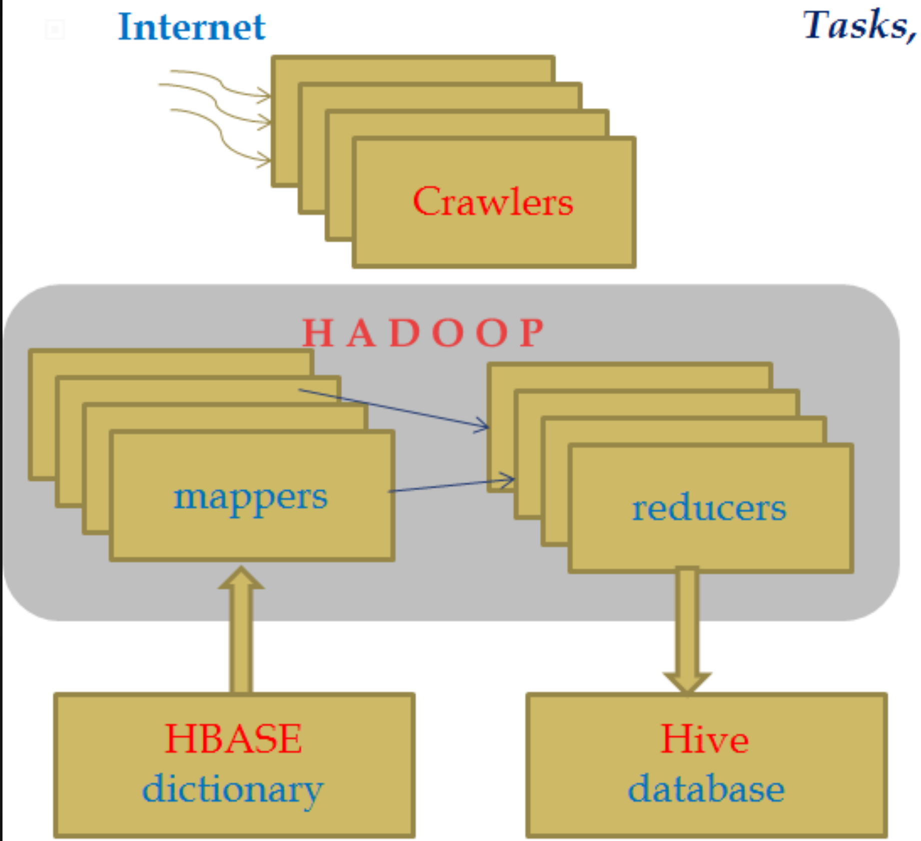
Research Laboratory

Java Web Crawler & Hadoop MapReduce

Anri Morchiladze && Bachana Dolidze
Supervisor Nodar Momtselidze

1. Java Web Crawler
   - Description
   - Java Code
2. MapReduce
   - Overview
   - Example of mapreduce program
   - Code & Run
   - Walk-Through



Tasks, have to be developed – this semester

- ***Create Hadoop multycomputer (5) System.***
- ***Install OS (Linux).***
- ***Install and configure map / reduce, Hive, Hbase, Crawler.***
- ***Develop programm to collect information from internet.***
- ***Develop Hbase dictionary, and Hive database.***
- ***Calculates number of needed word combination, using map / reduce technology, filters with the help of Hbase dictionary. Then summarize number of searched word combination and create database with Hive technology.***
- ***Create Data Warehouse***

```
class Mapper
    method Map(docid a, doc d)
        for all term $t \in$ doc d do
            if term t ! $\in$ HBase
                Emit(term t, count 1)
class Reducer
    method Reduce(term t, counts [c1, c2, . . .])
        sum $\leftarrow 0$
        for all count $c \in$ counts [c1, c2, . . .] do
            sum $\leftarrow$ sum + c
        Emit(term t, count sum)
        To Hive
```

Crawler

A web crawler (also known as a web spider or web robot) is a program or automated script which browses the World Wide Web in a methodical, automated manner.

This process is called Web crawling or spidering.

Description

- Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine, that will index the downloaded pages to provide fast searches.
- Crawlers can also be used for automating maintenance tasks on a Web site, such as checking links or validating HTML code.
- Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting e-mail addresses (usually for spam).

Our Aim

1. Our aim is to search specified word in web pages.
2. We used ibrary Jsoup for it and its commands. http://jsoup.org/

Java Code

1. Using Eclipse
2. Add a Jsoup library to project
3. You can change web site url or text word in this program and see results.
4. results are kept in a file

Mapreduce

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

A MapReduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

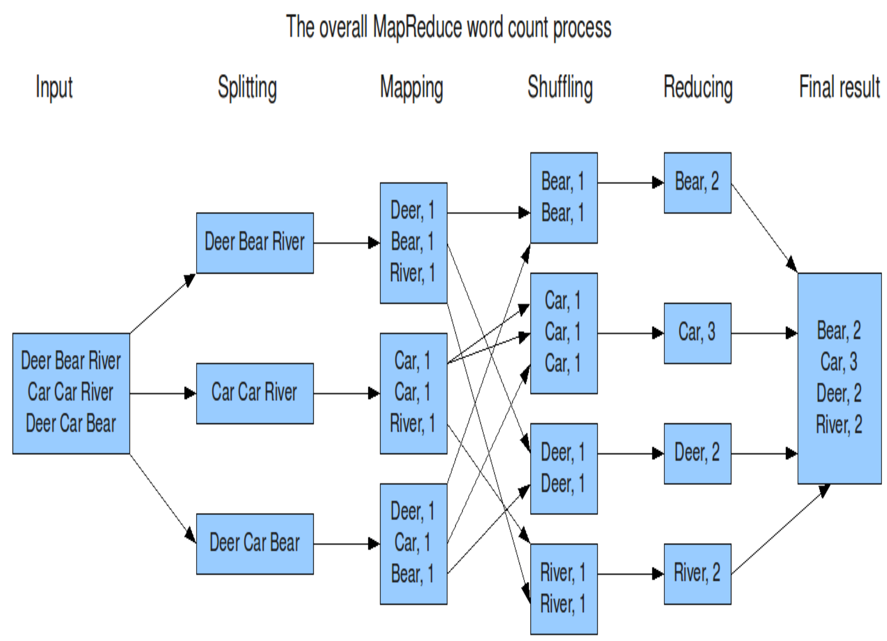
Overview

The MapReduce framework consists of a single master `JobTracker` and one slave `TaskTracker` per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

The MapReduce framework operates exclusively on `<key, value>` pairs, that is, the framework views the input to the job as a set of `<key, value>` pairs and produces a set of `<key, value>` pairs as the output of the job, conceivably of different types.

Input and Output types of a MapReduce job:

(input) `<k1, v1>` -> **map** -> `<k2, v2>` -> **combine** -> `<k2, v2>` -> **reduce** -> `<k3, v3>` (output)



Example

Before we jump into the details, lets walk through an example MapReduce application to get a flavour for how they work.

`WordCount` is a simple application that counts the number of occurences of each word in a given input set.

This works with a local-standalone, pseudo-distributed or fully-distributed Hadoop installation (Single Node Setup)

Code & Run

Usage

Assuming `HADOOP_HOME` is the root of the installation and `HADOOP_VERSION` is the Hadoop version installed, compile `WordCount.java` and create a jar:

```
$ mkdir wordcount_classes
```

```
$ javac -classpath ${HADOOP_HOME}/hadoop-${HADOOP_VERSION}-core.jar -d wordcount_classes WordCount.java
```

```
$ jar -cvf /usr/joe/wordcount.jar -C wordcount_classes/ .
```

Assuming that:

- `/usr/joe/wordcount/input` - input directory in HDFS
- `/usr/joe/wordcount/output` - output directory in HDFS

Code & Run

Sample text-files as input:

```
$ bin/hadoop dfs -ls /usr/joe/wordcount/input/

/usr/joe/wordcount/input/file01

/usr/joe/wordcount/input/file02



$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file01

Hello World Bye World

$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file02

Hello Hadoop Goodbye Hadoop
```

Run the application:

```
$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount /usr/joe/wordcount/input
/usr/joe/wordcount/output
```

Output

```
$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```

Walk-Through

The `WordCount` application is quite straight-forward.The `Mapper` implementation (lines 14-26), via the `map` method (lines 18-25), processes one line at a time, as provided by the specified `TextInputFormat` (line 49). It then splits the line into tokens separated by whitespaces, via the `StringTokenizer`, and emits a key-value pair of `< <word>, 1>`.

For the given sample input the first map emits:

```
< Hello, 1>

< World, 1>

< Bye, 1>

< World, 1>
```

The second map emits:

```
< Hello, 1>

< Hadoop, 1>

< Goodbye, 1>

< Hadoop, 1>
```

Walk-Through

`WordCount` also specifies a `combiner` (line 46). Hence, the output of each map is passed through the local combiner (which is same as the `Reducer` as per the job configuration) for local aggregation, after being sorted on the *keys*.

The output of the first map:

```
< Bye, 1>

< Hello, 1>

< World, 2>
```

The output of the second map:

```
< Goodbye, 1>

< Hadoop, 2>

< Hello, 1>
```

Walk-Through

The `Reducer` implementation (lines 28-36), via the `reduce` method (lines 29-35) just sums up the values, which are the occurence counts for each key (i.e. words in this example).

Thus the output of the job is:

```
< Bye, 1>

< Goodbye, 1>

< Hadoop, 2>

< Hello, 2>

< World, 2>
```

The `run` method specifies various facets of the job, such as the input/output paths (passed via the command line), key/value types, input/output formats etc., in the`JobConf`. It then calls the `JobClient.runJob` (line 55) to submit the and monitor its progress.

Electrical Consumption

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Avg
1979	23	23	2	43	24	25	26	26	26	26	25	26	25
1980	26	27	28	28	28	30	31	31	31	30	30	30	29
1981	31	32	32	32	33	34	35	36	36	34	34	34	34
1984	39	38	39	39	39	41	42	43	40	39	38	38	40
1985	38	39	39	39	39	41	41	41	00	40	39	39	45

If the above data is given as input, we have to write applications to process it and produce results such as finding the year of maximum usage, year of minimum usage, and so on. This is a walkover for the programmers with finite number of records. They will simply write the logic to produce the required output, and pass the data to the application written.

But, think of the data representing the electrical consumption of all the largescale industries of a particular state, since its formation.

When we write applications to process such bulk data,

- They will take a lot of time to execute.
- There will be a heavy network traffic when we move data from source to network server and so on.

To solve these problems, we have the MapReduce framework.

Thank you for pay attention!!!