Programming with Hadoop's Map/Reduce

Owen O'Malley
Yahoo!
owen@yahoo-inc.com

ApacheCon EU 2008

Problem

- How do you scale up applications?
  - 100's of terabytes of data
  - Takes 11 days to read on 1 computer
- Need lots of cheap computers
  - Fixes speed problem (15 minutes on 1000 computers), but…
  - Reliability problems
    - In large clusters, computers fail every day
    - Cluster size is not fixed
- Need common infrastructure
  - Must be efficient and reliable

Solution

- Apache Project
- Hadoop Core includes:
  - Distributed File System - distributes data
  - Map/Reduce - distributes application
- Written in Java
- Runs on
  - Linux, Mac OS/X, Windows, and Solaris
  - Commodity hardware

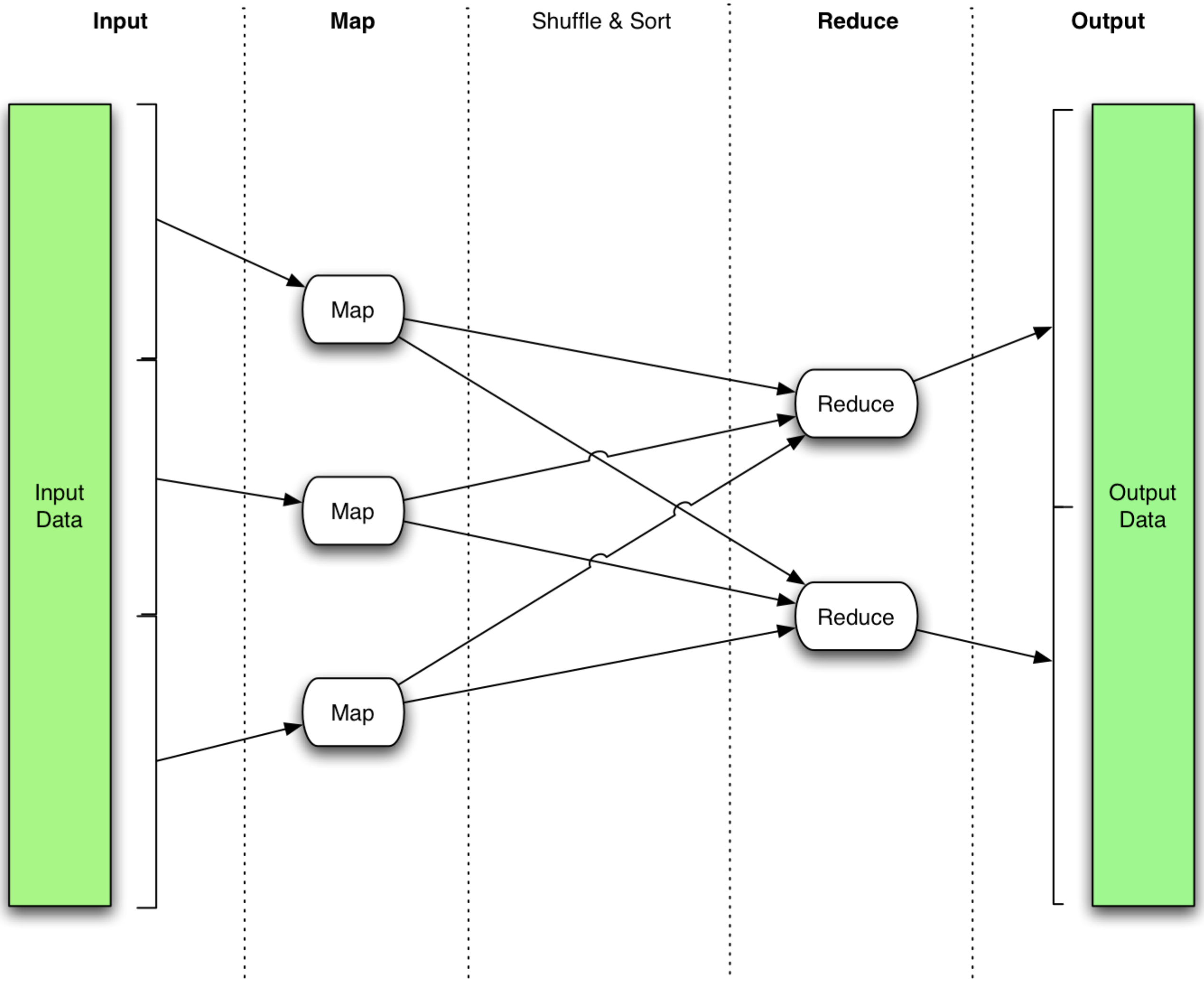
Distributed File System

- Designed to store large files
- Stores files as large blocks (eg. 128 MB)
- Each block stored on multiple servers
- Data is automatically re-replicated on need
- Accessed from command line, Java, or C

Map/Reduce

- Map/Reduce is a programming model for efficient distributed computing
- It works like a Unix pipeline:
  - cat input | grep | sort | uniq -c | cat > output
  - **Input** | **Map** | Shuffle & Sort | **Reduce** | **Output**
- Efficiency from
  - Streaming through data, reducing seeks
  - Pipelining
- A good fit for a lot of applications
  - Log processing
  - Web index building

Map/Reduce Dataflow



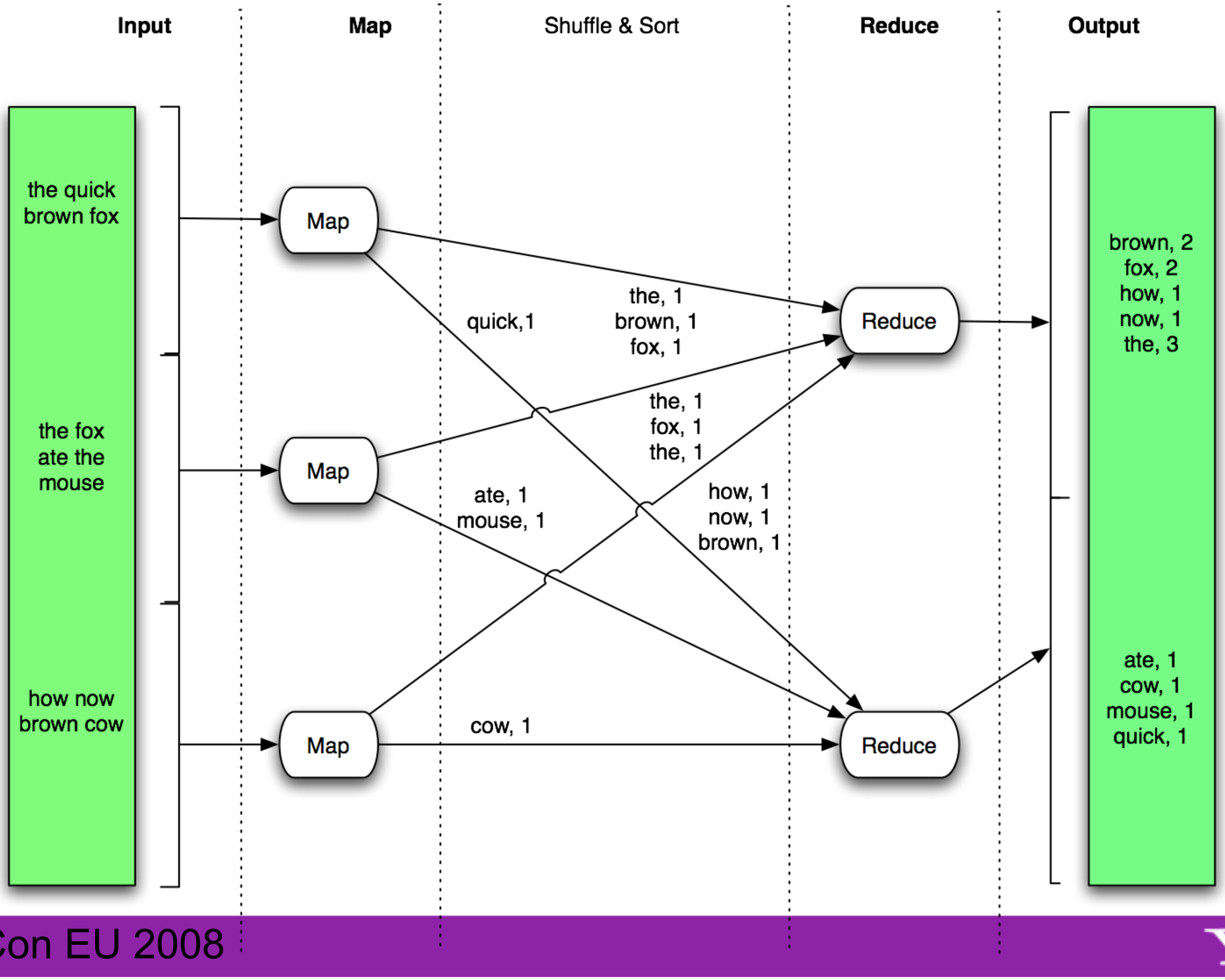
Map/Reduce features

- Fine grained Map and Reduce tasks
  - Improved load balancing
  - Faster recovery from failed tasks
- Automatic re-execution on failure
  - In a large cluster, some nodes are always slow or flaky
  - Framework re-executes failed tasks
- Locality optimizations
  - With large data, bandwidth to data is a problem
  - Map-Reduce + HDFS is a very effective solution
  - Map-Reduce queries HDFS for locations of input data
  - Map tasks are scheduled close to the inputs when possible

Word Count Example

- Mapper
  - Input: value: lines of text of input
  - Output: key: word, value: 1
- Reducer
  - Input: key: word, value: set of counts
  - Output: key: word, value: sum
- Launching program
  - Defines the job
  - Submits job to cluster

Word Count Dataflow



Example: Word Count Mapper

```
public static class MapClass extends MapReduceBase
    implements 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,
                    OutputCollector<Text, IntWritable> output,
                    Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer itr = new StringTokenizer(line);
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        output.collect(word, one);
      }
    }
  }
```

Example: Word Count Reducer

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

   public void reduce(Text key, Iterator<IntWritable> values,
                      OutputCollector<Text, IntWritable> output,
                      Reporter reporter) throws IOException {
     int sum = 0;
     while (values.hasNext()) {
       sum += values.next().get();
     }
     output.collect(key, new IntWritable(sum));
   }
 }
```

Configuring a Job

- Jobs are controlled by configuring *JobConfs*
- JobConfs are maps from attribute names to string value
- The framework defines attributes to control how the job is executed.

```
conf.set("mapred.job.name", "MyApp");
```

- Applications can add arbitrary values to the JobConf

```
conf.set("my.string", "foo");
conf.setInteger("my.integer", 12);
```

- JobConf is available to all of the tasks

Putting it all together

- Create a launching program for your application
- The launching program configures:
  - The *Mapper* and *Reducer* to use
  - The output key and value types (input types are inferred from the *InputFormat)*
  - The locations for your input and output
- The launching program then submits the job and typically waits for it to complete

Putting it all together

```
public class WordCount {
……
public static void main(String[] args) throws IOException {
    JobConf conf = new JobConf(WordCount.class);
    // the keys are words (strings)
    conf.setOutputKeyClass(Text.class);
    // the values are counts (ints)
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(MapClass.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputPath(new Path(args[0]);
    conf.setOutputPath(new Path(args[1]);
    JobClient.runJob(conf);
…. .
```

Input and Output Formats

- A Map/Reduce may specify how it’s input is to be read by specifying an *InputFormat* to be used
- A Map/Reduce may specify how it’s output is to be written by specifying an *OutputFormat* to be used
- These default to *TextInputFormat* and *TextOutputFormat,* which process line-based text data
- Another common choice is SequenceFileInputFormat and SequenceFileOutputFormat for binary data
- These are file-based, but they are not required to be

Non-Java Interfaces

- Streaming
- Pipes (C++)
- Pig

Streaming

- What about non-programmers?
  - Can define Mapper and Reducer using Unix text filters
  - Typically use grep, sed, python, or perl scripts
- Format for input and output is: **key \t value \n**
- Allows for easy debugging and experimentation
- Slower than Java programs

```
bin/hadoop jar hadoop-streaming.jar -input in-dir -output out-dir
  -mapper streamingMapper.sh -reducer streamingReducer.sh
```

- Mapper: `sed -e 's| |\n|g' | grep .`
- Reducer: `uniq -c | awk '{print $2 "\t" $1}'`

Pipes (C++)

- C++ API and library to link application with
- C++ application is launched as a sub-process of the Java task
- Keys and values are std::string with binary data
- Word count map looks like:

```
class WordCountMap: public HadoopPipes::Mapper {
public:
 WordCountMap(HadoopPipes::TaskContext& context){}
 void map(HadoopPipes::MapContext& context) {
  std::vector<std::string> words =
   HadoopUtils::splitString(context.getInputValue(), " ");
  for(unsigned int i=0; i < words.size(); ++i) {
   context.emit(words[i], "1");
  }}};
```

Pipes (C++)

- The reducer looks like:

```
class WordCountReduce: public HadoopPipes::Reducer {
public:
  WordCountReduce(HadoopPipes::TaskContext& context){ }
  void reduce(HadoopPipes::ReduceContext& context) {
    int sum = 0;
    while (context.nextValue()) {
      sum += HadoopUtils::toInt(context.getInputValue());
    }
    context.emit(context.getInputKey(),
     HadoopUtils::toString(sum));
  }
};
```

Pipes (C++)

- And define a main function to invoke the tasks:

```
int main(int argc, char *argv[]) {
  return HadoopPipes::runTask(
    HadoopPipes::TemplateFactory<WordCountMap,
                                 WordCountReduce, void,
                                 WordCountReduce>());
}
```

Pig

- Scripting language that generates Map/Reduce jobs
- User uses higher level operations
  - Group by
  - Foreach
- Word Count:

```
input = LOAD ’in-dir' USING TextLoader();
words = FOREACH input GENERATE
    FLATTEN(TOKENIZE(*));
grouped = GROUP words BY $0;
counts = FOREACH grouped GENERATE group,
    COUNT(words);
STORE counts INTO ‘out-dir’;
```

How many Maps and Reduces

- Maps
  - Usually as many as the number of HDFS blocks being processed, this is the default
  - Else the number of maps can be specified as a hint
  - The number of maps can also be controlled by specifying the *minimum split size*
  - The actual sizes of the map inputs are computed by:
    - *max(min(block_size, data/#maps), min_split_size)*
- Reduces
  - Unless the amount of data being processed is small
    - *0.95*num_nodes*mapred.tasktracker.tasks.maximum*

Performance Example

- Bob wants to count lines in text files totaling several terabytes
- He uses
  - Identity Mapper (input: text, output: same text)
  - A single Reducer that counts the lines and outputs the total
- What is he doing wrong ?
- This happened, really !
  - I am not kidding !

Some handy tools

- Partitioners
- Combiners
- Compression
- Counters
- Speculation
- Zero reduces
- Distributed File Cache
- Tool

Partitioners

- Partitioners are application code that define how keys are assigned to reduces
- Default partitioning spreads keys evenly, but randomly
  - Uses *key.hashCode() % num_reduces*
- Custom partitioning is often required, for example, to produce a total order in the output
  - Should implement *Partitioner* interface
  - Set by calling conf.setPartitionerClass(MyPart.class)
  - To get a total order, sample the map output keys and pick values to divide the keys into roughly equal buckets and use that in your partitioner

Combiners

- When *maps* produce many repeated keys
  - It is often useful to do a local aggregation following the *map*
  - Done by specifying a *Combiner*
  - Goal is to decrease size of the transient data
  - Combiners have the same interface as Reduces, and often are the same class.
  - Combiners must **not** have side effects, because they run an indeterminate number of times.
  - In *WordCount*, *conf.setCombinerClass(Reduce.class);*

Compression

- Compressing the outputs and intermediate data will often yield huge performance gains
  - Can be specified via a configuration file or set programatically
  - Set *mapred.output.compress* to *true* to compress job output
  - Set *mapred.compress.map.output* to *true* to compress map outputs
- Compression Types *(mapred(.map)?.output.compression.type)*
  - “block” - Group of keys and values are compressed together
  - “record” - Each value is compressed individually
  - Block compression is almost always best
- Compression Codecs *(mapred(.map)?.output.compression.codec)*
  - Default (zlib) - slower, but more compression
  - LZO - faster, but less compression

Counters

- Often Map/Reduce applications have countable events
- For example, framework counts records in to and out of Mapper and Reducer
- To define user counters:

```
static enum Counter {EVENT1, EVENT2};
reporter.incrCounter(Counter.EVENT1, 1);
```

- Define nice names in a MyClass_Counter.properties file

```
CounterGroupName=My Counters
EVENT1.name=Event 1
EVENT2.name=Event 2
```

Speculative execution

- The framework can run multiple instances of slow tasks
  - Output from instance that finishes first is used
  - Controlled by the configuration variable *mapred.speculative.execution*
  - Can dramatically bring in long tails on jobs

Zero Reduces

- Frequently, we only need to run a filter on the input data
  - No sorting or shuffling required by the job
  - Set the number of reduces to 0
  - Output from maps will go directly to OutputFormat and disk

Distributed File Cache

- Sometimes need read-only copies of data on the local computer.
  - Downloading 1GB of data for each Mapper is expensive
- Define list of files you need to download in JobConf
- Files are downloaded once per a computer
- Add to launching program:

```
DistributedCache.addCacheFile(new URI(“hdfs://nn:8020/foo”), conf);
```

- Add to task:

```
Path[] files = DistributedCache.getLocalCacheFiles(conf);
```

Tool

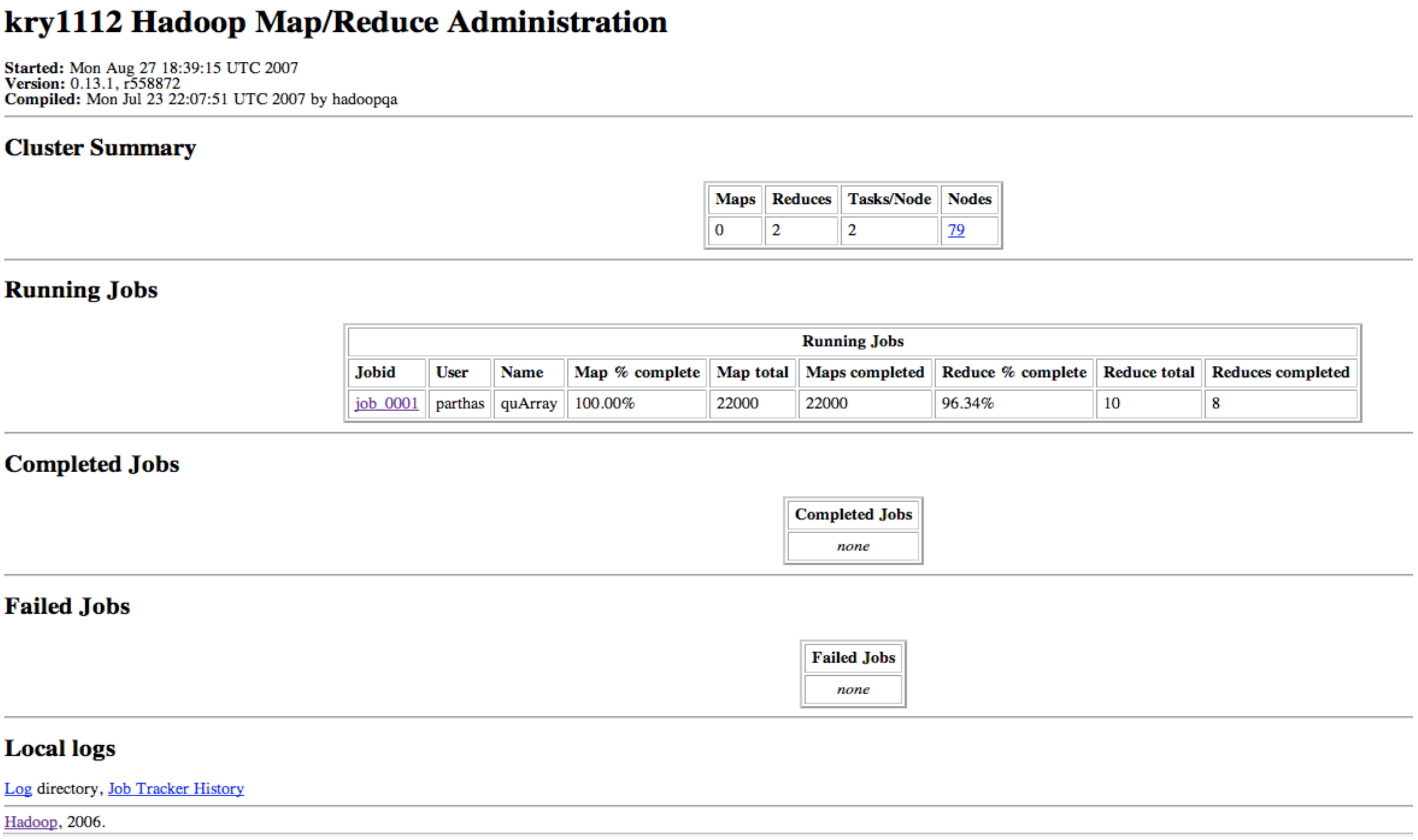
- Handle “standard” Hadoop command line options:
  - -conf file - load a configuration file named file
  - -D prop=value - define a single configuration property prop
- Class looks like:

```
public class MyApp extends Configured implements Tool {
 public static void main(String[] args) throws Exception {
   System.exit(ToolRunner.run(new Configuration(),
                      new MyApp(), args));
 }
 public int run(String[] args) throws Exception {
   .... getConf() ...
 }
}
```

Debugging & Diagnosis

- Run job with the Local Runner
  - Set mapred.job.tracker to “local”
  - Runs application in a single process and thread
- Run job on a small data set on a 1 node cluster
  - Can be done on your local dev box
- Set *keep.failed.task.files* to true
  - This will keep files from failed tasks that can be used for debugging
  - Use the IsolationRunner to run just the failed task
- Java Debugging hints
  - Send a *kill -QUIT* to the Java process to get the call stack, locks held, deadlocks

Jobtracker front page



kry1112 Hadoop Map/Reduce Administration

Started: Mon Aug 27 18:39:15 UTC 2007
Version: 0.13.1, r558872
Compiled: Mon Jul 23 22:07:51 UTC 2007 by hadoopqa

Cluster Summary

Maps	Reduces	Tasks/Node	Nodes
0	2	2	79

Running Jobs

Running Jobs

Jobid	User	Name	Map % complete	Map total	Maps completed	Reduce % complete	Reduce total	Reduces completed
job_0001	parthas	quArray	100.00%	22000	22000	96.34%	10	8

Completed Jobs

Completed Jobs
none

Failed Jobs

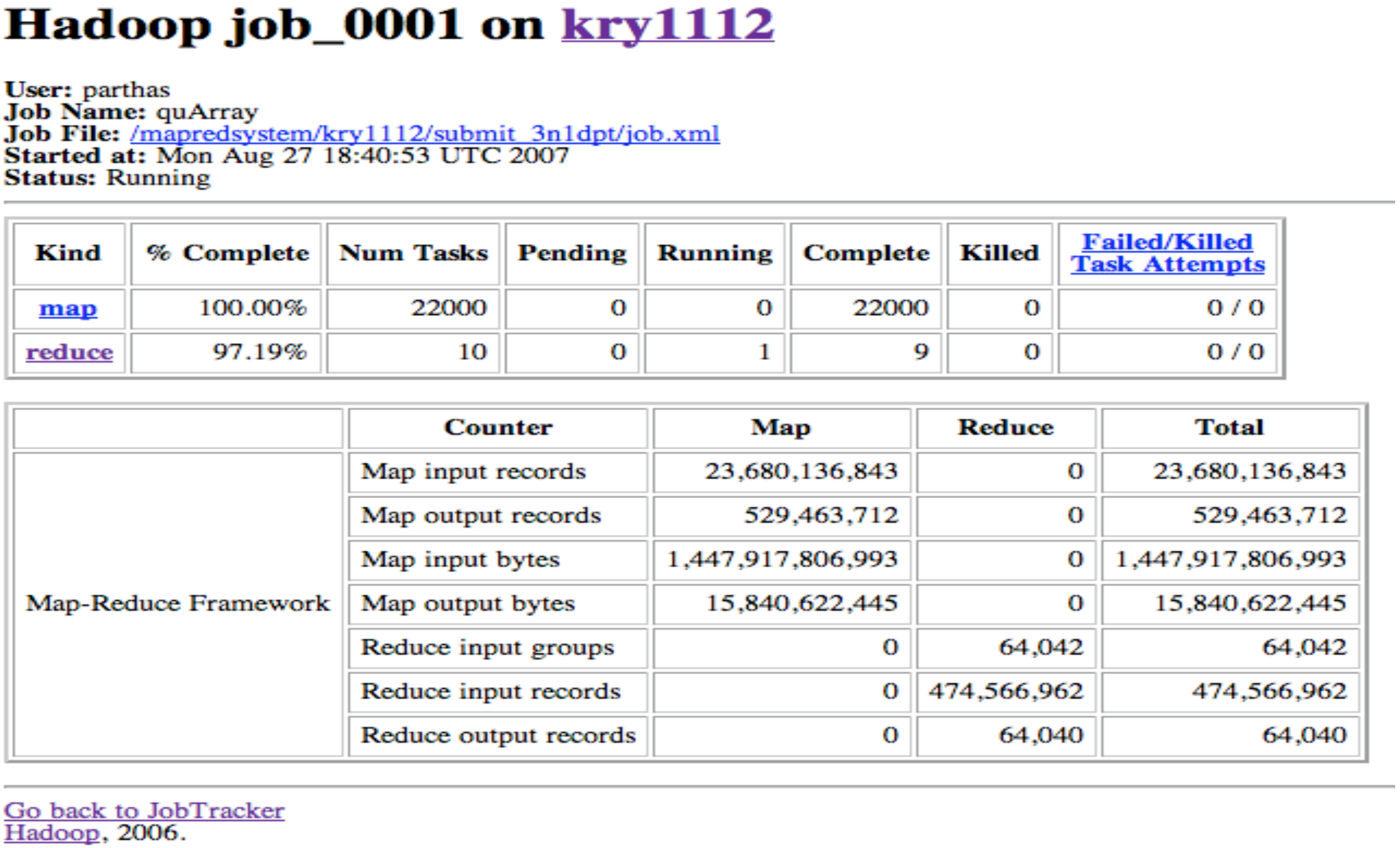
Failed Jobs
none

Local logs

Log directory, Job Tracker History

Hadoop, 2006.

Job counters



Hadoop job_0001 on kry1112

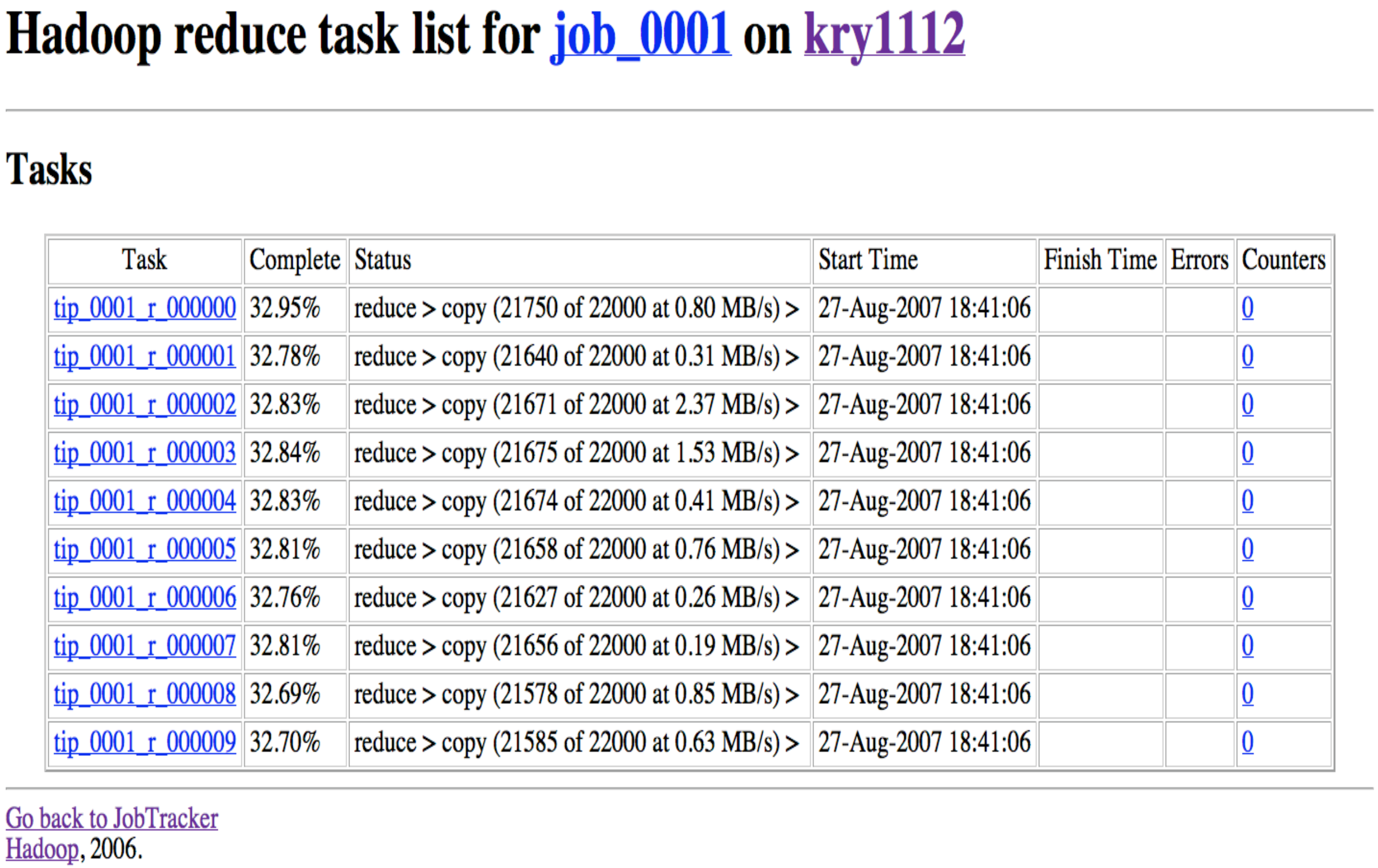
User: parthas
Job Name: quArray
Job File: /mapredsystem/kry1112/submit_3n1dpt/job.xml
Started at: Mon Aug 27 18:40:53 UTC 2007
Status: Running

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	22000	0	0	22000	0	0 / 0
reduce	97.19%	10	0	1	9	0	0 / 0

	Counter	Map	Reduce	Total
Map-Reduce Framework	Map input records	23,680,136,843	0	23,680,136,843
	Map output records	529,463,712	0	529,463,712
	Map input bytes	1,447,917,806,993	0	1,447,917,806,993
	Map output bytes	15,840,622,445	0	15,840,622,445
	Reduce input groups	0	64,042	64,042
	Reduce input records	0	474,566,962	474,566,962
	Reduce output records	0	64,040	64,040

Go back to JobTracker
Hadoop, 2006.

Task status



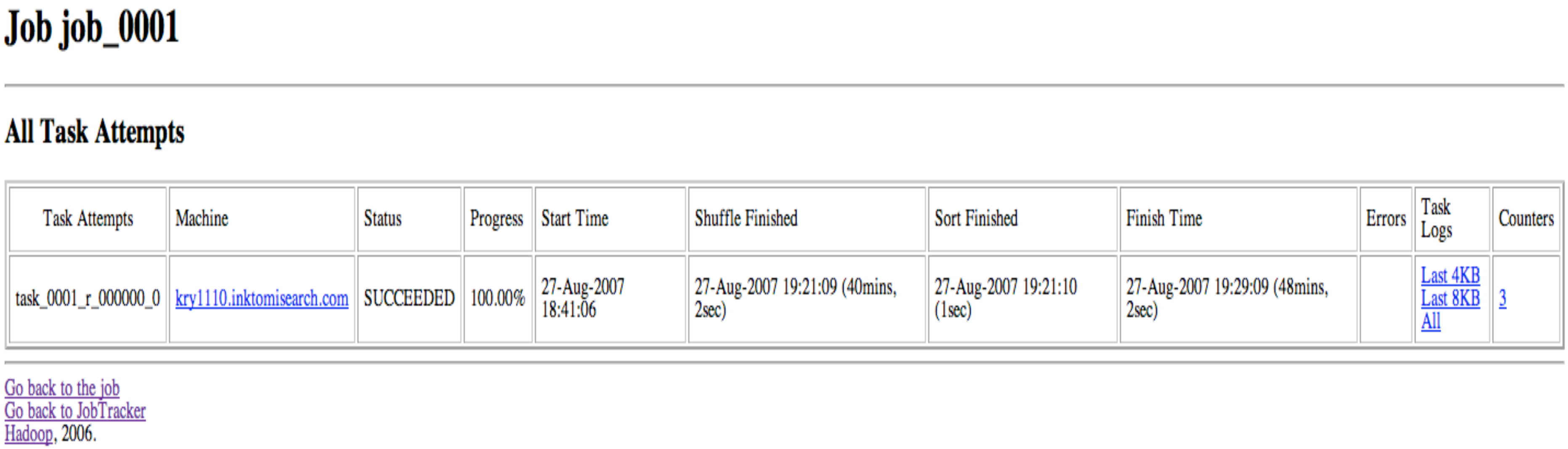
Hadoop reduce task list for job_0001 on kry1112

Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
tip_0001_r_000000	32.95%	reduce > copy (21750 of 22000 at 0.80 MB/s) >	27-Aug-2007 18:41:06			0
tip_0001_r_000001	32.78%	reduce > copy (21640 of 22000 at 0.31 MB/s) >	27-Aug-2007 18:41:06			0
tip_0001_r_000002	32.83%	reduce > copy (21671 of 22000 at 2.37 MB/s) >	27-Aug-2007 18:41:06			0
tip_0001_r_000003	32.84%	reduce > copy (21675 of 22000 at 1.53 MB/s) >	27-Aug-2007 18:41:06			0
tip_0001_r_000004	32.83%	reduce > copy (21674 of 22000 at 0.41 MB/s) >	27-Aug-2007 18:41:06			0
tip_0001_r_000005	32.81%	reduce > copy (21658 of 22000 at 0.76 MB/s) >	27-Aug-2007 18:41:06			0
tip_0001_r_000006	32.76%	reduce > copy (21627 of 22000 at 0.26 MB/s) >	27-Aug-2007 18:41:06			0
tip_0001_r_000007	32.81%	reduce > copy (21656 of 22000 at 0.19 MB/s) >	27-Aug-2007 18:41:06			0
tip_0001_r_000008	32.69%	reduce > copy (21578 of 22000 at 0.85 MB/s) >	27-Aug-2007 18:41:06			0
tip_0001_r_000009	32.70%	reduce > copy (21585 of 22000 at 0.63 MB/s) >	27-Aug-2007 18:41:06			0

Go back to JobTracker
Hadoop, 2006.

Drilling down



Job job_0001

All Task Attempts

Task Attempts	Machine	Status	Progress	Start Time	Shuffle Finished	Sort Finished	Finish Time	Errors	Task Logs	Counters
task_0001_r_000000_0	kry1110.inktomisearch.com	SUCCEEDED	100.00%	27-Aug-2007 18:41:06	27-Aug-2007 19:21:09 (40mins, 2sec)	27-Aug-2007 19:21:10 (1sec)	27-Aug-2007 19:29:09 (48mins, 2sec)		Last 4KB Last 8KB All	3

Go back to the job
Go back to JobTracker
Hadoop, 2006.

Performance

- Is your input splittable?
  - Gzipped files are NOT splittable
- Are partitioners uniform?
- Buffering sizes (especially io.sort.mb)
- Do you need to Reduce?
- Only use singleton reduces for very small data
  - Use Partitioners and cat to get a total order
- Memory usage
  - Please do not load all of your inputs into memory!

Hadoop clusters

- We have ~10,000 machines running Hadoop
- Our largest cluster is currently 2000 nodes
- 1 petabyte of user data (compressed, unreplicated)
- We run roughly 10,000 research jobs / week

Who Uses Hadoop?

- Amazon/A9
- Facebook
- Google
- IBM
- Joost
- Last.fm
- New York Times
- PowerSet
- Veoh
- Yahoo!

Q&A

- For more information:
  - Website: http://hadoop.apache.org/core
  - Mailing lists:
    - core-dev@hadoop.apache
    - core-user@hadoop.apache
  - IRC: #hadoop on irc.freenode.org