

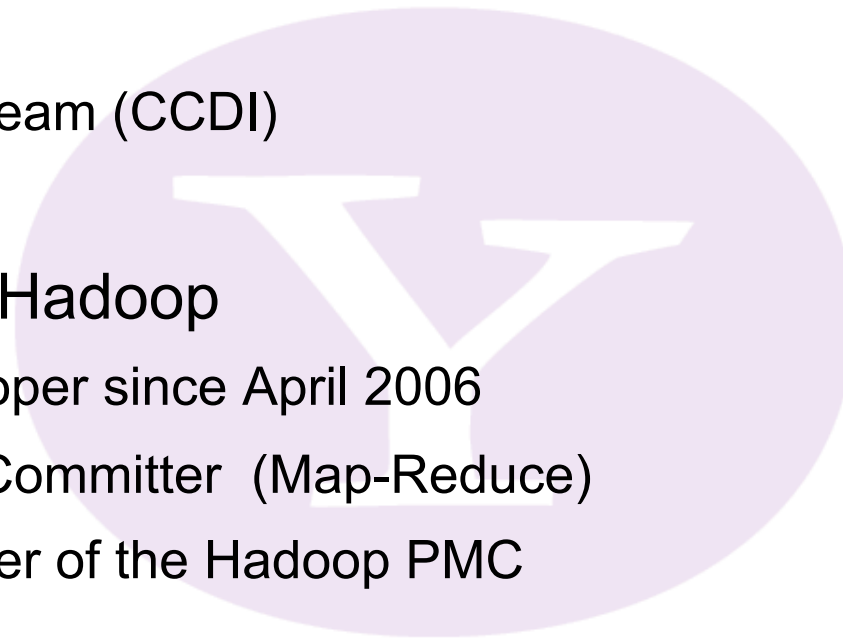
Programming Hadoop Map-Reduce Programming, Tuning & Debugging

Arun C Murthy
Yahoo! CCDI
acm@yahoo-inc.com
ApacheCon US 2008

YAHOO!

Existential angst: Who am I?

- Yahoo!
 - Grid Team (CCDI)
- Apache Hadoop
 - Developer since April 2006
 - Core Committer (Map-Reduce)
 - Member of the Hadoop PMC



YAHOO!



Hadoop - Overview

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

YAHOO!



Distributed File System

- Designed to store large files
- Stores files as large blocks (64 to 128 MB)
- Each block stored on multiple servers
- Data is automatically re-replicated on need
- Accessed from command line, Java API, or C API
 - `bin/hadoop fs -put my-file hdfs://node1:50070/foo/bar`
 - ```
Path p = new Path("hdfs://node1:50070/foo/bar");
FileSystem fs = p.getFileSystem(conf);
DataOutputStream file = fs.create(p);
file.writeUTF("hello\n");
file.close();
```

YAHOO!



# Map-Reduce

- Map-Reduce is a programming model for efficient distributed computing
- It works like a Unix pipeline:
  - `cat input | grep | sort | unique -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

YAHOO!



# 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
  - Introduces long tails or failures in computation
  - Framework re-executes failed tasks
- Locality optimizations
  - With big 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 local to the inputs when possible

YAHOO!



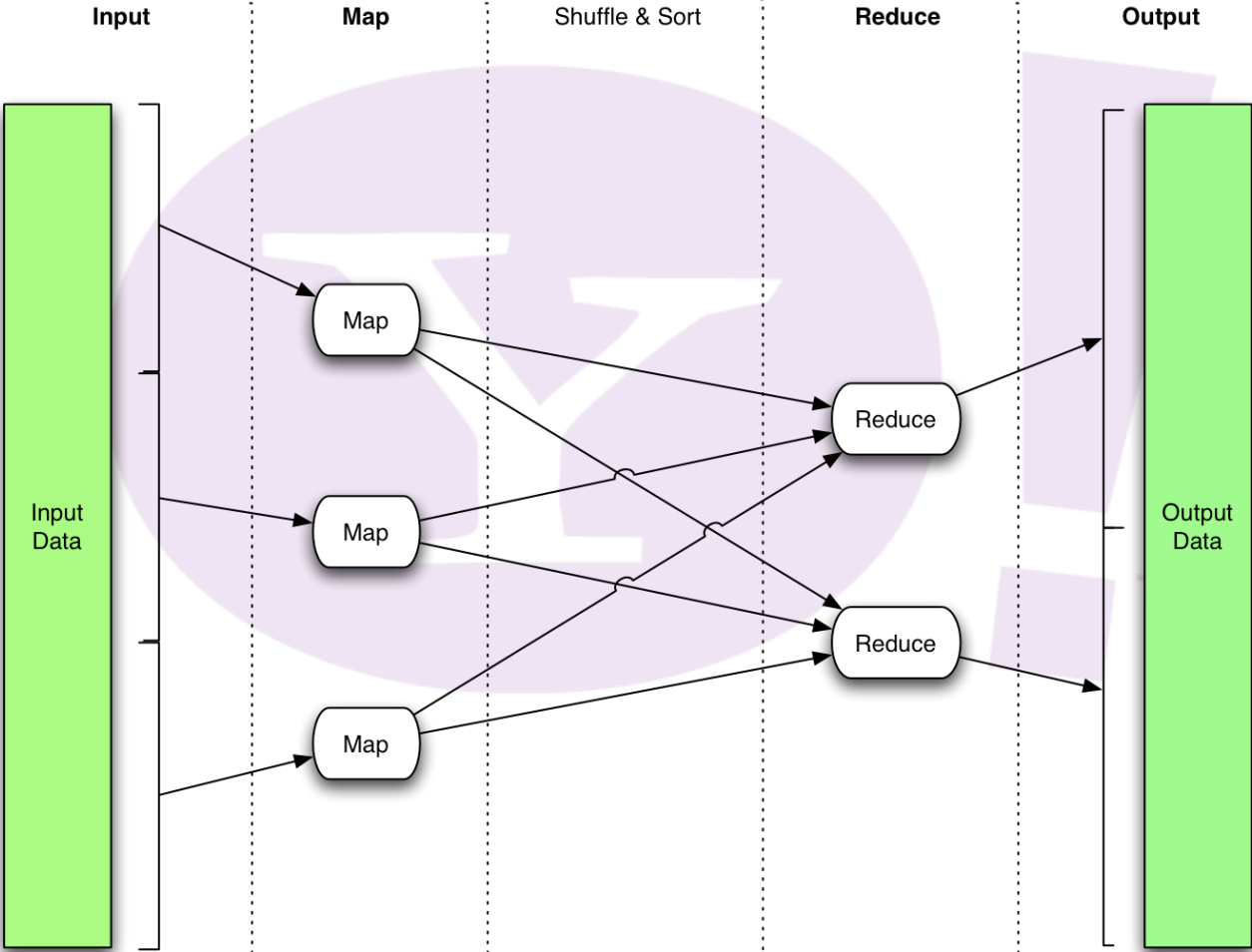
# Mappers and Reducers

- Every Map/Reduce program must specify a *Mapper* and typically a *Reducer*
- The *Mapper* has a *map* method that transforms input *(key, value)* pairs into any number of intermediate *(key', value')* pairs
- The *Reducer* has a *reduce* method that transforms intermediate *(key', value'\*)* aggregates into any number of output *(key'', value'')* pairs

YAHOO!



# Map/Reduce Dataflow



YAHOO!





# Example...

“45% of all Hadoop tutorials count words. 25% count sentences. 20% are about paragraphs. 10% are log parsers. The remainder are helpful.”

jandersen @<http://twitter.com/jandersen/statuses/926856631>

YAHOO!



# Example: Wordcount 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);
 }
 }
}
```

YAHOO!



# Example: Wordcount 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));
 }
}
```

YAHOO!



# Input and Output Formats

- A Map/Reduce may specify how it's input is to be read by specifying an *InputFormat* to be used
  - InputSplit
  - RecordReader
- 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
- SequenceFile: *SequenceFileInputFormat* and *SequenceFileOutputFormat*
- These are file-based, but they are not required to be

# Configuring a Job

- Jobs are controlled by configuring *JobConf*
- 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

YAHOO!



# 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
  - Optionally the *InputFormat* and *OutputFormat* to use
- The launching program then submits the job and typically waits for it to complete

YAHOO!



# Putting it all together

```
public class WordCount {
.....
public static void main(String[] args) throws IOException {
 JobConf conf = new JobConf(WordCount.class);
 conf.setJobName("wordcount");

 // the keys are words (strings)
 conf.setOutputKeyClass(Text.class);
 // the values are counts (ints)
 conf.setOutputValueClass(IntWritable.class);

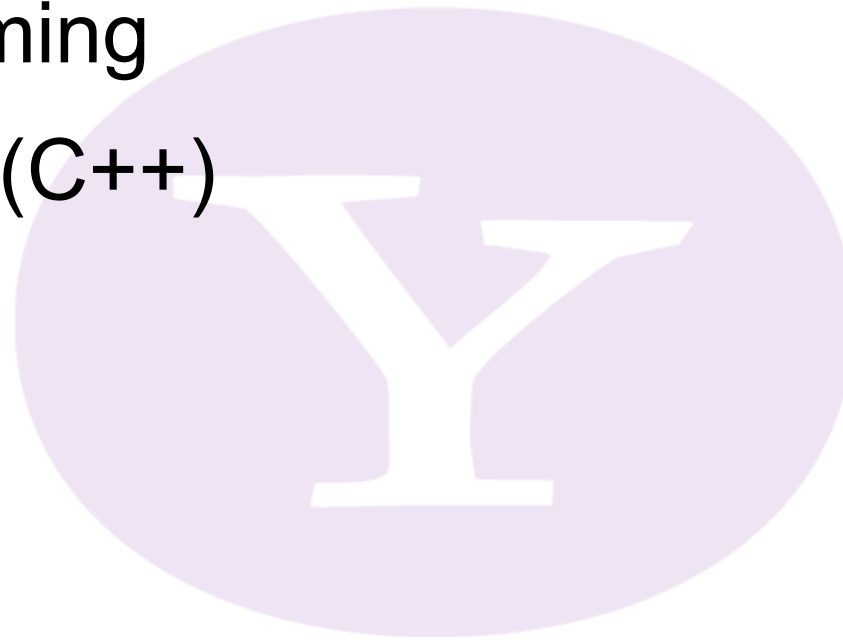
 conf.setMapperClass(MapClass.class);
 conf.setReducerClass(Reduce.class);
 conf.setInputPath(new Path(args[0]));
 conf.setOutputPath(new Path(args[1]));
 JobClient.runJob(conf);
.....
}
```

YAHOO!



# Non-Java Interfaces

- Streaming
- Pipes (C++)
- Pig
- Hive
- Jaql
- Cascading
- ...



YAHOO!





# Streaming

- What about Unix hacks?
  - Can define Mapper and Reduce 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: `/bin/sed -e 's| |\n|g' | /bin/grep .`
- Reducer: `/usr/bin/uniq -c | /bin/awk '{print $2 "\t" $1}'`

YAHOO!



# 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");
 }
 }
};
```

YAHOO!



# 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));
 }
};
```

YAHOO!



# 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>());
}
```

YAHOO!



# Pig – Hadoop Sub-project

- 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';
```

YAHOO!



# Hive – Hadoop Sub-project

- SQL-like interface for querying tables stored as flat-files on HDFS, complete with a meta-data repository
- Developed at Facebook
- In the process of moving from Hadoop contrib to a stand-alone Hadoop sub-project

YAHOO!



# 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:
    - $\max(\min(\text{block\_size}, \text{data}/\#\text{maps}), \text{min\_split\_size})$
- Reduces
  - Unless the amount of data being processed is small
    - $0.95 * \text{num\_nodes} * \text{mapred.tasktracker.reduce.tasks.maximum}$

# 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 !

YAHOO!





# Some handy tools

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



YAHOO!



# 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.output.compression.type*) for SequenceFiles
  - “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
  - LZ0 - faster, but less compression

YAHOO!



# 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

YAHOO!



# 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

YAHOO!



# 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);
```

YAHOO!





# 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() ...
 }
}
```

YAHOO!



# 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

# Profiling

- Set `mapred.task.profile` to `true`
- Use `mapred.task.profile.{maps | reduces}`
- `hprof` support is built-in
- Use `mapred.task.profile.params` to set options for the debugger
- Possibly use `DistributedCache` for the profiler's agent

YAHOO!



# 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          | <a href="#">79</a> |

### Running Jobs

| Running Jobs             |         |         |                |           |                |                   |              |                   |
|--------------------------|---------|---------|----------------|-----------|----------------|-------------------|--------------|-------------------|
| Jobid                    | User    | Name    | Map % complete | Map total | Maps completed | Reduce % complete | Reduce total | Reduces completed |
| <a href="#">job_0001</a> | parthas | quArray | 100.00%        | 22000     | 22000          | 96.34%            | 10           | 8                 |

### Completed Jobs

| Completed Jobs |
|----------------|
| <i>none</i>    |

### Failed Jobs

| Failed Jobs |
|-------------|
| <i>none</i> |

### Local logs

[Log](#) directory, [Job Tracker History](#)

[Hadoop](#), 2006.

YAHOO!



# 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 | <a href="#">Failed/Killed Task Attempts</a> |
|------------------------|------------|-----------|---------|---------|----------|--------|---------------------------------------------|
| <a href="#">map</a>    | 100.00%    | 22000     | 0       | 0       | 22000    | 0      | 0 / 0                                       |
| <a href="#">reduce</a> | 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.

YAHOO!



# Task status

## Hadoop reduce task list for [job\\_0001](#) on [kry1112](#)

---

### Tasks

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

---

[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 | <a href="http://kry1110.inktomisearch.com">kry1110.inktomisearch.com</a> | 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) |        | <a href="#">Last 4KB</a><br><a href="#">Last 8KB</a><br><a href="#">All</a> | <a href="#">3</a> |

[Go back to the job](#)  
[Go back to JobTracker](#)  
[Hadoop, 2006.](#)

YAHOO!





# Drilling down -- logs

## Task Logs: 'task\_0001\_r\_000000\_0'

### STDOUT logs

### STDERR logs

### SYSLOG logs

```
2007-08-27 19:29:05,663 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51883001/6545/0 in:18033=51883001/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,664 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51883101/6545/0 in:18033=51883101/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,664 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51883201/6545/0 in:18033=51883201/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,665 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51883301/6545/0 in:18033=51883301/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,665 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51883401/6545/0 in:18033=51883401/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,667 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51883501/6545/0 in:18033=51883501/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,668 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51883601/6545/0 in:18033=51883601/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,669 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51883701/6545/0 in:18033=51883701/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,671 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51883801/6545/0 in:18033=51883801/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,672 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51883901/6545/0 in:18034=51883901/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,673 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51884001/6545/0 in:18034=51884001/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,673 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51884101/6545/0 in:18034=51884101/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,675 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51884201/6545/0 in:18034=51884201/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,675 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51884301/6545/0 in:18034=51884301/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,676 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51884401/6545/0 in:18034=51884401/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,677 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51884501/6545/0 in:18034=51884501/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,678 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51884601/6545/0 in:18034=51884601/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,679 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51884701/6545/0 in:18034=51884701/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,680 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51884801/6545/0 in:18034=51884801/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,681 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51884901/6545/0 in:18034=51884901/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,681 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51885001/6545/0 in:18034=51885001/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,682 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51885101/6545/0 in:18034=51885101/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,683 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51885201/6545/0 in:18034=51885201/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,683 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51885301/6545/0 in:18034=51885301/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,684 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51885401/6545/0 in:18034=51885401/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,686 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51885501/6545/0 in:18034=51885501/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,686 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51885601/6545/0 in:18034=51885601/2877 [rec/s] out:2=6545/2877 [rec/s]
2007-08-27 19:29:05,687 INFO org.apache.hadoop.streaming.PipeMapRed: R/W/S=51885701/6545/0 in:18034=51885701/2877 [rec/s] out:2=6545/2877 [rec/s]
```

YAPPO!





# Performance

- Is your input splittable?
  - Gzipped files are NOT splittable
  - Use compressed SequenceFiles
- Are partitioners uniform?
- Buffering sizes (especially io.sort.mb)
- Can you avoid Reduce step?
- 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!

YAHOO!



# Q&A

- For more information:
  - Website: <http://hadoop.apache.org/core>
  - Mailing lists:
    - [general@hadoop.apache.org](mailto:general@hadoop.apache.org)
    - [core-dev@hadoop.apache.org](mailto:core-dev@hadoop.apache.org)
    - [core-user@hadoop.apache.org](mailto:core-user@hadoop.apache.org)
  - IRC: #hadoop on irc.freenode.org

YAHOO!

