

=====

TP2

Le traitement Batch avec Hadoop

HDFS- MapReduce

=====

Map Reduce

Présentation

Un Job Map-Reduce se compose principalement de deux types de programmes :

- **Mappers** : permettent d'extraire les données nécessaires sous forme de clef/valeur, pour pouvoir ensuite les trier selon la clef
- **Reducers** : prennent un ensemble de données triées selon leur clef, et effectuent le traitement nécessaire sur ces données (somme, moyenne, total...)

Wordcount

Nous allons tester un programme MapReduce grâce à un exemple très simple, le *WordCount*, l'équivalent du *HelloWorld* pour les applications de traitement de données. Le Wordcount permet de calculer le nombre de mots dans un fichier donné, en décomposant le calcul en deux étapes:

- L'étape de *Mapping*, qui permet de découper le texte en mots et de délivrer en sortie un flux textuel, où chaque ligne contient le mot trouvé, suivi de la valeur 1 (pour dire que le mot a été trouvé une fois)
- L'étape de *Reducing*, qui permet de faire la somme des 1 pour chaque mot, pour trouver le nombre total d'occurrences de ce mot dans le texte.

Commençons par créer un projet Maven dans IntelliJ IDEA. Nous utiliserons dans notre cas JDK 1.8.

- Définir les valeurs suivantes pour votre projet:
 - **GroupId**: hadoop.mapreduce
 - **ArtifactId**: wordcount
 - **Version**: 1
- Ouvrir le fichier *pom.xml*, et ajouter les dépendances suivantes pour Hadoop, HDFS et Map Reduce:

```
<dependencies>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>2.7.2</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-mapreduce-
client-core -->
  <dependency>
    <groupId>org.apache.hadoop</groupId>
```

```

        <artifactId>hadoop-mapreduce-client-core</artifactId>
        <version>2.7.2</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs -->
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-hdfs</artifactId>
        <version>2.7.2</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-mapreduce-client-common</artifactId>
        <version>2.7.2</version>
    </dependency>
</dependencies>

```

- Créer un package *ma.ensam.tp1* sous le répertoire *src/main/java*
- Créer la classe *TokenizerMapper*, contenant ce code:

```

package ma.ensam.tp1;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

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

public class TokenizerMapper extends Mapper<Object, Text, Text,
IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Mapper.Context context) throws
IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}

```

- Créer la classe *IntSumReducer*.

```

package ma.ensam.tp1;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

```

```

public class IntSumReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            System.out.println("value: " + val.get());
            sum += val.get();
        }
        System.out.println("--> Sum = " + sum);
        result.set(sum);
        context.write(key, result);
    }
}

```

- Enfin, créer la classe *WordCount*.

```

package ma.ensam.tp1;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

LANCER MAP REDUCE SUR LE CLUSTER

Dans votre projet Eclipse:

- Créer une configuration Maven avec la ligne de commande: `package install`
- Lancer la configuration. Un fichier `job-0.0.1-SNAPSHOT.jar` sera créé dans le répertoire *target* du projet.
- Taper la commande suivante :

```
docker cp job-0.0.1-SNAPSHOT.jar hadoop-master:/root/ job-0.0.1-SNAPSHOT.jar
```

- Revenir au shell du conteneur master, et lancer le job map reduce avec cette commande:

```
hadoop jar job-0.0.1-SNAPSHOT.jar ma.ensam.tp1.WordCount input output
```

Le Job sera lancé sur le fichier *purchases.txt* que vous aviez préalablement chargé dans le répertoire *input* de HDFS. Une fois le Job terminé, un répertoire *output* sera créé. Si tout se passe bien, vous obtiendrez un affichage ressemblant au suivant:

```
[root@hadoop-master:~# hadoop jar wordcount-1.jar tn.insat.tp1.WordCount input output
18/01/27 10:58:13 INFO client.RMProxy: Connecting to ResourceManager at hadoop-master/172.22.0.2:8032
18/01/27 10:58:14 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute y
our application with ToolRunner to remedy this.
18/01/27 10:58:14 INFO input.FileInputFormat: Total input paths to process : 1
18/01/27 10:58:14 INFO mapreduce.JobSubmitter: number of splits:1
18/01/27 10:58:15 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1517050438813_0001
18/01/27 10:58:15 INFO impl.YarnClientImpl: Submitted application application_1517050438813_0001
18/01/27 10:58:16 INFO mapreduce.Job: The url to track the job: http://hadoop-master:8088/proxy/application_1517050438813_0001/
18/01/27 10:58:16 INFO mapreduce.Job: Running job: job_1517050438813_0001
18/01/27 10:58:29 INFO mapreduce.Job: Job job_1517050438813_0001 running in uber mode : false
18/01/27 10:58:29 INFO mapreduce.Job: map 0% reduce 0%
18/01/27 10:58:47 INFO mapreduce.Job: map 42% reduce 0%
18/01/27 10:59:11 INFO mapreduce.Job: map 67% reduce 0%
18/01/27 10:59:28 INFO mapreduce.Job: map 100% reduce 0%
18/01/27 10:59:37 INFO mapreduce.Job: map 100% reduce 100%
18/01/27 10:59:37 INFO mapreduce.Job: Job job_1517050438813_0001 completed successfully
18/01/27 10:59:38 INFO mapreduce.Job: Counters: 49
  File System Counters
    FILE: Number of bytes read=2568324
    FILE: Number of bytes written=4086870
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=44337931
    HDFS: Number of bytes written=491081
    HDFS: Number of read operations=6
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
  Job Counters
    Launched map tasks=1
    Launched reduce tasks=1
    Data-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=57326
    Total time spent by all reduces in occupied slots (ms)=6032
    Total time spent by all map tasks (ms)=57326
    Total time spent by all reduce tasks (ms)=6032
    Total vcore-milliseconds taken by all map tasks=57326
    Total vcore-milliseconds taken by all reduce tasks=6032
    Total megabyte-milliseconds taken by all map tasks=58701824
    Total megabyte-milliseconds taken by all reduce tasks=6176768
  Map-Reduce Framework
    Map input records=868279
    Map output records=5872188
    Map output bytes=67826564
    Map output materialized bytes=1284159
    Input split bytes=120
    Combine input records=5872188
    Combine output records=101438
    Reduce input groups=50766
    Reduce shuffle bytes=1284159
    Reduce input records=101438
    Reduce output records=50766
    Spilled Records=304314
    Shuffled Maps =1
    Failed Shuffles=0
    Merged Map outputs=1
    GC time elapsed (ms)=225
    CPU time spent (ms)=51590
    Physical memory (bytes) snapshot=549974016
    Virtual memory (bytes) snapshot=1775112192
    Total committed heap usage (bytes)=292028416
  Shuffle Errors
    BAD_ID=0
    CONNECTION=0
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
    WRONG_REDUCE=0
  File Input Format Counters
    Bytes Read=44337811
  File Output Format Counters
    Bytes Written=491081
```

En affichant les dernières lignes du fichier généré *output/part-r-00000*, avec `hadoop fs -tail output/part-r-00000`, vous obtiendrez l'affichage suivant:

Petersburg	8430
Philadelphia	8471
Phoenix	8431
Pittsburgh	8470
Plano	8323
Portland	8367
Raleigh	8345
Reno	8334
Richmond	8388
Riverside	8338
Rochester	8440
Rouge	8396
Sacramento	8597
Saint	8494
San	42110
Santa	8416
Scottsdale	8443
Seattle	8339
Spokane	8356
Sporting	48207
Springs	8534
St.	16881
Stockton	8289
Supplies	48265
Tampa	8400
Toledo	8314
Toys	48463
Tucson	8546
Tulsa	8444
Vegas	16957
Video	48439
Virginia	8465
Visa	174018
Vista	8510
Washington	8477
Wayne	8527
Wichita	8547
Winston-Salem	8459
Women's	48252
Worth	8462
York	8529
and	48408

Il vous est possible de monitorer vos Jobs Map Reduce, en allant à la page: <http://localhost:8088>. Vous trouverez votre Job dans la liste des applications comme suit:



Logged in as: dr.who

All Applications

Cluster

- About
- Nodes
- Node Labels
- Applications
- NEW
- NEW SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED
- Scheduler

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
1	0	0	1	0	0 B	16 GB	0 B	0	16	0	2	0	0	0	0

Scheduler Metrics

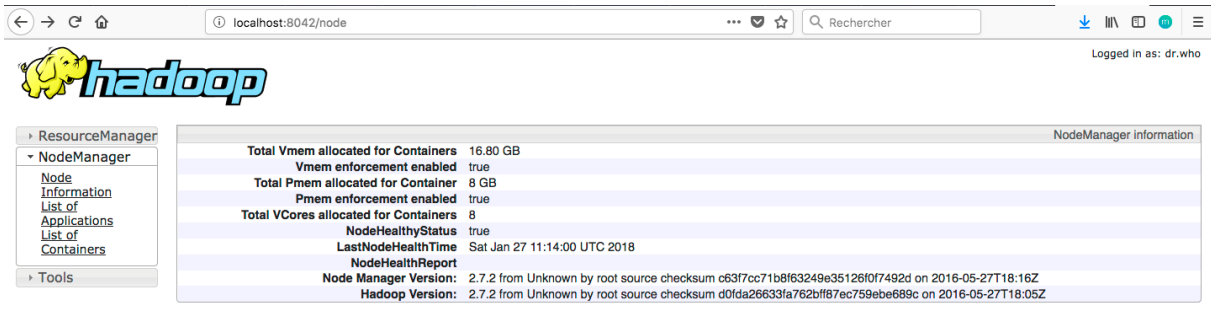
Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Show 20 entries

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1517050438813_0001	root	word count	MAPREDUCE	default	Sat Jan 27 11:58:15 +0100 2018	Sat Jan 27 11:59:36 +0100 2018	FINISHED	SUCCEEDED		History	N/A

Showing 1 to 1 of 1 entries

Il est également possible de voir le comportement des noeuds esclaves, en allant à l'adresse: `http://localhost:8041` pour *slave1*, et `http://localhost:8042` pour *slave2*. Vous obtiendrez ce qui suit:



The screenshot shows the Hadoop NodeManager web interface in a browser. The address bar displays `localhost:8042/node`. The page features the Hadoop logo and a navigation menu on the left with options: ResourceManager, NodeManager (selected), Node Information, List of Applications, List of Containers, and Tools. The main content area, titled "NodeManager information", displays the following details:

Total Vmem allocated for Containers	16.80 GB
Vmem enforcement enabled	true
Total Pmem allocated for Container	8 GB
Pmem enforcement enabled	true
Total VCores allocated for Containers	8
NodeHealthyStatus	true
LastNodeHealthTime	Sat Jan 27 11:14:00 UTC 2018
NodeHealthReport	
Node Manager Version:	2.7.2 from Unknown by root source checksum c63f7cc71b8f63249e35126f0f7492d on 2016-05-27T18:16Z
Hadoop Version:	2.7.2 from Unknown by root source checksum d0da26633fa762bf67ec759ebe689c on 2016-05-27T18:05Z

Écrire un Job Map Reduce permettant, à partir du fichier `purchases initial`, de déterminer le total des ventes par magasin.