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
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…

640K ought to be enough for anybody.
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" → \( w_1 \) → "worker" → \( r_1 \) → "Result"

\( w_2 \) → "worker" → \( r_2 \)

\( w_3 \) → "worker" → \( r_3 \)
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 \((k, v) \rightarrow <k', v'>\)*
  - reduce \((k', v') \rightarrow <k', v'>\)*

  - All values with the same key are sent to the same reducer

- The execution framework handles everything else…
Map (String linenum, 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);
Shuffle and Sort: aggregate values by keys

**Map Operations:**
- \( k_1, v_1 \)
- \( k_2, v_2 \)
- \( k_3, v_3 \)
- \( k_4, v_4 \)
- \( k_5, v_5 \)
- \( k_6, v_6 \)

**Reduce Operations:**
- Reduce 1: \( r_1, s_1 \)
- Reduce 2: \( r_2, s_2 \)
- Reduce 3: \( r_3, s_3 \)
MapReduce

- Developer specifies two functions:
  \[ \text{map} \ (k, v) \rightarrow <k', v'>^* \]
  \[ \text{reduce} \ (k', v') \rightarrow <k', v'>^* \]
  - All values with the same key are sent to the same reducer
- The execution framework handles everything else...
MapReduce

- Developer specifies two functions:
  
  **map** \((k, v) \rightarrow <k', v'>\)*
  
  **reduce** \((k', v') \rightarrow <k', v'>\)*

  ▷ 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** \((k', \text{number of partitions}) \rightarrow \text{partition for } k'\)
  
  ▷ Often a simple hash of the key – e.g., \(\text{hash}(k') \mod n\) – that divides up key space for parallel reduce operations

- **combine** \((k', v') \rightarrow <k', v'>\)*
  
  ▷ Mini-reducers that run in memory after the map phase, used as an optimization to reduce network traffic
Shuffle and Sort: aggregate values by keys

k₁ v₁  k₂ v₂  k₃ v₃  k₄ v₄  k₅ v₅  k₆ v₆

map

map

map

map

combine

combine

combine

combine

partition

partition

partition

partition

reduce

reduce

reduce

r₁ s₁

r₂ s₂

r₃ s₃
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?

Shuffle and Sort: aggregate values by keys

- 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

reduce

reduce

reduce

r1 s1
r2 s2
r3 s3
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
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

Hadoop Client

NameNode

SNameNode

read file

return DNs, block ids, etc.

(fsimage/edit)

checkpoint

heartbeat/block report

read blocks

DN | TT
DN | TT
DN | TT
DN | TT
DN | TT
DN | TT
DN | TT
DN | TT
DN | TT
DN | TT

Rack1
Rack2
Rack3
RackN
Writing files

- Hadoop Client requests write
- Write blocks
- Return DNs, etc.
- Checkpoint
- Block report
- Replication pipelining
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

- Submit job
- Deploy job
- Hadoop Client
- JobTracker

Jobs:
- Rack1
- Rack2
- Rack3
- RackN

Tasks:
- Map
- Reduce
- Shuffle
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
package org.apache.hadoop;
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.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, InterruptedException {
      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(Reducer.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    TextInputFormat.addInputPath(job, new Path(args[0]));
    TextOutputFormat.setOutputPath(job, new Path(args[1]));
    job.waitForCompletion(true);
  }
}
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);
}
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, InterruptedException {
            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.setOutputFormatClass(TextOutputFormat.class);
        job.setInputFormatClass(TextInputFormat.class);

        job.setJarByClass(WordCount.class);

        FileInputFormat.addInputPath(job, new Path("input"));
        FileOutputFormat.setOutputPath(job, new Path("output"));

        try {
            job.waitForCompletion(true);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}


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

...
### Job Overview

- **Job Name:** PigLatin.mbf_nonpage_raw.pig
- **State:** RUNNING
- **Uberized:** false
- **Started:** Tue Apr 08 14:39:04 UTC 2014
- **Elapsed:** 12 mins, 58 sec

### ApplicationMaster

<table>
<thead>
<tr>
<th>Attempt Number</th>
<th>Start Time</th>
<th>Node</th>
<th>Logs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tue Apr 08 14:39:01 UTC 2014</td>
<td>gsta641n00.tan.ygrid.yahoo.com:8042</td>
<td>logs</td>
</tr>
</tbody>
</table>

### Task Type

<table>
<thead>
<tr>
<th>Task Type</th>
<th>Progress</th>
<th>Total</th>
<th>Pending</th>
<th>Running</th>
<th>Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td></td>
<td>1517</td>
<td>0</td>
<td>0</td>
<td>1517</td>
</tr>
<tr>
<td>Reduce</td>
<td>10</td>
<td></td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

### Attempt Type

- **Maps:** 0 New, 0 Running, 0 Failed, 23 Killed, 1517 Successful
- **Reduces:** 0 New, 10 Running, 0 Failed, 0 Killed, 0 Successful
<table>
<thead>
<tr>
<th>Task</th>
<th>Progress</th>
<th>State</th>
<th>Start Time</th>
<th>Finish Time</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_1394563654794_387572_m_000236</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:39:36 GMT</td>
<td>29sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000160</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:44:52 GMT</td>
<td>5mins, 45sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000227</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:39:42 GMT</td>
<td>35sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000186</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:45:02 GMT</td>
<td>5mins, 55sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000196</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:45:16 GMT</td>
<td>6mins, 9sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000320</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:39:26 GMT</td>
<td>19sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000018</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:44:54 GMT</td>
<td>5mins, 47sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000020</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:45:04 GMT</td>
<td>31sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000224</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:39:38 GMT</td>
<td>5mins, 50sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000095</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:44:57 GMT</td>
<td>6mins, 34sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000138</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:45:41 GMT</td>
<td>17sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000164</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:44:59 GMT</td>
<td>5mins, 52sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000257</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:39:24 GMT</td>
<td>35sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000213</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:39:42 GMT</td>
<td>36sec</td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000222</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:39:43 GMT</td>
<td></td>
</tr>
<tr>
<td>task_1394563654794_387572_m_000286</td>
<td></td>
<td>SUCCEEDED</td>
<td>Tue, 08 Apr 2014 14:39:06 GMT</td>
<td>Tue, 08 Apr 2014 14:39:24 GMT</td>
<td>17sec</td>
</tr>
</tbody>
</table>
Reduce Tasks for
job_1394563654794_387572

<table>
<thead>
<tr>
<th>Task</th>
<th>Progress</th>
<th>State</th>
<th>Start Time</th>
<th>Finish Time</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>task 1394563654794_387572 r 000001</td>
<td></td>
<td>RUNNING</td>
<td>Tue, 08 Apr 2014 14:45:17 GMT</td>
<td>N/A</td>
<td>7mins, 37sec</td>
</tr>
<tr>
<td>task 1394563654794_387572 r 000003</td>
<td></td>
<td>RUNNING</td>
<td>Tue, 08 Apr 2014 14:45:17 GMT</td>
<td>N/A</td>
<td>7mins, 37sec</td>
</tr>
<tr>
<td>task 1394563654794_387572 r 000007</td>
<td></td>
<td>RUNNING</td>
<td>Tue, 08 Apr 2014 14:45:17 GMT</td>
<td>N/A</td>
<td>7mins, 37sec</td>
</tr>
<tr>
<td>task 1394563654794_387572 r 000008</td>
<td></td>
<td>RUNNING</td>
<td>Tue, 08 Apr 2014 14:45:17 GMT</td>
<td>N/A</td>
<td>7mins, 37sec</td>
</tr>
<tr>
<td>task 1394563654794_387572 r 000009</td>
<td></td>
<td>RUNNING</td>
<td>Tue, 08 Apr 2014 14:45:17 GMT</td>
<td>N/A</td>
<td>7mins, 37sec</td>
</tr>
<tr>
<td>task 1394563654794_387572 r 000002</td>
<td></td>
<td>RUNNING</td>
<td>Tue, 08 Apr 2014 14:45:17 GMT</td>
<td>N/A</td>
<td>7mins, 37sec</td>
</tr>
<tr>
<td>task 1394563654794_387572 r 000000</td>
<td></td>
<td>RUNNING</td>
<td>Tue, 08 Apr 2014 14:45:17 GMT</td>
<td>N/A</td>
<td>7mins, 37sec</td>
</tr>
<tr>
<td>task 1394563654794_387572 r 000005</td>
<td></td>
<td>RUNNING</td>
<td>Tue, 08 Apr 2014 14:45:17 GMT</td>
<td>N/A</td>
<td>7mins, 37sec</td>
</tr>
<tr>
<td>task 1394563654794_387572 r 000006</td>
<td></td>
<td>RUNNING</td>
<td>Tue, 08 Apr 2014 14:45:17 GMT</td>
<td>N/A</td>
<td>7mins, 37sec</td>
</tr>
<tr>
<td>task 1394563654794_387572 r 000004</td>
<td></td>
<td>RUNNING</td>
<td>Tue, 08 Apr 2014 14:45:17 GMT</td>
<td>N/A</td>
<td>7mins, 37sec</td>
</tr>
</tbody>
</table>

Showing 1 to 10 of 10 entries
### Counters for job_1394563654794_387572

#### File System Counters

<table>
<thead>
<tr>
<th>Counter Group</th>
<th>Name</th>
<th>Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FILE: Number of bytes read</td>
<td>449441443106</td>
</tr>
<tr>
<td></td>
<td>FILE: Number of bytes written</td>
<td>884764345715</td>
</tr>
<tr>
<td></td>
<td>FILE: Number of large read operations</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>FILE: Number of read operations</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>FILE: Number of write operations</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>HDFS: Number of bytes read</td>
<td>247833528294</td>
</tr>
<tr>
<td></td>
<td>HDFS: Number of bytes written</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>HDFS: Number of large read operations</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>HDFS: Number of read operations</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>HDFS: Number of write operations</td>
<td>4521</td>
</tr>
</tbody>
</table>

#### Job Counters

<table>
<thead>
<tr>
<th>Name</th>
<th>Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-local map tasks</td>
<td>0</td>
</tr>
<tr>
<td>Killed map tasks</td>
<td>0</td>
</tr>
<tr>
<td>Launched map tasks</td>
<td>0</td>
</tr>
<tr>
<td>Launched reduce tasks</td>
<td>0</td>
</tr>
<tr>
<td>Other local map tasks</td>
<td>0</td>
</tr>
<tr>
<td>Rack-local map tasks</td>
<td>0</td>
</tr>
<tr>
<td>Total time spent by all maps in occupied slots (ms)</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Map-Reduce Framework

<table>
<thead>
<tr>
<th>Name</th>
<th>Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combine input records</td>
<td>0</td>
</tr>
<tr>
<td>Combine output records</td>
<td>0</td>
</tr>
<tr>
<td>Failed Shuffles</td>
<td>0</td>
</tr>
<tr>
<td>Failed splits</td>
<td>0</td>
</tr>
<tr>
<td>GC time elapsed (ms)</td>
<td>5800996</td>
</tr>
<tr>
<td>Input split bytes</td>
<td>635152</td>
</tr>
<tr>
<td>Map input records</td>
<td>1793252681</td>
</tr>
<tr>
<td>Map output bytes</td>
<td>230078595054</td>
</tr>
<tr>
<td>Map output split bytes</td>
<td>0</td>
</tr>
<tr>
<td>Map output bytes</td>
<td>344190857084</td>
</tr>
<tr>
<td>Map output committed bytes</td>
<td>0</td>
</tr>
<tr>
<td>Map output records</td>
<td>1793252681</td>
</tr>
<tr>
<td>Merged Map outputs</td>
<td>0</td>
</tr>
<tr>
<td>Physical memory (bytes)</td>
<td>836515094528</td>
</tr>
<tr>
<td>Reduce input groups</td>
<td>0</td>
</tr>
<tr>
<td>Reduce input records</td>
<td>0</td>
</tr>
<tr>
<td>Reduce output groups</td>
<td>0</td>
</tr>
<tr>
<td>Reduce output records</td>
<td>0</td>
</tr>
<tr>
<td>Reduce shuffle bytes</td>
<td>0</td>
</tr>
<tr>
<td>Shuffle Maps</td>
<td>0</td>
</tr>
<tr>
<td>Spilled Records</td>
<td>34063523042</td>
</tr>
<tr>
<td>Total committed heap usage (bytes)</td>
<td>879833172796</td>
</tr>
<tr>
<td>Virtual memory (bytes)</td>
<td>2811068023488</td>
</tr>
</tbody>
</table>

#### MultiInputCounters

<table>
<thead>
<tr>
<th>Name</th>
<th>Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input records from <em>0</em>.part</td>
<td>0</td>
</tr>
<tr>
<td>Input records from <em>1</em>.tmp-1398980530</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Shuffle Errors

<table>
<thead>
<tr>
<th>Name</th>
<th>Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ID CONNECTION</td>
<td>0</td>
</tr>
<tr>
<td>BAD_ID ID</td>
<td>0</td>
</tr>
<tr>
<td>BAD_ID ERROR</td>
<td>0</td>
</tr>
<tr>
<td>BAD_ID WRONG</td>
<td>0</td>
</tr>
<tr>
<td>BAD_ID WRONG陟</td>
<td>0</td>
</tr>
</tbody>
</table>

#### File Input Format Counters

<table>
<thead>
<tr>
<th>Name</th>
<th>Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes Read</td>
<td>0</td>
</tr>
</tbody>
</table>

#### File Output Format Counters

<table>
<thead>
<tr>
<th>Name</th>
<th>Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes Written</td>
<td>0</td>
</tr>
</tbody>
</table>
$ 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

```bash
$ 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)

- 0.20.1
- 0.20.2
- 0.20.203
- 0.20.204
- 0.20.205
- 0.23
- 0.23.1

Year:
- 2009
- 2010
- 2011
- 2012
Hadoop Releases

- **Download**
  - 20 February, 2014: Release 2.3.0 available
  - 11 December, 2013: Release 0.23.10 available
  - 15 October, 2013: Release 2.2.0 available
- **News**
  - 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
  - 13 April, 2013: Release 2.0.4-beta 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
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
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

Hadoop Client

request write

return DNs, etc.

write blocks

NN1/ns1  NN2/ns2  NN3/ns3  NN4/ns4

SNameNode per NN

fsimage/edit copy

checkpoint

fs sync

Backup NN per NN

block report

block report

replication pipelining

DN | NM  DN | NM  DN | NM  DN | NM
DN | NM  DN | NM  DN | NM  DN | NM
DN | NM  DN | NM  DN | NM  DN | NM
DN | NM  DN | NM  DN | NM  DN | NM

Rack1  Rack2  Rack3  RackN
Running jobs
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/](http://hadoopsummit.org/)
    - similar ones for HBase, Hive, Pig, …
Questions?

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