Hadoop In 45 Minutes or Less

Large-Scale Data Processing for Everyone

Tom Wheeler

Who Am I?

- Principal Software Engineer at Object Computing, Inc.
- I worked on large-scale data processing in a previous job
  - *If only I'd had Hadoop back then...*

What I'm Going to Cover

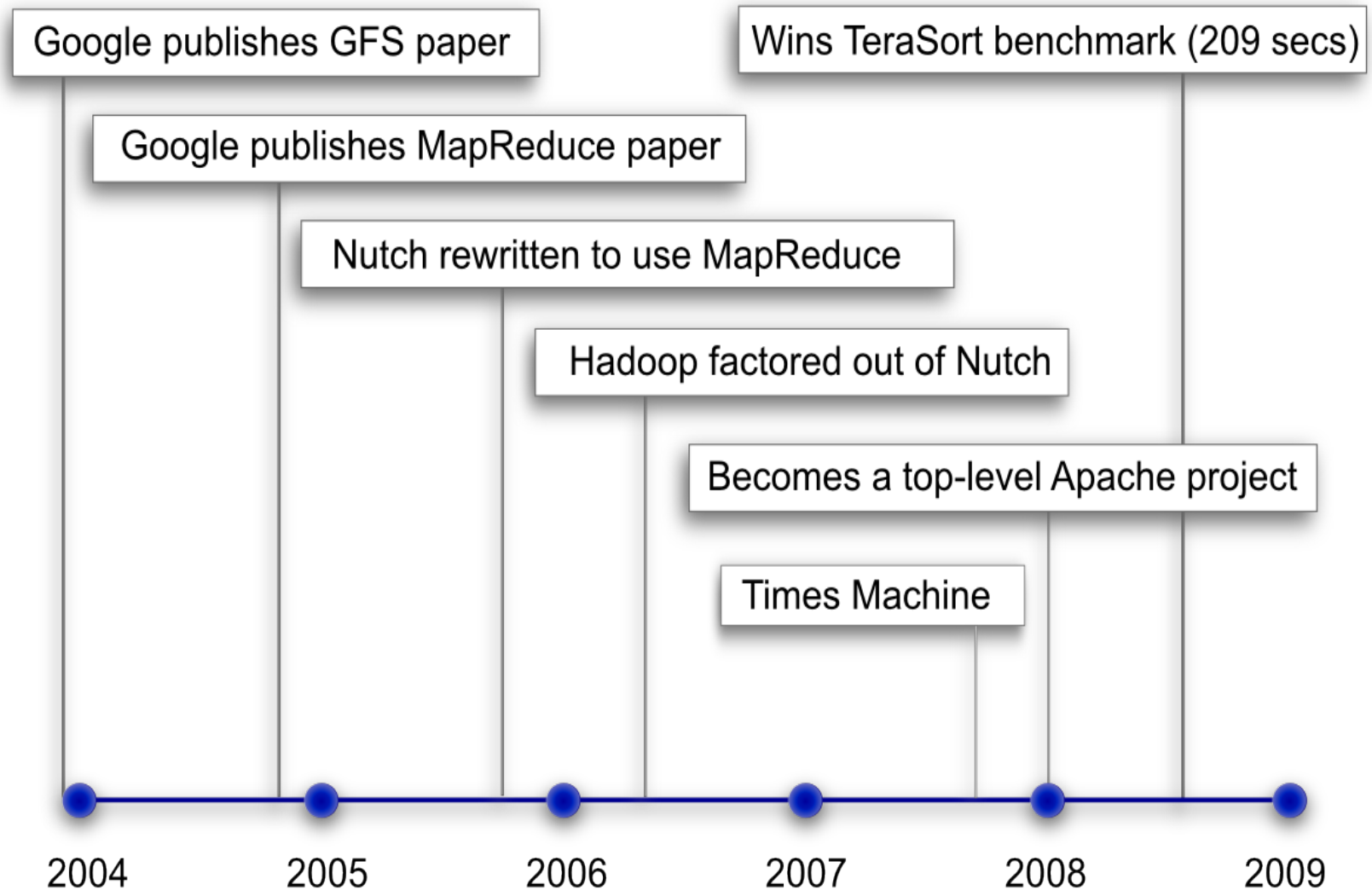
- I'll explain what Hadoop is
- I'll tell you what problems it can *(and can't)* solve
- I'll describe how it works
- I'll show examples so you can see it in action

What is Hadoop?

It's a framework for **large**-scale **data** processing:

- Inspired by Google's architecture: Map Reduce and GFS
- A top-level Apache project – Hadoop is open source
- Written in Java, plus a few shell scripts

How Did Hadoop Originate?



Why Should I Care About Hadoop?

- Fault-tolerant hardware is **expensive**
- Hadoop is designed to run on cheap commodity hardware
- It automatically handles data replication and node failure
- It does the hard work – you can focus on processing data

Who's Using Hadoop?

Source: Hadoop Wiki, September 2009

What Features Does Hadoop Offer?

- API + implementation for working with Map Reduce
- More importantly, it provides infrastructure

Hadoop Infrastructure

- Job configuration and efficient scheduling
- Browser-based monitoring of important cluster stats
- Handling failures in both computation and data nodes
- A distributed FS optimized for *HUGE* amounts of data

When is Hadoop a Good Choice?

- When you must process *lots* of unstructured data
- When your processing can easily be made parallel
- When running batch jobs is acceptable
- When you have access to lots of cheap hardware

When is Hadoop Not A Good Choice?

- For intense calculations with little or no data
- When your processing cannot be easily made parallel
- When your data is not self-contained
- When you need interactive results
- If you own stock in Cray!

Hadoop Examples/Anti-Examples

Hadoop would be a good choice for...

- Indexing log files
- Sorting vast amounts of data
- Image analysis

Hadoop would be a poor choice for...

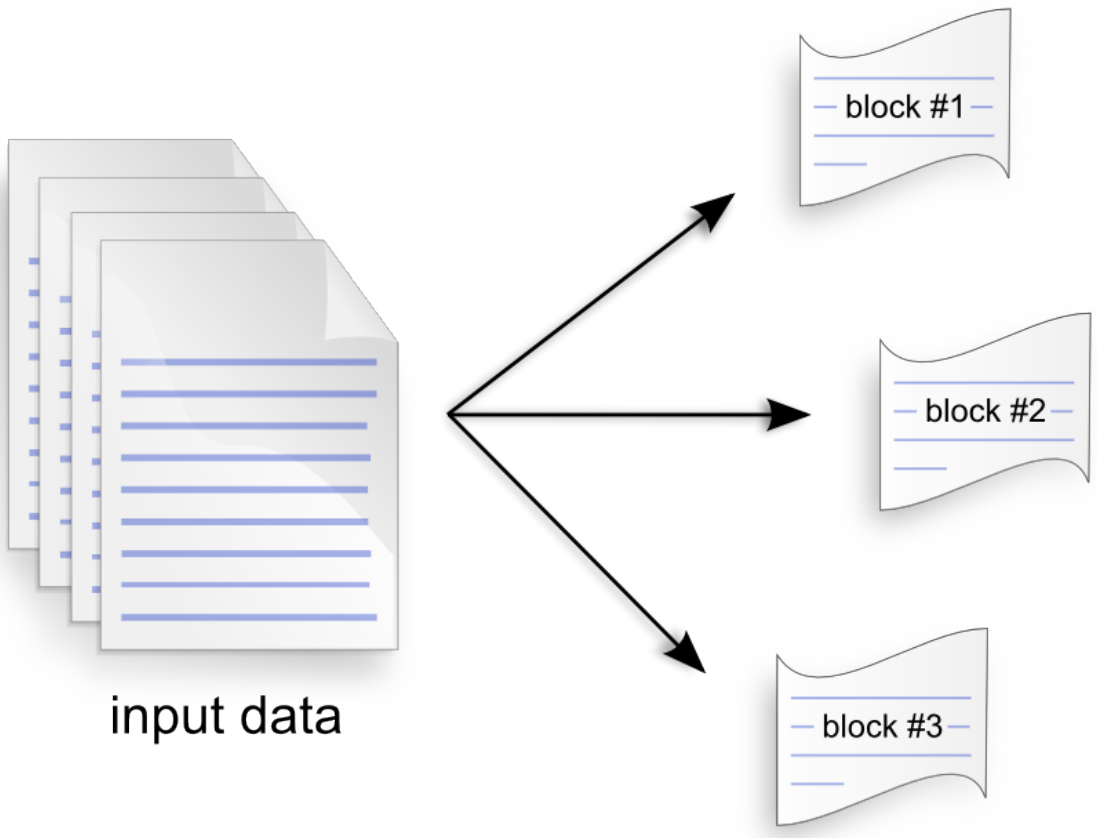
- Figuring Pi to 1,000,000 digits
- Calculating Fibonacci sequences
- A general RDBMS replacement

HDFS Overview

HDFS is perhaps Hadoop's most interesting feature.

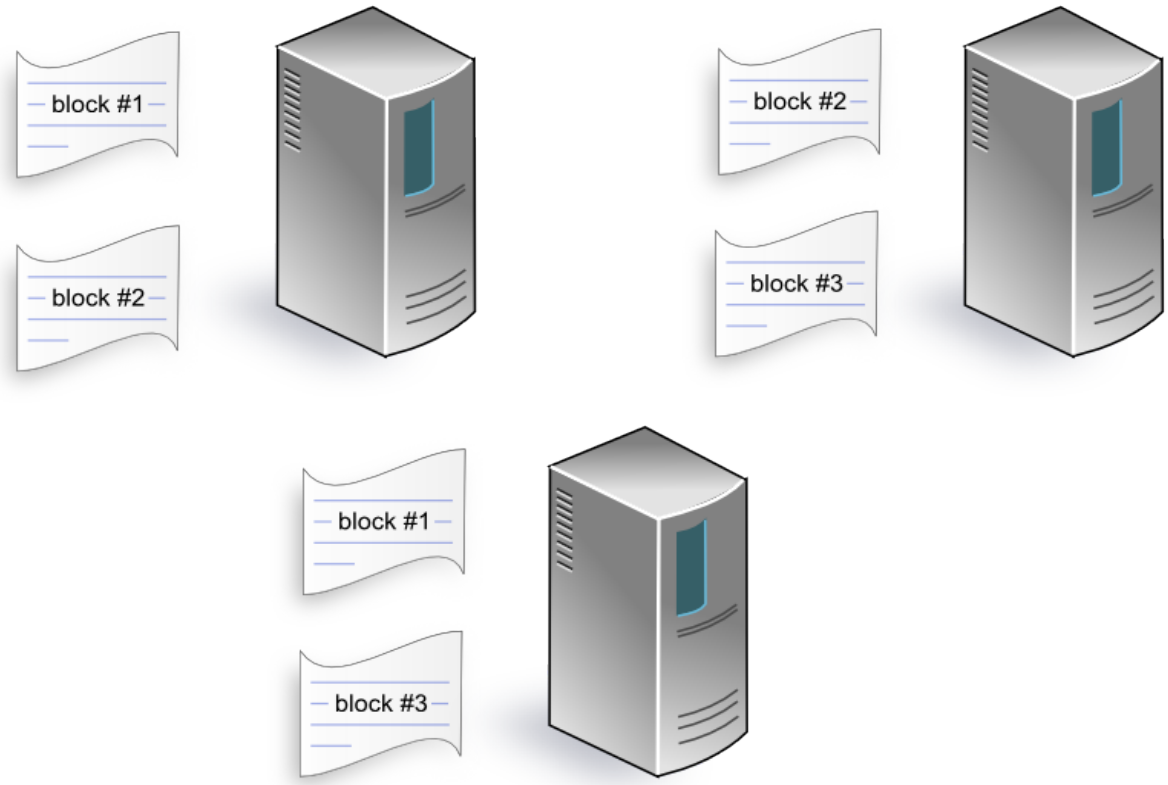
- HDFS = Hadoop Distributed Filesystem (userspace)
- Inspired by Google File System
- High aggregate throughput for streaming large files
- Replication and locality

How HDFS Works: Splits



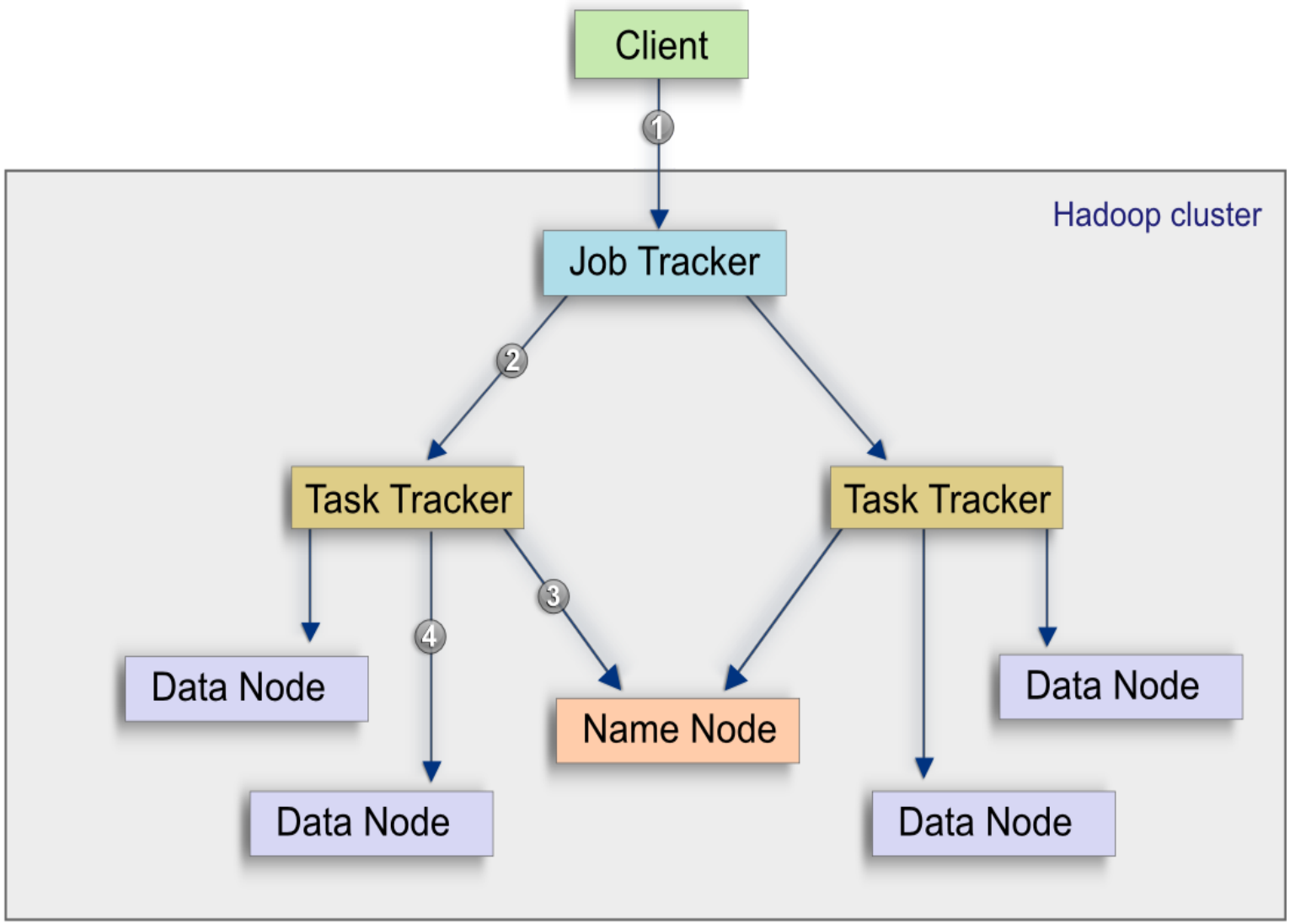
- Data copied into HDFS is split into blocks
- Typical block size: UNIX = 4KB vs. HDFS = 128MB

How HDFS Works: Replication

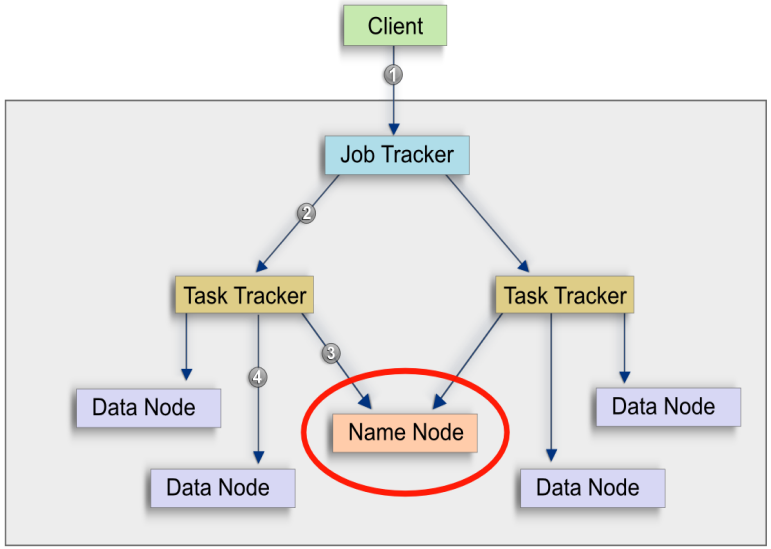


- Each data blocks is replicated to multiple machines
- Allows for node failure without data loss

Hadoop Architecture Overview

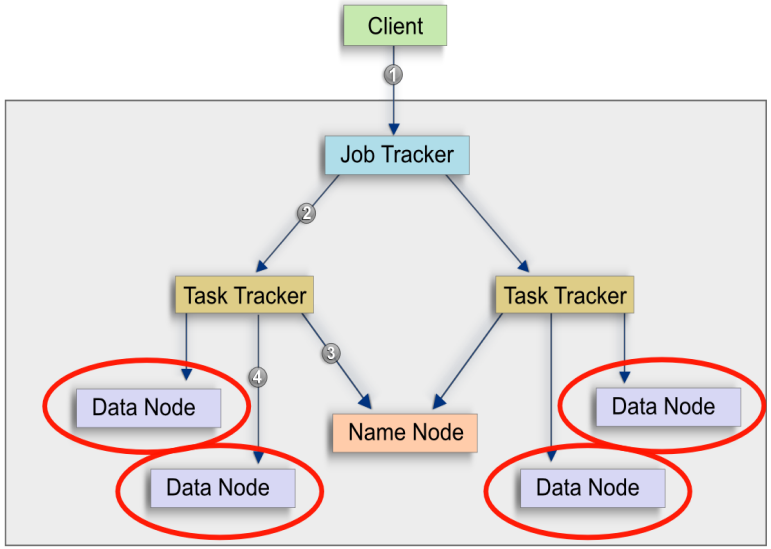


The Hadoop Cast of Characters: Name Node



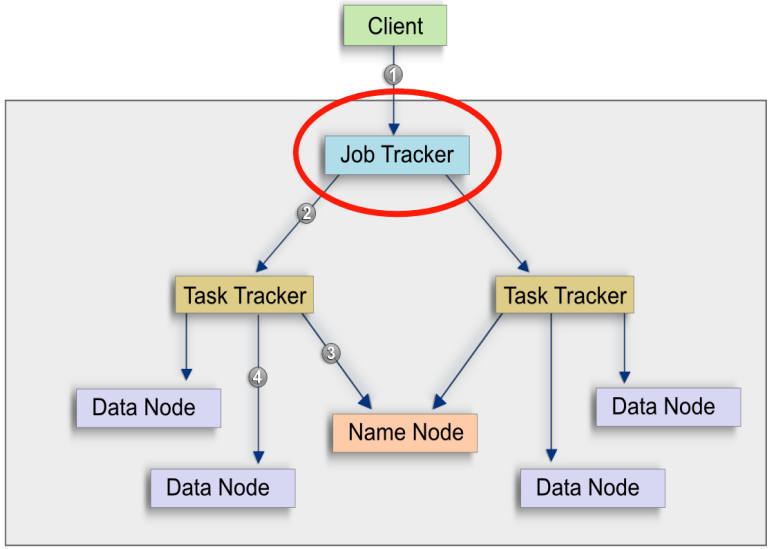
- There is only one (active) name node per cluster
- It manages the filesystem namespace and metadata
- SPOF: the one place to spend $$$ for good hardware

The Hadoop Cast of Characters: Data Node



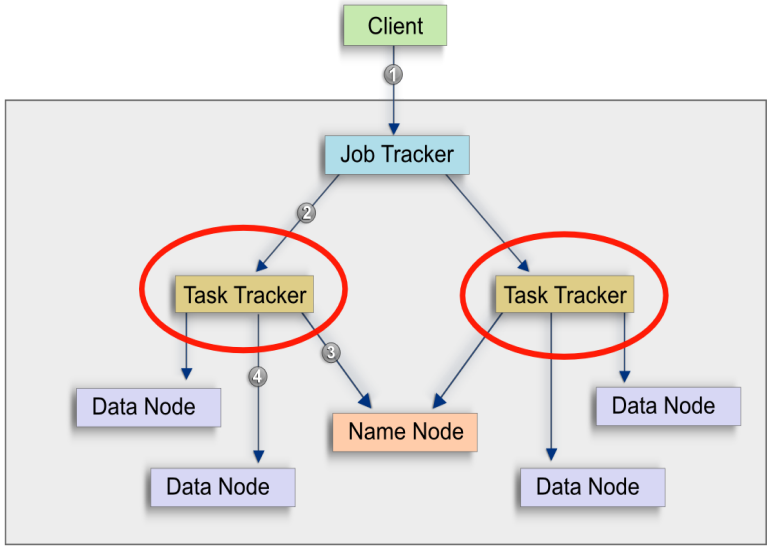
- There are typically lots of data nodes
- It manages data blocks + serves them to clients
- Data is replicated – failure is no big deal

The Hadoop Cast of Characters: Job Tracker



- There is exactly one job tracker per cluster
- Receives job requests submitted by client
- Schedules and monitors MR jobs on task trackers

The Hadoop Cast of Characters: Task Tracker



- There are typically lots of task trackers
- Responsible for executing MR operations
- Reads blocks from data nodes

Hadoop Modes of Operation

Hadoop supports three modes of operation:

- Standalone
- Pseudo-distributed
- Fully-distributed

Installing Hadoop

The installation process, for distributed modes:

- Requirements: Linux, Java 1.6, sshd, rsync
- Configure SSH for password-free authentication
- Unpack Hadoop distribution
- Edit a few configuration files
- Format the DFS on the name node
- Start all the daemon processes

Running Hadoop

The basic steps for running a Hadoop job are:

- Compile your job into a JAR file
- Copy input data into HDFS
- Execute `bin/hadoop jar` with relevant args
- Monitor tasks via Web interface (optional)
- Examine output when job is complete

A Simple Hadoop Job

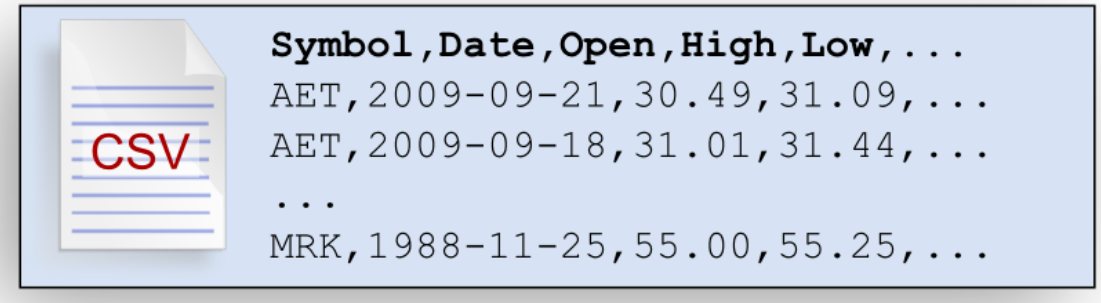
I'll demonstrate Hadoop with a simple Map Reduce example

- Input is historical data for 30 stocks, 1987-2009
- Records are CSV: symbol, low price, high price, etc.
- Goal is to find largest intra-day price fluctuation
- Output is one record per stock, showing max delta

Our Hadoop Job Illustrated

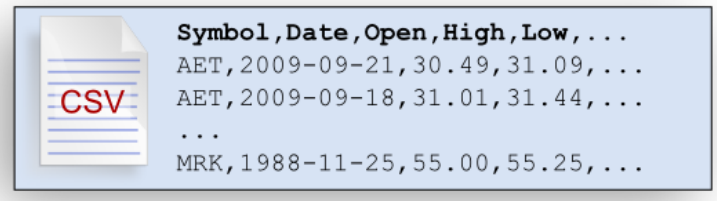
- Hadoop is all about data processing
- Seeing the data will help to explain the job

Our Hadoop Job Illustrated: Mapper Input



Input to map function

Our Hadoop Job Illustrated: Mapper Output



Input to map function

```
key=AET, value=0.61
key=AET, value=0.33
...
key=MRK, value=0.25
```

Output of map function

Our Hadoop Job Illustrated: Reducer Output

```
key=AET, value=0.61
key=AET, value=0.33
...
key=MRK, value=0.25
```

Output of map function

```
key=AET, value=13.75
key=AXP, value=15.12
...
key=MRK, value=29.0
```

Output of reduce function

Show Me the Code

- Now that you understand the data... let's see the code!

Mapper for Stock Analyzer (part 1)

```
1  public class StockAnalyzerMapper extends MapReduceBase
2      implements Mapper<LongWritable, Text, Text,
           FloatWritable> {
3
4    @Override
5    public void map(LongWritable key, Text value,
6          OutputCollector<Text, FloatWritable> output,
               Reporter reporter)
7          throws IOException {
8
9       String record = value.toString();
10
11      if (record.startsWith("Symbol")) {
12         // ignore header row
13         return;
14      }
```

Mapper for Stock Analyzer (part 2)

```
1
2         String[] fields = record.split(",");
3         String symbol = fields[0];
4         String high = fields[3];
5         String low = fields[4];
6
7         float lowValue = Float.parseFloat(low);
8         float highValue = Float.parseFloat(high);
9         float delta = highValue - lowValue;
10
11        output.collect(new Text(symbol),
12                       new FloatWritable(delta));
13    }
14 }
```

Reducer for Stock Analyzer

```
1  public class StockAnalyzerReducer extends MapReduceBase
2          implements Reducer<Text, FloatWritable, Text,
               FloatWritable> {
3
4      @Override
5      public void reduce(Text key, Iterator<FloatWritable>
           values,
6          OutputCollector<Text, FloatWritable> output,
               Reporter reporter)
7          throws IOException {
8
9        float maxValue = Float.MIN_VALUE;
10       while (values.hasNext()) {
11          maxValue = Math.max(maxValue, values.next().get());
12       }
13
14       output.collect(key, new FloatWritable(maxValue));
15     }
16 }
```

Job Conf for Stock Analyzer (part 1)

```
1
2  public class StockAnalyzerConfig {
3
4     public static void main(String[] args) throws Exception {
5        if (args.length != 2) {
6           System.err.println("Usage: StockAnalyzerConfig <
                input> <output>");
7           System.exit(1);
8        }
9
10       JobConf conf = new JobConf(StockAnalyzerConfig.class);
11       conf.setJobName("Stock analyzer");
12
13       Path inputPath = new Path(args[0]);
14       FileInputFormat.addInputPath(conf, inputPath);
15
16       Path outputPath = new Path(args[1]);
17       FileOutputFormat.setOutputPath(conf, outputPath);
```

Job Conf for Stock Analyzer (part 2)

```
1
2           conf.setMapperClass(StockAnalyzerMapper.class);
3           conf.setReducerClass(StockAnalyzerReducer.class);
4
5           conf.setOutputKeyClass(Text.class);
6           conf.setOutputValueClass(FloatWritable.class);
7
8           JobClient.runJob(conf);
9       }
10  }
```

Demonstration

Let's run the example so we can see Hadoop in action!

Conclusion

- Tonight I've explained the basics of Hadoop
- It's a highly-scalable framework for data processing
- It's inspired by Google
- And essential to Yahoo, Facebook and many others
- Hopefully you can find a use for it too!