合肥学院
HEFEI UNIVERSITY

Distributed Computing

Lesson 23: MapReduce with Hadoop

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://www.it-weise.de

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

1. MapReduce & Hadoop
2. Examples

website

Overview

- What is MapReduce?
- Distinguish use cases of Hadoop/MapReduce, MPI, Servlets
- Getting to know the MapReduce support of the Hadoop framework

HTTP, Web Services, and Java Servlets are ideal for

HTTP/Web Services/Java Servlet Use Case

HTTP, Web Services, and Java Servlets are ideal for:

- Applications with request-response and client-server scheme

HTTP/Web Services/Java Servlet Use Case

HTTP, Web Services, and Java Servlets are ideal for:

- Applications with request-response and client-server scheme
- Data processing does not take place in a natural distributed fashion

HTTP/Web Services/Java Servlet Use Case

HTTP, Web Services, and Java Servlets are ideal for:

- Applications with request-response and client-server scheme
- Data processing does not take place in a natural distributed fashion
- Requests are answered by single threads, cooperative parallelism does not improve overall performance

HTTP/Web Services/Java Servlet Use Case

HTTP, Web Services, and Java Servlets are ideal for:

- Applications with request-response and client-server scheme
- Data processing does not take place in a natural distributed fashion
- Requests are answered by single threads, cooperative parallelism does not improve overall performance
- We want to combine different applications in a heterogeneous system

MPI Use Case

MPI is ideal for the following situation

MPI Use Case

MPI is ideal for the following situation:

- Communication is expensive and the bottleneck of our application. It must be done as efficiently as possible.

MPI Use Case

MPI is ideal for the following situation:

- Communication is expensive and the bottleneck of our application. It must be done as efficiently as possible.
- Available hardware is homogenous, e.g., we have a cluster of the same server components connected by an Infiniband network, so we can use highly-specialized communication libraries.

MPI Use Case

MPI is ideal for the following situation:

- Communication is expensive and the bottleneck of our application. It must be done as efficiently as possible.
- Available hardware is homogenous, e.g., we have a cluster of the same server components connected by an Infiniband network, so we can use highly-specialized communication libraries.
- Processes need to be organized in groups or topological structures (see the heat simulation example) to make efficient use of collective communication to achieve high performance.

MPI Use Case

MPI is ideal for the following situation:

- Communication is expensive and the bottleneck of our application. It must be done as efficiently as possible.
- Available hardware is homogenous, e.g., we have a cluster of the same server components connected by an Infiniband network, so we can use highly-specialized communication libraries.
- Processes need to be organized in groups or topological structures (see the heat simulation example) to make efficient use of collective communication to achieve high performance. $\Longrightarrow$ Problems have regular patterns.

MPI Use Case

MPI is ideal for the following situation:

- Communication is expensive and the bottleneck of our application. It must be done as efficiently as possible.
- Available hardware is homogenous, e.g., we have a cluster of the same server components connected by an Infiniband network, so we can use highly-specialized communication libraries.
- Processes need to be organized in groups or topological structures (see the heat simulation example) to make efficient use of collective communication to achieve high performance. $\Longrightarrow$ Problems have regular patterns.
- Basically anything where we need lots of/repetitive interprocess communication *during* the computation.

MPI Use Case

MPI is ideal for the following situation:

- Communication is expensive and the bottleneck of our application. It must be done as efficiently as possible.
- Available hardware is homogenous, e.g., we have a cluster of the same server components connected by an Infiniband network, so we can use highly-specialized communication libraries.
- Processes need to be organized in groups or topological structures (see the heat simulation example) to make efficient use of collective communication to achieve high performance. $\Longrightarrow$ Problems have regular patterns.
- Basically anything where we need lots of/repetitive interprocess communication *during* the computation.
- Size of data that needs to be transmitted is smaller in comparison to runtime of computations (see point 1).

MPI Use Case

MPI is ideal for the following situation:

- Communication is expensive and the bottleneck of our application. It must be done as efficiently as possible.
- Available hardware is homogenous, e.g., we have a cluster of the same server components connected by an Infiniband network, so we can use highly-specialized communication libraries.
- Processes need to be organized in groups or topological structures (see the heat simulation example) to make efficient use of collective communication to achieve high performance. $\Longrightarrow$ Problems have regular patterns.
- Basically anything where we need lots of/repetitive interprocess communication *during* the computation.
- Size of data that needs to be transmitted is smaller in comparison to runtime of computations (see point 1).
- We are in a scientific environment and do not need to connect to other software such as enterprise systems.

Other Scenario

But what if we have distributed computations. . .

Other Scenario

But what if we have distributed computations with

- Communication is not the bottleneck: We spend much more time computing than communicating.

Other Scenario

But what if we have distributed computations with

- Communication is not the bottleneck: We spend much more time computing than communicating.
- Hardware is commodity PCs or heterogeneous

Other Scenario

But what if we have distributed computations with

- Communication is not the bottleneck: We spend much more time computing than communicating.
- Hardware is commodity PCs or heterogeneous
- Processes do not need to be organized in any special way, data and workload can be distributed in an arbitrary fashion.

Other Scenario

But what if we have distributed computations with

- Communication is not the bottleneck: We spend much more time computing than communicating.
- Hardware is commodity PCs or heterogeneous
- Processes do not need to be organized in any special way, data and workload can be distributed in an arbitrary fashion. $\Longrightarrow$ Problems have parallel, unstructured, or batch job character.

Other Scenario

But what if we have distributed computations with

- Communication is not the bottleneck: We spend much more time computing than communicating.
- Hardware is commodity PCs or heterogeneous
- Processes do not need to be organized in any special way, data and workload can be distributed in an arbitrary fashion. $\Longrightarrow$ Problems have parallel, unstructured, or batch job character.
- Problems require communication only during startup and end, similar to a single *scatter/reduce* step in MPI.

Other Scenario

But what if we have distributed computations with

- Communication is not the bottleneck: We spend much more time computing than communicating.
- Hardware is commodity PCs or heterogeneous
- Processes do not need to be organized in any special way, data and workload can be distributed in an arbitrary fashion. $\Longrightarrow$ Problems have parallel, unstructured, or batch job character.
- Problems require communication only during startup and end, similar to a single *scatter/reduce* step in MPI.
- Alternatively: Data is unstructured (e.g., text) and potentially huge. Communication will take up lots of time (and thus would eat up all advantages of MPI).

Other Scenario

But what if we have distributed computations with

- Communication is not the bottleneck: We spend much more time computing than communicating.
- Hardware is commodity PCs or heterogeneous
- Processes do not need to be organized in any special way, data and workload can be distributed in an arbitrary fashion. $\Longrightarrow$ Problems have parallel, unstructured, or batch job character.
- Problems require communication only during startup and end, similar to a single *scatter/reduce* step in MPI.
- Alternatively: Data is unstructured (e.g., text) and potentially huge. Communication will take up lots of time (and thus would eat up all advantages of MPI).
- Our data comes from and results need to be passed to other applications, such as enterprise systems, which may use stacks such as HTTP/Java Servlet/Web Service.

Use Cases

http://www.hadoopilluminated.com/hadoop_illuminated/Hadoop_Use_Cases.html reports many use cases

Use Cases

`http://www.hadoopilluminated.com/hadoop_illuminated/Hadoop_Use_Cases.html` reports many use cases:

- analysis of large-scale data sets (e.g., 1TB/day) for banks, health care, mobile phone carriers, travel industry, . . .

Use Cases

http://www.hadoopilluminated.com/hadoop_illuminated/Hadoop_Use_Cases.html reports many use cases:

- analysis of large-scale data sets (e.g., 1TB/day) for banks, health care, mobile phone carriers, travel industry, ...
- processing of human genome data

Use Cases

http://www.hadoopilluminated.com/hadoop_illuminated/Hadoop_Use_Cases.html reports many use cases:

- analysis of large-scale data sets (e.g., 1TB/day) for banks, health care, mobile phone carriers, travel industry, ...
- processing of human genome data
- data mining for online sales platform, travel industry, marketing campaigns

Use Cases

http:
//www.hadoopilluminated.com/hadoop_illuminated/Hadoop_Use_Cases.html
reports many use cases:

- analysis of large-scale data sets (e.g., 1TB/day) for banks, health care, mobile phone carriers, travel industry, ...
- processing of human genome data
- data mining for online sales platform, travel industry, marketing campaigns
- data analysis backend for software-as-a-service / business intelligence

Use Cases

http:
//www.hadoopilluminated.com/hadoop_illuminated/Hadoop_Use_Cases.html
reports many use cases:

- analysis of large-scale data sets (e.g., 1TB/day) for banks, health care, mobile phone carriers, travel industry, ...
- processing of human genome data
- data mining for online sales platform, travel industry, marketing campaigns
- data analysis backend for software-as-a-service / business intelligence
- large-scale parallel image processing

- There are large-scale distributed computations for which MPI is not the right answer.

- There are large-scale distributed computations for which MPI is not the right answer.
- MapReduce [1, 2] with Hadoop [1–3] fills this gap.

- There are large-scale distributed computations for which MPI is not the right answer.
- MapReduce [1, 2] with Hadoop [1–3] fills this gap.
- *MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.* [4]

MapReduce with Hadoop

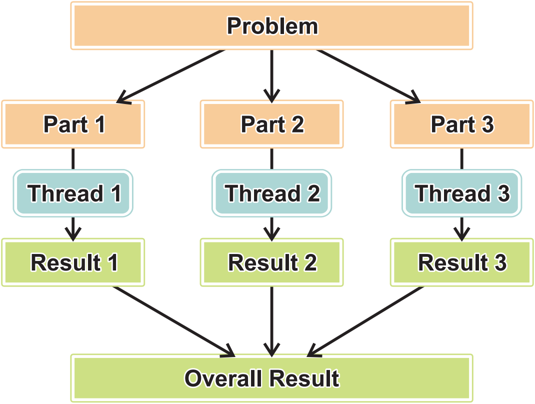
- There are large-scale distributed computations for which MPI is not the right answer.
- MapReduce [1, 2] with Hadoop [1–3] fills this gap.
- *MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.* [4]
- Conceptually similar to scatter/reduce in MPI

- Data is divided into smaller pieces, each piece corresponds to a problem part. The parts are solved separately and the separate solutions are combined to final results.

Divide & Conquer

- Data is divided into smaller pieces, each piece corresponds to a problem part. The parts are solved separately and the separate solutions are combined to final results.

- Data is divided into smaller pieces, each piece corresponds to a problem part. The parts are solved separately and the separate solutions are combined to final results.



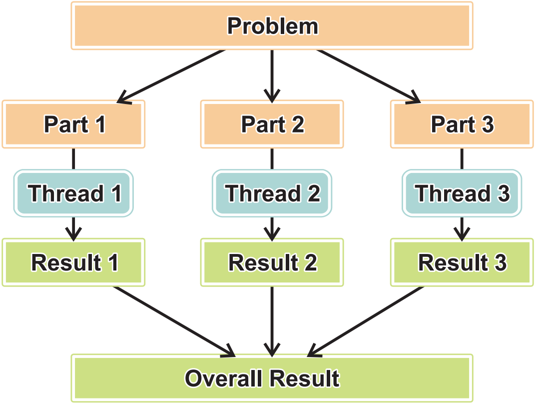
Partition = Divide

Conquer

Combine

image source: [5]

- solving of partial problems = mapping; combination of partial results to final results = reduce



Partition = Divide

Mapping

Reducing

Hadoop

- Apache Hadoop is a huge framework in Java that – amongst many other things – provides MapReduce support

Hadoop

- Apache Hadoop is a huge framework in Java that – amongst many other things – provides MapReduce support
- Scalable and fault-tolerant system for data processing and storage [6]

- Apache Hadoop is a huge framework in Java that – amongst many other things – provides MapReduce support
- Scalable and fault-tolerant system for data processing and storage [6]
- Two main components in core Hadoop [6]

Hadoop

- Apache Hadoop is a huge framework in Java that – amongst many other things – provides MapReduce support
- Scalable and fault-tolerant system for data processing and storage [6]
- Two main components in core Hadoop [6]:
  - Hadoop Distributed File System: self-healing high-bandwidth clustered storage: distributed fault-tolerant resource management and scheduling coupled with a scalable data programming abstraction.

Hadoop

- Apache Hadoop is a huge framework in Java that – amongst many other things – provides MapReduce support
- Scalable and fault-tolerant system for data processing and storage [6]
- Two main components in core Hadoop [6]:
  - Hadoop Distributed File System: self-healing high-bandwidth clustered storage: distributed fault-tolerant resource management and scheduling coupled with a scalable data programming abstraction.
  - MapReduce

Hadoop

- Apache Hadoop is a huge framework in Java that – amongst many other things – provides MapReduce support
- Scalable and fault-tolerant system for data processing and storage [6]
- Two main components in core Hadoop [6]:
  - Hadoop Distributed File System: self-healing high-bandwidth clustered storage: distributed fault-tolerant resource management and scheduling coupled with a scalable data programming abstraction.
  - MapReduce
- `http://hadoop.apache.org/`, current version 2.7.2

Hadoop

- Apache Hadoop is a huge framework in Java that – amongst many other things – provides MapReduce support
- Scalable and fault-tolerant system for data processing and storage [6]
- Two main components in core Hadoop [6]:
  - Hadoop Distributed File System: self-healing high-bandwidth clustered storage: distributed fault-tolerant resource management and scheduling coupled with a scalable data programming abstraction.
  - MapReduce
- `http://hadoop.apache.org/`, current version 2.7.2
- Due to time restrictions, we will *only* consider the MapReduce part

Word Count

- The most well-known example for Hadoop

Word Count

- The most well-known example for Hadoop
- Input: A (potentially large) text

Word Count

- The most well-known example for Hadoop
- Input: A (potentially large) text
- Output: A list of words and how often they occur in the text

Word Count

- The most well-known example for Hadoop
- Input: A (potentially large) text
- Output: A list of words and how often they occur in the text
- *Map*

Word Count

- The most well-known example for Hadoop
- Input: A (potentially large) text
- Output: A list of words and how often they occur in the text
- *Map*:
  1. divide text into pieces

Word Count

- The most well-known example for Hadoop
- Input: A (potentially large) text
- Output: A list of words and how often they occur in the text
- *Map*:
  1. divide text into pieces
  2. assign one piece to each worker

Word Count

- The most well-known example for Hadoop
- Input: A (potentially large) text
- Output: A list of words and how often they occur in the text
- *Map*:
  1. divide text into pieces
  2. assign one piece to each worker
  3. worker creates `<word, 1>` tuples for each word it finds

Word Count

- The most well-known example for Hadoop
- Input: A (potentially large) text
- Output: A list of words and how often they occur in the text
- *Map*:
  1. divide text into pieces
  2. assign one piece to each worker
  3. worker creates `<word, 1>` tuples for each word it finds
- *Reduce*

Word Count

- The most well-known example for Hadoop
- Input: A (potentially large) text
- Output: A list of words and how often they occur in the text
- *Map*:
  1. divide text into pieces
  2. assign one piece to each worker
  3. worker creates `<word, 1>` tuples for each word it finds
- *Reduce*:
  1. the `word` elements of the tuples are keys

Word Count

- The most well-known example for Hadoop
- Input: A (potentially large) text
- Output: A list of words and how often they occur in the text
- *Map*:
  1. divide text into pieces
  2. assign one piece to each worker
  3. worker creates `<word, 1>` tuples for each word it finds
- *Reduce*:
  1. the `word` elements of the tuples are keys

Word Count

- The most well-known example for Hadoop
- Input: A (potentially large) text
- Output: A list of words and how often they occur in the text
- *Map*:
  1. divide text into pieces
  2. assign one piece to each worker
  3. worker creates `<word, 1>` tuples for each word it finds
- *Reduce*:
  1. the `word` elements of the tuples are keys
  2. we add up the corresponding values (the `1`s) per word

Word Count

- The most well-known example for Hadoop
- Input: A (potentially large) text
- Output: A list of words and how often they occur in the text
- *Map*:
  1. divide text into pieces
  2. assign one piece to each worker
  3. worker creates `<word, 1>` tuples for each word it finds
- *Reduce*:
  1. the `word` elements of the tuples are keys
  2. we add up the corresponding values (the `1`s) per word
- We can also do this locally on each node, after *Map* and before *Reduce*, to reduce the amount of communication needed (this is called *combination*)

WordCount Mapper

- The input of the MapReduce process are text files

WordCount Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)

WordCount Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)
- Input of Mapper: tuples of the form `<Integer (line number), Text (the line contents)>`

WordCount Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)
- Input of Mapper: tuples of the form
  `<Integer (line number), Text (the line contents)>`
- The mapper will split each line into words (ignoring punctuation marks). For each word of the like, it will emit a tuple

WordCount Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)
- Input of Mapper: tuples of the form `<Integer (line number), Text (the line contents)>`
- The mapper will split each line into words (ignoring punctuation marks). For each word of the like, it will emit a tuple
- Output of Mapper: `<Text (the word), WriteableInteger (always value 1)>`, where the `WriteableInteger` is stands for the number of appearences

WordCount Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)
- Input of Mapper: tuples of the form `<Integer (line number), Text (the line contents)>`
- The mapper will split each line into words (ignoring punctuation marks). For each word of the like, it will emit a tuple
- Output of Mapper: `<Text (the word), WriteableInteger (always value 1)>`, where the `WriteableInteger` is stands for the number of appearences
- If a word occurs multiple times in a line, one token is emitted for each occurrence.

Listing: The mapper class.

```java
package wordCount;
public class WordCountMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

  public static final IntWritable ONE = new IntWritable(1);

  @Override
  protected void map(final LongWritable offset, final Text line,
      final Context context) throws IOException, InterruptedException {
    for (String word : line.toString()// replace punctuation and other
        .replace('.', '␣').replace(',', '␣').replace('/', '␣')// strange
        .replace(']', '␣').replace('[', '␣').replace('_', '␣')// chars
        .replace(')', '␣').replace('(', '␣').replace('#', '␣')// with
        .replace('!', '␣').replace('?', '␣').replace("-", "")// spaces
        .replace("\"", "").replace("\'", "").replaceAll("[0-9]+", "␣")//
        .replace(':', '␣').replace('\t', '␣').replace('\f', '␣')//
        .split("\\s+")) {// iterate over all space-separated words
      word = word.trim();
      if (word.length() > 0) {// emit one tuple <WORD, 1> for each WORD
        context.write(new Text(word.toLowerCase()), WordCountMapper.ONE);
      }
    }
  }
}
```

WordCount Reducer

- After this mapping step, the reducer is applied.

WordCount Reducer

- After this mapping step, the reducer is applied.
- Input:

```
<Text (the word), Iterable<WriteableInteger> (number of occurrences)>
```

WordCount Reducer

- After this mapping step, the reducer is applied.
- Input:
  `<Text (the word), Iterable<WriteableInteger> (number of occurrences)>`
- Hadoop has put all the `<WriteableInteger>` for the same `<Text (the word)>` key into a list for us!

WordCount Reducer

- After this mapping step, the reducer is applied.
- Input:
  `<Text (the word), Iterable<WriteableInteger> (number of occurrences)>`
- Hadoop has put all the `<WriteableInteger>` for the same `<Text (the word)>` key into a list for us!
- We now only need to add them up

WordCount Reducer

- After this mapping step, the reducer is applied.
- Input:
  `<Text (the word), Iterable<WriteableInteger> (number of occurrences)>`
- Hadoop has put all the `<WriteableInteger>` for the same `<Text (the word)>` key into a list for us!
- We now only need to add them up
- Output:
  `<Text (the word), WriteableInteger (count of occurrences)>`

- After this mapping step, the reducer is applied.
- Input:
  `<Text (the word), Iterable<WriteableInteger> (number of occurrences)>`
- Hadoop has put all the `<WriteableInteger>` for the same `<Text (the word)>` key into a list for us!
- We now only need to add them up
- Output:
  `<Text (the word), WriteableInteger (count of occurrences)>`
- The reducer can also be used as combinator

WordCount Reducer

- After this mapping step, the reducer is applied.
- Input:
  `<Text (the word), Iterable<WriteableInteger> (number of occurrences)>`
- Hadoop has put all the `<WriteableInteger>` for the same `<Text (the word)>` key into a list for us!
- We now only need to add them up
- Output:
  `<Text (the word), WriteableInteger (count of occurrences)>`
- The reducer can also be used as combinator: Before sending the results of the mapper to the central reducer, we can add up the tuples (`WriteableInteger`s) for the same key (word, `Text`).

WordCount Reducer

- After this mapping step, the reducer is applied.
- Input:
  `<Text (the word), Iterable<WriteableInteger> (number of occurrences)>`
- Hadoop has put all the `<WriteableInteger>` for the same `<Text (the word)>` key into a list for us!
- We now only need to add them up
- Output:
  `<Text (the word), WriteableInteger (count of occurrences)>`
- The reducer can also be used as combinator: Before sending the results of the mapper to the central reducer, we can add up the tuples (`WriteableInteger`s) for the same key (word, `Text`). This way we reduce the data volume.

Listing: The reducer class.

```
package wordCount;
public class WordCountReducer
    extends Reducer<Text, IntWritable, Text,
        IntWritable> {

  @Override
  protected void reduce(final Text key, final
      Iterable<IntWritable> values,
      final Context context) throws IOException,
        InterruptedException {
    // we receive tuples of the type <WORD,
        IntWritable> for each WORD
    int count = 0;
    for (final IntWritable current : values) { //
        we add up all the ints
      count += current.get();
    }
    context.write(key, new IntWritable(count));//
        and emit the final count
  }
}
```

WordCount Driver

- We put everything together in a "driver" class

Driver Class

Listing: The driver class.

```java
package wordCount;
public class WordCountDriver extends Configured implements Tool {

  public static void main(final String[] args) throws Exception {
    System.exit(ToolRunner.run(new Configuration(), //
        new WordCountDriver(), args));
  }

  @Override
  public int run(final String[] args) throws Exception {
    final Configuration conf;
    final Job job;

    conf = new Configuration();
    job = Job.getInstance(conf, "Word Count Map-Reduce");

    job.setJarByClass(WordCountDriver.class);

    if (args.length < 2) {
      return 1;
    }

    job.setMapperClass(WordCountMapper.class);// set mapper
    job.setReducerClass(WordCountReducer.class);// set reducer
    // a combiner performs something like a reduction step right after
    // mapping, on the mapper's computer, before sending on the data
    job.setCombinerClass(WordCountReducer.class);// set combiner

    job.setOutputKeyClass(Text.class);// set output key class
    job.setOutputValueClass(IntWritable.class);// set output value class

    job.setInputFormatClass(TextInputFormat.class);// set input format
    job.setOutputFormatClass(TextOutputFormat.class);// set output format

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    return 0;
  }
}
```

Maven Setup

- We can set up this project by using Maven

Maven Setup

- We can set up this project by using Maven
- And create an executable `jar` with `mvn clean compile package`

pom.xml

Listing: [pom.xml] – Part 1: Basic Project Information

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0␣
      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>thomasWeise</groupId>
  <artifactId>wordCount</artifactId>
  <version>0.8.0</version>
  <packaging>jar</packaging>
  <name>Hadoop Word Counting Example</name>
  <description>The famous word counting example for
      Hadoop.</description>
```

Listing: [pom.xml] – Part 2: Information about Organization

```xml
<url>http://www.it-weise.de/</url>
<organization>
  <url>http://www.it-weise.de/</url>
  <name>thomasWeise</name>
</organization>
```

pom.xml

Listing: [pom.xml] – Part 3: Information about Developer

```
<developers>
  <developer>
    <id>thomasWeise</id>
    <name>Thomas Weise</name>
    <email>tweise@ustc.edu.cn</email>
    <url>http://www.it-weise.de/</url>
    <organization>University of Science and Technology of
       China (USTC)</organization>
    <organizationUrl>http://www.ustc.edu.cn/</organizationUrl>
    <roles>
      <role>architect</role>
      <role>developer</role>
    </roles>
    <timezone>China Time Zone</timezone>
  </developer>
</developers>
```

pom.xml

Listing: [pom.xml] – Part 4: Properties for Rest of pom

```
<properties>
  <encoding>UTF-8</encoding>
  <project.build.sourceEncoding>${encoding}</project.build.sourceEncoding>
  <project.reporting.outputEncoding>${encoding}</project.reporting.outputEncoding>
  <jdk.version>1.7</jdk.version>
  <project.mainClass>wordCount.WordCountDriver</project.mainClass>
```

pom.xml

Listing: [pom.xml] – Part 5: Licensing

```
<licenses>
  <license>
    <name>GNU GENERAL PUBLIC LICENSE Version 3, 29 June
        2007</name>
    <url>http://www.gnu.org/licenses/gpl-3.0-standalone.html</url>
    <distribution>repo</distribution>
  </license>
</licenses>
```

pom.xml

Listing: [pom.xml] – Part 6: SCM, Issue Management, and Inception Year

```
<issueManagement>
  <url>https://github.com/thomasWeise/distributedComputingExamples/issues</url>
  <system>GitHub</system>
</issueManagement>

<scm>
  <connection>scm:git:git@github.com:thomasWeise/distributedComputingExamples.git</connection>
  <developerConnection>scm:git:git@github.com:thomasWeise/distributedComputingExamples.git</developerConn
  <url>git@github.com:thomasWeise/distributedComputingExamples.git</url>
</scm>

<inceptionYear>2016</inceptionYear>
```

pom.xml

Listing: [pom.xml] – Part 7: Dependencies

```
<dependencies>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-client</artifactId>
    <version>2.7.2</version>
  </dependency>
</dependencies>
```

pom.xml

Listing: [pom.xml] – Part 8: Build

```
<build>
  <finalName>wordCount</finalName>

  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version>
      <configuration>
        <source>${jdk.version}</source>
        <target>${jdk.version}</target>
        <encoding>${encoding}</encoding>
        <showWarnings>true</showWarnings>
        <showDeprecation>true</showDeprecation>
      </configuration>
    </plugin>

    <!-- This one is for building the fat jar. -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>2.3</version>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>shade</goal>
          </goals>
          <configuration>

            <minimizeJar>false</minimizeJar>
            <shadedArtifactAttached>true</shadedArtifactAttached>
            <createDependencyReducedPom>false</createDependencyReducedPom>
            <finalName>wordCount-full</finalName>

            <filters>
              <filter>
                <artifact>*:*</artifact>
                <excludes>
                  <exclude>META-INF/*.SF</exclude>
                  <exclude>META-INF/*.DSA</exclude>
                  <exclude>META-INF/*.RSA</exclude>
                </excludes>
              </filter>
            </filters>

            <transformers>
              <transformer
                implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                <mainClass>${project.mainClass}</mainClass>
              </transformer>
              <transformer
                implementation="org.apache.maven.plugins.shade.resource.ApacheLicenseResourceTransformer" />
              <transformer
                implementation="org.apache.maven.plugins.shade.resource.ApacheNoticeResourceTransformer" />
              <transformer
                implementation="org.apache.maven.plugins.shade.resource.PluginXmlResourceTransformer" />
              <transformer
                implementation="org.apache.maven.plugins.shade.resource.ServicesResourceTransformer" />
            </transformers>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Executing the Example

- Enter the `hadoop` folder and perform the following steps

Executing the Example

- Enter the `hadoop` folder and perform the following steps:

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
    1. `bin/hdfs namenode -format`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. `bin/hdfs dfs -mkdir /user/<username>` (use your user name)

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. `bin/hdfs dfs -mkdir /user/<username>` (use your user name)
  5. `bin/hdfs dfs -put Y/distributedComputingExamples/hadoop/wordCount/input input`

     where `Y` is the folder where we have the `distributedComputingExamples` repository

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. bin/hdfs dfs -mkdir /user/<username> (use your user name)
  5. `bin/hdfs dfs -put Y/distributedComputingExamples/hadoop/wordCount/input input`
     where `Y` is the folder where we have the `distributedComputingExamples` repository
  6.

```
bin/hadoop jar Y/distributedComputingExamples/hadoop/wordCount/target/wordCount-full.
input output
```

Executing the Example

- Enter the `hadoop` folder and perform the following steps:

  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. `bin/hdfs dfs -mkdir /user/<username>` (use your user name)
  5. `bin/hdfs dfs -put Y/distributedComputingExamples/hadoop/wordCount/input input` where `Y` is the folder where we have the `distributedComputingExamples` repository
  6.
     ```
     bin/hadoop jar Y/distributedComputingExamples/hadoop/wordCount/target/wordCount-full.
     input output
     ```
  7. `bin/hdfs dfs -ls output`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. `bin/hdfs dfs -mkdir /user/<username>` (use your user name)
  5. `bin/hdfs dfs -put Y/distributedComputingExamples/hadoop/wordCount/input input`
     where `Y` is the folder where we have the `distributedComputingExamples` repository
  6.
     ```
     bin/hadoop jar Y/distributedComputingExamples/hadoop/wordCount/target/wordCount-full.
     input output
     ```
  7. `bin/hdfs dfs -ls output`
  8. `bin/hdfs dfs -cat output/part-r-00000 | less`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. bin/hdfs dfs -mkdir /user/<username> (use your user name)
  5. `bin/hdfs dfs -put Y/distributedComputingExamples/hadoop/wordCount/input input`
     where `Y` is the folder where we have the `distributedComputingExamples` repository
  6.
```
bin/hadoop jar Y/distributedComputingExamples/hadoop/wordCount/target/wordCount-full.
input output
```
  7. `bin/hdfs dfs -ls output`
  8. `bin/hdfs dfs -cat output/part-r-00000 | less`
  9. `bin/hdfs dfs -copyToLocal output/part-r-00000`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. `bin/hdfs dfs -mkdir /user/<username>` (use your user name)
  5. `bin/hdfs dfs -put Y/distributedComputingExamples/hadoop/wordCount/input input`
     where `Y` is the folder where we have the `distributedComputingExamples` repository
  6. `bin/hadoop jar Y/distributedComputingExamples/hadoop/wordCount/target/wordCount-full.`
     `input output`
  7. `bin/hdfs dfs -ls output`
  8. `bin/hdfs dfs -cat output/part-r-00000 | less`
  9. `bin/hdfs dfs -copyToLocal output/part-r-00000`
  10. `sbin/stop-dfs.sh`

Web Finder Count

- Try to find the interconnection between pages

Web Finder Count

- Try to find the interconnection between pages
- Input to Mapper: List of URLs (as `Text`s)

Web Finder Count

- Try to find the interconnection between pages
- Input to Mapper: List of URLs (as `Text`s)
- Output of Mapper: Tuples of URL to resources and URLs from input referencing them

Web Finder Count

- Try to find the interconnection between pages
- Input to Mapper: List of URLs (as `Text`s)
- Output of Mapper: Tuples of URL to resources and URLs from input referencing them
- Input of Reducer: Tuples of URL to a resources and *list* of URLs (from Mapper input) referencing them

Web Finder Count

- Try to find the interconnection between pages
- Input to Mapper: List of URLs (as `Text` s)
- Output of Mapper: Tuples of URL to resources and URLs from input referencing them
- Input of Reducer: Tuples of URL to a resources and *list* of URLs (from Mapper input) referencing them
- Output of Reducer: List of URLs to *shared* resources and URLs (from Mapper input) referencing them

Web Finder Count

- Try to find the interconnection between pages
- Input to Mapper: List of URLs (as `Text` s)
- Output of Mapper: Tuples of URL to resources and URLs from input referencing them
- Input of Reducer: Tuples of URL to a resources and *list* of URLs (from Mapper input) referencing them
- Output of Reducer: List of URLs to *shared* resources and URLs (from Mapper input) referencing them
- Having this, we can find out which resources are fundamental in the web, which are shared between different pages, and how the most important pages in China are interconnected

Web Finder Mapper

- The input of the MapReduce process are text files

Web Finder Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)

Web Finder Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)
- Input of Mapper: tuples of the form `<Integer (line number), Text (the line contents)>`

Web Finder Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)
- Input of Mapper: tuples of the form `<Integer (line number), Text (the line contents)>`
- Each line is a `URL` (let's call it `A`)

Web Finder Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)
- Input of Mapper: tuples of the form `<Integer (line number), Text (the line contents)>`
- Each line is a `URL` (let's call it `A`)
- We download the referenced web page

Web Finder Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)
- Input of Mapper: tuples of the form `<Integer (line number), Text (the line contents)>`
- Each line is a `URL` (let's call it `A`)
- We download the referenced web page
- We detect all references to other pages, CSS, and javscript in the page. Each reference corresponds to a URL, let's call it `B`

Web Finder Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)
- Input of Mapper: tuples of the form `<Integer (line number), Text (the line contents)>`
- Each line is a `URL` (let's call it `A`)
- We download the referenced web page
- We detect all references to other pages, CSS, and javscript in the page. Each reference corresponds to a URL, let's call it `B`
- We emit all tuples `<B as Text, A as Text>`

Web Finder Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)
- Input of Mapper: tuples of the form `<Integer (line number), Text (the line contents)>`
- Each line is a `URL` (let's call it `A`)
- We download the referenced web page
- We detect all references to other pages, CSS, and javscript in the page. Each reference corresponds to a URL, let's call it `B`
- We emit all tuples `<B as Text, A as Text>`
- The process is applied *recursively* (up to a maximum depth)

Web Finder Mapper

- The input of the MapReduce process are text files
- The `TextInputFormat` splits the text files into single lines (of type `Text`, the Hadoop version of Strings)
- Input of Mapper: tuples of the form
  `<Integer (line number), Text (the line contents)>`
- Each line is a `URL` (let's call it `A`)
- We download the referenced web page
- We detect all references to other pages, CSS, and javscript in the page. Each reference corresponds to a URL, let's call it `B`
- We emit all tuples `<B as Text, A as Text>`
- The process is applied *recursively* (up to a maximum depth)
- So we have tuples of resources and the URLs referencing them

Listing: The mapper class.

```java
package webFinder;
/**
 * This is the Mapper component of the Web Finder example. Its input are
 * text lines, where each line stands for a website URL. It finds all
 * resources that are loaded by a given website URL and emits tuples of
 * kind {@code <resource URL, website URL>}.
 */
public class WebFinderMapper
    extends Mapper<LongWritable, Text, Text, Text> {

  /** the logger we use */
  private static Logger LOGGER = Logger.getLogger(WebFinderMapper.class);

  /**
   * Map tuples of type {@code <line number, website url text>} to tuples
   * of kind {@code <resource url text, website url text>}.
   */
  @Override
  protected void map(final LongWritable offset, final Text line,
      final Context context) throws IOException, InterruptedException {
    final URL baseUrl;
    final URI baseUri;
    final int maxDepth;
    final Text baseUrlText;
    final HashSet<URL> done;
    String str;

    str = WebFinderMapper.__prepare(line.toString(), true);
    if (str == null) {// prepare base url
      return;
    }
    // set maximum depth of spider
    maxDepth = context.getConfiguration().getInt("maxDepth", 1);

    baseUri = URI.create(str).normalize();
    baseUrl = baseUri.toURL();
    done = new HashSet<>();// URLs that have been processed
    done.add(baseUrl);
```

- The input of the Reducer are tuples of resources and *lists* of URLs referencing them

Web Finder Reducer

- The input of the Reducer are tuples of resources and *lists* of URLs referencing them
- For each resource URL, we compute the set of unique URLs referencing them

Web Finder Reducer

- The input of the Reducer are tuples of resources and *lists* of URLs referencing them
- For each resource URL, we compute the set of unique URLs referencing them
- If the set contains more then one resource, we have a resource shared among multiple of the originally provided URLs

Web Finder Reducer

- The input of the Reducer are tuples of resources and *lists* of URLs referencing them
- For each resource URL, we compute the set of unique URLs referencing them
- If the set contains more then one resource, we have a resource shared among multiple of the originally provided URLs
- We will only output such elements, as tuple `<URL, list of URLs referencing it>`

Listing: The reducer class.

```java
package webFinder;
/**
 * This is the reducer component of the web finder example. For each key (
 * {@code resource URL}) of the tuples produced by the mapper, it receives
 * the list of all values ({@code website URLs}). If such a list contains
 * more than one unique element, this means that the resource is shared by
 * multiple websites. This reducer emits tuples of the form
 * {@code <resource URL, list of website urls>}.
 */
public class WebFinderReducer
    extends Reducer<Text, Text, Text, List<Text>> {

  /**
   * The actual reduction step: From the tuples of form
   * {@code <resource URL, iterable of referencing website URLs>}, select
   * all resources referenced by more than one unique website. For these,
   * output tuples of the form {@code <resource URL, list of website URLs>}
   * .
   */
  @Override
  protected void reduce(final Text key, final Iterable<Text> values,
      final Context context) throws IOException, InterruptedException {
    final HashSet<URL> set;
    final int size;
    final ArrayList list;
    String string;
    URL add;
    int index;

    set = new HashSet<>();
    looper: for (final Text url : values) {
      string = url.toString();// convert value to a URL
      try {
        add = new URI(string).normalize().toURL();
      } catch (@SuppressWarnings("unused") final Throwable error) {
        try {
          add = new URL(string).toURI().normalize().toURL();
        } catch (@SuppressWarnings("unused") final Throwable error2) {
          try {
            add = new URL(string);
          } catch (@SuppressWarnings("unused") final Throwable error3) {
            continue looper;
          }
        }
      }
      set.add(add); // store value in set of URLs pointing to this resource
```

WebFinder Driver

- We put everything together in a “driver” class

The Driver Class

Listing: The driver class.

```
package webFinder;
/**
 * The driver of the web finder sets up the distributed computation by
 * defining what the mapper and reducer classes, amongst other things.
 */
public class WebFinderDriver extends Configured implements Tool {

  public static void main(final String[] args) throws Exception {
    try {
      final int res = ToolRunner.run(new Configuration(),
          new WebFinderDriver(), args);
      System.exit(res);
    } catch (final Exception e) {
      e.printStackTrace();
      System.exit(255);
    }
  }

  /** Setting up the computation. */
  @Override
  public int run(final String[] args) throws Exception {
    final Configuration conf;
    final Job job;

    conf = new Configuration();
    job = Job.getInstance(conf, "WebFinder MapReduce");

    job.setJarByClass(WebFinderDriver.class);// use current jar

    if (args.length < 2) {
      return 1;
    }
    if (args.length > 2) {// set max depth and pass parameter to mapper
      conf.setInt("maxDepth", Integer.parseInt(args[2]));
    }

    job.setMapperClass(WebFinderMapper.class);// set mapper
    job.setMapOutputKeyClass(Text.class);// set mapper output key type
    job.setMapOutputValueClass(Text.class); // set mapper output value type

    job.setReducerClass(WebFinderReducer.class);// set reducer
    job.setOutputKeyClass(Text.class);// set reducer output key type
    job.setOutputValueClass(List.class);// set reducer output value

    job.setInputFormatClass(TextInputFormat.class);// set input format
    job.setOutputFormatClass(TextOutputFormat.class);// set output format

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

Executing the Example

- Enter the `hadoop` folder and perform the following steps

Executing the Example

- Enter the `hadoop` folder and perform the following steps:

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. `bin/hdfs dfs -mkdir /user/<username>` (use your user name)

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. `bin/hdfs dfs -mkdir /user/<username>` (use your user name)
  5. `bin/hdfs dfs -put Y/distributedComputingExamples/hadoop/webFinder/input input` where `Y` is the folder where we have the `distributedComputingExamples` repository

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. `bin/hdfs dfs -mkdir /user/<username>` (use your user name)
  5. `bin/hdfs dfs -put Y/distributedComputingExamples/hadoop/webFinder/input input` where `Y` is the folder where we have the `distributedComputingExamples` repository
  6. `bin/hadoop jar Y/distributedComputingExamples/hadoop/webFinder/target/webFinder-full.jar output`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. `bin/hdfs dfs -mkdir /user/<username>` (use your user name)
  5. `bin/hdfs dfs -put Y/distributedComputingExamples/hadoop/webFinder/input input` where `Y` is the folder where we have the `distributedComputingExamples` repository
  6. `bin/hadoop jar Y/distributedComputingExamples/hadoop/webFinder/target/webFinder-full.jar input output`
  7. `bin/hdfs dfs -ls output`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. `bin/hdfs dfs -mkdir /user/<username>` (use your user name)
  5. `bin/hdfs dfs -put Y/distributedComputingExamples/hadoop/webFinder/input input` where `Y` is the folder where we have the `distributedComputingExamples` repository
  6. `bin/hadoop jar Y/distributedComputingExamples/hadoop/webFinder/target/webFinder-full.jar output`
  7. `bin/hdfs dfs -ls output`
  8. `bin/hdfs dfs -cat output/part-r-00000 | less`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. `bin/hdfs dfs -mkdir /user/<username>` (use your user name)
  5. `bin/hdfs dfs -put Y/distributedComputingExamples/hadoop/webFinder/input input` where `Y` is the folder where we have the `distributedComputingExamples` repository
  6. `bin/hadoop jar Y/distributedComputingExamples/hadoop/webFinder/target/webFinder-full.jar output`
  7. `bin/hdfs dfs -ls output`
  8. `bin/hdfs dfs -cat output/part-r-00000 | less`
  9. `bin/hdfs dfs -copyToLocal output/part-r-00000`

Executing the Example

- Enter the `hadoop` folder and perform the following steps:
  1. `bin/hdfs namenode -format`
  2. `sbin/start-dfs.sh`
  3. `bin/hdfs dfs -mkdir /user`
  4. `bin/hdfs dfs -mkdir /user/<username>` (use your user name)
  5. `bin/hdfs dfs -put Y/distributedComputingExamples/hadoop/webFinder/input input` where `Y` is the folder where we have the `distributedComputingExamples` repository
  6. `bin/hadoop jar Y/distributedComputingExamples/hadoop/webFinder/target/webFinder-full.jar output`
  7. `bin/hdfs dfs -ls output`
  8. `bin/hdfs dfs -cat output/part-r-00000 | less`
  9. `bin/hdfs dfs -copyToLocal output/part-r-00000`
  10. `sbin/stop-dfs.sh`

Listing: Output

```
http://c.youku.com/aboutcn/youtu  [http://www.tudou.com, http://www.youku.com]
http://c.youku.com/abouteg/youku  [http://www.tudou.com, http://www.youku.com]
http://c.youku.com/abouteg/youtu  [http://www.tudou.com, http://www.youku.com]
http://cbjs.baidu.com/js/m.js [http://www.baidu.com, http://www.qq.com]
http://css.tudouui.com/skin/__g/img/sprite.gif  [http://www.tudou.com, http://www.youku.com]
http://events.youku.com/global/scripts/jquery-1.8.3.js  [http://www.tudou.com, http://www.youku.com]
http://events.youku.com/global/scripts/youku.js [http://www.tudou.com, http://www.youku.com]
http://images.china.cn/images1/ch/appxz/2.jpg [http://www.qq.com, http://www.youku.com]
http://images.china.cn/images1/ch/appxz/3.jpg [http://www.qq.com, http://www.youku.com]
http://js.tudouui.com/v3/dist/js/lib_6.js [http://www.tudou.com, http://www.youku.com]
http://mail.qq.com  [http://www.baidu.com, http://www.qq.com]
http://minisite.youku.com/mini_common/urchin.js [http://www.tudou.com, http://www.youku.com]
http://player.youku.com/jsapi [http://www.tudou.com, http://www.youku.com]
http://qzone.qq.com [http://www.baidu.com, http://www.qq.com]
http://res.mfs.ykimg.com/051000004D92DF6197927339BA04E210.js  [http://www.tudou.com, http://www.youku.com]
http://static.youku.com/user/img/avatar/80/5.jpg  [http://www.tudou.com, http://www.youku.com]
http://static.youku.com/user/img/avatar/80/9.jpg  [http://www.tudou.com, http://www.youku.com]
http://weibo.com  [http://www.baidu.com, http://www.qq.com]
http://www.12377.cn [http://www.baidu.com, http://www.qq.com, http://www.youku.com]
http://www.12377.cn/node_548446.htm [http://www.qq.com, http://www.youku.com]
http://www.bjjubao.org  [http://www.baidu.com, http://www.youku.com]
http://www.china.com.cn/player/video.js [http://www.qq.com, http://www.youku.com]
http://www.ellechina.com  [http://www.qq.com, http://www.youku.com]
http://www.hao123.com [http://www.baidu.com, http://www.qq.com]
http://www.hd315.gov.cn/beian/view.asp?bianhao=010202006082400023 [http://www.tudou.com,
   http://www.youku.com]
http://www.miibeian.gov.cn  [http://www.qq.com, http://www.tudou.com, http://www.youku.com]
http://www.miibeian.gov.cn/publish/query/indexFirst.action  [http://www.tudou.com, http://www.youku.com]
http://www.pclady.com.cn  [http://www.baidu.com, http://www.qq.com]
http://www.qq.com [http://www.baidu.com, http://www.qq.com]
http://www.shjbzx.cn  [http://www.qq.com, http://www.tudou.com]
http://www.tudou.com  [http://www.tudou.com, http://www.youku.com]
http://www.tudou.com/about/cn [http://www.tudou.com, http://www.youku.com]
http://www.tudou.com/about/en [http://www.tudou.com, http://www.youku.com]
http://www.youku.com  [http://www.baidu.com, http://www.tudou.com, http://www.youku.com]
http://www.youku.com/show_page/id_z8dc3fdeedcb911e3a705.html  [http://www.tudou.com, http://www.youku.com]
http://y.qq.com [http://www.baidu.com, http://www.qq.com]
https://www.alipay.com  [http://www.baidu.com, http://www.youku.com]
```

Summary

- MapReduce via Hadoop can cover a use case of distributed computing that neither MPI nor Java Servlets can
- Easy to use with Java and Maven
- Apache Hadoop has many more features, which we cannot cover here

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://www.it-weise.de

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui, China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog

Bibliography

Bibliography I

1. Thilina Gunarathne. *Hadoop MapReduce v2 Cookbook*. Birmingham, UK: Packt Publishing Limited – ebooks Account, 2nd revised edition, January 2015. ISBN 978-1783285471.
2. Donald Miner and Adam Shook. *MapReduce Design Patterns – Building Effective Algorithms and Analytics for Hadoop and Other Systems*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2012. ISBN 978-1-4493-2717-0. URL `http://filepi.com/i/VhKExiV`.
3. Tom White. *Hadoop: The Definitive Guide – Storage and Analysis at Internet Scale*. Sebastopol, CA, USA: O'Reilly Media, Inc., 4th edition, March 2015. ISBN 978-1-4919-0163-2.
4. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. Technical report, Google, Inc., 2004. URL `http://static.googleusercontent.com/media/research.google.com/es/us/archive/mapreduce-osdi04.pdf`. Appeared in OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004 and Communications of the ACM – 50th anniversary issue: 1958-2008 CACM Homepage archive. 51(1):107–113, January 2008, ACM New York, NY, USA.
5. Jimmy Lin. *Cloud Computing Lecture – #1 What is Cloud Computing? (and an intro to parallel/distributed processing)*. College Park, MD, USA: University of Maryland, The iSchool, September 3, 2008. URL `http://www.umiacs.umd.edu/~jimmylin/cloud-2008-Fall/Session1.ppt`.
6. Amr Awadallah. Stanford ee380 computer systems colloquium – introducing apache hadoop: The modern data operating system, November 16, 2011. URL `http://web.stanford.edu/class/ee380/Abstracts/111116-slides.pdf`.