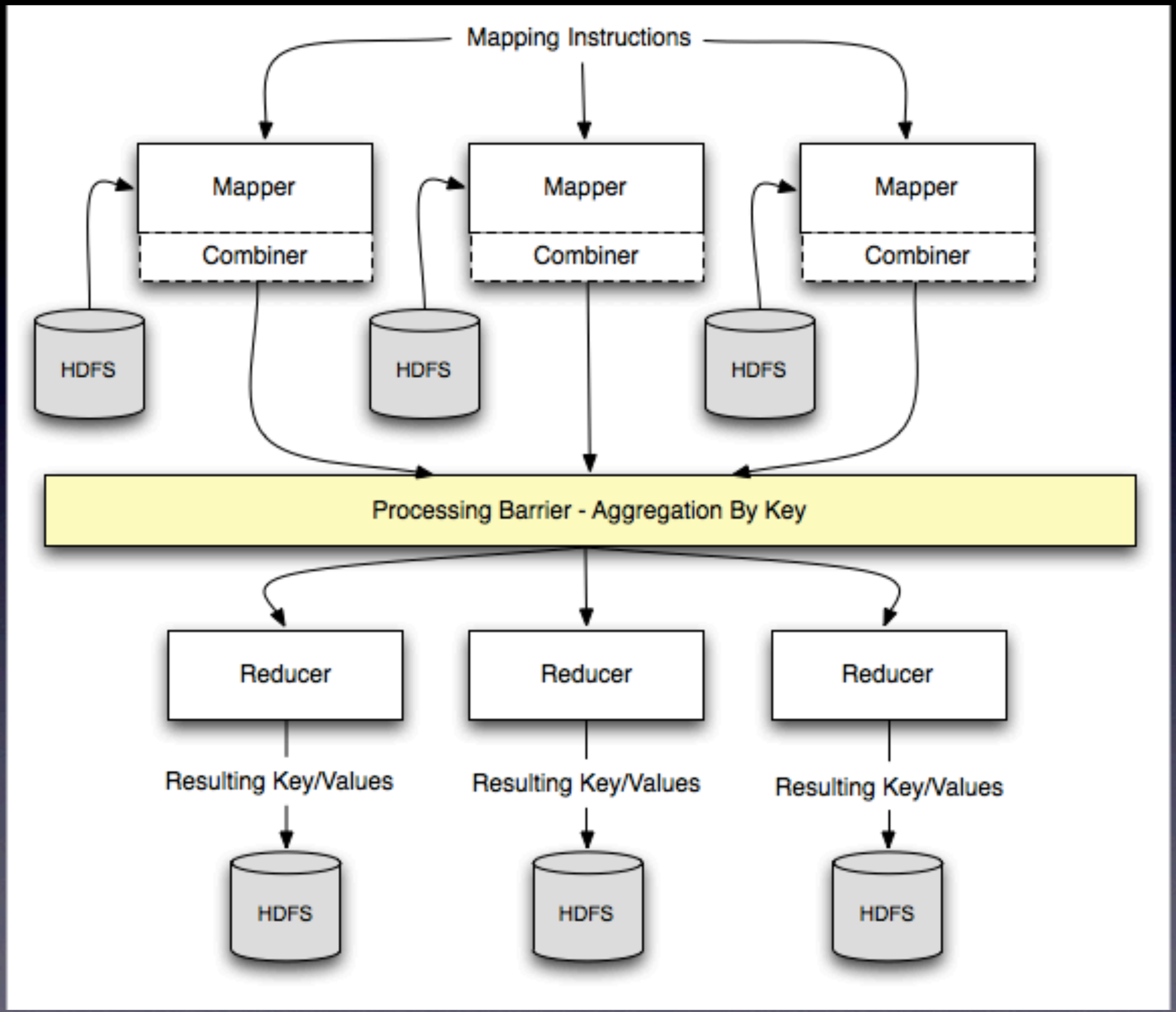
Wordcount App:

```
combine(k,v)
  count = 0
  foreach value in v
    count += value
  end
  output(k, count)
end

e.g.

combine(and, [1])
output(and, 1)
combine(around, [1,1])
output(around, 2)
```

Hadoop Map Reduce Data Processing Workflow

1. Map function is distributed
2. Data read from HDFS
3. Data fed to mapper
4. Output fed to Combiner
5. Output aggregated by key
6. Data distributed to reducers
7. Output written to HDFS




Mapping Instructions
Mapper
Combiner
Mapper
Combiner
Mapper
Combiner
HDFS
HDFS
HDFS
Processing Barrier - Aggregation By Key
Reducer
Reducer
Reducer
Resulting Key/Values
Resulting Key/Values
Resulting Key/Values
HDFS
HDFS
HDFS

Real Example

Google Ngram Dataset

- 1-gram dataset
- file format:

  `<ngram><tab><year><tab><count><tab><pages><tab><books>`

- map: extract nword + count
- reduce: sum ngram counts

justin-desktop-2 Hadoop Map/Reduce Administration

State: RUNNING
Started: Wed Mar 09 00:41:45 EST 2011
Version: 0.21.0, 985326
Compiled: Tue Aug 17 01:02:28 EDT 2010 by tomwhite from branches/branch-0.21
Identifier: 201103090041

Cluster Summary (Heap Size is 88.88 MB/888.94 MB)

Queues	Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Ma Ca
1	4	0	30	2	4	0	0	0	4

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce Comple
job_201103090041_0037	NORMAL	hadoop	streamjob5518569008219750944.jar	0.00%	15	0	0.00%

Retired Jobs

Jobid	Priority	User	Name	State	Start Time	Finish T
job_201103090041_0035	NORMAL	hadoop	streamjob1099387696950201093.jar	SUCCEEDED	Mon Mar 14 16:41:03 EDT 2011	Mon Mar 16:47:29

Hadoop job_201103090041_0037 on justin-desktop-2

User: hadoop
Job Name: streamjob5518569008219750944.jar
Job File: hdfs://justin-desktop-2:9000/tmp/hadoop-hadoop/mapred/staging/hadoop/.staging/job_201103090041_0037/job.xml
Job Setup: Successful
Status: Running
Started at: Tue Mar 15 01:33:31 EDT 2011
Running for: 1mins, 11sec
Job Cleanup: Pending

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	44.93%	15	5	4	6	0	0 / 0
reduce	8.88%	1	0	1	0	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	SLOTS_MILLIS_MAPS	0	0	216,760
	Launched reduce tasks	0	0	1
	Rack-local map tasks	0	0	6
	Launched map tasks	0	0	10
	Data-local map tasks	0	0	4
FileInputFormatCounters	BYTES_READ	402,653,193	0	402,653,193
FileSystemCounters	FILE_BYTES_READ	250,957,892	0	250,957,892
	HDFS_BYTES_READ	402,678,594	0	402,678,594
	FILE_BYTES_WRITTEN	501,915,940	0	501,915,940
	Map input records	20,128,016	0	20,128,016

Map Completion Graph - close

Reduce Completion Graph - close

Hard numbers

- 10 gigs of data
- 475 million lines
- Hadoop running with 1 gig memory
- 2 machines: 2x2.33ghz, 4x3ghz
- Word count job: 5 hours 18 minutes

How do I use Hadoop?

Install

- Download from http://hadoop.apache.org/
- tar -xzvf hadoop-0.21.0.tar.gz
- Need Java

Components

- One NameNode (coordinates HDFS)
- One JobTracker (coordinates data processing)
- Many DataNodes (stores HDFS blocks)
- Many TaskTrackers (runs mapper and reducer work)

Setup Config

- Configure NameNode (HDFS)
  - core-site.xml, hdfs-site.xml
- Configure JobTracker (Map/Reduce)
  - mapred-site.xml
- on every host
- slaves file on master hosts

core-site.xml

fs.default.name: NameNode endpoint

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://yourhost:9000</value>
  </property>
</configuration>
```

hdfs-site.xml

```
dfs.name.dir: directory to store NameNode data
dfs.data.dir: directory to store DataNode data

<configuration>
  <property>
    <name>dfs.name.dir</name>
    <value>/home/hadoop/hdfs/name/</value>
  </property>
  <property>
    <name>dfs.data.dir</name>
    <value>/home/hadoop/hdfs/data/</value>
  </property>
</configuration>
```

mapred-site.xml

```
mapred.job.tracker: JobTracker endpoint
mapred.local.dir: directories to store mapper output

<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>yourhost:9001</value>
  </property>
  <property>
    <name>mapred.local.dir</name>
    <value>/home/justin/mapred</value>
  </property>
</configuration>
```

slave

Define DataNodes and TaskTrackers

One host per line, used by NameNode and JobTracker to startup daemons

```
yourhost1.domain
yourhost2.domain
yourhost3.domain
```

Starting Up Hadoop

Format hdfs

```
$HADOOP_HOME/bin/hadoop namenode -format
```

Start NameNode

```
$HADOOP_HOME/bin/start-dfs.sh
```

Start JobTracker

```
$HADOOP_HOME/bin/start-mapred.sh
```

Hadoop Streaming

- Not a Java person?
- Streaming provides command-line interface
- Mapper: Provided
  - `<key><tab><value><newline>`
- Reducer: Provided
  - `<key><tab><value><newline>`

Quicky Ruby MR script

Mapper:

```
for line in STDIN.each_line do
  parts = line.split(/\t/)
  print "#{parts[0]}\t#{parts[2]}\n"
end
```

Reducer:

```
sum = {}
for line in STDIN.each_line do
    parts = line.split(/\t/)
    val = sum[parts[0]] || 0
    sum[parts[0]] = val+parts[1].to_i
end
for k in sum.keys do
    print "#{k}\t#{sum[k]}\n"
end
```

Hadoop Streaming Execution

```
$HADOOP_HOME/bin/hdfs dfs -put
/home/justin/localData/* inputDir/

$HADOOP_HOME/bin/hadoop jar hadoop-streamin.jar \
-input inputDir/* \
-output outputDir \
-mapper 'ruby map.rb' \
-reducer 'ruby reduce.rb' \
-file map.rb \
-file reduce.rb \
```

How do I get started?

- Borrow access on peer desktops
- Look to the cloud: Amazon Webservices
- Build your own cloud

Recap

- Overview and motivation for using Hadoop
- How Hadoop works
- How to use Hadoop
- How to get started

Final Thoughts

- Hadoop is a widely used open source framework for processing large datasets with multiple machines
- Hadoop is a tool simple enough to add to your back pocket

EOF

How to interface with the Mapper

- Hadoop comes with a variety of ways to get data in
- Pig for different languages other than native Java
- Hadoop-Streaming comes bundled, for command line style execution