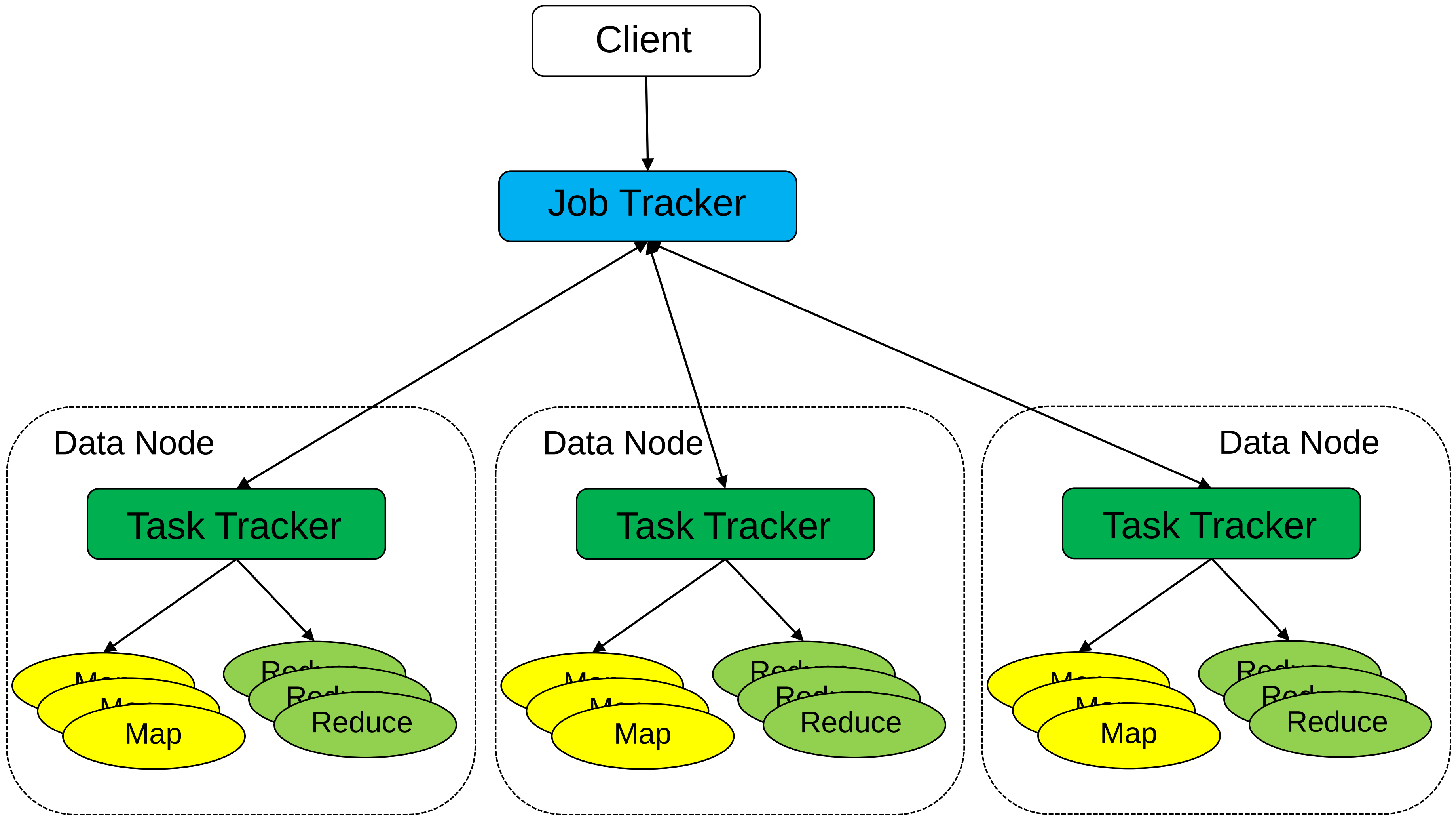
Introduction to Big Data Science

05th Period

Map-Reduce Programming Exercise
(Word Count, TF-IDF Calculation)

Contents

Map-Reduce Architecture on Hadoop



Map-Reduce Architecture on Hadoop

- Client: Map-Reduce API provided by Hadoop and Map-Reduce Program executed by a user.
- Job Tracker
  - A Map-Reducer program is managed as a work unit, “Job”.
  - Manage scheduling and monitoring of entire jobs on Hadoop cluster.
  - Running on a Name Node server usually, but necessarily.
- Description of Job Tracker
  - When a user requests a new job, it calculates how many Maps and Reduces will be executed for the job.

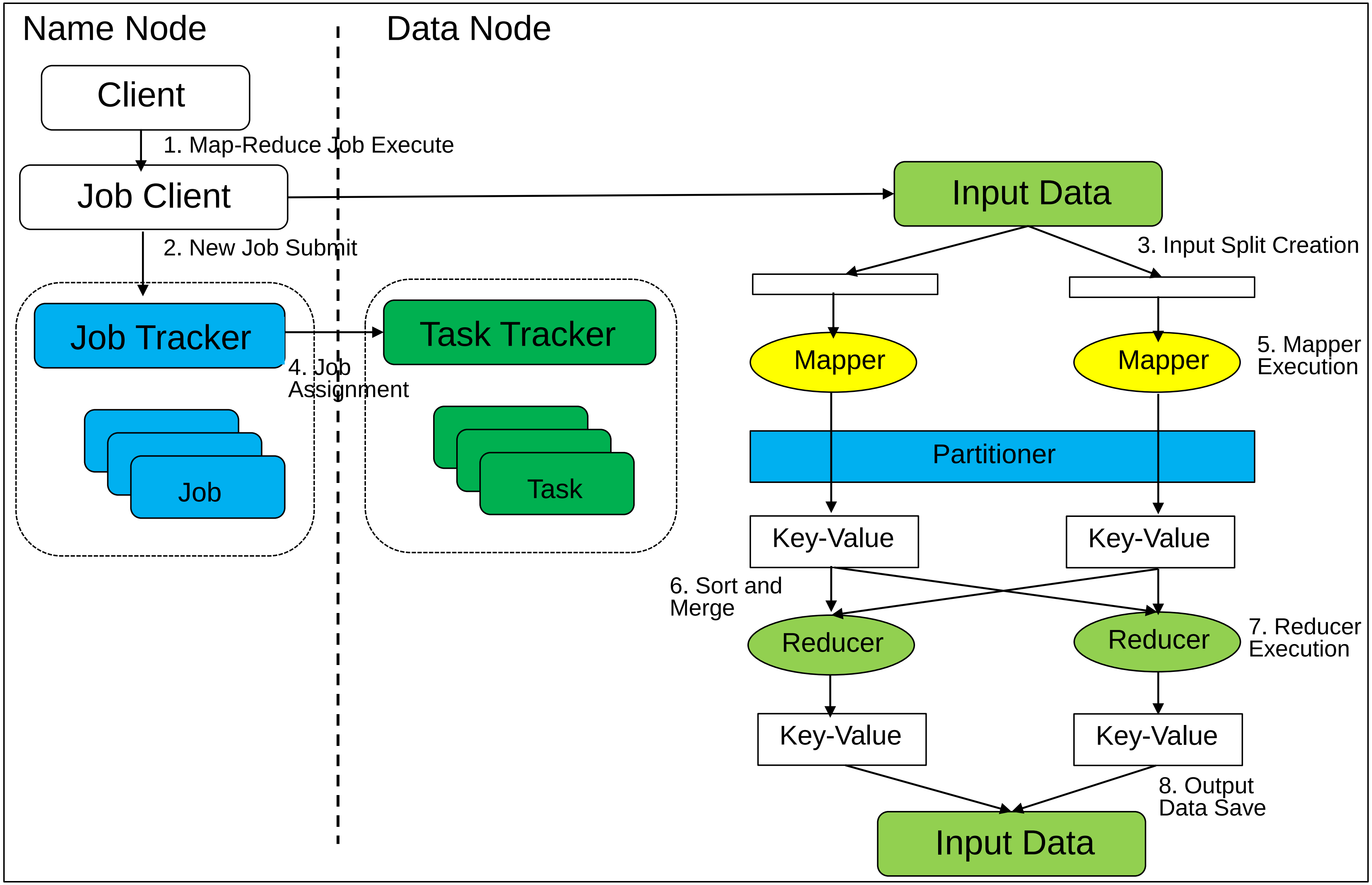
Map-Reduce Architecture on Hadoop

- Description of Job Tracker
  - A Map-Reduce program that executed by a client through Hadoop is managed by the unit of “Job”.
  - The Job Tracker monitors scheduling of all jobs registered on Hadoop cluster. But we don’t have to execute the job tracker on name node server.
  - Decide Task Trackers where the Maps-Reduces will be executed and assign the job to the Task Trackers decided. -> Then the task trackers execute the Map-Reduce programs.
  - The job tracker communicates with task trackers with heart-beats for checking status of task trackers and job execution information.

Map-Reduce Architecture on Hadoop

- Task Tracker
  - A demon program that executes Map-Reduce program of a user on a Data Node.
  - Creates Map-Tasks and Reduce-Tasks that the Job Tracker requested.
  - Execute the tasks (Map and Reduces) by running new JVM.
- Workflow of Map-Reduce
  - A user executes a job by invoking *waitForCompletion* method.
  - A job client object is created in the job interface, and the object access to the Job Tracker to execute the job.

Map-Reduce Operation on Hadoop



Map-Reduce Architecture on Hadoop

- Workflow of Map-Reduce
  - Then, the Job Tracker returns a new job ID to the Job Client. And the Job Client checks the output path defined by the user.
  - The Job Client calculates input split (Size of input file to be processed in a Map procedure) about the input data of the job. After the calculation of the input split, it saves input split information, setting file, and Map-Reduce JAR file to HDFS, and notifies completion of preparation for starting the job.
  - The Job Tracker registers the job on the Queue, Job scheduler initializes the job from the Queue. And then, the Job Scheduler creates Map tasks according to the input split information and assign IDs.

Map-Reduce Architecture on Hadoop

- Workflow of Map-Reduce
  - The Task Tracker gives its status to the Job Tracker periodically calling heartbeat method. In heartbeat information, there are resource information about CPU, memory, disk, number of tasks, and available maximum tasks, new tasks execution possibilities.
  - The Task Tracker executes the Map task assigned by the Job Tracker. The Map task executes the logic in the Map method, and save output data to memory buffer. At this time, a patitioner decides a Reduce task to which the output data should be transferred, and assign a suitable partition for it. The data in memory will be sorted by the keys, and saved to local hard disk. When the save is to be completed, the files will be sorted and merged into one file.

Map-Reduce Architecture on Hadoop

- Workflow of Map-Reduce
  - A reduce task is the Reducer class written by a user. The task can be executed when all output data are to be prepared. When the Map task completes output, it notifies its completion to the Task Tracker that ran itself. When the Task Tracker receives the message, it notifies status of the corresponding Map task and output data path of the Map task to the Job Tracker.
  - The Reduce task copies output data of all Map tasks, and merges output data of the Map tasks. When the merge is to be completed, it executes analysis logic to call Reduce method.
  - The Reduce task saves the output data to HDFS as name of “part-nnnnn”. Where “nnnnn” means partition ID.

Component of Map-Reduce Programming

- Data Types
  - The Map-Reduce is the optimized object for network communication, and provides “WritableComparable” interface.
  - All data types for ***Key*** and ***Value*** in Map-Reduce program should implement the “WritableComparable” interface.
  - The “WritableComparable” interface inherits the “Writable” and “Comparable” interfaces.
  - The “write” method serializes data value, and the “readFields” method unserializes to read the serialized data.

```
public interface Writable {
  void write (DataOutput out) throws IOException;
  void readFields (DataInput in) throws IOException;
}
```

Component of Map-Reduce Programming

- Data Types
  - The “Wrapper” class to implement the “WritableComparable” interface for each data type: BoolleanWritable, ByteWritable, DoubleWritable, FloatWritable, IntWritable, LongWritable, TextWritable, NullWritable.
- InputFormat
  - The “InputFormat” abstract class let us use it as input parameter of the “map” method.

```
public abstract class InputFormat(K, V> {
  public abstract List<InputSplit> getSplits(JobContext context) throws
IOException, InterruptedException;
  public abstract RecordReader<K, V> createRecordReader(InputSplit split,
TaskAttemptContext context) throws IOException, InterruptedException;
}
```

Component of Map-Reduce Programming

- InputFormat
  - "getSplits": the "map" can use input split.
  - "createRecordReader": creates "RecordReader" object so that the "map" method uses input split in the form of key and list.
  - The "map" method executes analysis logic to read key and value in the RecordReader object.
  - Kinds of the InputFormat: TextInputFormat, KeyValueTextInputFormat, NLineInputFormat, DelegatingInputFormat, CombineFileInputFormat, SequenceFileInputFormat, SequenceFileAsBinaryInputFormat, SequenceFileAsTextInputFormat

Component of Map-Reduce Programming

- Mapper Class
  - Carry out function of *map* method of Map-Reduce programming.
  - It receives input data consisting key and value, then process and classify the data to create new data list.
  - Class definition: using <Input Key Type, Input Value Type, Output Key Type, Output Value Type>
  - Context object: Getting information about job, and read input split as unit of record.
  - The *RecordReader* object is passed to the *map* method to read data in the form of key and value.
  - A Map programmer overwrites the *map* method.
  - Next, when the *run* method is to be executed, the map method will be executed for all keys in the *Context* object.

Component of Map-Reduce Programming

- Mapper Class

```
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
  public class Context extends MapContext<KEYIN,VALUEIN,KEYOUT,VALUEOUT> {
    public Context(Configuration conf, TaskAttemptID taskid,
                   RecordReader<KEYIN,VALUEIN> reader,
                   RecordWriter<KEYOUT,VALUEOUT> writer,
                   OutputCommitter committer,   StatusReporter reporter,
                   InputSplit split) throws IOException, InterruptedException {
      super(conf, taskid, reader, writer, committer, reporter, split);
    }
  }

@SuppressWarnings("unchecked")
  protected void map(KEYIN key, VALUEIN value,
                     Context context) throws IOException, InterruptedException {
    context.write((KEYOUT) key, (VALUEOUT) value);
  }

Public void run(Context context) throws IOException, InterruptedException {
  setup (context);
  while (context.nextKeyValue()) {
   map(context.getCurrentKey(), context.getCurrentValue(), context);
 }
}
```

Component of Map-Reduce Programming

- Partitioner
  - To decide to which Reduce task the data of Map task will be passed.
  - Calculating a partition as key.hashCode() & Integer.MAX_VALUE) % numReduceTasks
  - According to the result, the partition will be created at the node that Map task has been executed, and output data of the Map task is saved. When the Map task is completed, the data at the partition will be transmitted to corresponding Reduce task through network.
  - The getPartition method can be overwritten for other partitioning strategy.

```
public class HashPartitioner<K2, V2> implements Partitioner<K2, V2> {
  public int getPartition(K2 key, V2 value, int numReduceTasks) {
    return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
  }
}
```

Component of Map-Reduce Programming

- Reducer
  - The *Reducer* class receives the output data of the Map task as input data to execute aggregation operation.
  - Class definition: using <Input Key Type, Input Value Type, Output Key Type, Output Value Type>
  - Like *Mapper* class, the Context object: inherits the *ReduceContext*. Consult with job for information, getting the information as *RawKeyValueIterator* for checking input value list.
  - *RecordWriter* as parameter so that output result of map method can be as unit of record.
  - A *Reducer* programmer overwrites the *reduce* method.

Component of Map-Reduce Programming

◆ Reducer Class

```
public class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT> {
  public class  extends ReduceContext<KEYIN,VALUEIN,KEYOUT,VALUEOUT> {
    public Context(Configuration conf, TaskAttemptID  taskid,
       RawKeyValueIterator input, Counter inputKeyCounter,  Counter inputValueCounter,
       RecordWriter<KEYOUT,VALUEOUT> output, OutputCommitter committer, StatusReporter reporter,
       RawComparator<KEYIN> comparator,  Class<KEYIN> keyClass, Class<VALUEIN> valueClass
     ) throws IOException, InterruptedException {
        super(conf, taskid, input, inputKeyCounter, inputValueCounter,
            output, committer, reporter,   comparator, keyClass, valueClass);
      }
  }

protected void reduce(KEYIN key, Iterable<VALUEIN> values, Context context ) throws IOException,
InterruptedException {
    for(VALUEIN value: values) {  context.write((KEYOUT) key, (VALUEOUT) value);     }
}

public void run(Context context) throws IOException, InterruptedException {
    setup(context);
    while (context.nextKey()) { reduce(context.getCurrentKey(), context.getValues(), context);    }
    cleanup(context);
 }
} // end of class Reducer
```

Component of Map-Reduce Programming

- Combiner Class
  - Shuffle: Transportation process between Map task and Reduce task.
  - It need to reduce the amount of data that will be transferred through network to improve performance of entire jobs.
  - The Combiner class inputs the output data of Mapper and creates reduced data on the local node to send through network.
  - The Reducer should produce the same output in the both case: with the Combiner and without the class.

Component of Map-Reduce Programming

- OutputFormat
  - Output data format of Map-Reduce is created by the format of setOutputFormatClass method in Job interface.
  - Output data format is made by inheriting the abstract class "OutputFormat".
  - Kinds of the OutputFormat: TextOutputFormat, SequenceFileOutputFormat, SequenceFileAsBinaryOutputFormat, FilterOutputFormat, LazyOutputFormat, NullOutputFormat.
  - The classes inherit the FileOutputFormat class which inherits the OutputFormat class.

HADOOP MapReduce Program : WordCount.java

```
package net.kzk9;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
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 {
  // Mapperの実装
  public static class Map extends
    Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    @Override
      protected 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);
      }
    }
  }
```

```
  // Reducerの実装
  public static class Reduce extends
    Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable value = new IntWritable(0);

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context
context)
      throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable value : values)
        sum += value.get();
      value.set(sum);
      context.write(key, value);
    }
  }

  public static void main(String[] args) throws Exception {
    // 設定情報の読み込み
    Configuration conf = new Configuration();
    // 引数のパース
    GenericOptionsParser parser = new GenericOptionsParser(conf,
args);
    args = parser.getRemainingArgs();

    // ジョブの作成
    Job job = new Job(conf, "wordcount");
    job.setJarByClass(WordCount.class);
    // Mapper/Reducerに使用するクラスを指定
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    // ジョブ中の各種型を設定
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
```

HADOOP MapReduce Program : WordCount.java

```
// 入力 /出力パスを設定
   FileInputFormat.setInputPaths(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));
   // ジョブを JobTrackerにサブミット
   boolean success = job.waitForCompletion(true);
   System.out.println(success);
 }
}
```

```
TestWordCount.java
package net.kzk9;
import net.kzk9.WordCount;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import junit.framework.TestCase;
import org.junit.Before;
import org.junit.Test;

public class WordCountTest extends TestCase {
 // テストする対象の Mapper
 private WordCount.Map mapper;

 // Mapper実行用のドライバクラス
 private MapDriver driver;

 @Before
 public void setUp() {
   // テスト対象の Mapperの作成
   mapper = new WordCount.Map();
   // テスト実行用 MapDriverの作成
   driver = new MapDriver(mapper);
 }
```

```
  @Test
  public void testWordCountMapper() {
   // 入力を設定
   driver.withInput(new LongWritable(0), new Text("this is a pen"))
     // 期待される出力を設定
     .withOutput(new Text("this"), new IntWritable(1))
     .withOutput(new Text("is"), new IntWritable(1))
     .withOutput(new Text("a"), new IntWritable(1))
     .withOutput(new Text("pen"), new IntWritable(1))
     // テストを実行
     .runTest();
  }
}
```

```
#!/bin/bash
export HADOOP_HOME=/usr/lib/hadoop-0.20/
export DIR=wordcount_classes

# CLASSPATHの設定
CLASSPATH=$HADOOP_HOME/hadoop-core.jar
for f in $HADOOP_HOME/lib/*.jar; do
CLASSPATH=${CLASSPATH}:$f;
done

# ディレクトリの初期化
rm -fR $DIR
mkdir $DIR

# コンパイル
javac -classpath $CLASSPATH -d $DIR WordCount.java

# jarファイルの作成
jar -cvf wordcount.jar -C $DIR .
```

Term Frequency

- Consider the number of occurrences of a term in a document
  - Bag of words model
  - Document is a vector in N a column below
- Let’s consider the following document set.

D1	Today weather is sunny and cloudy. Rainy and cloudy tomorrow
D2	The soccer game is interesting. I like basketball game.
D3	Yesterday weather was cloudy and sunny. I like sunny day.
D4	The baseball game is not interesting. I win the tennis game.

Calculating Term Frequency (TF)

D1	Today weather is sunny and cloudy. Rainy and cloudy tomorrow
D2	The soccer game is interesting. I like basketball game.
D3	Yesterday weather was cloudy and sunny. I like sunny day.
D4	The baseball game is not interesting. I win the tennis game.

Term Frequency

Words / **Doc #**	Today	weather	is	sunny	and	Cloudy	rainy	tomorrow	The	soccer	game	interesting	I	like	basketball	Yesterday	day	baseball	not	win	tennis
D1	1	1	1	1	2	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
D2	0	0	1	0	0	0	0	0	1	1	2	1	1	1	1	0	0	0	0	0	0
D3	0	1	1	2	1	1	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0
D4	0	0	1	0	0	0	0	0	1	0	2	1	1	0	0	0	0	1	1	1	1

Problem of Term Frequency

- Which of these tells you more about a medical document?
  - 10 occurrences of hernia(脱腸)?
  - 10 occurrences of the ?
- Why is it?
  - Is the term a common word that exists in every document or a meaningful word that give feature to the document?
- How can we get the information?
  - If a term is found in more documents, it will have less meaning for feature of the document.
  - Document Frequency

Document Frequency (DF)

D1	Today weather is sunny and cloudy. Rainy and cloudy tomorrow
D2	The soccer game is interesting. I like basketball game.
D3	Yesterday weather was cloudy and sunny. I like sunny day.
D4	The baseball game is not interesting. I win the tennis game.

Document Frequency

Words / **Doc #**	Today	weather	is	sunny	and	Cloudy	rainy	tomorrow	The	soccer	game	interesting	I	like	basketball	Yesterday	day	baseball	not	win	tennis
DF	1	2	4	2	2	2	1	1	2	1	2	2	3	2	1	1	1	1	1	1	1

- **Usually, we use Inverse Document Frequency (IDF), and it can be calculated in the form of 1/DF.**
- **But by far the most commonly used version is: IDF = log (n/DF)**

Summary : TF × IDF (or tf.idf)

- Assign a tf.idf weight to each term i in each document d

$$w_{i,d} = tf_{i,d} \times \log(n / df_i)$$

What is the wt of a term that occurs in all of the docs?

$tf_{i,d}$ = frequency of term i in document j

n = total number of documents

df_i = the number of documents that contain term i

- Increases with the number of occurrences within a doc
- Increases with the rarity of the term across the whole corpus

Calculating TF-IDF

D1	Today weather is sunny and cloudy. Rainy and cloudy tomorrow
D2	The soccer game is interesting. I like basketball game.
D3	Yesterday weather was cloudy and sunny. I like sunny day.
D4	The baseball game is not interesting. I win the tennis game.

$$w_{i,d} = tf_{i,d} \times \log(n / df_i)$$

What is the wt of a term that occurs in all of the docs?

$tf_{i,d}$ = frequency of term i in document j

n = total number of documents

df_i = the number of documents that contain term i

TF-IDF

Words / Doc #	Today	weather	is	sunny	and	Cloudy	rainy	tomorrow	The	soccer	game	Interesting	I	Like	basketball	Yesterday	day	baseball	not	Win	Tennis
D1	1* log(4/1)	1* log(4/2)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
D2	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
D3	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
D4	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

TF × IDF term weights

- TF × IDF measure combines:
  - Term Frequency (TF)
    - or wf, some measure of term density in a doc
  - Inverse document frequency (IDF)
    - measure of informativeness of a term: its rarity across the whole corpus
    - Could just be raw count of number of documents the term occurs in ($IDF_i = 1/DF_i$)
    - but by far the most commonly used version is:

      $IDF_i = \log(n/DF_i)$

Summary : TF × IDF (or tf.idf)

- Assign a tf.idf weight to each term i in each document d

$$w_{i,d} = tf_{i,d} \times \log(n / df_i)$$

What is the wt of a term that occurs in all of the docs?

$tf_{i,d}$ = frequency of term i in document j

n = total number of documents

df_i = the number of documents that contain term i

- Increases with the number of occurrences within a doc
- Increases with the rarity of the term across the whole corpus

HADOOP MapReduce Program : Calculating TF-IDF

```
Function: Calculating TF-IDF using Map-Reduce on Hadoop
Input: word in document // set of words that are split
Output: TF-IDF for each word //category of words set
   1st Map function:
      for each word output <word@document title,1>
   1st Reduce function:
     for each word@document
        n = number of each word@document file
        output file “word@documet title, n”
   2nd Map function
      for each word@document title output <document title,word=n>
   2nd Reduce function
      for each document
         N = number of word in all documents
         for each word output “word@document title, n/N”
   3rd Map function
      for each word@document output <word,document title=n/N>
   3rd Reduce function
      for each word
      D = number of word in all documents
      for each document
         d = number of word in document
         calculate TF-IDF value use n, N, d and D
         output “word@documet,TF-IDF value”
}
```