Computing at Scale: Meet Hadoop

Edgar Meij

Outline

- Background
  - key ideas and intuitions
  - programming paradigm
- Hadoop in action
- The broader picture
  - Hadoop ecosystem
  - current developments and outlook
- Resources

Background

Data data data

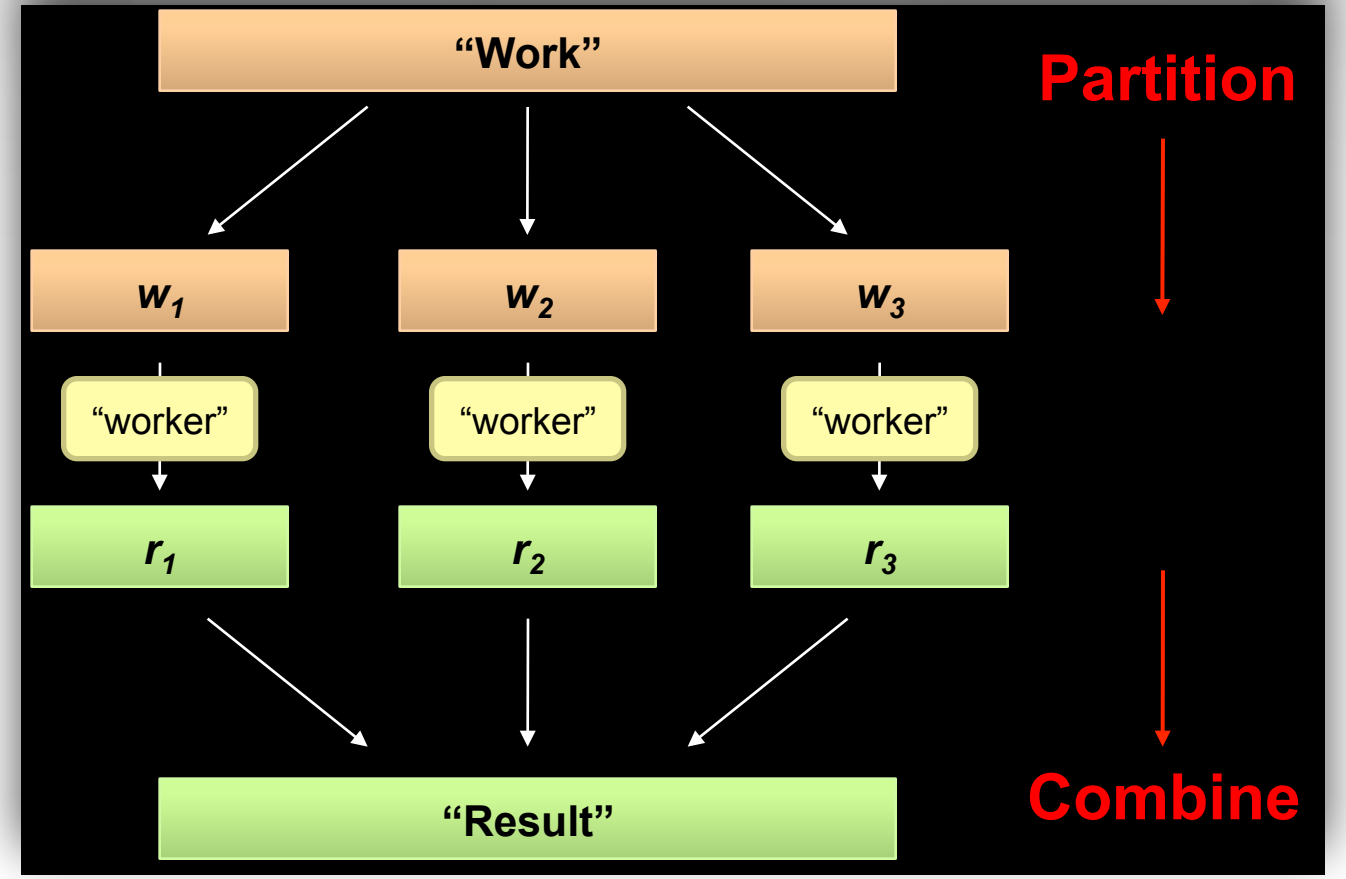
- Amount of data created and replicated in 2012: ~2.8 ZB
  - 1 Zettabyte = 1 Billion TB
- LHC generates ~15 PB per year
- Google processes 20 PB / day (2008)
- Facebook
  - 500+ TB of new data added / day (2012)
  - 60+ PB of storage
- etc…

Parallel computing is non-trivial

- Scheduling, synchronization, data distribution, fault tolerance, …
- Architectural issues…
- Programming models (message passing, shared memory, …)
- Deadlocks, racing conditions, queues, …
- *I want to develop/implement new algorithms, not debug such issues*

What is the (or “a”) solution?

- Hide system-level details: separate the *what* from the *how*
  - specify the computation that needs to be performed, the execution framework handles the actual execution
- Avoid random access
- Move processing to the data
- Scale out instead of up: ideal scaling characteristics
  - twice the data, twice the running time
  - twice the resources, half the running time
  - why can’t we typically achieve this?
    - synchronization requires communication and communication kills performance


"Work"
Partition
w_1
w_2
w_3
"worker"
"worker"
"worker"
r_1
r_2
r_3
"Result"
Combine

MapReduce (2004)

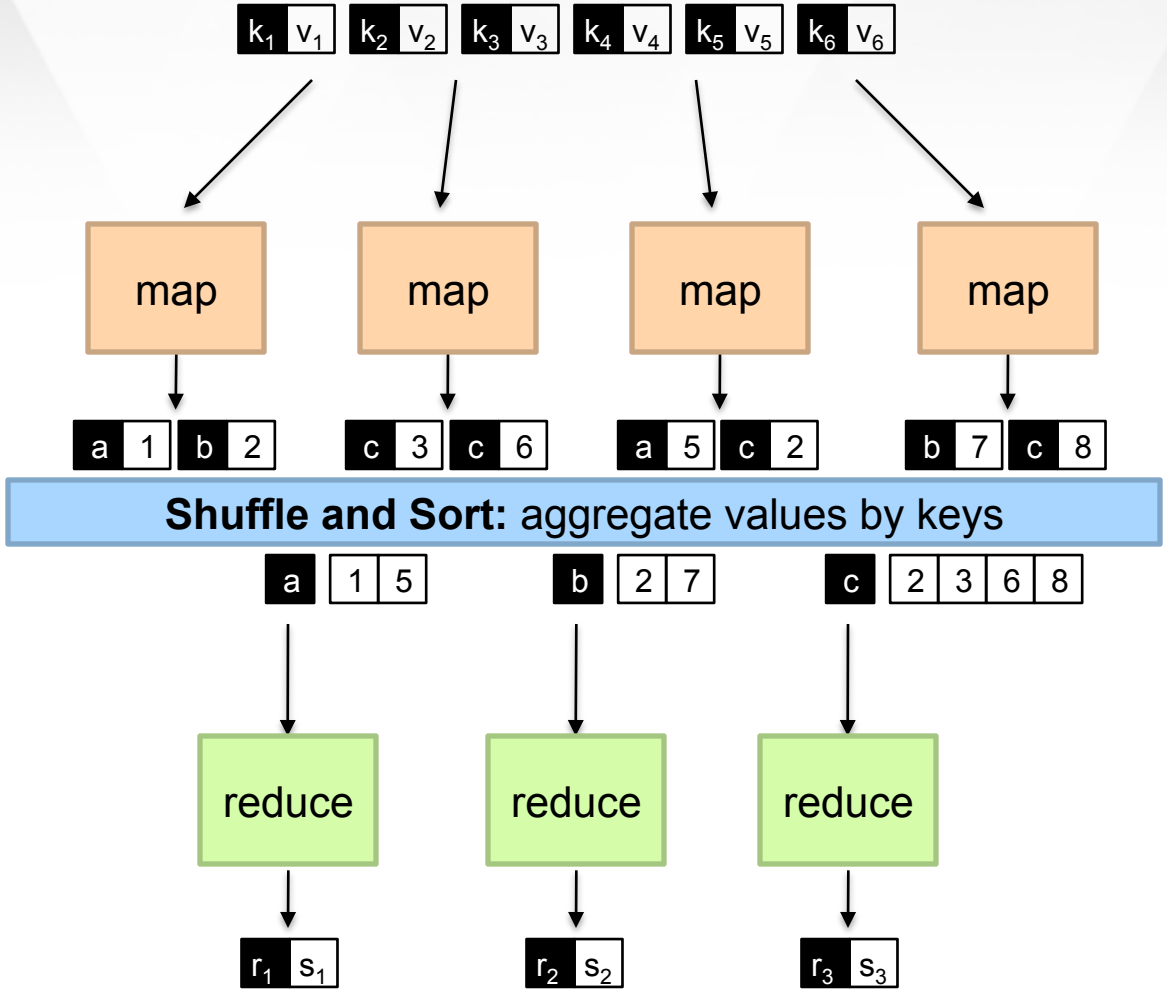
- Typical large data problem
  - iterate over (a large number of) records
  - extract something of interest from each — **Map**
  - shuffle and sort intermediate results
  - aggregate intermediate results — **Reduce**
  - generate final output
- Key idea
  - provide a functional abstraction for these two operations

MapReduce

- Developer specifies two functions:
  map (k, v) → <k', v'>*
  reduce (k', v') → <k', v'>*
  - All values with the same key are sent to the same reducer
- The execution framework handles everything else…

Word count example, in pseudocode

```
Map (String linenumber, String text):
      for each word w in text:
            Emit(w, 1);

Reduce (String term, Iterator<Int> values):
      int sum = 0;
      for each v in values:
            sum += v;
            Emit(term, sum);
```

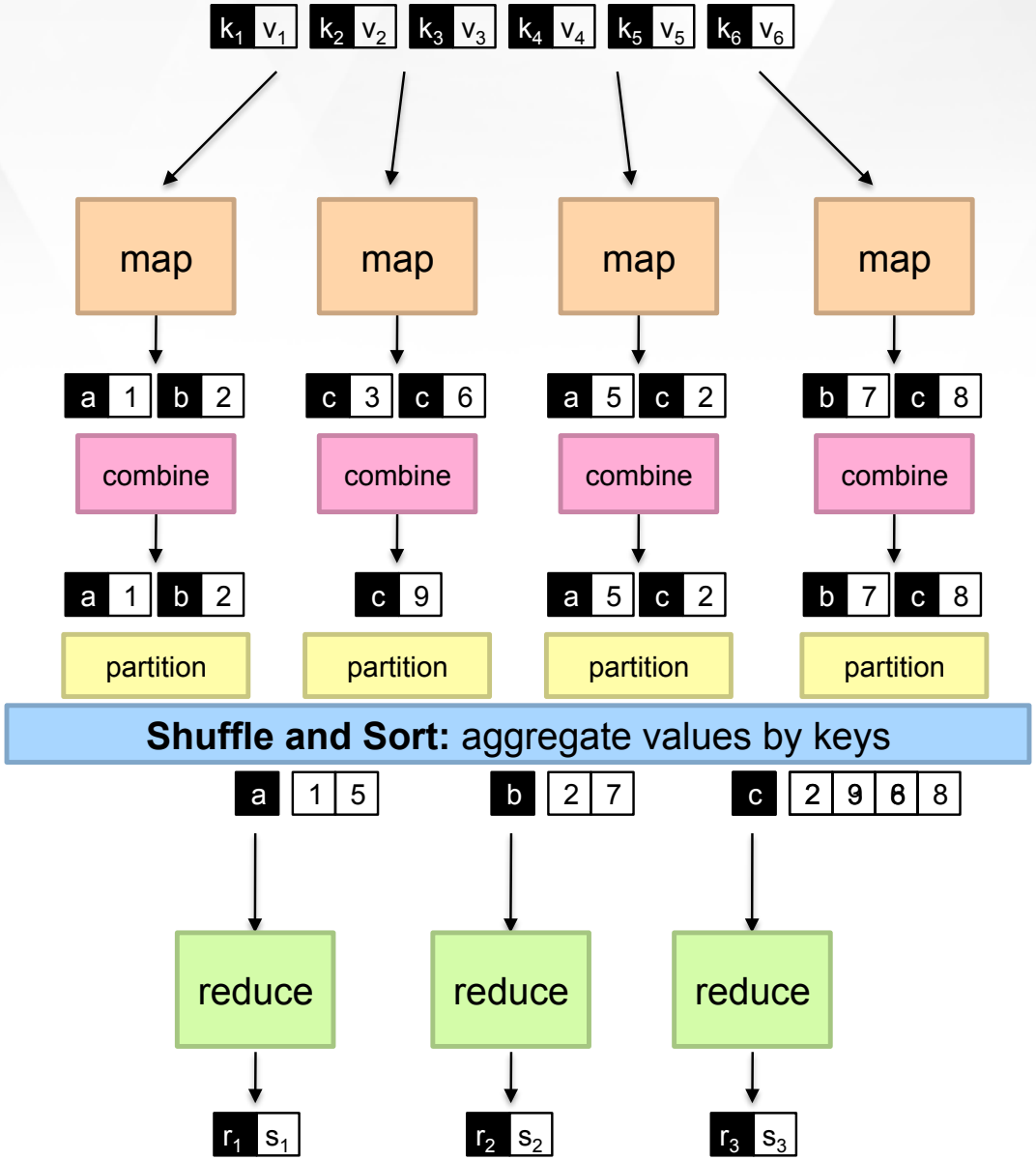

k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1
b 2
c 3
c 6
a 5
c 2
b 7
c 8
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 3 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

MapReduce

- Developer specifies two functions:
  map (k, v) → <k', v'>*
  reduce (k', v') → <k', v'>*
  - All values with the same key are sent to the same reducer
- The execution framework handles everything else…

MapReduce

- Developer specifies two functions:
  map (k, v) → <k', v'>*
  reduce (k', v') → <k', v'>*
  - All values with the same key are sent to the same reducer
- The execution framework handles everything else…
- Not quite… you can also specify…
- **partition** (k', number of partitions) → partition for k'
  - Often a simple hash of the key – e.g., hash(k') mod n – that divides up key space for parallel reduce operations
- **combine** (k', v') → <k', v'>*
  - Mini-reducers that run in memory after the map phase, used as an optimization to reduce network traffic


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

YAHOO!

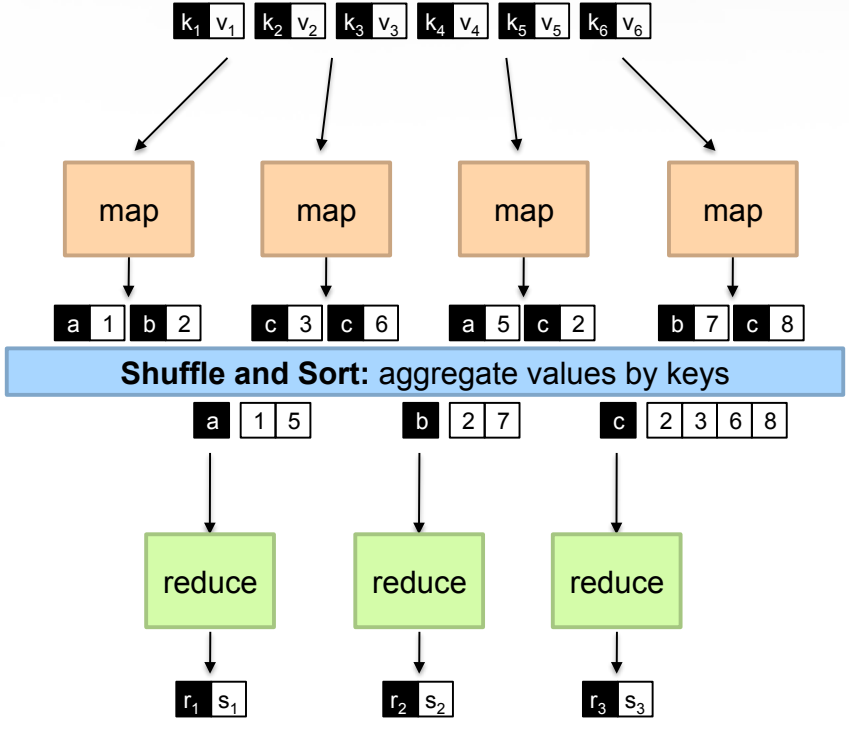
MapReduce runtime

- Handles
  - scheduling: assigns workers to map and reduce tasks
  - “data distribution”
  - synchronization: gathers, sorts, and shuffles intermediate data
  - errors and faults: detects worker failures and restarts
- On top of a distributed FS

MapReduce

- MapReduce can refer to
  - the programming model
  - the execution framework (aka “runtime”)
  - the specific implementation
- Google has a proprietary implementation in C++
- Hadoop is an open-source implementation in Java
  - original development led by Yahoo
  - now an Apache open source project
  - emerging as the de facto big data stack
  - big software ecosystem
- Lots of custom research implementations
  - for GPUs, cell processors, etc.
  - includes variations of the basic programming model

What is Hadoop?



YAHOO!

What is Hadoop?

- A simple distributed programming model (MapReduce)
- Distributed file system (HDFS)
- Plus some admin

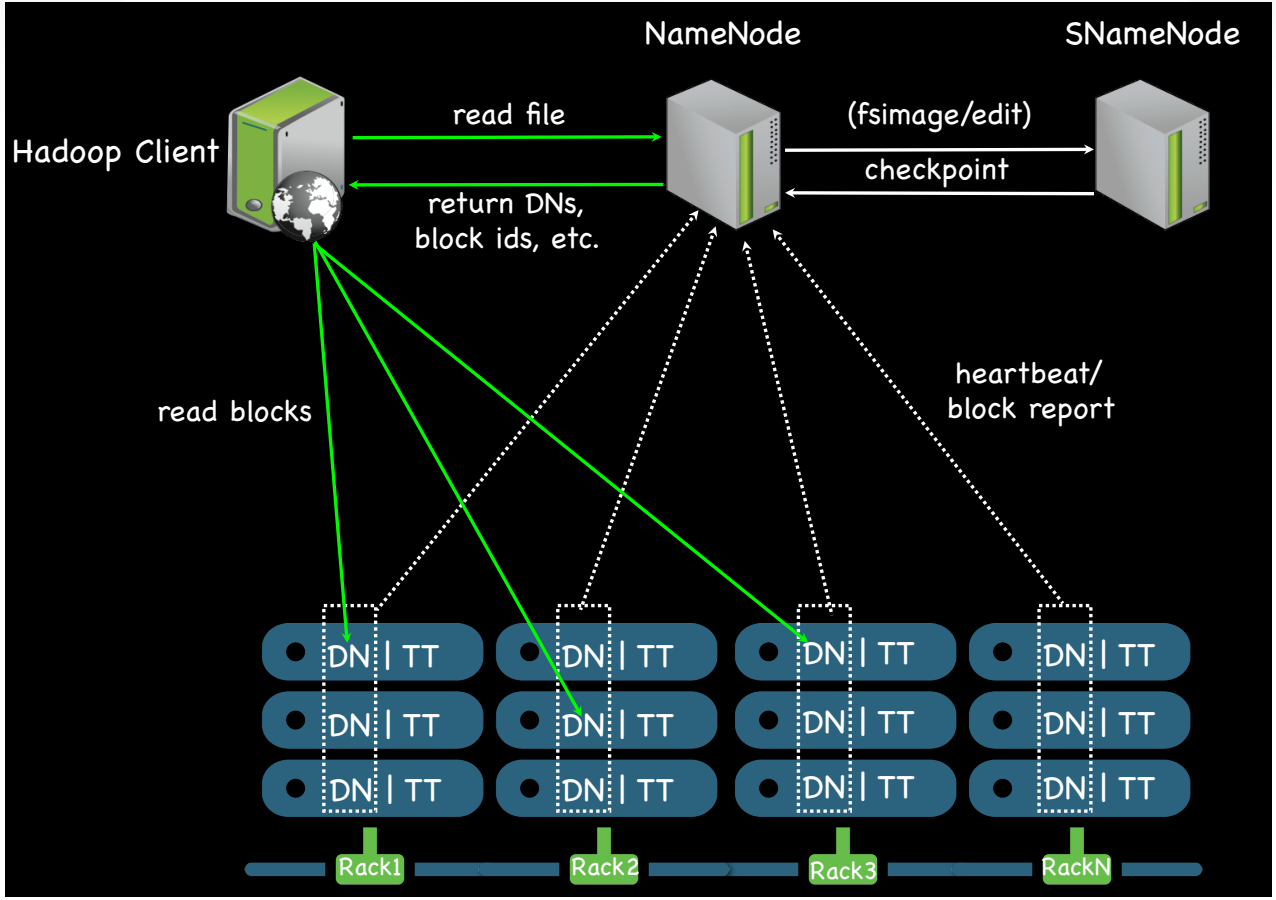
DFS

- Don’t move data to workers… move workers to the data!
  - store data on the local disks of nodes in the cluster (and replicate)
  - start up the workers on a node that has the data local
- A distributed file system is the answer
  - GFS (Google File System) for Google’s MapReduce
  - HDFS (Hadoop Distributed File System) for Hadoop

GFS/HDFS

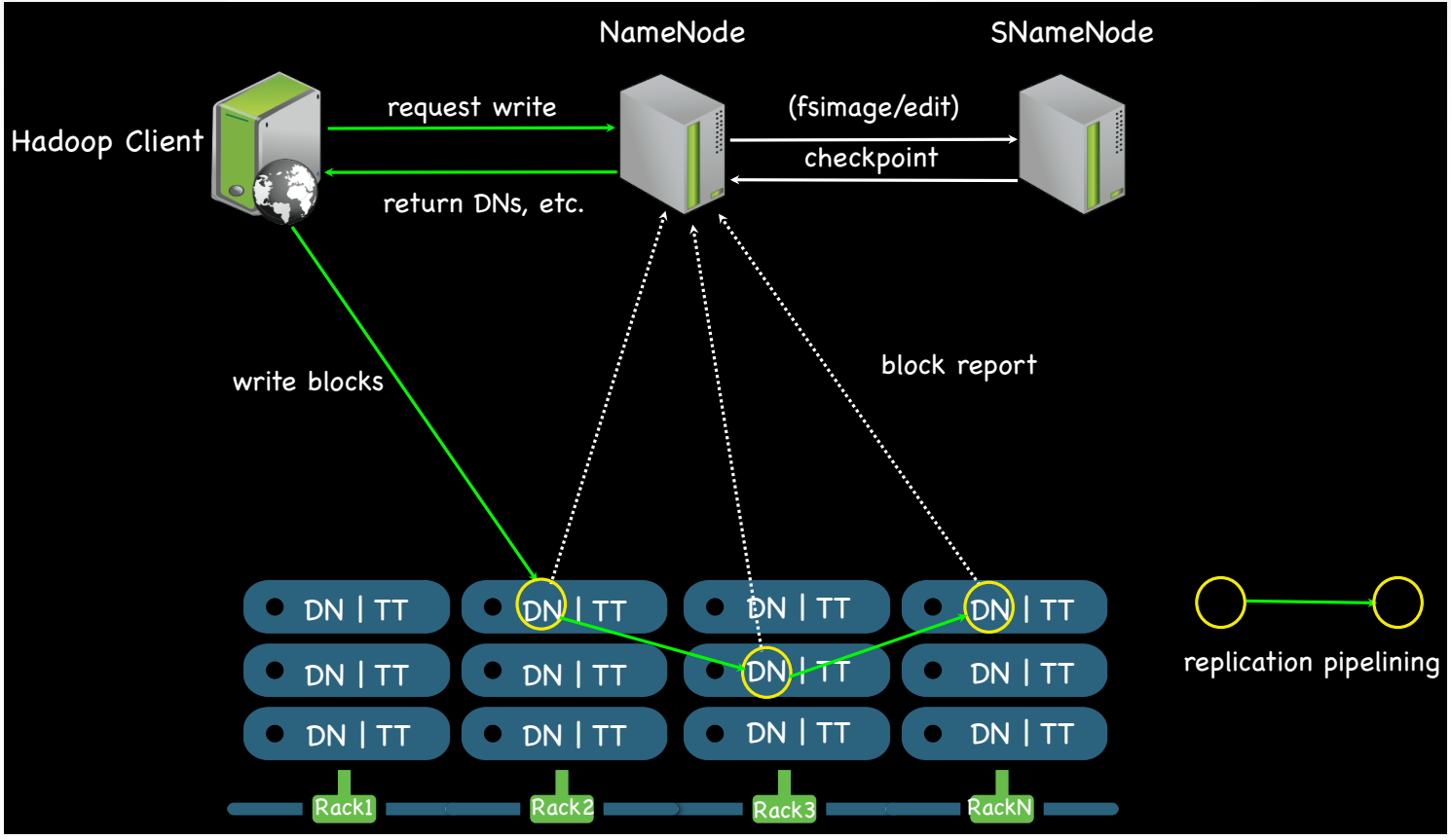
- Files stored as chunks of a fixed size (64MB)
- Reliability through replication: each chunk 3+ times replicated
- Single master to coordinate access, keep metadata
  - simple centralized management
- Simple API
  - push some of the issues (e.g., data layout) to the client

Reading files



YAHOO!

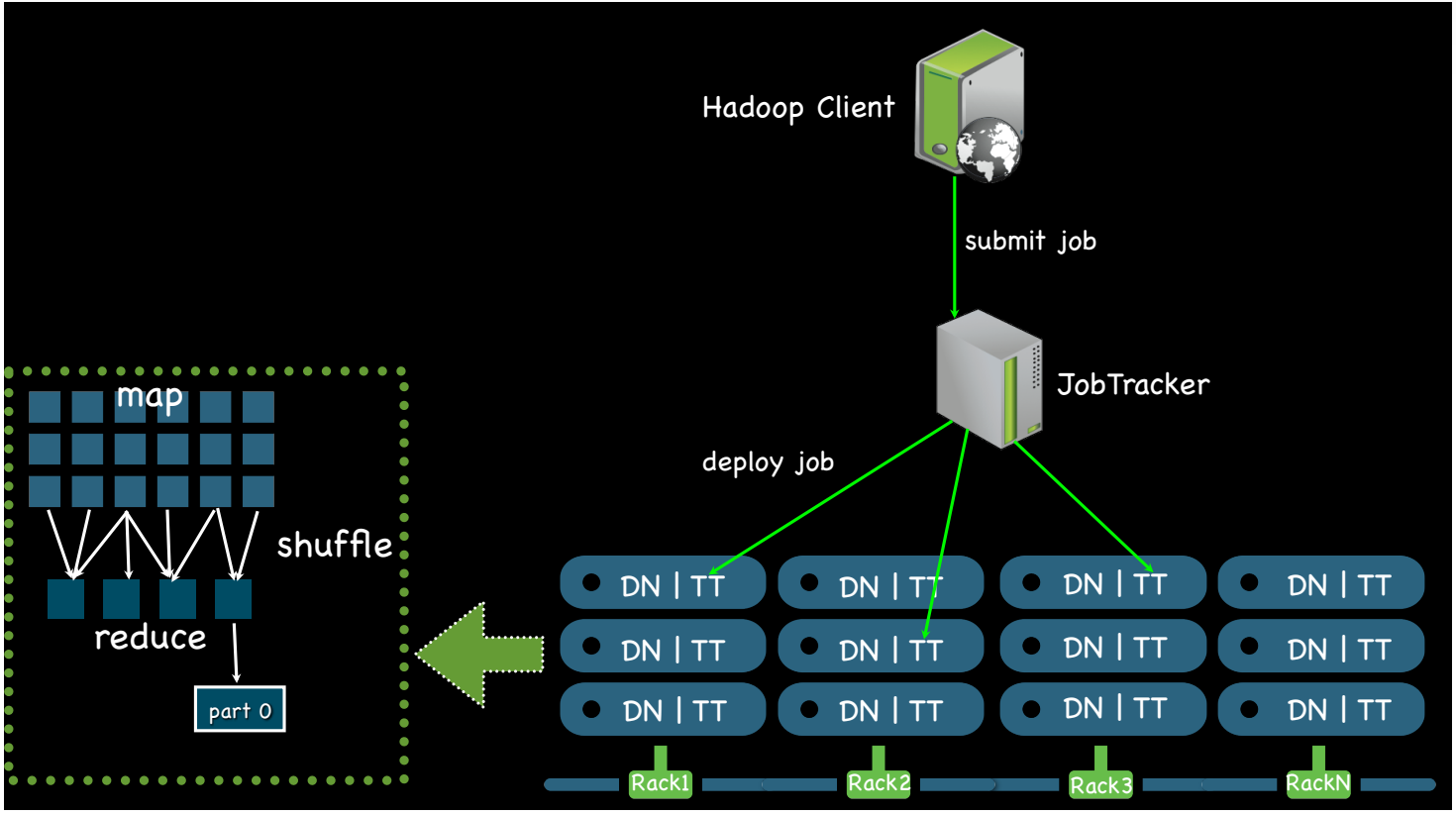
Writing files



Namenodes

- Manage the file system namespace
  - holds file/directory structure, metadata, file-to-block mapping, access permissions, etc.
- Coordinate file operations
  - directs clients to datanodes for reads and writes
  - no data is moved through the namenode
- Maintain overall health
  - periodic communication with the datanodes (heartbeats)
  - block re-replication and rebalancing
  - garbage collection

Running jobs



The execution framework handles everything else…

- The framework handles
  - scheduling: assign workers to map and reduce tasks
  - “data distribution”: move processes to data
  - synchronization: gather, sort, and shuffle intermediate data
  - errors and faults: detect worker failures and restarts
- Limited control over data and execution flow
  - Everything is expressed in m, r, c, p
- You don’t know
  - where mappers and reducers run
  - when a mapper or reducer begins or finishes
  - which input a particular mapper is processing
  - which intermediate key a particular reducer is processing

Hadoop in action

An example: counting words

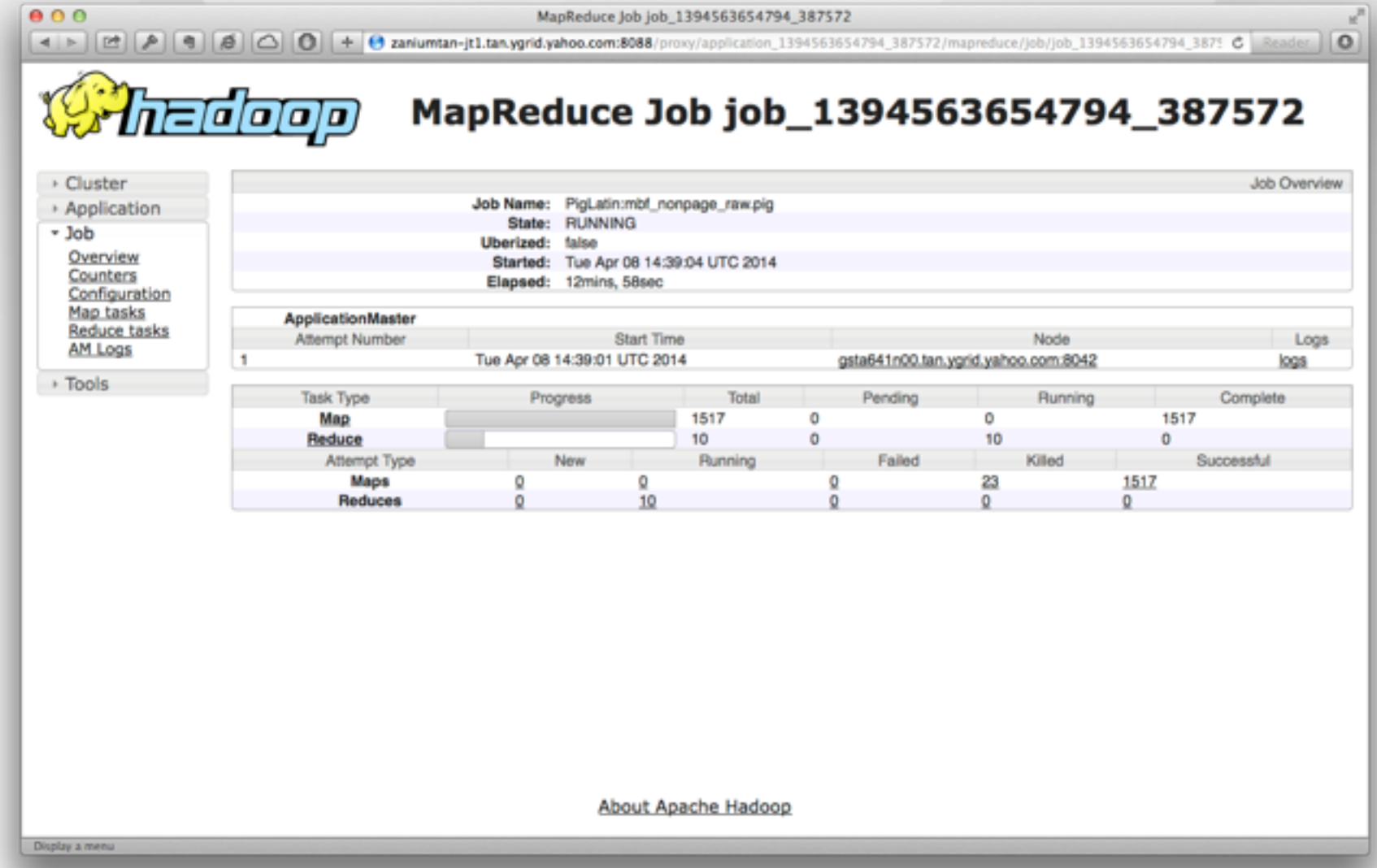
```java
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

  public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedI
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
  }

  public static void main(String[] args) throws Exception {

    Configuration conf = new Configuration();
    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
  }

}
```

```
41      }
42    }
43
44    public static void main(String[] args) throws Exception {
45
46      Configuration conf = new Configuration();
47      Job job = new Job(conf, "wordcount");
48
49      job.setOutputKeyClass(Text.class);
50      job.setOutputValueClass(IntWritable.class);
51
52      job.setMapperClass(Map.class);
53      job.setReducerClass(Reduce.class);
54
55      job.setInputFormatClass(TextInputFormat.class);
56      job.setOutputFormatClass(TextOutputFormat.class);
57
58      FileInputFormat.addInputPath(job, new Path(args[0]));
59      FileOutputFormat.setOutputPath(job, new Path(args[1]));
60
61      job.waitForCompletion(true);
62    }
63
64  }
```

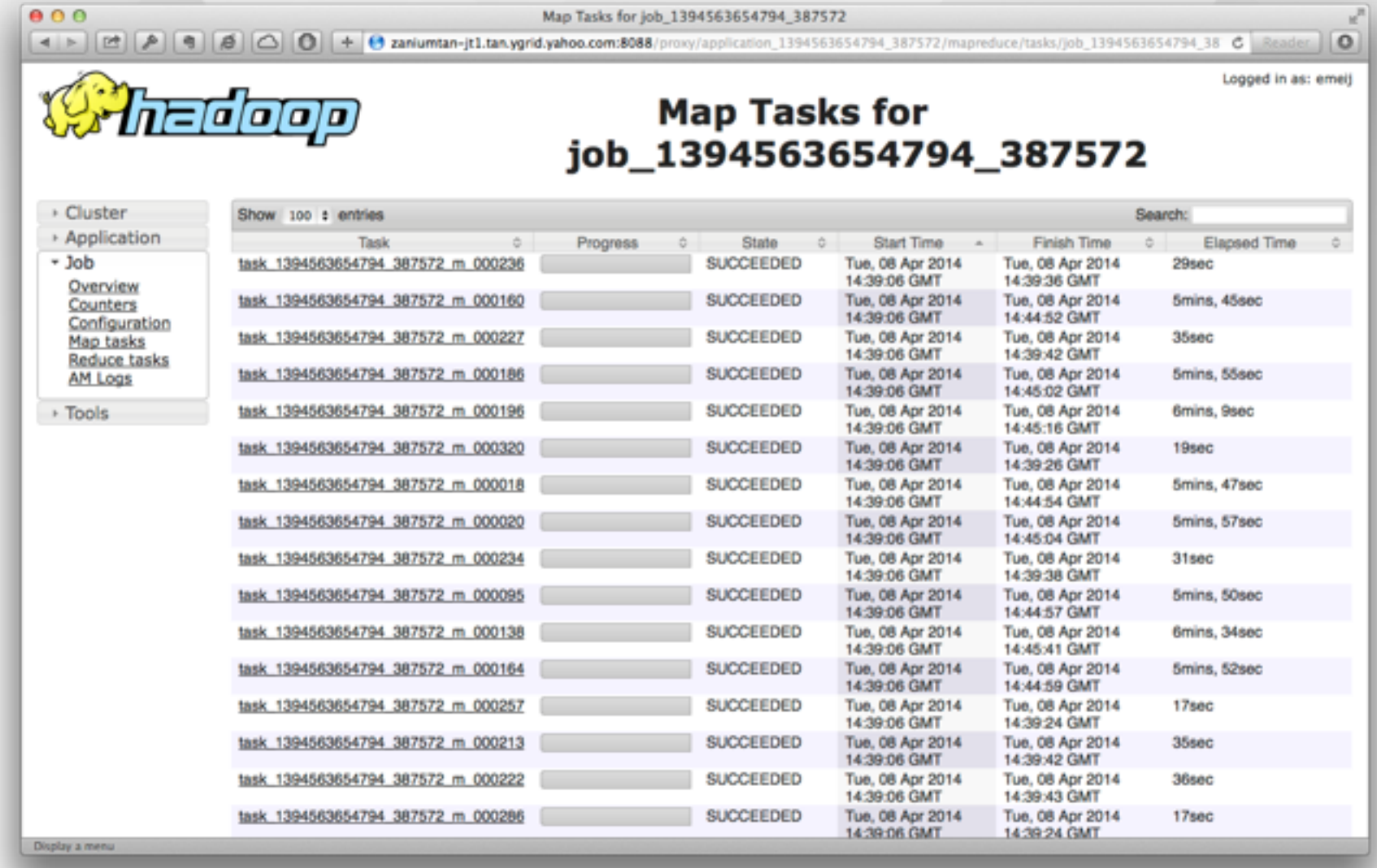
Line 1, Column 1 Spaces: 4 Java

```java
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

  public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedE
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
  }

  public static void main(String[] args) throws Exception {

    Configuration conf = new Configuration();
    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
```

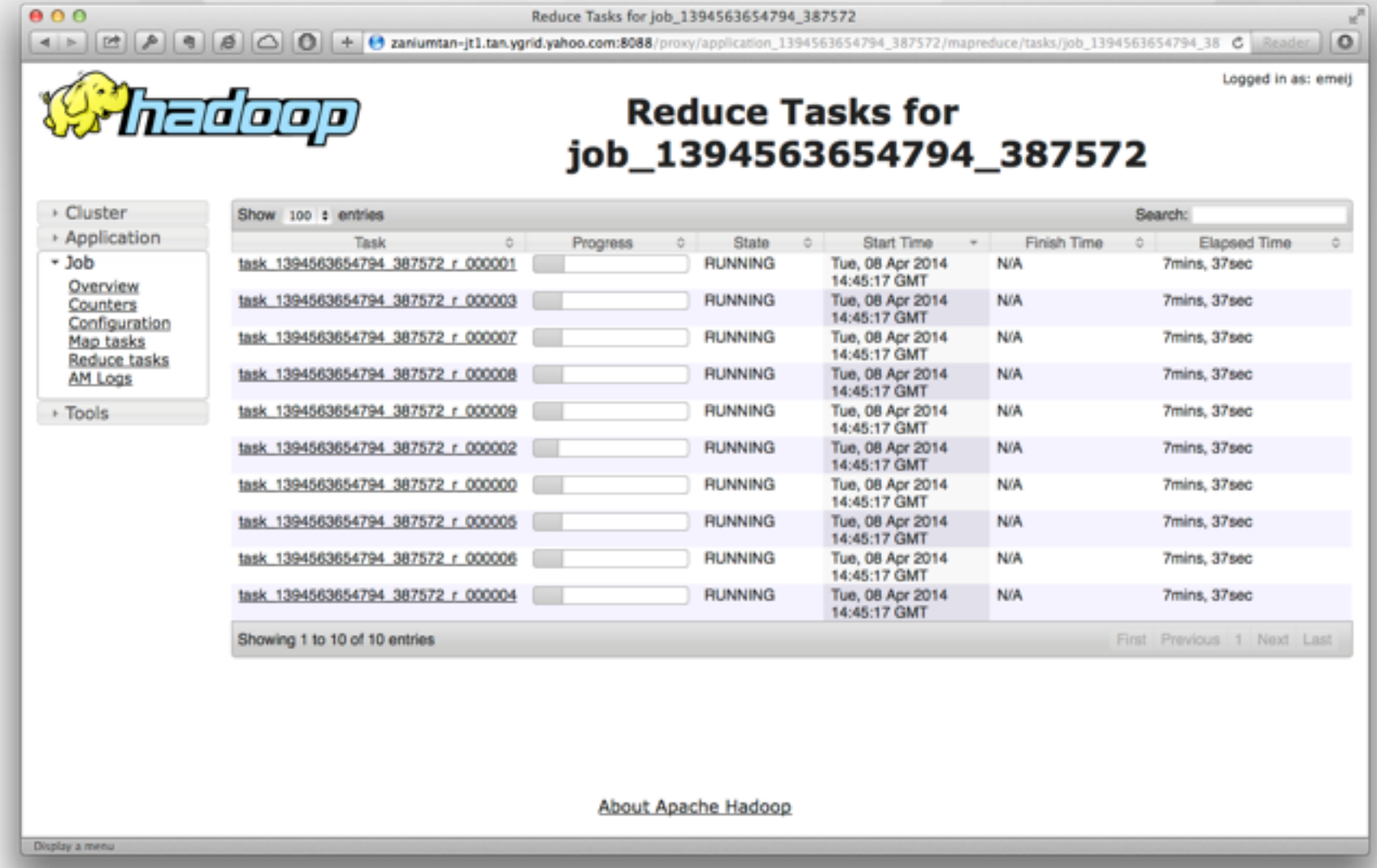
```
$ hadoop jar wordcount.jar org.myorg.WordCount $in $out

...
```


MapReduce Job job_1394563654794_387572
zaniumtan-jt1.tan.ygrid.yahoo.com:8088/proxy/application_1394563654794_387572/mapreduce/job/job_1394563654794_387
Reader
hadoop
MapReduce Job job_1394563654794_387572
Cluster
Application
Job
Overview
Counters
Configuration
Map tasks
Reduce tasks
AM Logs
Tools
Job Overview
Job Name: PigLatin:mbf_nonpage_raw.pig
State: RUNNING
Uberized: false
Started: Tue Apr 08 14:39:04 UTC 2014
Elapsed: 12mins, 58sec
ApplicationMaster
Attempt Number
Start Time
Node
Logs
1
Tue Apr 08 14:39:01 UTC 2014
gsta641n00.tan.ygrid.yahoo.com:8042
logs
Task Type
Progress
Total
Pending
Running
Complete
Map
1517
0
0
1517
Reduce
10
0
10
0
Attempt Type
New
Running
Failed
Killed
Successful
Maps
0
0
0
23
1517
Reduces
0
10
0
0
0
About Apache Hadoop
Display a menu

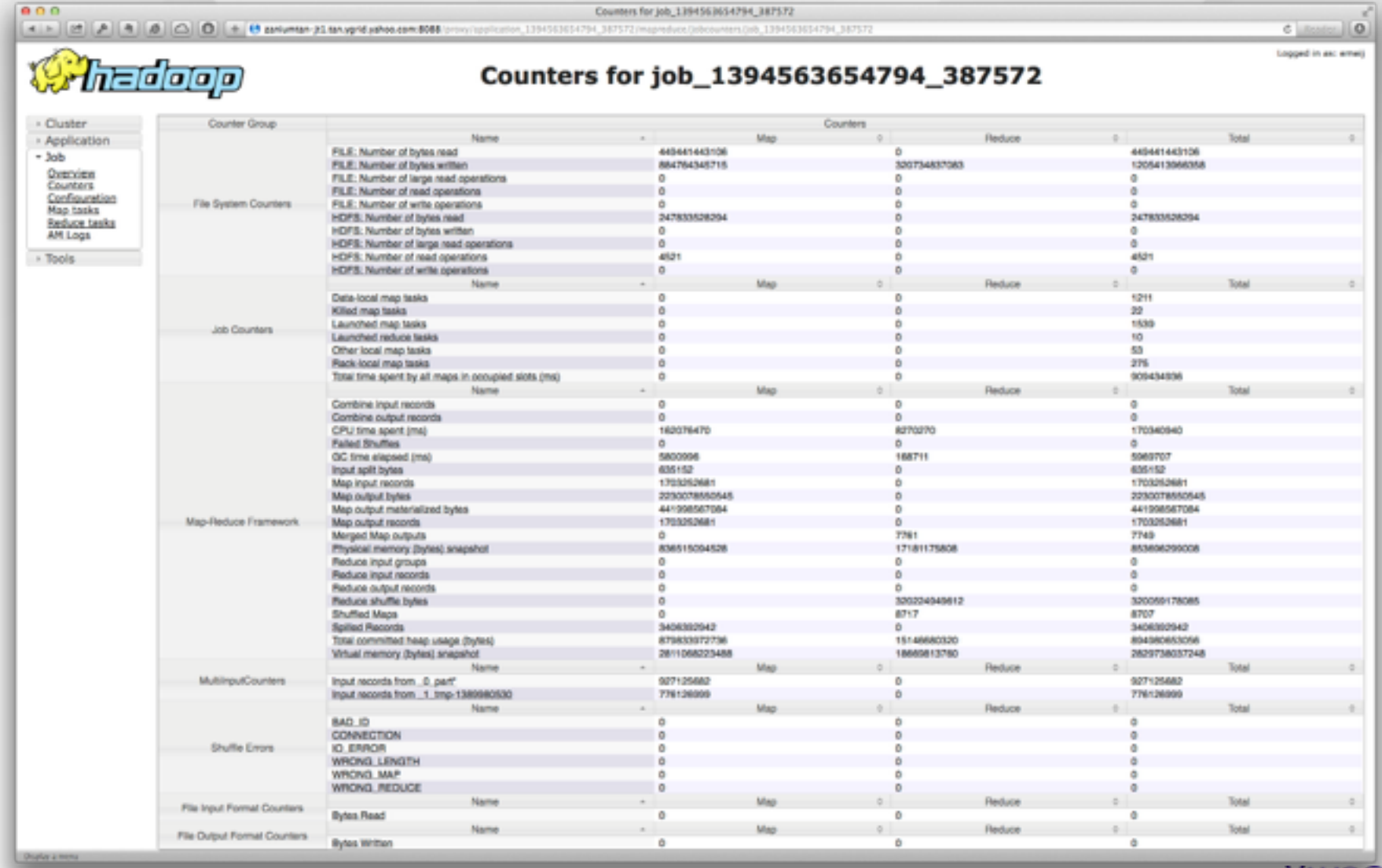
YAHOO!


Map Tasks for job_1394563654794_387572
zaniumtan-jt1.tan.ygrid.yahoo.com:8088/proxy/application_1394563654794_387572/mapreduce/tasks/job_1394563654794_38
Logged in as: emeij
Map Tasks for job_1394563654794_387572
Cluster
Application
Job
Overview
Counters
Configuration
Map tasks
Reduce tasks
AM Logs
Tools
Show 100 entries
Search:
Task | Progress | State | Start Time | Finish Time | Elapsed Time
task_1394563654794_387572_m_000236 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:39:36 GMT 29sec
task_1394563654794_387572_m_000160 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:44:52 GMT 5mins, 45sec
task_1394563654794_387572_m_000227 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:39:42 GMT 35sec
task_1394563654794_387572_m_000186 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:45:02 GMT 5mins, 55sec
task_1394563654794_387572_m_000196 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:45:16 GMT 6mins, 9sec
task_1394563654794_387572_m_000320 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:39:26 GMT 19sec
task_1394563654794_387572_m_000018 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:44:54 GMT 5mins, 47sec
task_1394563654794_387572_m_000020 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:45:04 GMT 5mins, 57sec
task_1394563654794_387572_m_000234 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:39:38 GMT 31sec
task_1394563654794_387572_m_000095 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:44:57 GMT 5mins, 50sec
task_1394563654794_387572_m_000138 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:45:41 GMT 6mins, 34sec
task_1394563654794_387572_m_000164 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:44:59 GMT 5mins, 52sec
task_1394563654794_387572_m_000257 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:39:24 GMT 17sec
task_1394563654794_387572_m_000213 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:39:42 GMT 35sec
task_1394563654794_387572_m_000222 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:39:43 GMT 36sec
task_1394563654794_387572_m_000286 SUCCEEDED Tue, 08 Apr 2014 14:39:06 GMT Tue, 08 Apr 2014 14:39:24 GMT 17sec
Display a menu

YAHOO!


Reduce Tasks for job_1394563654794_387572
zaniumtan-jt1.tan.ygrid.yahoo.com:8088/proxy/application_1394563654794_387572/mapreduce/tasks/job_1394563654794_38
Reader
Logged in as: emeij
hadoop
Reduce Tasks for job_1394563654794_387572
Cluster
Application
Job
Overview
Counters
Configuration
Map tasks
Reduce tasks
AM Logs
Tools
Show 100 entries
Search:
Task
Progress
State
Start Time
Finish Time
Elapsed Time
task_1394563654794_387572_r_000001 RUNNING Tue, 08 Apr 2014 14:45:17 GMT N/A 7mins, 37sec
task_1394563654794_387572_r_000003 RUNNING Tue, 08 Apr 2014 14:45:17 GMT N/A 7mins, 37sec
task_1394563654794_387572_r_000007 RUNNING Tue, 08 Apr 2014 14:45:17 GMT N/A 7mins, 37sec
task_1394563654794_387572_r_000008 RUNNING Tue, 08 Apr 2014 14:45:17 GMT N/A 7mins, 37sec
task_1394563654794_387572_r_000009 RUNNING Tue, 08 Apr 2014 14:45:17 GMT N/A 7mins, 37sec
task_1394563654794_387572_r_000002 RUNNING Tue, 08 Apr 2014 14:45:17 GMT N/A 7mins, 37sec
task_1394563654794_387572_r_000000 RUNNING Tue, 08 Apr 2014 14:45:17 GMT N/A 7mins, 37sec
task_1394563654794_387572_r_000005 RUNNING Tue, 08 Apr 2014 14:45:17 GMT N/A 7mins, 37sec
task_1394563654794_387572_r_000006 RUNNING Tue, 08 Apr 2014 14:45:17 GMT N/A 7mins, 37sec
task_1394563654794_387572_r_000004 RUNNING Tue, 08 Apr 2014 14:45:17 GMT N/A 7mins, 37sec
Showing 1 to 10 of 10 entries
First Previous 1 Next Last
About Apache Hadoop
Display a menu

YAHOO!



Counter Group	Name	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	449441443106	0	449441443106
	FILE: Number of bytes written	884764345715	320734837083	1205413966358
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	2478335526294	0	2478335526294
	HDFS: Number of bytes written	0	0	0
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of read operations	4521	0	4521
	HDFS: Number of write operations	0	0	0
Job Counters	Name	Map	Reduce	Total
	Data-local map tasks	0	0	1211
	Killed map tasks	0	0	22
	Launched map tasks	0	0	1539
	Launched reduce tasks	0	0	10
	Other local map tasks	0	0	53
	Rack-local map tasks	0	0	275
	Total time spent by all maps in occupied slots (ms)	0	0	909434936
Map-Reduce Framework	Name	Map	Reduce	Total
	Combine input records	0	0	0
	Combine output records	0	0	0
	CPU time spent (ms)	162076470	8270270	170346940
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	5800996	168711	5969707
	Input split bytes	635152	0	635152
	Map input records	1703252681	0	1703252681
	Map output bytes	22300785550545	0	22300785550545
	Map output materialized bytes	4411998567084	0	4411998567084
	Map output records	1703252681	0	1703252681
	Merged Map outputs	0	7761	7749
	Physical memory (bytes) snapshot	836515094528	171811175808	853696299008
	Reduce input groups	0	0	0
	Reduce input records	0	0	0
	Reduce output records	0	0	0
	Reduce shuffle bytes	0	3202240349612	3200591178085
	Shuffled Maps	0	8717	8707
	Spilled Records	3406392942	0	3406392942
	Total committed heap usage (bytes)	879633972736	151466680320	894980653056
	Virtual memory (bytes) snapshot	2811068223488	18660813760	2829738037248
MultiInputCounters	Name	Map	Reduce	Total
	Input records from _0_part*	927125682	0	927125682
	Input records from _1_tmp-1389980530	776126999	0	776126999
Shuffle Errors	Name	Map	Reduce	Total
	BAD_ID	0	0	0
	CONNECTION	0	0	0
	IO_ERROR	0	0	0
	WRONG_LENGTH	0	0	0
	WRONG_MAP	0	0	0
	WRONG_REDUCE	0	0	0
File Input Format Counters	Name	Map	Reduce	Total
	Bytes Read	0	0	0
File Output Format Counters	Name	Map	Reduce	Total
	Bytes Written	0	0	0

YAHOO!

```
$ hadoop jar wordcount.jar org.myorg.WordCount $in $out

...

$ hdfs dfs -cat $out/part-r-00000
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```

Hadoop Streaming…

- Allows MapReduce jobs with any executable/script as the mapper and/or the reducer
- Uses pipes

```
$ cat myInputDirs/* | wc -w
…
$ hadoop jar $HADOOP_HOME/hadoop-streaming.jar \
      -input myInputDirs \
      -output myOutputDir \
      -mapper /bin/cat \
      -reducer ‘/bin/wc -w’
```

Hadoop ecosystem

Projects “around” Hadoop

- Ambari
  - Managing Hadoop clusters
- Cassandra
  - “key-value store” (created by Facebook, used by Netflix, a.o.)
- Hbase
  - ~BigTable (used by Facebook, Twitter (non-prod), Mendeley, Y, a.o.)
- Hive
  - data warehousing (created by Facebook, used by Amazon, Netflix, Y, a.o.)
- Mahout
  - Machine learning for Hadoop
- Pig
- Zookeeper
  - Managing Hadoop clusters

Mahout

- Machine learning @ Hadoop
  - distributed or otherwise scalable algorithms
  - focusing on collaborative filtering, clustering, and classification
- Apache licensed
- Lots of “early” implementations
- JBOA

Hbase

- Distributed, wide-column store
  - random, realtime read/write access to large quantities of sparse data
  - non-relational
  - compression/Bloom filters
  - ~BigTable
- Based on HDFS
  - SPOFs: HDFS Name Node and HBase Master (unlike Cassandra)
- APIs: Hadoop, Java, REST, Avro, thrift

Pig

- High-level “platform” for creating MapReduce jobs
  - abstracts programming from MapReduce into a high-level notation
  - similar to SQL for DBs – Pig is more procedural than (declarative) SQL
  - developed at Yahoo Labs in ’06
- Can be extended with user-defined functions
  - Java, Python, JavaScript, Ruby, or Groovy
- Four modes
  - interactive (shell) vs batch (script)
  - local (single machine) vs mapreduce
- Users specify script in “Pig Latin”
  - ~ specifying a query execution plan
  - Pig translates this into MapReduce jobs

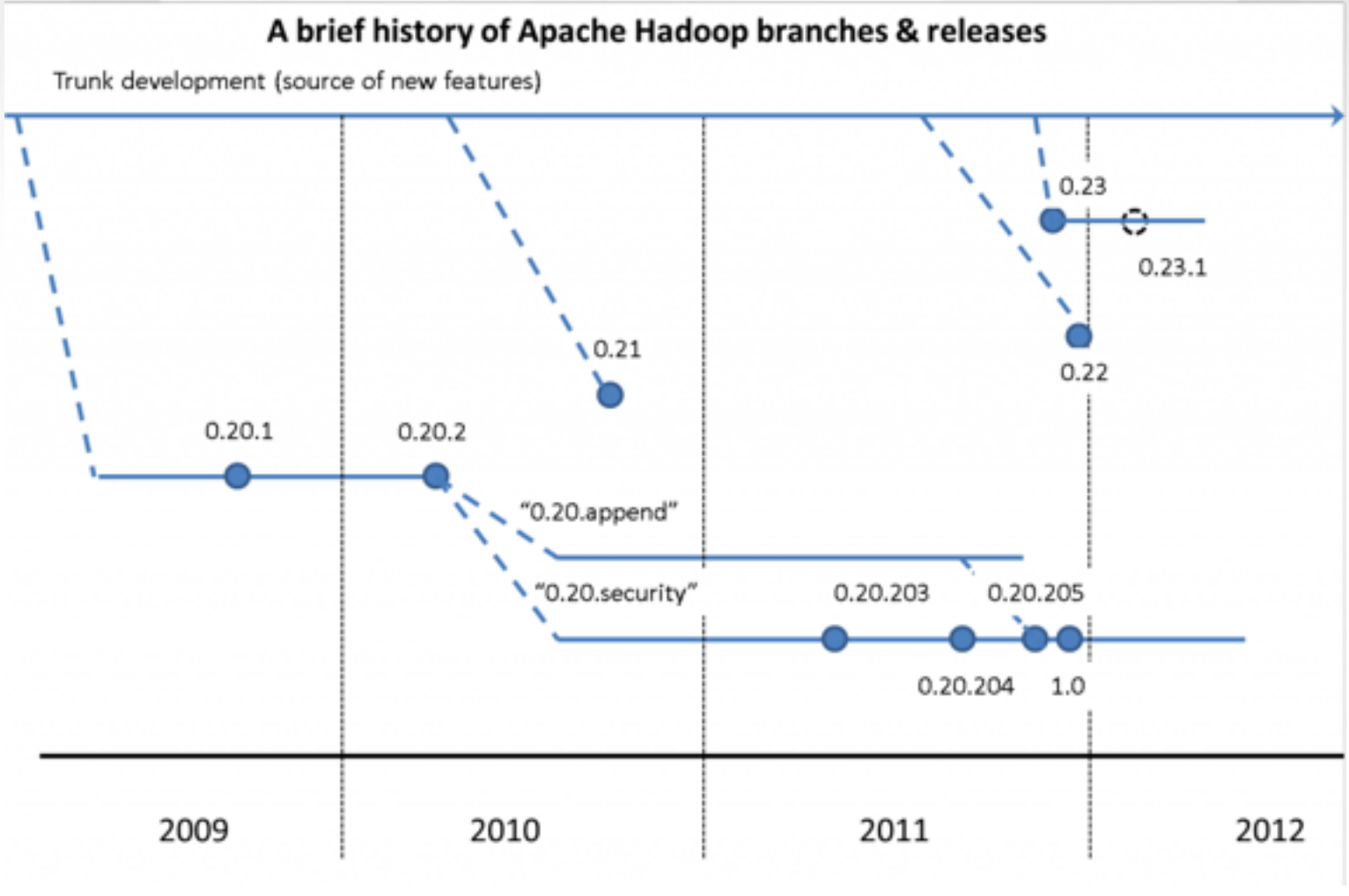
Pig Latin

```
A = load 'input.txt';
B = foreach A generate
      flatten(TOKENIZE((chararray)$0)) as word;
C = group B by word;
D = foreach C generate COUNT(B), group;
store D into 'wordcount.txt';
```

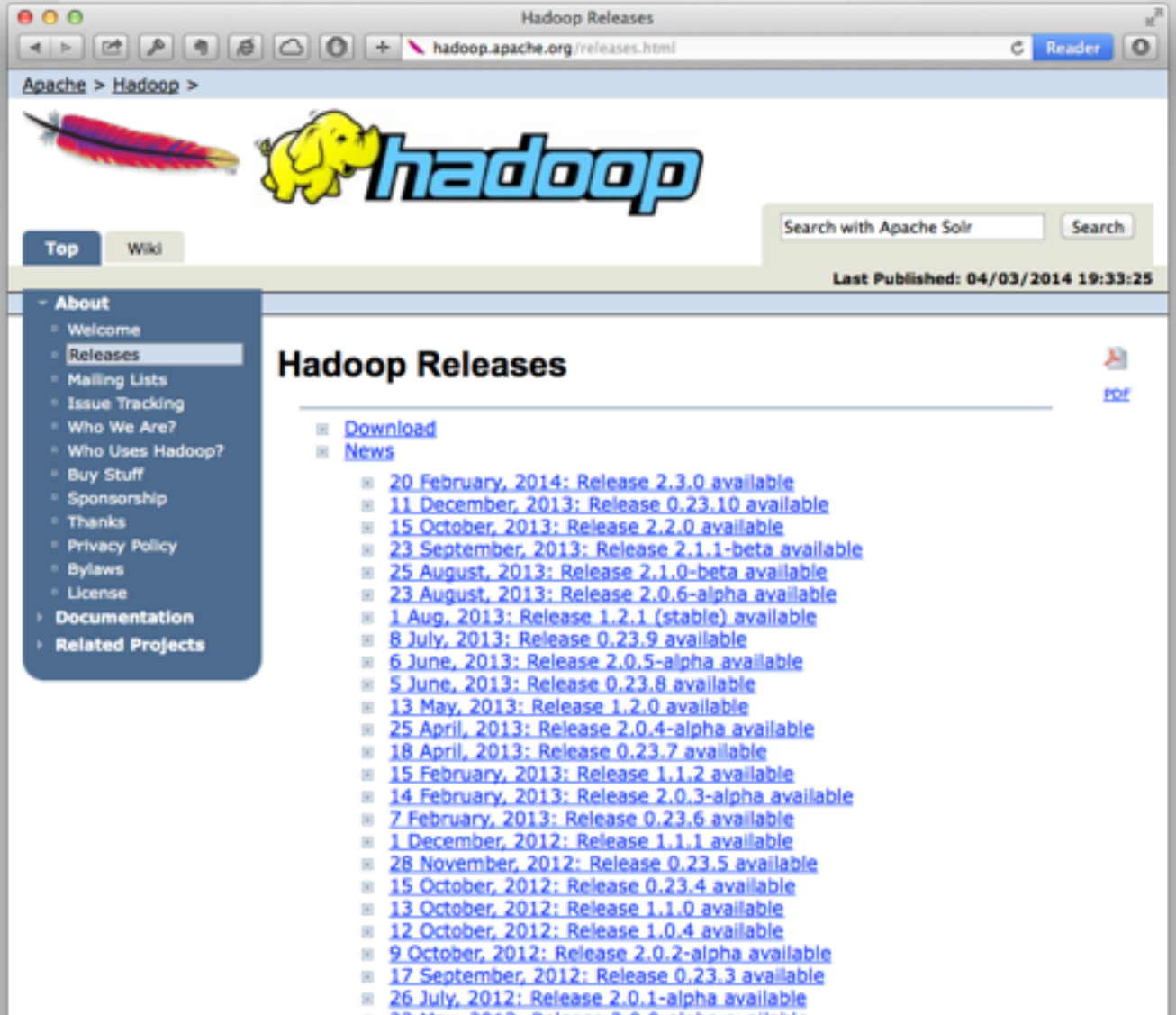
Current developments

Hadoop versioning…

- Typical version-controlled setup
  - trunk: main codeline
  - large features developed on branches: expected to merge with trunk at some later point in time
  - candidate releases branched from trunk
- However…


A brief history of Apache Hadoop branches & releases
Trunk development (source of new features)
0.23
0.23.1
0.21
0.22
0.20.1
0.20.2
"0.20.append"
"0.20.security"
0.20.203
0.20.205
0.20.204
1.0
2009
2010
2011
2012

YAHOO!


Hadoop Releases
hadoop.apache.org/releases.html
Reader
Apache > Hadoop >
hadoop
Search with Apache Solr
Search
Top
Wiki
Last Published: 04/03/2014 19:33:25
About
Welcome
Releases
Mailing Lists
Issue Tracking
Who We Are?
Who Uses Hadoop?
Buy Stuff
Sponsorship
Thanks
Privacy Policy
Bylaws
License
Documentation
Related Projects
Hadoop Releases
PDF
Download
News
20 February, 2014: Release 2.3.0 available
11 December, 2013: Release 0.23.10 available
15 October, 2013: Release 2.2.0 available
23 September, 2013: Release 2.1.1-beta available
25 August, 2013: Release 2.1.0-beta available
23 August, 2013: Release 2.0.6-alpha available
1 Aug, 2013: Release 1.2.1 (stable) available
8 July, 2013: Release 0.23.9 available
6 June, 2013: Release 2.0.5-alpha available
5 June, 2013: Release 0.23.8 available
13 May, 2013: Release 1.2.0 available
25 April, 2013: Release 2.0.4-alpha available
18 April, 2013: Release 0.23.7 available
15 February, 2013: Release 1.1.2 available
14 February, 2013: Release 2.0.3-alpha available
7 February, 2013: Release 0.23.6 available
1 December, 2012: Release 1.1.1 available
28 November, 2012: Release 0.23.5 available
15 October, 2012: Release 0.23.4 available
13 October, 2012: Release 1.1.0 available
12 October, 2012: Release 1.0.4 available
9 October, 2012: Release 2.0.2-alpha available
17 September, 2012: Release 0.23.3 available
26 July, 2012: Release 2.0.1-alpha available

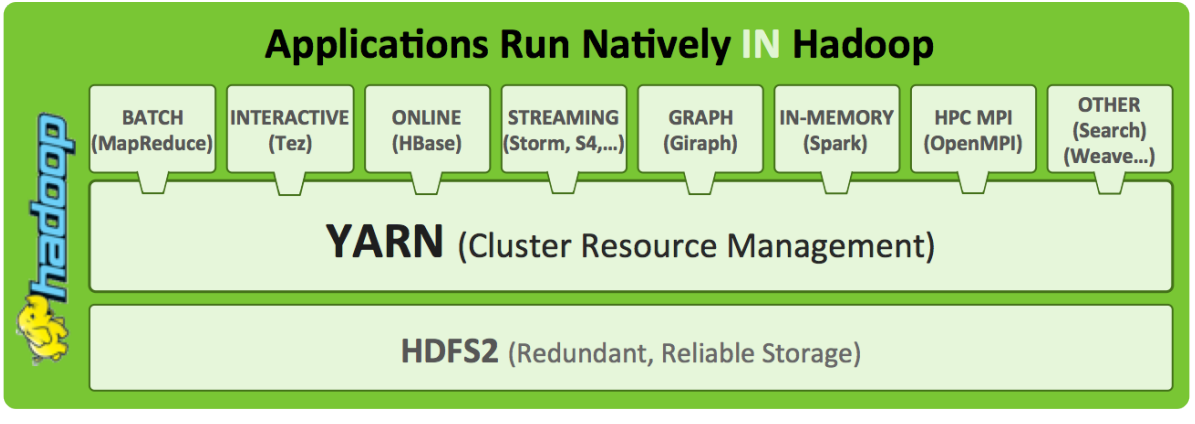
YAHOO!

Issues with Hadoop 1

- Limited to 4000 nodes per cluster
- JobTracker = bottleneck, single POF
- Only one HDFS namespace
- Static map and reduce slots per node
- Only MapReduce jobs
  - although some applications circumvent this

Hadoop 2

- Up to 10,000 nodes per cluster
- Multiple HDFS namespaces
- API compatible with Hadoop 1
- Beyond Java
- YARN

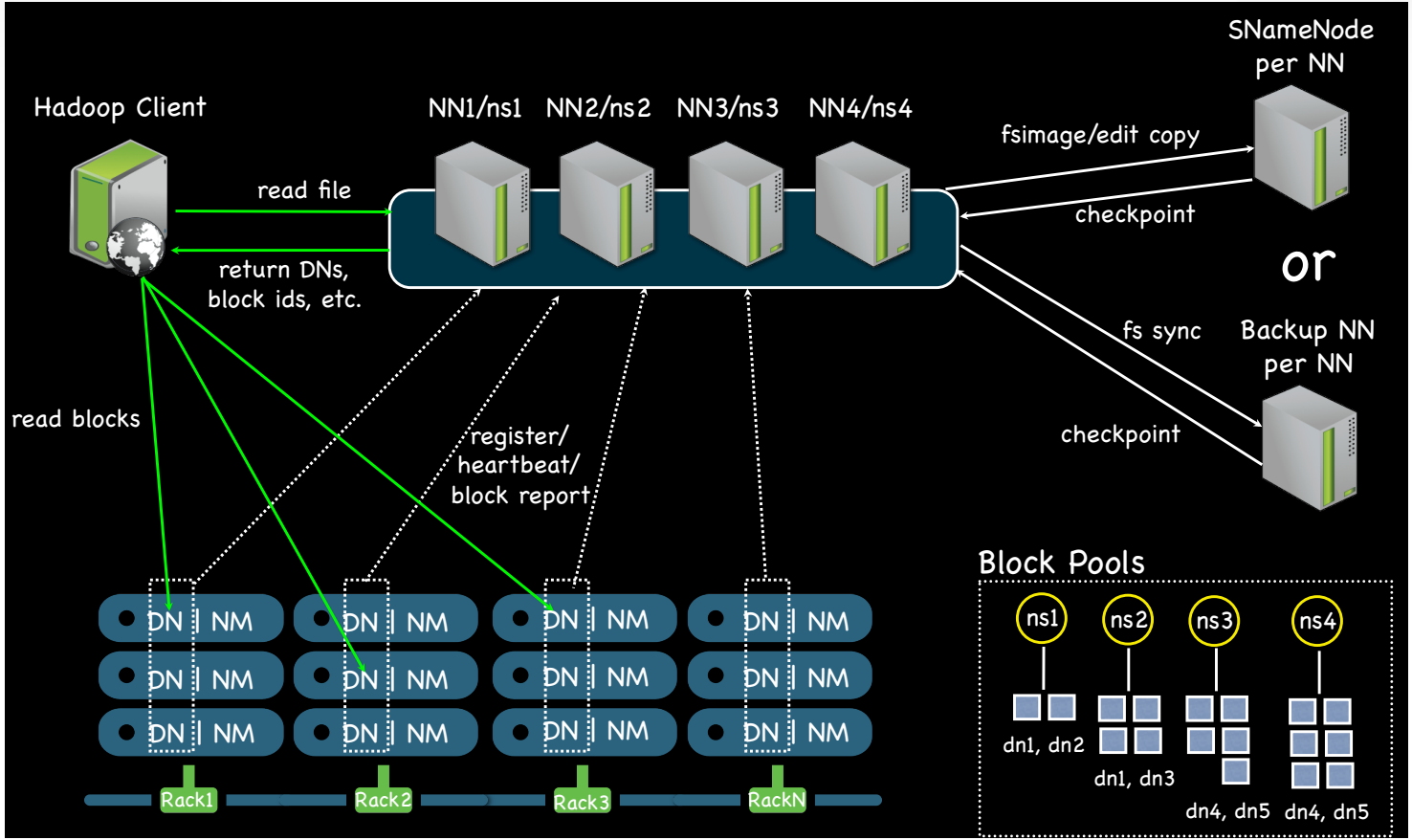


YAHOO!

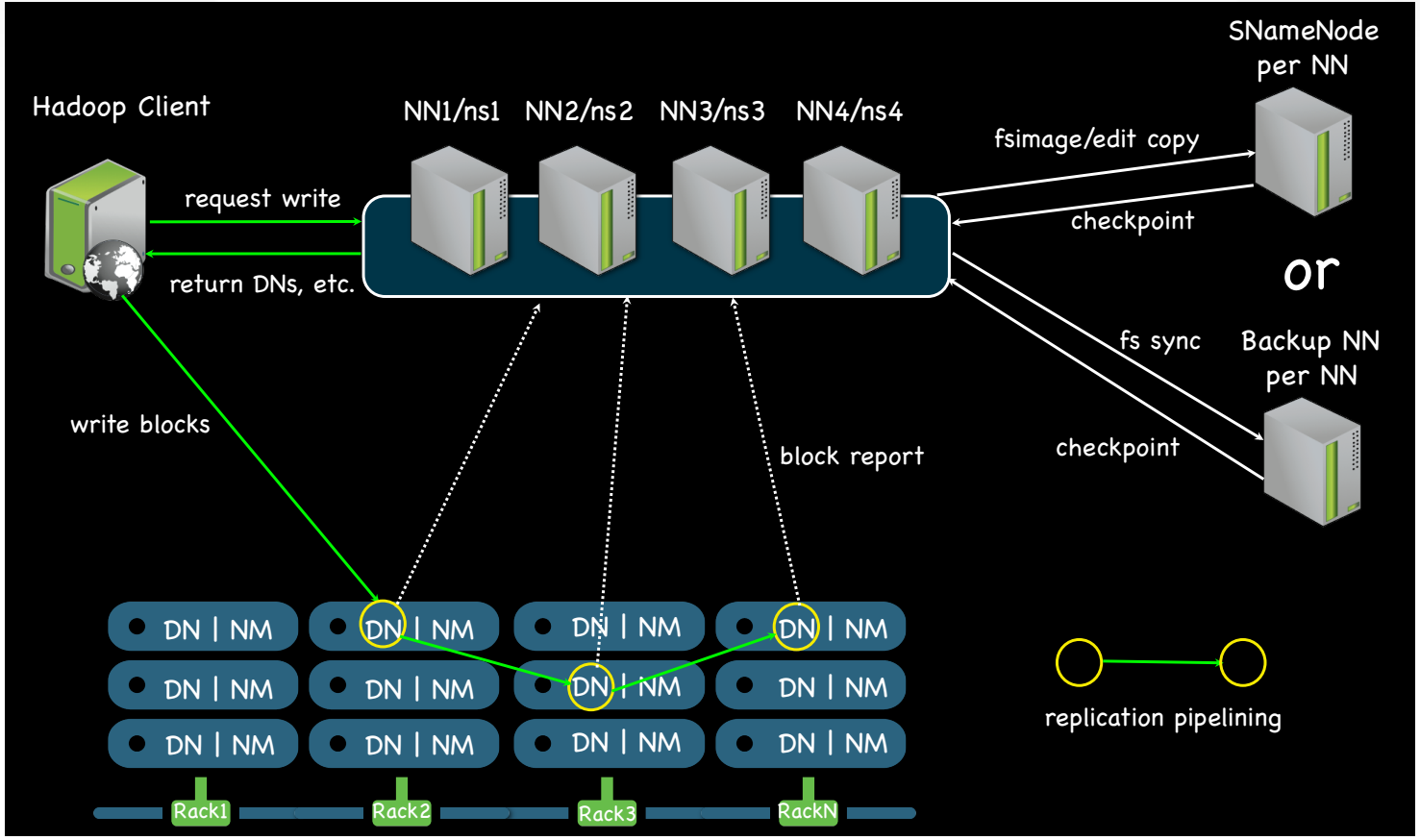
YARN (Yet Another Resource Negotiator)

- Introduced in Hadoop 0.23 (and is also in 2.x)
- Divides the two major functions of the JobTracker into:
  - **ResourceManager** – manages the global assignment of compute resources to applications
    - supports hierarchical application queues
    - pure scheduler: no monitoring or tracking of status for the application
    - resource requests include memory, CPU, disk, network etc.
  - **ApplicationMaster** – manages an application's scheduling and coordination
    - negotiates resource containers, launches tasks, tracks their status, and handles failures
- **NodeManager** manages the user processes on a machine
  - launches the applications' containers, monitors their resource usage (cpu, memory, disk, network), reports to the ResourceManager

Reading files

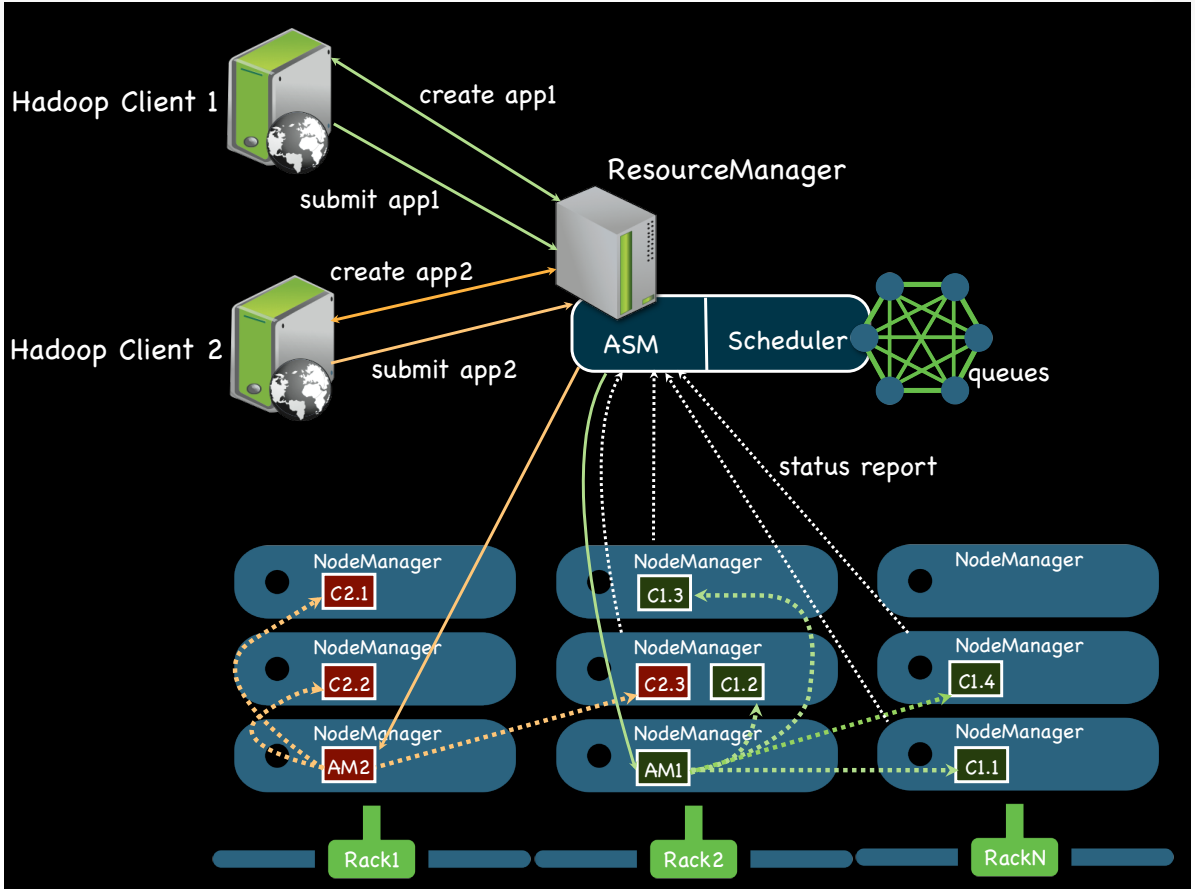


Writing files



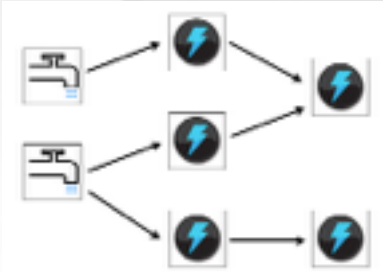
YAHOO!

Running jobs



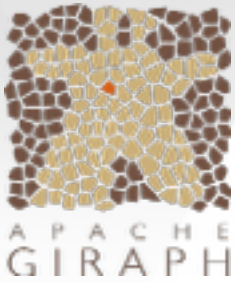
YAHOO!

Storm



- Stream-based processing
  - distributed, realtime computation
  - usable for analytics, online machine learning, continuous computation (sensor data, machine data, query log data, etc.)
- Based on
  - Topologies (~ MapReduce job)
  - Streams: unbounded sequence of tuples that is processed and created in parallel
  - Spouts: a source of streams in a topology
  - Bolts: do the processing on streams
- Moving to YARN
- Used by Twitter, Y, a.o.

Giraph



- Iterative graph processing system using high scalability
  - ~ Pregel
  - bulk synchronous parallel processing
- Runs on YARN
- Used by Facebook, Y, a.o.
  - analyze one trillion edges using 200 nodes in 4 minutes

Spark

- Provides primitives for **in-memory** cluster computing
  - supports streaming
  - Java, Scala or Python
  - speaks YARN (but not tied to Hadoop 2)
- Allows loading data into a cluster's memory and query it repeatedly
  - makes it well-suited to machine learning algorithms – 10x faster than Mahout
  - limited to physical memory sizes (although spillover to disk possible)
- Used by Baidu, Y, a.o.
- MLbase/MLlib
  - Machine learning based on Spark

Finally, some notes

- Resources
  - *Lots* of resources/courses/software/… can be found online
  - Cloudera, Hortonworks, …
  - Not just for industry
    - Amazon Elastic MapReduce
      - also Hadoop streaming
  - Main Hadoop conference: http://hadoopsummit.org/
    - similar ones for HBase, Hive, Pig, …

Questions?

emeij@yahoo-inc.com
http://edgar.meij.pro

YAHOO!