Cloud Computing: Laborator 2

Honorius Gâlmeanu
galmeanu@unitbv.ro

October 25, 2023

1 Termeni Hadoop

Platforma Hadoop oferă capabilităţi de procesare paralelă pentru seturi de date de volum mare. Instalarea pe care o aveţi pe maşina virtuală suportă deja atât programe Java pentru Hadoop cât şi programarea pe platformele Hive şi Spark (care sunt construite peste funcţionalitatea Hadoop). Se recomandă să rulaţi imaginea de VirtualBox pe un calculator cu cel puţin 4GB RAM, recomandat 8GB, şi dual-core. Setările maşinii virtuale trebuie să specifice atât faptul că maşinii virtuale îi sunt alocaţi 4GB cât şi alocarea a cel puţin două core-uri - fără ca acest lucru să producă warning-uri în VirtualBox.

Sistemul Hadoop are doi daemoni[1] care sunt activi permanent:

- HDFS (Hadoop Distributed File System), proces care pune la dispoziţia platformei un sistem de fişiere distribuit pe mai multe maşini. În general fişierele în HDFS sunt împărţite în blocuri (mărimea standard este 32/64MB), care sunt replicate pe mai multe maşini (factorul de replicare implicit este 3). Fiecare astfel de bloc este replicat în mai multe shard-uri (mai multe shard-uri sunt în fapt imaginea binară a unui aceluiaşi bloc);
- YARN (Yet Another Resource Negotiator), este planificatorul de procese care rulează pe toate nodurile şi este responsabil cu pornirea, oprirea, supravegherea şi alocarea pentru ele.

Pentru YARN, am prezentat în materialele de curs arhitectura şi modul de interacţiune dintre Resource Manager (care rulează pe nodul master), Application Master (proces responsabil cu crearea task-urilor şi preluarea rezultatelor unui job), Node Manager (dispatcher de task-uri la nivel de nod) respectiv containere (sloturile ce ruleaza un task). O astfel de structură există şi pentru buna funcţionare a HDFS-ului: Name Node, Data Node, Secondary Name Node. Job Tracker şi Task Tracker sunt denumiri pentru Application Master şi Node Manager pentru versiuni anterioare versiunii Hadoop 2.0.

Name Node este nodul master care supervizează activitatea HDFS-ului. El decide cum se face împărţirea fişierelor pe blocuri, cum funcţionează mecanismul de replicare al blocurilor şi ce nod deţine ce blocuri. Deoarece există un singur nod de acest tip, el constituie un bottleneck în sistem.

Secondary Name Node este nodul pe care funcţionează un proces ce preia, în timp real, toate alocările realizate de Name Node. Este un backup al acestuia din urmă. Dacă nu mai sunt primite heartbeat-uri de la acesta, Secondary Name Node are toate metainformaţiile privind alocarea HDFS-ului pentru a putea prelua 'din zbor' funcţionarea ca Name Node. Un Secondary Name Node nu comunică decât cu Name Node-ul.

Un Data Node rulează efectiv procesul de alocare şi stocare a fişierelor distribuite. El dialoghează cu Name Node precum şi cu celelalte Data Nodes pentru a stoca date şi a realiza redundanţa, folosind schema de alocare dictată de Name Node.

2 Pornirea platformei Hadoop

Cei doi daemoni Hadoop nu funcţionează implicit; aceştia trebuiesc porniţi. Ei lucrează pe sistemul Linux sub credenţialele userului 'hadoop'; de regulă programatorul lucrează ca 'user'. Credenţialele de 'root' se pot folosi, dar de regulă lucrul ca 'root' este descurajat. Dacă nu ştiţi exact ce faceţi este bine să nu folosiţi credenţialele root-ului.

Pornirea se realizează folosind comanda ce pornește atât daemon-ul de DFS cât și cel de YARN:

[1] daemon = proces care rulează (într-un sistem de tip Unix) fără a necesita intervenţia utilizatorului şi fără a tipări date la consolă

Command Line

```
$ /home/user/apps/hadoop/sbin/start-all.sh
```

Notice: Observaţi că se lucrează implicit ca 'user'; instalarea făcută rulează sistemul Hadoop cu credențialele user-ului.

Notice: Dacă nu se specifică explicit, întotdeauna se va folosi rolul 'user'.

Oprirea celor două procese daemon se va face astfel:

Command Line

```
$ /home/user/apps/hadoop/sbin/stop-all.sh
```

Notice: Nu opriţi maşina virtuală cu 'Reset' sau 'Power Off'! Acest lucru va conduce la coruperea sistemului de fişiere HDFS. Folosiţi întotdeauna secvenţa Start | Logout | Shutdown din maşina virtuală Linux.

3 Regenerarea sistemului HDFS (opţional)

Dacă totuşi aţi corupt sistemul HDFS, veţi sesiza o serie de erori de tip 'cannot read file' sau similare atunci când vreţi să scrieţi / citiţi fişiere pe / de pe HDFS. Va trebui să-l regeneraţi (cu pierderea informaţiilor stocate acolo).

Mai întâi va trebui să opriţi serviciile:

Command Line

```
$ sudo -u hadoop /home/user/apps/hadoop/sbin/stop-all.sh
```

Regenerați sistemul HDFS precum este descris în laboratorul anterior:

Command Line

```
$ cd /home/user/apps/hadoop
$ bin/hdfs namenode -format
```

Se repornesc serviciile şi apoi se creează şi home-ul user-ului în HDFS (dacă nu facem asta, programele pe care le vom rula ca user nu vor putea să-şi scrie rezultatele):

Command Line

```
$ /home/user/apps/hadoop/sbin/start-dfs.sh
$ hdfs dfs -mkdir -p /user/user
$ hdfs dfs -chown user:user /user/user
```

Se creeaza si folder-ul /tmp:

Command Line

```
$ hdfs dfs -mkdir -p /tmp
$ hdfs dfs -chmod 777 /tmp
$ hdfs dfs -chmod a+t /tmp
```

4 Programul Word count

Ne propunem să determinăm frecvenţa cuvintelor folosite într-un text. Pentru aceasta, folosim iniţial următorul scurt text:

minibook.txt

```
Hello to you all!
Hello, World!
Other types of 'hello'.
All of hello types..
Hello to you, all other.
```

Vom procesa acest text în două etape: map şi reduce. Mapper-ul va avea rolul de a împărţi textul în cuvinte şi de a emite toate perechile de tip (cuvânt, 1) existente[2]

mapper.py

```
#!/usr/bin/env python
#
import sys
import re

# read input from stdin, line by line
for line in sys.stdin:
  # strip trailing spaces and split into words
  words = re.split('[\W]+', line.strip())

  # emit <key, value> pairs
  for word in words:
    if word != '':
      print '%s %s' % (word.lower(), 1)
```

Partitioner-ul va agrega aceste perechi folosind cuvântul drept cheie, Reducer-ul va prelua aceste perechi şi va aduna toate counter-ele asociate aceluiaşi cuvânt:

[2]Pentru a vedea ce este cu acea metodă re.split, puteţi deschide un interpretor Python şi apela comanda **help("re.split").**

reducer.py

```
#!/usr/bin/env python

import sys

d = {}

for line in sys.stdin:
  # strip and split considering only the first gap
  word, count = line.strip().split(' ', 1)

  # convert the string to integer
  try:
    count = int(count)
  except ValueError:
    # if the value is not a number, discard line
    continue

  c = d.get(word)
  c = 0 if c == None else c
  d[word] = c + count

for (k, v) in d.items():
  print '%s %s' % (k, v)
```

La rulare off-line (fără Hadoop) se generează următorul rezultat:

Command Line

```
$ cat minibook.txt | python mapper.py | python reducer.py
all 3
of 2
to 2
other 2
world 1
you 2
hello 5
types 2
```

Pentru a rula pe Hadoop, mai întâi creem directoarele input şi output pentru job-ul nostru:

Command Line

```
$ hdfs dfs -mkdir input/
$ hdfs dfs -put minibook.txt input/
$ hdfs dfs -ls input/
```

Interfaţa web de inspectare a HDFS-ului

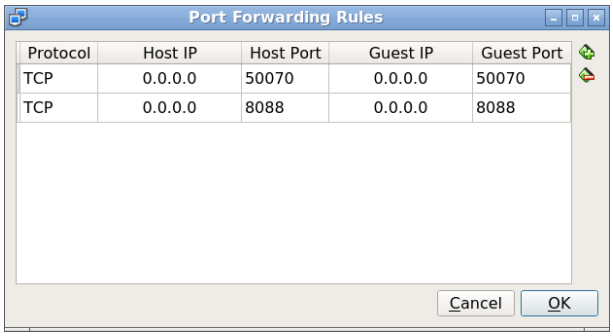
- Pentru a vizualiza starea HDFS-ului precum şi fişierele stocate acolo, aveţi la dispoziţie URL-ul **http://localhost:9870**.
- Folosiţi meniul Utilities | Browse the filesystem pentru aceasta.
- Meniul Utilities | Logs, userlogs/ mai exact, vă va arăta log-urile tuturor aplicaţiilor.

Interfaţa web de inspectare a job-urilor

- Pentru a vizualiza starea joburilor aveţi la dispoziţie URL-ul **http://localhost:8088**.

Interfaţa web nu este vizibilă decât în maşina virtuală. Pentru a redirecta traficul de pe porturile 9870 şi 8088 din VirtualBox în maşina host (Windows), va trebui să faceţi două lucruri:

1. să intraţi în meniul Devices | Network | Network settings şi de aici la Advanced | Port Forwarding şi să adăugaţi două rute, pentru fiecare port:



2. să adăugaţi două rute în firewall-ul din Linux, pentru porturile respective:

```
Command Line
    $ sudo firewall-config
```

   La zona publică veţi adăuga cele două porturi TCP, iar apoi veţi face permanente schimbările folosind opţiunea “Runtime to permanent”.

După adăugarea fişierului veţi rula efectiv codul folosind Streamin API-ul:

```
Command Line
  $ hadoop jar /home/user/apps/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.4.jar
    -file mapper.py -mapper mapper.py -file reducer.py -reducer reducer.py
    -input input/ -output output/
```

Dacă rulaţi succesiv, folder-ul de output/ va trebui şters:

```
Command Line
  $ hdfs dfs -rm -r -f output/
```

După execuţie, puteţi inspecta folder-ul output/ fie din interfaţa web, fie cu comanda “ls”:

```
Command Line
  $ hdfs dfs -ls output/
```

Vizualizarea directă, respectiv extragerea fişierelor din HDFS se face astfel:

```
Command Line
  $ hdfs dfs -cat output/part-00000 | less
  $ hdfs dfs -get output/part-00000 local.txt
```

5 Word count în Java

Programul Java echivalent vine sub forma unui proiect Maven. Ar fi de dorit să exersaţi crearea unui proiect Maven. Pentru aceasta aveţi nevoie de acces la Inernet.

Mai întâi creaţi un proiect folosind generatorul:

Command Line

```
$ mvn archetype:generate -DgroupId=net.examples
  -DartifactId=WordCount -DinteractiveMode=false
  -DarchetypeArtifactId=maven-archetype-quickstart
```

Adăugaţi o secţiune <build> după </dependencies>, cu următorul conţinut:

pom.xml

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.3.2</version>
        <configuration>
            <source>1.6</source>
            <target>1.6</target>
        </configuration>
    </plugin>
</plugins>
```

Adăugaţi şi dependinţele Hadoop:

pom.xml

```
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-common</artifactId>
  <version>3.3.4</version>
</dependency>
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-mapreduce-client-core</artifactId>
  <version>3.3.4</version>
</dependency>
```

Creaţi JAR-ul şi rulaţi-l:

Command Line

```
$ mvn package
$ hadoop jar target/WordCount-1.0-SNAPSHOT.jar net.examples.WordCount input/
output/
```

6 Exerciţii

Question 1 **(Pipe)**

Realizaţi pornirea mediului Hadoop şi parcurgeţi toate instrucţiunile pentru a vă crea mediul de lucru şi de a realiza rularea codului Python pentru mapper şi reducer. Rulaţi exemplul pentru minibook.txt atât în pipe command line cât şi folosind Hadoop. Comparaţi rezultatele (ar trebui să vă dea acelaşi lucru).

Question 2 **(Controlul numărului de reduceri)**

Pentru codul Python deja dat, vom experimenta numărul de reducers. Implicit, vă este creat un singur reducer. Controlaţi numărul de reduceri (să zicem 10) folosind comanda:

```
$ hadoop jar /home/user/apps/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.4.jar
-file mapper.py -mapper mapper.py -file reducer.py -reducer reducer.py
-input input/ -output output/ -numReduceTasks 10
```

Observaţi care din fişierele din HDFS au conţinut şi care nu. Concatenaţi rezultatele. Ar trebui să obţineţi acelaşi rezultat ca şi la rularea prin pipe.

Question 3 **(Observarea rezultatelor mapping-ului)**

De data aceasta setaţi pe 0 numarul de reduceri (deja ştiţi cum). Etapa de reduce va fi sărită. Observaţi câte fişiere se creează după map - aceasta dă numărul mapper-ilor. Câţi mapperi sunt implicit? Inspectaţi listing-ul generat şi căutaţi după “number of splits”.

Question 4 **(Schimbarea numărului de mapperi)**

Realizaţi un scurt program Python care împarte un fişier text în blocuri de mărime cvasi-egală, considerând un număr de mapperi. De exemplu, fişierul are 5 linii; dacă sunt 3 mapperi, atunci primul bloc cuprinde primele 2 linii, următorul bloc următoarele 2 linii din fişier iar ultimul bloc doar o linie. Atenţie să nu rupeţi liniile pentru că puteţi rupe şi cuvintele!

Cu programul astfel creat, alimentaţi folder-ul input din HDFS cu un numar de 3 fişiere. Rulaţi doar mapper-ele (numReduceTasks = 0). Câte fişiere se scriu în HDFS?

Question 5 **(Rulări cu număr diferit de mapperi)**

Partiţionaţi de data aceasta fişierul bigbook.txt în 2, 4 şi 8 blocuri. Notaţi timpul total de rulare (folosiţi comanda ’time’).

Question 6 **(Modificare cod Java)**

Folosiţi codul Java. Observaţi că acesta nu partiţionează corect cuvintele (de exemplu, el consideră ’again!’, ’Again’ şi ’again’ 3 cuvinte diferite). Modificaţi codul Java în aşa fel încât cuvintele să fie formate doar din litere (nu şi semne de punctuaţie) şi să convertiţi cuvintele în lowercase. Rulaţi atât codul Python cât şi codul Java şi comparaţi rezultatele.

Bibliografie

1. http://www.mkyong.com/maven/how-to-create-a-java-project-with-maven/
2. https://hadoopi.wordpress.com/2013/05/25/setup-maven-project-for-hadoop-in-5mn/
3. http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/
4. https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/ hadoop-mapreduce-client-core/ MapReduceTutorial.html