分散処理アプリ演習　第2回
MapReduceアプリケーションの概要

（株）NTTデータ

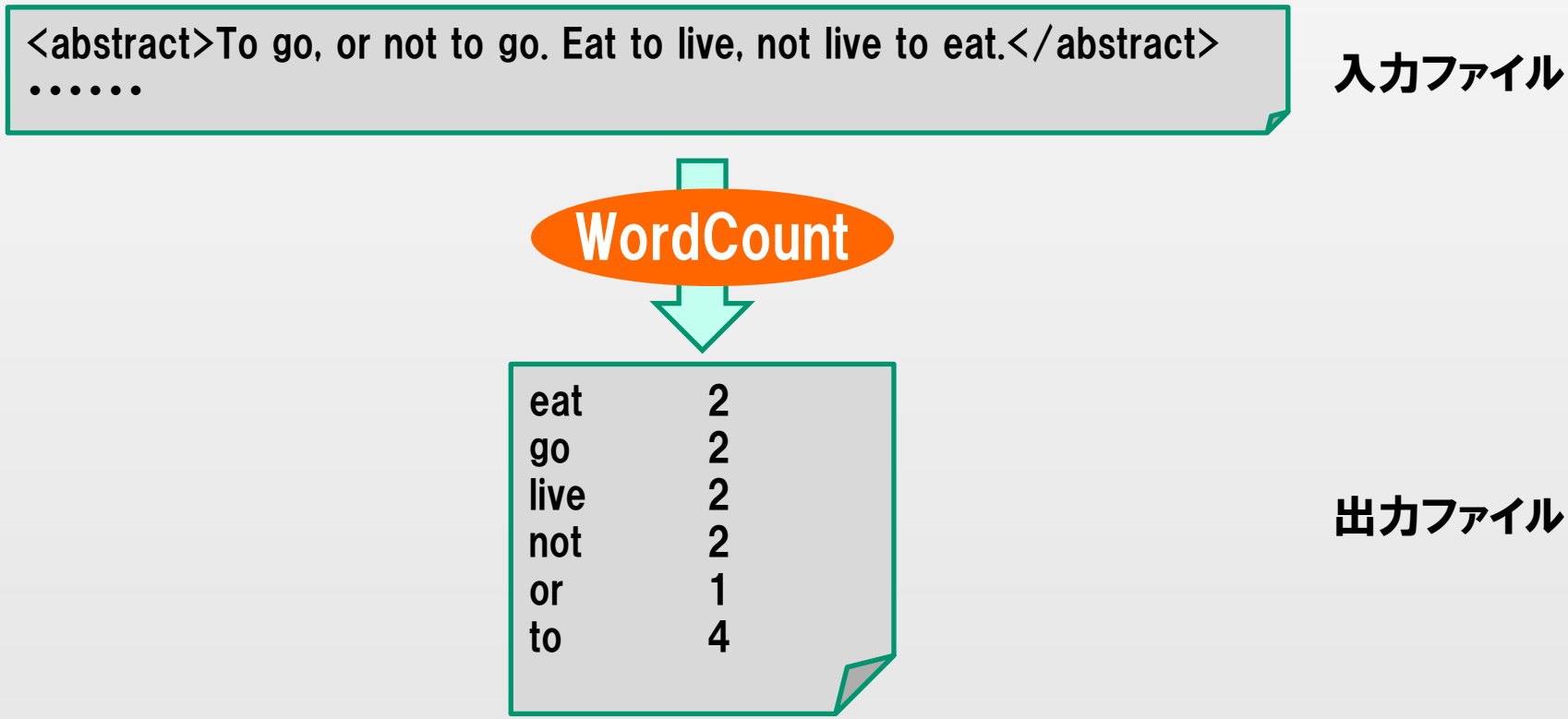
講義内容

1. WordCountプログラムの概要
2. MapReduceプログラムのコンポーネント
3. 演習：WordCountプログラムの実行
4. Streaming API
5. レポート課題

1. WordCountプログラムの概要

MapReduceプログラム事例：WordCount

- 入力のテキストファイル中にある各単語の出現回数をカウントする
  - 本講義では、入力ファイルとしてWikipedia英語版のabstract版のダンプファイル（xmlファイル）の一部とする
    - さらにこのファイルの中で、<abstract>タグで囲まれた文章のみを対象とする



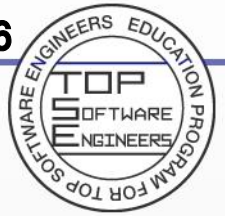
Map関数／Reduce関数の設計

- 擬似コード
  - Mapper

```
map(offset, line)
        words = line.split
        for w in words:
                emit(w, 1)
```

  - Reducer

```
reduce(key, values)
        sum = 0
        for v in values:
                sum += v
        emit(key, sum)
```



2. MapReduceプログラムのコンポーネント

MapReduceプログラムのコンポーネント

- MapReduceプログラムは通常3つの部分から構成される
  - Mapper
  - Reducer
  - ドライバー

コードの全体構成

コードの概要

```
import •••
•••

public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text,
  IntWritable> {
    •••
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text,
  IntWritable> {
    •••
 }

 public static void main(String[] args) throws Exception {
    •••
 }

}
```

Importステートメント

Mapper

Reducer

ドライバー

importステートメント

コードの内容

```
import java.io.IOException;
import java.util.*;
import java.util.regex.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
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;
```

必要となるクラスをインポート

（赤字部分は本講義用に独自に追加）

Mapper

コードの内容

```
public static class Map extends Mapper<LongWritable, Text, Text,
 IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
       String line = value.toString();
       Matcher m =
         Pattern.compile("<abstract>(.*)</abstract>").matcher(line);
       line = (m.matches()) ? m.group(1) : "";
       StringTokenizer tokenizer = new StringTokenizer(line);
       while (tokenizer.hasMoreTokens()) {
           word.set(tokenizer.nextToken());
           context.write(word, one);
       }
   }
}
```

入力のKey、Value、出力のKey、Valueの型を指定

書き込みの2つのオブジェクトを生成

<abstract>タグで囲まれた文字列を抽出

文字列を単語に分解し、(単語、1)を出力

（赤字部分は本講義用に独自に追加）

Reducer

コードの内容

```
public static class Reduce extends Reducer<Text, IntWritable, Text,
 IntWritable> {

   public void reduce(Text key, Iterable<IntWritable> values,
     Context context) throws IOException, InterruptedException {
       int sum = 0;
       for (IntWritable val : values) {
           sum += val.get();
       }
       context.write(key, new IntWritable(sum));
   }
}
```

入力のKey、Value、出力のKey、Valueの型を指定

出現回数をカウント

（単語、出現回数）を出力

ドライバー

コードの内容

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");
    job.setJarByClass(WordCount.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

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

    job.waitForCompletion(true);
}
```

Jobに名前を付与し生成

main関数のクラスを指定

Mapフェーズで出力されるKey、Valueそれぞれの型情報を指定

Mapper、Reducerのクラスを指定

入力と出力のフォーマット情報を指定

入力ディレクトリと出力ディレクトリを指定

3. 演習：WordCountプログラムの実行

演習：WordCountプログラムの実行

- 1. クライアントの仮想マシンにログインする。
- 2. 第1回のHDFSの演習で行ったのと同様に、ローカルファイルシステムにある入力データファイルをHDFS上へコピーする。（第1回に続けて行う場合は不要）

```
$ hadoop fs -put /root/hadoop_exercise/01/enwiki /enwiki
$ hadoop fs -lsr /enwiki
```

  - 以下の2つのデータファイルが存在
    - enwiki-long1.xml ： 150MB程度のファイル
    - enwiki-short1.xml ： 1MB程度のファイル
- 3. WordCountのJavaプログラムファイルがローカルファイルシステムにあることを確認する。また、内容を確認する。

```
$ cd /root/hadoop_exercise/02/wordcount
$ ls -l
$ less WordCount.java
```

演習：WordCountプログラムの実行（続き）

- 4. Javaプログラムをコンパイルする。コンパイル後、Classファイルが生成されたことを確認する。

```
$ mkdir wordcount_classes
$ javac -classpath /usr/lib/hadoop-0.20/hadoop-core.jar ¥
   -d wordcount_classes WordCount.java
$ ls -l wordcount_classes
```

  - 以下のClassファイルが生成される
    - WordCount.class：ドライバークラス
    - WordCount$Map.class：Mapクラス
    - WordCount$Reduce.class：Reduceクラス
- 5. コンパイルしたプログラムをJARファイルに入れる。

```
$ jar cvf wordcount.jar -C wordcount_classes .
```

演習：WordCountプログラムの実行（続き）

- 6. JARファイルを使ってMapReduceジョブをサブミットする。入力ファイルは1つを指定する。

```
$ hadoop jar wordcount.jar WordCount ¥
   /enwiki/enwiki-short1.xml /wordcount01
```

  - hadoop jar wordcount.jar：wordcount.jarファイルを使ってMapReduceジョブをサブミット
  - WordCount：WordCountクラスのmainメソッドを実行
  - /enwiki /enwiki-short1.xml：HDFS上の入力ファイル
  - /wordcount 01: HDFS上の出力ディレクトリ
  - MapとReduceの処理の進行状況が以下のように標準出力に表示されるのを確認する（mapが完了した後、reduceが開始される）

```
yy/mm/dd hh:mm:ss INFO mapred.JobClient:  map 0% reduce 0%
yy/mm/dd hh:mm:ss INFO mapred.JobClient:  map 100% reduce 0%
yy/mm/dd hh:mm:ss INFO mapred.JobClient:  map 100% reduce 33%
yy/mm/dd hh:mm:ss INFO mapred.JobClient:  map 100% reduce 100%
```

演習：WordCountプログラムの実行（続き）

- 7. 出力ディレクトリを確認する。

```
$ hadoop fs -lsr /wordcount01
```

  - _SUCCESS：ジョブが成功して完了したことを示すファイル
  - _logs：ログディレクトリ
  - part-r-00000：出力ファイル（Reducerが1つの場合）
- 8. 出力ファイルの内容を確認する。

```
$ hadoop fs -cat /wordcount01/part-r-00000 | less
```

  - 各行は、単語 [空白] 出現回数
  - 単語は昇順にソートされている
- 9. 出力ファイルの内容を出現回数（2フィールド目）の降順にソートして表示する。出現回数の最も多い単語と、その出現回数を確認する。

```
$ hadoop fs -cat /wordcount01/part-r-00000 ¥
    | sort -k2 -n -r | less
```

演習：WordCountプログラムの実行（続き）

- 10. JARファイルを使ってMapReduceジョブをサブミットする。入力ファイルは6とは別のファイルを指定する。

```
$ hadoop jar wordcount.jar WordCount ¥
    /enwiki/enwiki-long1.xml /wordcount02
```

- 11. 出力ディレクトリを確認する。

```
$ hadoop fs -lsr /wordcount02
```

- 12. 出力ファイルの内容を出現回数（2フィールド目）の降順にソートして表示する。出現回数の最も多い単語と、その出現回数を確認する。

```
$ hadoop fs -cat /wordcount02/part-r-00000 ¥
    | sort -k2 -n -r | less
```

演習：WordCountプログラムの実行（続き）

- 13. JARファイルを使ってMapReduceジョブをサブミットする。入力はディレクトリ全体（配下の全てのファイルが対象となる）を指定する。

```
$ hadoop jar wordcount.jar WordCount /enwiki /wordcount03
```

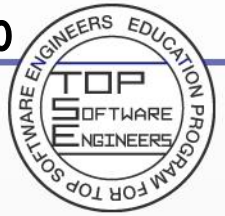
- 14. 出力ディレクトリを確認する。

```
$ hadoop fs -lsr /wordcount03
```

- 15. 出力ファイルの内容を出現回数（2フィールド目）の降順にソートして表示する。出現回数の最も多い単語と、その出現回数を確認する。

```
$ hadoop fs -cat /wordcount03/part-r-00000 ¥
   | sort -k2 -n -r | less
```

  - 出現回数が、9と12で確認した各出現回数の合計（各入力ファイルの結果の合計）となっていることを確認する



4. Streaming API

Streaming API

- MapReduceを利用したいが、Java以外の言語を得意とする開発者も数多く存在している
  - Perl
  - Python
  - その他
- HadoopのStreaming APIは、 Java以外の言語を使用してMapReduceプログラムを書くことができる
  - 標準入力から読み込み、標準出力に書き出すことができる言語であればよい

Streaming APIの仕組み

- MapperとReducerのプログラムをそれぞれ記述する
  - 標準入力から入力を受け取る
  - 標準出力に出力を書き込む
- 入力フォーマットは、Key [TAB] Value
- 出力フォーマットは、Key [TAB] Value [改行]
- TAB以外の区切り文字も指定可能

Streamingジョブの実行

Streamingジョブの実行方法

```
$ hadoop jar ¥
/usr/lib/hadoop-0.20/contrib/streaming/hadoop-streaming*.jar ¥
-input   入力ディレクトリ ¥
-output   出力ディレクトリ ¥
-file   Mapperプログラムのパス ¥
-file   Reducerプログラムのパス ¥
-mapper   Mapperプログラムのファイル名 ¥
-reducer   Reducerプログラムのファイル名
```

例

```
$ hadoop jar ¥
/usr/lib/hadoop-0.20/contrib/streaming/hadoop-streaming*.jar ¥
-input /enwiki ¥
-output /wordcount04 ¥
-file mapper.py ¥
-file reducer.py ¥
-mapper mapper.py ¥
-reducer reducer.py
```

まとめ

本講義で学んだ内容

- WordCountプログラムの概要
  - 文章中の単語の出現回数をカウント、入力データとしてWikipediaのダンプファイルを利用
- MapReduceプログラムのコンポーネント
  - Mapper、Reducer、ドライバー
- 演習：WordCountプログラムの実行
  - hadoop fsコマンドで入力データファイルをHDFSへコピー、javacコマンドでプログラムをコンパイル、hadoop jarコマンドでジョブをサブミット（プログラムの実行）、hadoop fsコマンドで出力ファイルを確認
- Streaming API
  - Java以外の言語を使用してMapReduceプログラムを書くことが可能

5. レポート課題

レポート課題：Streaming APIを用いたWordCountプログラムの実装

- Java以外で自分の得意なプログラミング言語を用いて、本講義と同じ仕様のWordCountプログラムを実装する
  - 入力ファイルは、本講義で用いたWikipedia英語版のabstract版のダンプファイルとする
  - このファイルの中で、<abstract>タグで囲まれた文章のみを対象とする
  - 本講座の演習環境に実行環境がインストール済みのプログラミング言語を選択すること
- 提出物
  - ソースコード（コメントを含めること）
  - 実行コマンド
  - 実行結果（カウント数の多いものから降順にソートして上位の一部のみを表示したもの）
  - 未完成であっても部分点を出すので提出すること
- 提出方法：上記のファイル群をtar.gzまたはzipで1つにまとめてLMSに提出
- 提出期限：当日周知
- 質問先：当日周知