Hadoop

Dawid Weiss

Institute of Computing Science
Poznań University of Technology

2008

1 Open Source Map-Reduce: Hadoop
- About
- Cluster Configuration

2 Programming Hadoop

3 Hadoop in Reality

1 Open Source Map-Reduce: Hadoop
- About
- Cluster Configuration

2 Programming Hadoop

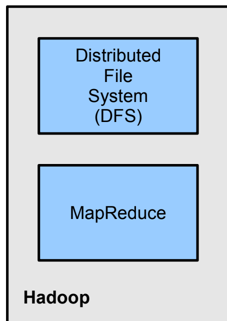
3 Hadoop in Reality

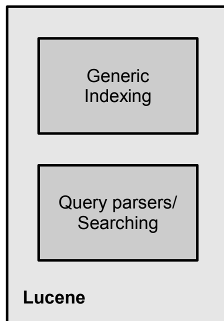
The Hadoop project

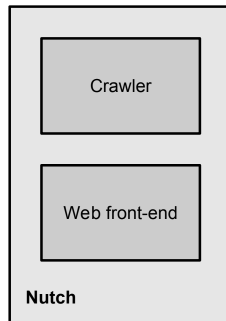
- Mike Cafarella, Doug Cutting and others.
- Pronunciation not entirely clear. . .
- Project forked from Apache Nutch.
- Impressive, dynamic growth. Yahoo! involvement.
- Apache-license.

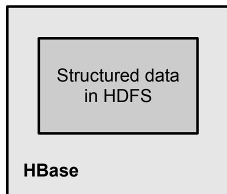
http://lucene.apache.org/hadoop/

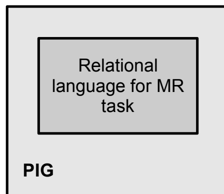
The open source MR landscape











HDFS assumptions

HDFS is inspired by GFS (Google File System).

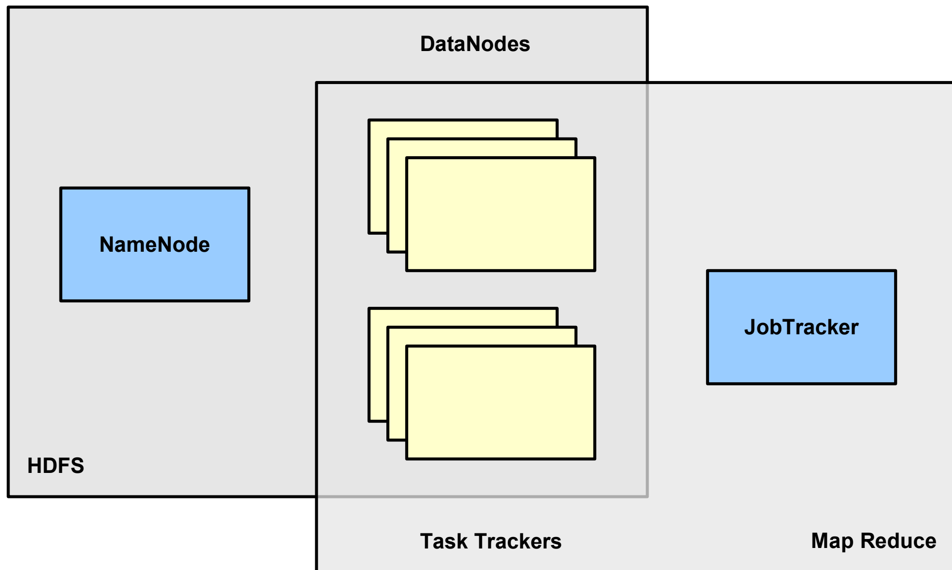
Design goals:

- expect hardware failures (processes, disk, nodes),
- streaming data access, large files (TB of data),
- simple coherence model (one writer, many readers),
- optimization of computation in map-reduce (data locality),
- single master (name node), multiple slaves (data nodes).

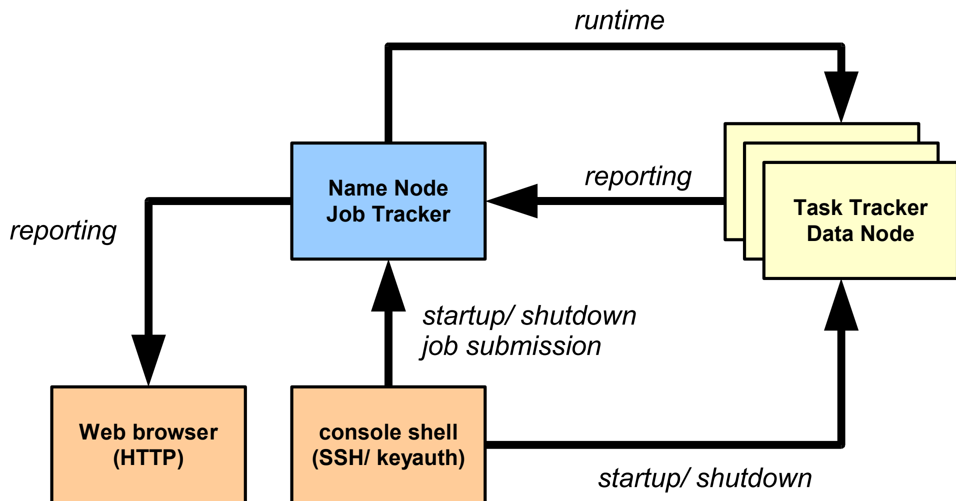
1 Open Source Map-Reduce: Hadoop
- About
- Cluster Configuration

2 Programming Hadoop

3 Hadoop in Reality



Logical Hadoop components



Typical physical layout

Prerequisites

Installation/ operation recommendations:

- Java `1.5.x` or higher.
- Linux (or Windows under CygWin).
- Password-less SSH/ key cert.
- rsync, NFS and homogeneous paths make life easier.

Setting up a cluster (in a pill)

1. Unpack Hadoop.
2. Configure: `hadoop-env.sh`, `hadoop-site.xml` and `slaves`.

```
1 fs.default.name      -> masterhost:50010
2 mapred.job.tracker   -> masterhost:50551
```

3. Prepare password-less SSH to the slaves.
4. Check firewall (!).
5. Distribute files.
6. Start the cluster.

Hadoop Web site contains more verbose instructions.

Setting up local development environment

1. Unpack Hadoop.
2. Configure Hadoop as if you worked with a cluster.
3. Start Eclipse (Linux) or Eclipse from CygWin's shell (Windows).
4. Develop your map and reduce classes.
5. Set your job configuration to use local file system and in-process job tracker:

```
1 conf.set("mapred.job.tracker", "local");
2 conf.set("fs.default.name", "local");
```

NameNode '150.254.130.24:50010'

Started: Wed Nov 28 19:00:28 CET 2007
Version: 0.15.0, r589881
Compiled: Mon Oct 29 15:01:10 PDT 2007 by cutting

Browse the filesystem

Cluster Summary

Capacity	:	476.85 GB
DFS Remaining	:	354.72 GB
DFS Used	:	260 KB
DFS Used%	:	0 %
Live Nodes	:	13
Dead Nodes	:	0

Live Datanodes : 13

Node	Last Contact	Admin State	Size (GB)	Used (%)	Used (%)	Remaining (GB)	Blocks
lab-143-1	1	In Service	36.68	0		27.27	0
lab-143-10	1	In Service	36.68	0		27.29	0
lab-143-11	1	In Service	36.68	0		27.26	0
lab-143-13	1	In Service	36.68	0		27.28	0
lab-143-14	1	In Service	36.68	0		27.28	0
lab-143-15	1	In Service	36.68	0		27.29	0
lab-143-2	1	In Service	36.68	0		27.28	0
lab-143-3	1	In Service	36.68	0		27.28	0
lab-143-4	1	In Service	36.68	0		27.28	0
lab-143-5	1	In Service	36.68	0		27.29	0
lab-143-6	0	In Service	36.68	0		27.29	0
lab-143-7	1	In Service	36.68	0		27.3	0
lab-143-8	1	In Service	36.68	0		27.32	0

Dead Datanodes : 0

1 Open Source Map-Reduce: Hadoop

2 Programming Hadoop
- Jobs
- Reading and splitting the input data
- Affecting shuffling: partitioning and sorting
- Tracking progress: counters
- Job launching

3 Hadoop in Reality

1 Open Source Map-Reduce: Hadoop

2 Programming Hadoop
- Jobs
- Reading and splitting the input data
- Affecting shuffling: partitioning and sorting
- Tracking progress: counters
- Job launching

3 Hadoop in Reality

Map Reduce jobs:

- Classes (JARs) in Java.
- Hadoop Streaming (arbitrary shell commands).
- C/C++ APIs to HDFS.

This lecture focuses on Java jobs.

Hadoop Java API: Fundamentals

```
1 public interface Mapper<
2     K1 extends WritableComparable, V1 extends Writable,
3     K2 extends WritableComparable, V2 extends Writable>
4   extends JobConfigurable, Closeable
5 {
6   void map(K1 key, V1 value, OutputCollector<K2, V2> output,
7         Reporter reporter)
8     throws IOException;
9 }
```

```
1 public interface Reducer<
2     K2 extends WritableComparable, V2 extends Writable,
3     K3 extends WritableComparable, V3 extends Writable>
4     extends JobConfigurable, Closeable
5 {
6   void reduce(K2 key, Iterator<V2> values, OutputCollector<K3, V3> output,
7         Reporter reporter)
8     throws IOException;
9 }
```

Note shared parameter names between the mapper and the reducer, these must be compatible types.

Writable and WritableComparable

Hadoop uses custom data type serialization/ RPC mechanism. Hence the two `Writable` and `WritableComparable` interfaces.

A number of default implementations are available:

- Primitive types:
  `IntWritable`, `Text`, `FloatWritable`, ...
- Arrays and dictionaries:
  `MapWritable`, `ArrayWritable`, ...
- Record compiler generated classes (`rcc`, later).

Example: word count

```
1 /**
2  * Counts the words in each line.
3  *
4  * For each line of input, break the line into words and emit them as
5  * (<b>word</b>, <b>1</b>).
6  */
7 public class Map extends MapReduceBase
8   implements Mapper<LongWritable, Text, Text, IntWritable> {
9
10   private final static IntWritable one = new IntWritable(1);
11   private Text word = new Text();
12
13   public void map(LongWritable key, Text value,
14     OutputCollector<Text, IntWritable> output,
15     Reporter reporter) throws IOException
16   {
17     final String line = value.toString();
18     final StringTokenizer itr = new StringTokenizer(line);
19     while (itr.hasMoreTokens()) {
20       word.set(itr.nextToken());
21       output.collect(word, one);
22     }
23   }
24 }
```

Example: word count

```
1 /**
2  * A reducer class that just emits the sum of the input values.
3  */
4 public class Reduce extends MapReduceBase
5   implements Reducer<Text, IntWritable, Text, IntWritable> {
6 
7   public void reduce(Text key, Iterator<IntWritable> values,
8     OutputCollector<Text, IntWritable> output,
9     Reporter reporter) throws IOException
10  {
11    int sum = 0;
12    while (values.hasNext()) {
13      sum += values.next().get();
14    }
15    output.collect(key, new IntWritable(sum));
16  }
17 }
```

Example: word count

```
1 public static void main(String[] args) throws IOException {
2   final JobConf conf = new JobConf(WordCount.class);
3   conf.setJobName("wordcount");
4
5   // The keys are words (strings).
6   conf.setOutputKeyClass(Text.class);
7
8   // The values are counts (ints).
9   conf.setOutputValueClass(IntWritable.class);
10
11  // Set mapper, reducer and combiner (‘local’ aggregation)
12  conf.setMapperClass(MapClass.class);
13  conf.setCombinerClass(Reduce.class);
14  conf.setReducerClass(Reduce.class);
15
16  // Set input and output paths. Note ‘virtual’ paths.
17
18  conf.setInputPath(new Path(input));
19  conf.setOutputPath(new Path(output));
20
21  // Set job properties here (to run locally for example).
22  // ...
23
24  JobClient.runJob(conf);
25 }
```

1 Open Source Map-Reduce: Hadoop

2 Programming Hadoop
- Jobs
- Reading and splitting the input data
- Affecting shuffling: partitioning and sorting
- Tracking progress: counters
- Job launching

3 Hadoop in Reality

Data access abstraction layer

How does Hadoop split input files into keys an values?

Data access abstraction layer

How does Hadoop split input files into keys an values?

Record access API:

`InputFormat` → `InputSplit[]` → `RecordReader`

Default implementation:

`TextInputFormat` → `InputSplit[]` → `LineRecordReader`

InputFormat

```
1 public interface InputFormat<K extends WritableComparable,V extends Writable>
2 {
3   void validateInput(JobConf job);
4
5   InputSplit[] getSplits(JobConf job, int numSplits /* hint! */);
6
7   RecordReader<K,V> getRecordReader(
8       InputSplit split, JobConf job, Reporter reporter);
9 }
```

Note `InputFormat` is not tied to the filesystem, but implies the record format (returns a compatible `RecordReader`). Examples:

- `TextInputFormat` (file-based, K: line number, V: a single line, splits span many files),
- `KeyValueTextInputFormat` (file-based, K,V: lines split on a given byte separator),
- `SequenceFileInputFormat` (file-based, list of binary key/ value pairs).

Sequence files

- Sequence files store arbitrary `Writeable` key/ value pairs.
- Sequence files can be compressed internally (block and record compression).
- Complex structures can be easily created/stored with a combination of `rcc` compiler and sequence files.

Sequence file using local file system

```
// These would be normally provided by Hadoop's MR
final FileSystem fs = new RawLocalFileSystem();
fs.setConf(new Configuration());

// Output path for the sequence file.
final Path output = new Path("sequence-file.sf");

// Create a writer,
final SequenceFile.Writer writer = SequenceFile.createWriter(
    fs, conf, output,
    Text.class, IntWritable.class,
    SequenceFile.CompressionType.BLOCK);

// Write to the writer.
writer.append(new Text("stary"), new IntWritable(0xbaca));

// Close it.
writer.close();
```

Using sequence files in a Map-Reduce job

```
1    // ...
2    jobConf.setInputFormat(SequenceFileInputFormat.class);
3    jobConf.setOutputFormat(SequenceFileOutputFormat.class);
4    // ...
```

Note that key and value classes must be compatible with the mapper and the reducer.

Writing a custom InputFormat?

- Start from one of the defined base classes: FileInputFormat, MultiFileInputFormat.
- Implement a custom RecordReader, look and extensively reuse existing code (LineRecordReader):

```
boolean next(K key, V value) throws IOException;
K createKey();
V createValue();
```

Writing a custom `InputFormat`?

- Start from one of the defined base classes: `FileInputFormat`, `MultiFileInputFormat`.
- Implement a custom `RecordReader`, look and extensively reuse existing code (`LineRecordReader`):

```
1 boolean next(K key, V value) throws IOException;
2 K createKey();
3 V createValue();
```

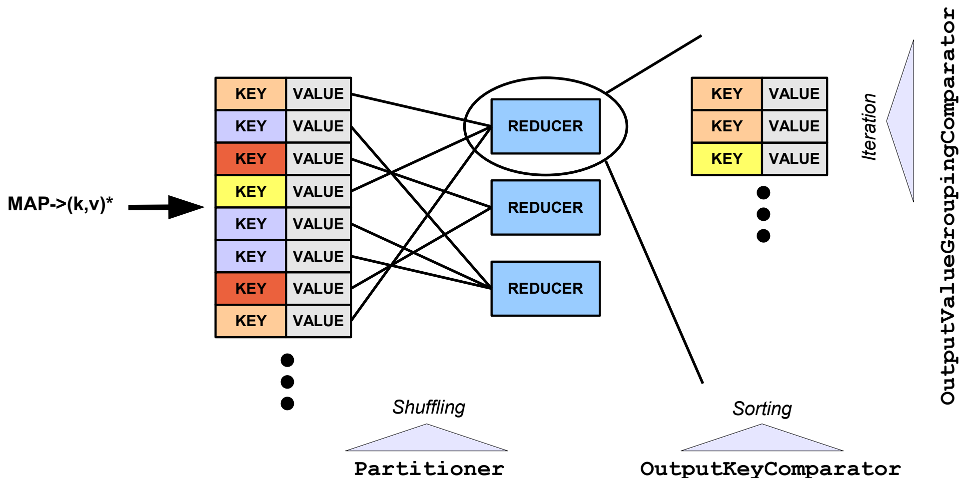
What happens if a split is not on a record boundary?

1 Open Source Map-Reduce: Hadoop

2 Programming Hadoop
- Jobs
- Reading and splitting the input data
- Affecting shuffling: partitioning and sorting
- Tracking progress: counters
- Job launching

3 Hadoop in Reality

Data shuffling and sorting



Shuffling and sorting

- Partitioning is performed immediately, unless a combiner function has been specified.
- For the combiner, data is sorted locally, the combiner is executed, the result is partitioned.
- Sorting uses merge sort. The in-memory buffer size is configurable (larger buffer means fewer passes).

Overriding the defaults

Possible via properties on `JobConf`:

```
1 conf.setOutputKeyComparatorClass(MyComparator.class);
```

- Partitioner should **balance** keys among reducers.
- Partitioner may work on a subset of the key (i.e., a domain of the entire URI).
- The comparator above is not `java.util.Comparator`!

Custom comparator example

```
1 public class CustomTextComparator extends WritableComparator {
2     private Collator collator;
3 
4     public CustomTextComparator() {
5         super(Text.class);
6 
7         final Locale locale = new Locale("pl");
8         collator = Collator.getInstance(locale);
9     }
10 
11     public int compare(WritableComparable a, WritableComparable b) {
12         synchronized (collator) {
13             return collator.compare(
14                 ((Text) a).toString(), ((Text) b).toString());
15         }
16     }
17 }
```

1 Open Source Map-Reduce: Hadoop

2 Programming Hadoop
- Jobs
- Reading and splitting the input data
- Affecting shuffling: partitioning and sorting
- Tracking progress: counters
- Job launching

3 Hadoop in Reality

Counters

Maps and reduces receive a `Reporter` object, which has the following methods:

```
1 void setStatus(String status);
2 void incrCounter(Enum key, long amount);
3 // ... (and other methods, not relevant here)
```

Note that counters are subclasses of `Enum`. This is quite convenient:

```
1 public enum MyStatistics {
2     COUNTER_NAME, ANOTHER_COUNTER,
3 }
```

```
1 public void map(...) throws IOException {
2         reporter.incrCounter(MyStatistics.COUNTER_NAME, 1);
3         reporter.incrCounter(MyStatistics.ANOTHER_COUNTER, 5);
4         // ...
5 }
```

1 Open Source Map-Reduce: Hadoop

2 Programming Hadoop
- Jobs
- Reading and splitting the input data
- Affecting shuffling: partitioning and sorting
- Tracking progress: counters
- Job launching

3 Hadoop in Reality

- Pack your Java job as a `jar` file.
- Place any extra jars under `/lib` folder inside the `jar`.
- Submit the job from command line:

```
1 hadoop jar [job-file.jar] my.package.MyMainClass param1 param2 ...
```

1 Open Source Map-Reduce: Hadoop

2 Programming Hadoop

3 Hadoop in Reality

The trickery of Hadooping. . .

- Very fast development.
- Windows installation often broken (scripts, paths).
- Not source-level documentation scarce and not really up-to-speed with the code.
- Real setup of a distributed cluster requires some initial work (account setup, moving distributions around).

Power of Hadoop

- Writing map reduce jobs relatively easy.
- It really works.
- Compatible with Amazon's paid cluster infrastructure.