

Report

Class 데이터과학

Title | Hadoop

학 과 명	컴퓨터공학과
교 수 명	이 영 석 교수님
학 번	201601989
이 름	김 진 섭
제 출 일	2019. 06.01



충남대학교
Chungnam National University

목 차

1. 과제 목표

2. [과제1] 3-Node Hadoop 세팅

3. [과제2]Spark 설치

<과제 목표>

작업 환경

- Ubuntu 18.04
- Hadoop 2.9.x
- Java 8.x

1. Hadoop 3 Node Cluster 설정

- 3개의 노드 세팅
- Wordcount V1
- Wordcount V2

2. Spark 설정

- 3개의 Worker 등록
- WordCount

-과제 목표-

- > 3-Node Hadoop을 세팅한다.
- > Hadoop에서 3-cluster와 standalone의 성능을 비교한다.
- > MapReduce를 활용하여 WordCount를 수행할 수 있다.
- > History Server를 통한 Task 확인
- > 3개의 Worker를 동작한다.
- > PySpark를 이용하여 wordcount를 수행한다.
- > PySpark를 이용한 경우와 Hadoop Node의 속도를 비교한다.

[과제 1] 3-Node Hadoop 세팅

```
11 192.168.56.100 master
12 192.168.56.101 slave1
13 192.168.56.102 slave2
```

IP주소들을 저장하여준다.

- 192.168.56.100 master node
- 192.168.56.101 slave1 node
- 192.168.56.102 slave2 node

\$ sudo vi /etc/netplan/50-cloud-init.yaml

위 명령어를 사용해 들어가서, ip들을 고정으로 변경하여 준다.

The image shows three screenshots of Oracle VM VirtualBox terminal windows, each displaying a netplan configuration file for a different node in a 3-node Hadoop setup.

standalone [실행 중] - Oracle VM VirtualBox

```
1 # This file is generated from information provided by
2 # the datasource. Changes to it will not persist across an instance.
3 # To disable cloud-init's network configuration capabilities, write a file
4 # /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
5 # network: {config: disabled}
6 network:
7   ethernets:
8     enp0s3:
9       dhcp4: true
10    enp0s8:
11      addresses:
12        - 192.168.56.100/24
13      nameservers:
14        addresses:
15          - 8.8.8.8
16  version: 2
```

standalone_1 [실행 중] - Oracle VM VirtualBox

```
7   ethernets:
8     enp0s3:
9       dhcp4: true
10    enp0s8:
11      addresses:
12        - 192.168.56.101/24
13      nameservers:
14        addresses:
15          - 8.8.8.8
16  version: 2
```

standalone_2 [실행 중] - Oracle VM VirtualBox

```
6 network:
7   ethernets:
8     enp0s3:
9       dhcp4: true
10    enp0s8:
11      addresses:
12        - 192.168.56.102/24
13      nameservers:
14        addresses:
15          - 8.8.8.8
16  version: 2
```

\$ sudo netplan apply

위 명령어를 활용하여 고정으로 ip들을 위치럼 변경하여준다.

<RSA key>

각 노드는 비밀번호가 없이 다른 노드에 접근을 해야하므로 RSA key를 이용한다.

```
$ ssh-keygen -t rsa
```

key를 생성한다.

```
$ ssh-copy-id -i id_rsa.pub [hostname]
```

hostname에는 각각 slave1, slave2를 넣어주고 실행하여준다.

<Pseudo-Distributed Operation>

각각의 세 개의 가상 컴퓨터의 hadoop폴더 내에서 작업을 수행하여준다.

```
$vi etc/hadoop/core-site.xml
```

```
19 <configuration>
20   <property>
21     <name>fs.defaultFS</name>
22     <value>hdfs://master:9000</value>
23   </property>
24 </configuration>
```

```
$vi etc/hadoop/hdfs-site.xml
```

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

```
$vi etc/hadoop/mapred-site.xml
```

```
15 <configuration>
16
17 <!-- Site specific YARN configuration properties -->
18   <property>
19     <name>yarn.resourcemanager.hostname</name>
20     <value>master</value>
21   </property>
22   <property>
23     <name>yarn.nodemanager.aux-services</name>
24     <value>mapreduce_shuffle</value>
25   </property>
26 </configuration>
```

```
$vi etc/hadoop/yarn-site.xml
```

```
<configuration>
<!-- Site specific YARN configuration properties -->
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

- master와 slave에서 모두 위처럼 작업을 수행하여준다.

<slave>

```
$vi etc/hadoop/slaves
```

```
1 master
2 slave1
3 slave2
```

master에서는 위 명령어로 접속하여 위처럼 제가 만들어준 host ip들을 작성해 줍니다.

[1-a]HDFS 세팅 결과 (노드 3개 정상 동작 과정)

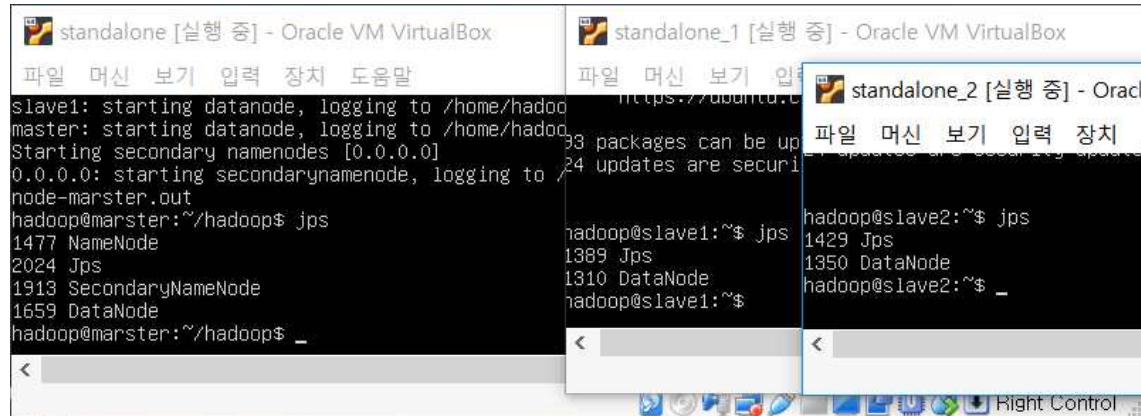
hadoop파일 내에서

```
$bin/hdfs namenode -format
```

```
$start-dfs.sh
```

를 수행하여 3개의 노드가 정상적으로 세팅되어있는지 확인한다.

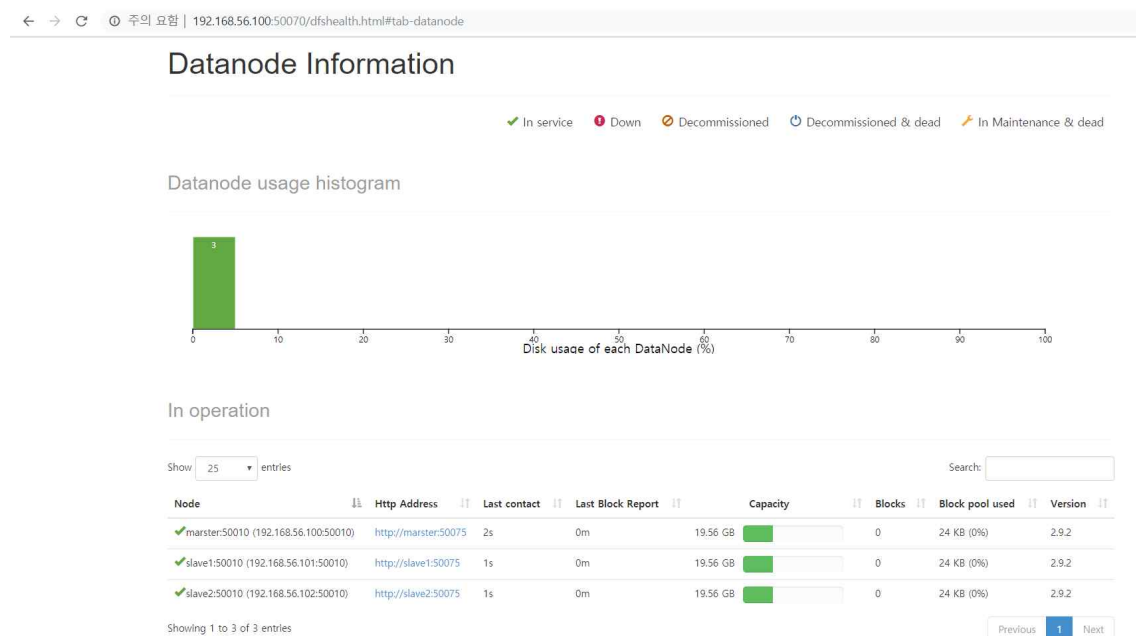
3개의 노드가 정상적으로 수행되는 것을 확인할 수 있다.



Marster

Slave1

Slave2

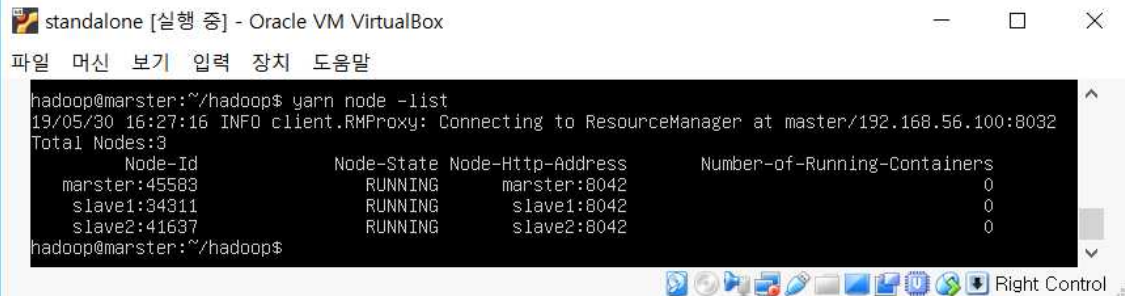


192.168.56.100:50070에 접속하면 위와 같이 3개의 노드가 정상적으로 작동되는 것을 알 수 있다.

[1-a] YARN 세팅 결과

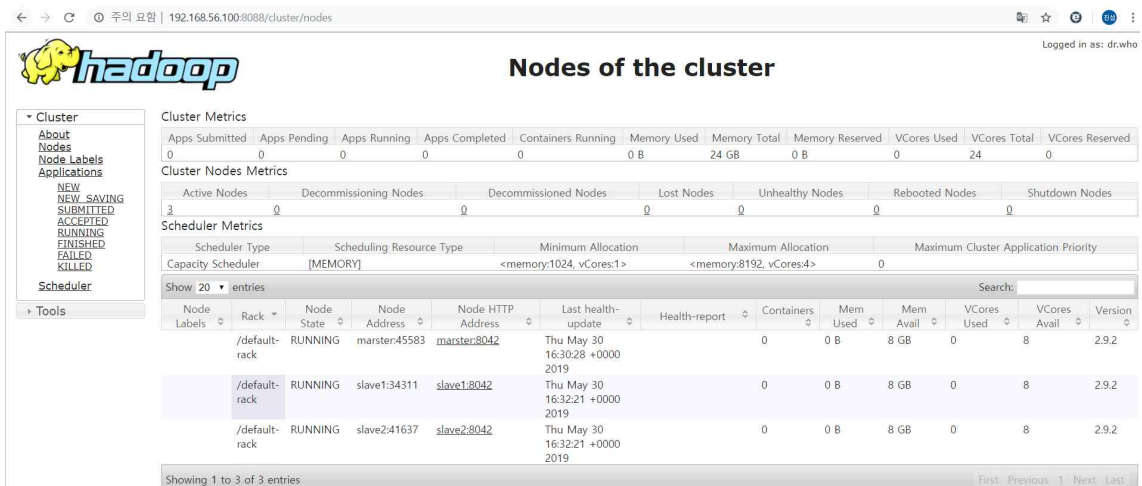
hadoop파일 내에서

\$start-yarn.sh를 실행하여준다.



```
standalone [실행 중] - Oracle VM VirtualBox
파일 머신 보기 입력 장치 도움말
hadoop@marster:~/hadoop$ yarn node -list
19/05/30 16:27:16 INFO client.RMProxy: Connecting to ResourceManager at master/192.168.56.100:8032
Total Nodes:3
Node-Id Node-State Node-Http-Address Number-of-Running-Containers
marster:45583 RUNNING marster:8042 0
slave1:34311 RUNNING slave1:8042 0
slave2:41637 RUNNING slave2:8042 0
hadoop@marster:~/hadoop$
```

\$yarn node -list를 사용하면 현재 각 노드가 Running 상태임을 확인할 수 있다.

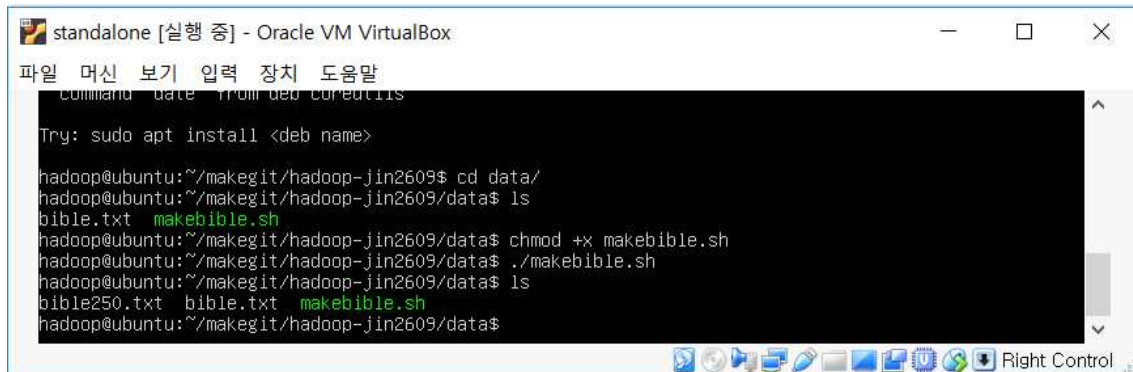


Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail	Version
/default-rack		RUNNING	marster:45583	marster:8042	Thu May 30 16:30:28 +0000 2019		0	0 B	8 GB	0	8	2.9.2
/default-rack		RUNNING	slave1:34311	slave1:8042	Thu May 30 16:32:21 +0000 2019		0	0 B	8 GB	0	8	2.9.2
/default-rack		RUNNING	slave2:41637	slave2:8042	Thu May 30 16:32:21 +0000 2019		0	0 B	8 GB	0	8	2.9.2

192.168.56.100:8088에 접속하면 위처럼 현재 각 노드의 Running 상태를 확인할 수 있다.

[1-b]MapReduce WordCount V1 실행

우선 Git에서 `git clone`을 활용하여 데이터를 가져옵니다. 저는 이때, 직접 MapReduce jar파일을 생성해보는 과정도 수행해보고자 java 코드도 넣어서 클론해 주었습니다. (git에 class파일이 있는 이유)



```
standalone [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
Command: apt-get install <deb name>
Try: sudo apt install <deb name>
hadoop@ubuntu:~/makegit/hadoop-jin2609$ cd data/
hadoop@ubuntu:~/makegit/hadoop-jin2609/data$ ls
bible.txt  makebible.sh
hadoop@ubuntu:~/makegit/hadoop-jin2609/data$ chmod +x makebible.sh
hadoop@ubuntu:~/makegit/hadoop-jin2609/data$ ./makebible.sh
hadoop@ubuntu:~/makegit/hadoop-jin2609/data$ ls
bible250.txt  bible.txt  makebible.sh
hadoop@ubuntu:~/makegit/hadoop-jin2609/data$
```

위는 data파일로 들어가서, bible250.txt를 생성하는 과정입니다. 추후에 standalone 과 3-cluster의 비교를 위해서 수행해 주어야합니다. 분산 처리된 결과로 볼 수 있도록 올려주는 과정입니다.(수행 안하게 되면 하나의 노드에서 처리하게 됨)



```
hadoop@ubuntu:~$ ls
hadoop  hadoop-2.9.2.tar.gz  makegit  spark  spark-2.4.3-bin-hadoop2.7.tgz  wordcountH_classes
hadoop@ubuntu:~$ cd hadoop/
hadoop@ubuntu:~/hadoop$ ls
bin  include  libexec  logs  README.txt  share
etc  lib  LICENSE.txt  NOTICE.txt  stub  WordCount.java
hadoop@ubuntu:~/hadoop$ bin/hadoop com.sun.tools.javac.Main WordCount.java
hadoop@ubuntu:~/hadoop$ ls
bin  lib  logs  stub  'WordCount$IntSumReducer.class'
etc  libexec  NOTICE.txt  share  WordCount.java
include  LICENSE.txt  README.txt  WordCount.class  'WordCount$TokenizerMapper.class'
hadoop@ubuntu:~/hadoop$ jar cf wc.jar WordCount*.class
hadoop@ubuntu:~/hadoop$ ls
bin  libexec  README.txt  WordCount.class
etc  LICENSE.txt  stub  'WordCount$IntSumReducer.class'
include  logs  share  WordCount.java
lib  NOTICE.txt  wc.jar  'WordCount$TokenizerMapper.class'
```

위 작업은 wc.jar파일을 생성하여주는 작업입니다.

```
$hdfs dfs -mkdir /data
```

```
$hdfs dfs -mkdir /data/input
```

```
$hdfs dfs -put bible250.txt /data/input
```

```
$bin/hadoop jar wc.jar WordCount /data/input/bible250.txt /data/output
```

하둡에서 실행을 위해서 위와 같은 명령어를 수행하여줍니다.

mkdir를 활용해 /data 디렉토리를 생성하고, /data에 input 디렉토리를 생성합니다.

그리고 put을 활용하여 bible250.txt를 넣어줍니다.

그 뒤 생성한 jar파일을 활용하여 /data/output에 Map Reduce를 사용해 수행한 결과를 넣어줍니다.

(Word Count v1를 수행한 결과)

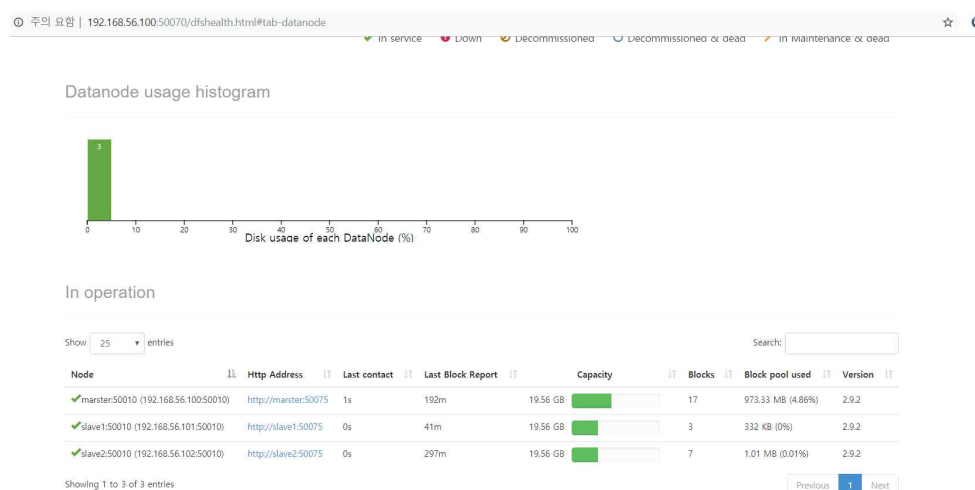
```
hadoop@ubuntu:~/hadoop$ hadoop fs -ls /data/output
Found 2 items
-rw-r--r-- 1 hadoop supergroup          0 2019-05-31 02:45 /data/output/_SUCCESS
-rw-r--r-- 1 hadoop supergroup 365531 2019-05-31 02:45 /data/output/part-r-000000
```

mapreduce가 실행되어 wordcount를 수행하고 생성된 두 개의 파일입니다.

```
young: 250
younger 5500
younger, 1250
younger. 500
younger: 250
younger; 250
youngest 3500
youngest, 500
youngest: 500
your 434000
your's 500
your's. 500
your's: 250
your's; 500
yours 500
yours, 500
yours: 250
yourselves 28500
yourselves, 12750
yourselves. 3000
yourselves: 2250
yourselves; 500
yourselves? 500
youth 3500
youth, 6000
youth. 3750
youth: 1750
youth; 2000
youth? 500
youthful 250
youths 250
youths, 250
zeal 2750
zeal, 750
zealous 2000
zealously 500
```

cat을 사용하여 part-r-000000파일을 읽어본 결과의 일부입니다.

hadoop fs -cat /data/output/part-r-000000를 입력하여 실행하였습니다.



데이터 노드가 어떻게 사용되었는지 확인해 볼 수 있습니다.

<Hadoop History Server> (History server를 통한 Task 확인)

hadoop 디렉토리에서 sbin 디렉토리로 이동합니다.

\$. /mr-jobhistory-daemon.sh start historyserver를 입력하고

192.168.56.100:19888에 접속하여 어떤 노드에서 task가 수행되었나 확인합니다.

주요 요약 | 192.168.56.100:19888/jobhistory/attempts/job_1559298266650_0001/mrSUCCESSFUL
Logged in as: dr.who

hadoop SUCCESSFUL MAP attempts in job_1559298266650_0001										
Show 20 entries										
Attempt	State	Status	Node	Logs	Start Time	Finish Time	Elapsed Time	Note		
attempt_1559298266650_0001_m_000000_1	SUCCEEDED	map > sort	/default-rack/slave2:8042	logs	Fri May 31 19:33:11 +0900 2019	Fri May 31 19:37:17 +0900 2019	4mins, 6sec			
attempt_1559298266650_0001_m_000001_1	SUCCEEDED	map > sort	/default-rack/slave2:8042	logs	Fri May 31 19:32:51 +0900 2019	Fri May 31 19:37:07 +0900 2019	4mins, 16sec			
attempt_1559298266650_0001_m_000002_1	SUCCEEDED	map > sort	/default-rack/slave1:8042	logs	Fri May 31 19:33:24 +0900 2019	Fri May 31 19:37:21 +0900 2019	3mins, 56sec			
attempt_1559298266650_0001_m_000003_1	SUCCEEDED	map > sort	/default-rack/slave1:8042	logs	Fri May 31 19:33:38 +0900 2019	Fri May 31 19:37:31 +0900 2019	3mins, 52sec			
attempt_1559298266650_0001_m_000004_1	SUCCEEDED	map > sort	/default-rack/slave1:8042	logs	Fri May 31 19:34:08 +0900 2019	Fri May 31 19:37:36 +0900 2019	3mins, 28sec			
attempt_1559298266650_0001_m_000005_1	SUCCEEDED	map > sort	/default-rack/slave2:8042	logs	Fri May 31 19:33:55 +0900 2019	Fri May 31 19:37:25 +0900 2019	3mins, 29sec			
attempt_1559298266650_0001_m_000006_0	SUCCEEDED	map > sort	/default-rack/slave2:8042	logs	Fri May 31 19:31:43 +0900 2019	Fri May 31 19:33:48 +0900 2019	2mins, 5sec			
attempt_1559298266650_0001_m_000007_0	SUCCEEDED	map > sort	/default-rack/slave1:8042	logs	Fri May 31 19:31:45 +0900 2019	Fri May 31 19:32:32 +0900 2019	47sec			

Map에 대한 Task는 총 8회 Reduce의 Task는 총 1회가 있는데, Map에 대한 Task는 slave2가 총 4개 slave1이 4개 수행하였음을 알 수 있습니다.

attempt_1559298266650_0001_r_000000_0	SUCCEEDED	reduce > reduce	/default-rack/slave2:8042	logs	Fri May 31 19:32:44 +0900 2019	Fri May 31 19:37:37 +0900 2019	Fri May 31 19:37:38 +0900 2019	Fri May 31 19:37:39 +0900 2019	4mins, 53sec	0sec	1sec	4mins, 55sec
---------------------------------------	-----------	-----------------	---------------------------	------	--------------------------------	--------------------------------	--------------------------------	--------------------------------	--------------	------	------	--------------

Reduce에 대한 Task입니다. slave2가 수행하였음을 확인할 수 있습니다.

attempt_1559298266650_0001_m_000000_0	KILLED		/default-rack/master:8042	logs	Fri May 31 19:31:42 +0900 2019	Fri May 31 19:37:20 +0900 2019	5mins, 37sec	Speculation: attempt_1559298266650_0001_m_000000_1 succeeded first!
attempt_1559298266650_0001_m_000001_0	KILLED		/default-rack/master:8042	logs	Fri May 31 19:31:42 +0900 2019	Fri May 31 19:37:18 +0900 2019	5mins, 36sec	Speculation: attempt_1559298266650_0001_m_000001_1 succeeded first!
attempt_1559298266650_0001_m_000002_0	KILLED		/default-rack/master:8042	logs	Fri May 31 19:31:42 +0900 2019	Fri May 31 19:37:21 +0900 2019	5mins, 39sec	Speculation: attempt_1559298266650_0001_m_000002_1 succeeded first!
attempt_1559298266650_0001_m_000003_0	KILLED		/default-rack/master:8042	logs	Fri May 31 19:31:42 +0900 2019	Fri May 31 19:37:31 +0900 2019	5mins, 48sec	Speculation: attempt_1559298266650_0001_m_000003_1 succeeded first!
attempt_1559298266650_0001_m_000004_0	KILLED		/default-rack/master:8042	logs	Fri May 31 19:31:42 +0900 2019	Fri May 31 19:37:36 +0900 2019	5mins, 54sec	Speculation: attempt_1559298266650_0001_m_000004_1 succeeded first!
attempt_1559298266650_0001_m_000005_0	KILLED		/default-rack/master:8042	logs	Fri May 31 19:31:42 +0900 2019	Fri May 31 19:37:25 +0900 2019	5mins, 43sec	Speculation: attempt_1559298266650_0001_m_000005_1 succeeded first!

Kill을 당한 경우입니다. Master Node도 수행을 하려 하였으나 어떠한 작업이 우선되어야하므로 Kill된 경우를 분석결과 알 수 있었다.

각각의 Task를 클릭하면 어떤 노드가 Task를 수행하였는지 확인이 가능합니다. 각각을 들어가 확인해보면 Kill당한 Master 노드, 그리고 해당 Task를 수행한 노드들을 볼 수 있습니다.

[1-c] WordCount V2

- 문장부호, 접속사를 제외한 Word Count

https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v2.0

해당 링크에 제공되어있는 소스 코드를 git에 올려, clone을 통해 받아서 jar 파일을 생성하도록 하겠습니다.

```
hadoop@master:~/makegit$ git clone https://github.com/cnu-cse/hadoop-jin2609.git
Cloning into 'hadoop-jin2609'...
Username for 'https://github.com': jin2609@cnu.ac.kr
Password for 'https://jin2609@cnu.ac.kr@github.com':
remote: Enumerating objects: 39, done.
remote: Counting objects: 100% (39/39), done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 39 (delta 16), reused 25 (delta 11), pack-reused 0
Unpacking objects: 100% (39/39), done.
hadoop@master:~/makegit$ _
```

git clone을 수행하여줍니다.

cp명령어를 활용하여 hadoop 디렉토리로 wordcount2.java를 옮겨주고 작업을 진행합니다.

```
hadoop@master:~/hadoop$ bin/hadoop com.sun.tools.javac.Main WordCount2.java
hadoop@master:~/hadoop$ ls
bible250.txt  LICENSE.txt  wc.jar
bin           logs        WordCount2.class
etc          NOTICE.txt 'WordCount2$IntSumReducer.class'
include      README.txt  WordCount2.java
lib          sbin       'WordCount2$TokenizerMapper.class'
libexec      share     'WordCount2$TokenizerMapper$CountersEnum.class'
hadoop@master:~/hadoop$ jar cf wc2.jar WordCount2*.class
hadoop@master:~/hadoop$ ls
bible250.txt  libexec      sbin           'WordCount2$IntSumReducer.class'
bin           LICENSE.txt  share         WordCount2.java
etc          logs        wc2.jar       'WordCount2$TokenizerMapper.class'
include      NOTICE.txt wc.jar        'WordCount2$TokenizerMapper$CountersEnum.class'
lib          README.txt  WordCount2.class
```

\$bin/hadoop com.sun.tools.javac.Main WordCount2.java

\$jar cf wc2.jar WordCount2*.class

위 두 가지 명령어를 활용하여 wc2.jar파일을 생성하여줍니다.

```
1 \.
2 \,
3 \!
4 to
```

\$vi pattenrns

patterns 생성하여 줍니다. ignore해 줄 값들을 넣어주는 것입니다.

```
hadoop@master:~/home/hadoop/hadoop$ hdfs dfs -put patterns.txt /data/input
```

ignore해 줄 pattern을 넣어줍니다.

이제 실행을 해줄 준비를 모두 마쳤습니다. bible250.txt는 위에서 넣어준 input을 사용하도록 하겠습니다.

```
hadoop@master:~/hadoop$ bin/hadoop jar wc2.jar WordCount2 -Dwordcount.case.sensitive=true  
e /data/input/bible250.txt /data/outputv2 -skip /data/input/patterns.txt
```

\$bin/hadoop jar wc2.jar WordCount2 -Dwordcount.case.sensitive=true
/data/input/bible250.txt /data/outputv2 -skip /data/input/patterns.txt
위 명령어를 사용하여 Map Reduce를 수행하여줍니다.

(Word Count v2를 수행한 결과)

```
hadoop@master:~/hadoop$ hdfs dfs -ls /data/outputv2  
Found 2 items  
-rw-r--r-- 1 hadoop supergroup 0 2019-05-31 17:38 /data/outputv2/_SUCCESS  
-rw-r--r-- 1 hadoop supergroup 252291 2019-05-31 17:38 /data/outputv2/part-r-00000
```

성공적으로 수행되었음을 확인할 수 있습니다.

```
yonder 1250  
yonder; 250  
you 574750  
you) 500  
you-ward 500  
you-ward: 250  
you: 38250  
you; 21500  
you? 11500  
young 72250  
young: 500  
young; 250  
younger 7250  
younger: 250  
younger; 250  
youngest 4000  
youngest: 500  
your 434000  
your's 1000  
your's: 250  
your's; 500  
yours 1000  
yours: 250  
yourselves 44250  
yourselves: 2250  
yourselves; 500  
yourselves? 500  
youth 13250  
youth: 1750  
youth; 2000  
youth? 500  
youthful 250  
youths 500  
zeal 3500  
zealous 2000  
zealously 500  
hadoop@master:~/hadoop$
```

\$hdfs dfs -cat /data/outputv2/part-r-00000

수행 결과를 cat을 활용하여 가져온 결과입니다. pattern에 쓰여져 있던 문구들이 ignore된 결과를 확인할 수 있습니다.

(History server를 통한 v2 Task 확인)

Attempt	State	Status	Node	Logs	Start Time	Finish Time	Elapsed Time	Note
attempt_1559324976831_0001_m_000000_1	SUCCEEDED	map > sort	/default-rack/slave1:8042	logs	Sat Jun 1 03:04:05 +0900 2019	Sat Jun 1 03:08:06 +0900 2019	4mins, 1sec	
attempt_1559324976831_0001_m_000001_1	SUCCEEDED	map > sort	/default-rack/slave2:8042	logs	Sat Jun 1 03:04:51 +0900 2019	Sat Jun 1 03:08:31 +0900 2019	3mins, 39sec	
attempt_1559324976831_0001_m_000002_1	SUCCEEDED	map > sort	/default-rack/slave2:8042	logs	Sat Jun 1 03:04:39 +0900 2019	Sat Jun 1 03:08:27 +0900 2019	3mins, 47sec	
attempt_1559324976831_0001_m_000003_1	SUCCEEDED	map > sort	/default-rack/slave2:8042	logs	Sat Jun 1 03:03:36 +0900 2019	Sat Jun 1 03:06:38 +0900 2019	3mins, 1sec	
attempt_1559324976831_0001_m_000004_1	SUCCEEDED	map > sort	/default-rack/slave1:8042	logs	Sat Jun 1 03:04:20 +0900 2019	Sat Jun 1 03:08:44 +0900 2019	4mins, 24sec	
attempt_1559324976831_0001_m_000005_0	SUCCEEDED	map > sort	/default-rack/master:8042	logs	Sat Jun 1 02:52:10 +0900 2019	Sat Jun 1 03:08:50 +0900 2019	16mins, 39sec	
attempt_1559324976831_0001_m_000006_0	SUCCEEDED	map > sort	/default-rack/master:8042	logs	Sat Jun 1 02:52:10 +0900 2019	Sat Jun 1 03:09:01 +0900 2019	16mins, 50sec	
attempt_1559324976831_0001_m_000007_0	SUCCEEDED	map > sort	/default-rack/master:8042	logs	Sat Jun 1 02:52:10 +0900 2019	Sat Jun 1 03:03:33 +0900 2019	11mins, 22sec	
Attempt	State	Status	Node	Logs	Start Time	Finish Time	Elapsed Time	Note

Map에 대한 Task는 총 8개이다. Task들은 각각 slave1이 2개 slave2가 3개 master가 3개를 수행하였다.

Attempt	State	Status	Node	Logs	Start Time	Shuffle Finish Time	Merge Finish Time	Finish Time	Elapsed Time Shuffle	Elapsed Time Merge	Elapsed Time Reduce	Elapsed Time	Note
attempt_1559324976831_0001_r_000000_0	SUCCEEDED	reduce > reduce	/default-rack/slave1:8042	logs	Sat Jun 1 03:03:36 +0900 2019	Sat Jun 1 03:09:01 +0900 2019	Sat Jun 1 03:09:01 +0900 2019	Sat Jun 1 03:09:02 +0900 2019	5mins, 25sec	0sec	1sec	5mins, 26sec	
Attempt	State	Status	Node	Logs	Start Time	Shuffle Fin	Merge Fin	Finish Time	Elapsed S	Elapsed M	Elapsed R	Elapsed Ti	Note

Reduce에 대한 Task는 1개로 Slave1이 수행하였음을 확인할 수 있다.

Attempt	State	Status	Node	Logs	Start Time	Finish Time	Elapsed Time	Note
attempt_1559324976831_0001_m_000000_0	KILLED		/default-rack/master:8042	logs	Sat Jun 1 02:52:10 +0900 2019	Sat Jun 1 03:08:06 +0900 2019	15mins, 55sec	Speculation: attempt_1559324976831_0001_m_000000_1 succeeded first!
attempt_1559324976831_0001_m_000001_0	KILLED		/default-rack/master:8042	logs	Sat Jun 1 02:52:10 +0900 2019	Sat Jun 1 03:08:31 +0900 2019	16mins, 20sec	Speculation: attempt_1559324976831_0001_m_000001_1 succeeded first!
attempt_1559324976831_0001_m_000002_0	KILLED		/default-rack/master:8042	logs	Sat Jun 1 02:52:10 +0900 2019	Sat Jun 1 03:08:27 +0900 2019	16mins, 17sec	Speculation: attempt_1559324976831_0001_m_000002_1 succeeded first!
attempt_1559324976831_0001_m_000003_0	KILLED		/default-rack/master:8042	logs	Sat Jun 1 02:52:10 +0900 2019	Sat Jun 1 03:06:44 +0900 2019	14mins, 33sec	Speculation: attempt_1559324976831_0001_m_000003_1 succeeded first!
attempt_1559324976831_0001_m_000004_0	KILLED		/default-rack/master:8042	logs	Sat Jun 1 02:52:10 +0900 2019	Sat Jun 1 03:08:44 +0900 2019	16mins, 33sec	Speculation: attempt_1559324976831_0001_m_000004_1 succeeded first!
attempt_1559324976831_0001_m_000005_1	KILLED		/default-rack/slave1:8042	logs	Sat Jun 1 03:05:06 +0900 2019	Sat Jun 1 03:08:50 +0900 2019	3mins, 43sec	Speculation: attempt_1559324976831_0001_m_000005_0 succeeded first!
attempt_1559324976831_0001_m_000006_1	KILLED		/default-rack/master:8042	logs	Sat Jun 1 03:03:50 +0900 2019	Sat Jun 1 03:09:01 +0900 2019	5mins, 10sec	Speculation: attempt_1559324976831_0001_m_000006_0 succeeded first!
Attempt	State	Status	Node	Logs	Start Time	Finish Time	Elapsed Time	Note

Kill을 당한 결과를 확인할 수 있다. master와 slave1이 succeeded first! 문제를 일으키며 kill을 당한 log를 볼 수 있다.

Task를 클릭하면 어떤노드가 Task를 수행하였는지 확인이 가능합니다. 각각을 확인해보면 Kill당한 Master 노드, 그리고 작업을 수행한 노드들을 볼 수 있습니다.

<성능 비교 standalone vs 3-cluster>

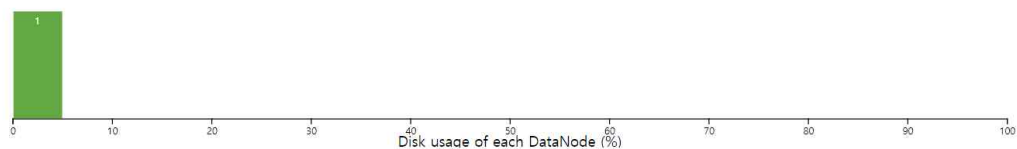
```
11         addresses:
12         - 192.168.56.105/24
13         nameservers:
14         addresses:
15         - 8.8.8.8
16         version: 2

"/etc/netplan/50-cloud-init.yaml" 16L, 532C written
hadoop@ubuntu:~$ sudo netplan apply
```

stand alone을 생성하고, IP를 위와 같이 설정하여준다.

그리고 Pseudo-Distributed Operation을 세팅하여준다.

Datanode usage histogram



In operation

Show entries

Search:

Node	Http Address	Last contact	Last Block Report	Capacity	Blocks	Block pool used	Version
✓ ubuntu:50010 (192.168.56.105:50010)	http://ubuntu:50075	2s	6m	19.56 GB	0	24 KB (0%)	2.9.2

Showing 1 to 1 of 1 entries

Previous **1** Next

Standalone의 HDFS의 세팅 결과이다.

Cluster Metrics										
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
0	0	0	0	0	0 B	8 GB	0 B	0	8	0

Cluster Nodes Metrics							
Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes	
1	0	0	0	0	0	0	

Scheduler Metrics					
Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority	
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0	

Show entries

Search:

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail	Version
/default-rack		RUNNING	ubuntu:42829	ubuntu:8042	Fri May 31 13:00:29 +0000 2019		0	0 B	8 GB	0	8	2.9.2

Showing 1 to 1 of 1 entries

First Previous **1** Next Last

위처럼 세팅을 완료하여주고 Map Reduce를 수행하여준다.

*Git에서 가져와서 jar파일을 생성하는 부분은 생략합니다. 성능 비교만 수행하도록 하겠습니다.

hoop

SUCCESSFUL MAP attempts in job_1559307372353_0002

Logged in as:

Attempt	State	Status	Node	Logs	Start Time	Finish Time	Elapsed Time	Note
attempt_1559307372353_0002_m_000000_0	SUCCEEDED	map > sort	/default-rack/ubunturack:8042	logs	Fri May 31 22:34:54 +0900 2019	Fri May 31 22:53:30 +0900 2019	18mins, 35sec	
attempt_1559307372353_0002_m_000001_0	SUCCEEDED	map > sort	/default-rack/ubunturack:8042	logs	Fri May 31 22:34:54 +0900 2019	Fri May 31 22:53:36 +0900 2019	18mins, 42sec	
attempt_1559307372353_0002_m_000002_0	SUCCEEDED	map > sort	/default-rack/ubunturack:8042	logs	Fri May 31 22:34:54 +0900 2019	Fri May 31 22:53:41 +0900 2019	18mins, 47sec	
attempt_1559307372353_0002_m_000003_0	SUCCEEDED	map > sort	/default-rack/ubunturack:8042	logs	Fri May 31 22:34:54 +0900 2019	Fri May 31 22:53:29 +0900 2019	18mins, 35sec	
attempt_1559307372353_0002_m_000004_0	SUCCEEDED	map > sort	/default-rack/ubunturack:8042	logs	Fri May 31 22:34:54 +0900 2019	Fri May 31 22:53:31 +0900 2019	18mins, 36sec	
attempt_1559307372353_0002_m_000005_0	SUCCEEDED	map > sort	/default-rack/ubunturack:8042	logs	Fri May 31 22:34:54 +0900 2019	Fri May 31 22:53:31 +0900 2019	18mins, 37sec	
attempt_1559307372353_0002_m_000006_0	SUCCEEDED	map > sort	/default-rack/ubunturack:8042	logs	Fri May 31 22:53:32 +0900 2019	Fri May 31 22:56:58 +0900 2019	3mins, 26sec	
attempt_1559307372353_0002_m_000007_0	SUCCEEDED	map > sort	/default-rack/ubunturack:8042	logs	Fri May 31 22:53:32 +0900 2019	Fri May 31 22:56:15 +0900 2019	2mins, 43sec	

2개의 실행 결과 중 2번째 실행한 것의 Map에 대한 Task에 대하여만 보여드리겠습니다. (standalone으로 실행되어지는지 확인하기 위하여)

보여지는 결과와 마찬가지로 하나의 노드에서 모든 작업을 수행하는 standalone임을 알 수 있습니다.

(Standalone, 3-Cluster 속도 비교)

*성능 비교는 history의 Elapsed를 활용하여 진행하도록 하겠습니다.

MapReduce Job job_1559298266650_0001

Job Name:	word count
User Name:	hadoop
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri May 31 10:31:27 UTC 2019
Started:	Fri May 31 10:31:39 UTC 2019
Finished:	Fri May 31 10:37:39 UTC 2019
Elapsed:	5mins, 59sec
Diagnostics:	
Average Map Time	3mins, 15sec
Average Shuffle Time	4mins, 53sec
Average Merge Time	0sec
Average Reduce Time	1sec

위 결과는 3-cluster에서 WordCount V1을 활용하여 WordCount를 수행한 결과입니다. 성능은 elapsed를 확인하면 알 수 있습니다. 약 5분 59초가 걸림을 확인 할 수 있습니다.

MapReduce Job job_1559307372353_0001

Job Name:	word count
User Name:	hadoop
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri May 31 13:13:41 UTC 2019
Started:	Fri May 31 13:13:55 UTC 2019
Finished:	Fri May 31 13:34:10 UTC 2019
Elapsed:	20mins, 14sec
Diagnostics:	
Average Map Time	13mins, 16sec
Average Shuffle Time	3mins, 25sec
Average Merge Time	0sec
Average Reduce Time	1sec

위 결과는 standalone에서 WordCount V1을 활용하여 WordCount를 수행한 결과입니다. 성능은 약 20분 14초가 걸림을 알 수 있습니다.

성능은 3-cluster가 훨씬 좋음을 확인할 수 있습니다.

(컴퓨터의 상태에 따라서 Elapsed가 달라질 수 있어서, 극심한 결과가 도출되었습니다.)

Job Overview	
Job Name:	word count
User Name:	hadoop
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri May 31 17:51:53 UTC 2019
Started:	Fri May 31 17:52:07 UTC 2019
Finished:	Fri May 31 18:09:03 UTC 2019
Elapsed:	16mins, 55sec
Diagnostics:	
Average Map Time	7mins, 58sec
Average Shuffle Time	5mins, 25sec
Average Merge Time	0sec
Average Reduce Time	1sec

위 결과는 3-cluster에서 WordCount V2를 활용하여 WordCount를 수행한 결과입니다. 성능은 elapsed를 확인하면 알 수 있습니다. 약 16분 55초가 걸림을 확인할 수 있습니다. (컴퓨터의 메모리가 8GB인데, 4개의 가상환경을 돌리다보니, 다운되어 버리는 문제가 발생하여, 재 측정하였습니다)

MapReduce Job job_1559307372353_0002

Job Name:	word count
User Name:	hadoop
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri May 31 13:34:39 UTC 2019
Started:	Fri May 31 13:34:51 UTC 2019
Finished:	Fri May 31 13:57:01 UTC 2019
Elapsed:	22mins, 10sec
Diagnostics:	
Average Map Time	14mins, 45sec
Average Shuffle Time	3mins, 26sec
Average Merge Time	0sec
Average Reduce Time	1sec

위 결과는 standalone에서 WordCount V2를 활용하여 WordCount를 수행한 결과입니다. 성능은 약 22분 10초가 걸림을 알 수 있습니다.

역시 동일하게 성능은 3-cluster가 Standalone보다 좋음을 확인할 수 있습니다.

[과제2]SPARK 설치

[2-a]모든 노드가 SPARK와 연결된 상태

spark디렉토리에 들어가 수행한다.

```
$cd conf
```

```
$vi slaves
```

```
19 slave1
20 slave2
21 master
```

만약 slaves가 없으면 mv명령어로 template을 slaves로 변경하여주고 실행한다.

master, slave1, slave2 (hosts에 있는 값들)을 넣어준다.


다시 spark 디렉토리로 접속한다.

```
$cd sbin/
```

```
$./start-master.sh
```

```
$./start-slaves.sh 를 수행하여 spark를 실행한다.
```

(3개의 Worker 동작)

 **Spark Master at spark://master:7077**

URL: spark://master:7077
Alive Workers: 3
Cores in use: 3 Total, 0 Used
Memory in use: 3.0 GB Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory
worker-20190601084604-192.168.56.100-38751	192.168.56.100:38751	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20190601090136-192.168.56.101-37003	192.168.56.101:37003	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20190601090136-192.168.56.102-42849	192.168.56.102:42849	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

192.168.56.100:8080에 접속하여 3개의 노드가 Spark와 연결된 상태를 확인할 수 있다. 모든 노드가 Alive Worker임을 알 수 있다.

[2-b] Spark Quick Start

우선 수행에 앞서서, hdfs에 사용할 bible250.txt를 넣어 두어야합니다.

```
$hdfs dfs -mkdir /data
```

```
$hdfs dfs -mkdir /data/input
```

```
$hdfs dfs -put bible250.txt /data/input
```

위 과정을 수행하여 bible250.txt파일을 hdfs에 넣어줍니다.

```
1 import time
2 from pyspark import SparkContext
3
4 sc = SparkContext('spark://master:7077')
5 starttime = time.time()
6 textFile = sc.textFile('hdfs://master:9000/data/input/bible250.txt')
7 print('words :{0}'.format(textFile.count()))
8 print('processing time: {0}s'.format(time.time()-starttime))
```

```
$vi WordCount.py
```

실행을 위하여 파이썬 스크립트를 작성하여줍니다.

SparkContext를 'spark://master:7077로 세팅하여 줌으로써, 3개의 Worker 노드로 동작시켜주기 위해 master를 바꾸어주는 과정이다.

starttime을 사용하여 측정을 측정의 시작시간을 체크하여준다.

textFile을 sc.textFile을 활용하여 읽어주는데,

'hdfs://master:9000/data/input/bible250.txt'에서 읽어주는 즉 hdfs에서 데이터를 가져오는 방식을 취해준다.

master:9000으로 시작하는 이유는,

```
19 <configuration>
20   <property>
21     <name>fs.defaultFS</name>
22     <value>hdfs://master:9000</value>
23   </property>
24 </configuration>
```

하둡을 설정할 때 fs.defaultFS를 (core-site.xml) 위와 같이 설정하였기 때문이다.

따라서, hdfs에 넣어둔 파일을 가져올 수 있다.

그리고 count()를 수행하여주고 얼마나 오래 걸렸는지 processing time을 계산하여 보여준다.

*과제 수행시, master에서 수행을 할 때, 계속해서 error가 났었는데, 그 원인이 바로 master를 local에서 원격 cluster로 변경하게 되면 파일을 열어볼 권한이 없을 수 있다는 것으로 추측하였습니다. 따라서, hdfs를 활용하여 파일을 읽을 수 있도록 설정하여 과제를 해결해 주었습니다.

(PySpark를 이용한 wordcount-v1 활용)

이제 실행을 하여줍니다.

```
hadoop@master:~/spark$ ./bin/spark-submit WordCount.py
```

위처럼 spark-submit을 활용하여 수행하여줍니다.

```
19/06/01 12:52:45 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 16290 ms on 192.168.56.101 (executor 2) (1/8)
19/06/01 12:52:45 INFO PythonAccumulatorV2: Connected to AccumulatorServer at host: 127.0.0.1 port: 58207
19/06/01 12:52:45 INFO TaskSetManager: Starting task 4.0 in stage 0.0 (TID 4, 192.168.56.102, executor 0, partition 4, ANY, 7906 bytes)
19/06/01 12:52:45 INFO TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in 16356 ms on 192.168.56.102 (executor 0) (2/8)
19/06/01 12:52:49 INFO TaskSetManager: Starting task 5.0 in stage 0.0 (TID 5, 192.168.56.100, executor 1, partition 5, ANY, 7906 bytes)
19/06/01 12:52:49 INFO TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 18756 ms on 192.168.56.100 (executor 1) (3/8)
```

wordcount가 진행되고 있는 과정입니다. local에서 하나의 worker로 수행하게 되면 약 30개정도를 수행하게 되는데, 지금은 위처럼 8개만 수행함을 알 수 있습니다.

```
words :7595750
processing time: 43.3030149937s
```

print로 출력된 결과입니다.

약 43초가 걸렸음을 확인할 수 있습니다.

The screenshot shows the Spark Master web interface at `spark://master:7077`. It displays the following information:

- URL:** spark://master:7077
- Alive Workers:** 3
- Cores in use:** 3 Total, 0 Used
- Memory in use:** 3.0 GB Total, 0.0 B Used
- Applications:** 0 Running, 5 Completed
- Drivers:** 0 Running, 0 Completed
- Status:** ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory
worker-20190601124151-192.168.56.101-41553	192.168.56.101:41553	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20190601124151-192.168.56.102-41349	192.168.56.102:41349	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20190601124155-192.168.56.100-41015	192.168.56.100:41015	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	----------------	------	-------	----------

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20190601131430-0004	WordCount.py	3	1024.0 MB	2019/06/01 13:14:30	hadoop	FINISHED	45 s

192.168.56.100:8080에 접속하면 수행한 Application을 확인할 수 있습니다.

확인 결과 정상적으로 3개의 worker로 동작하였음을 알 수 있습니다.

<PySpark를 이용한 경우와 Hadoop Node의 속도 비교>

(PySpark를 이용한 경우와 Hadoop Node의 속도비교)

MapReduce Job job_1559298266650_0001

Job Name:	word count
User Name:	hadoop
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri May 31 10:31:27 UTC 2019
Started:	Fri May 31 10:31:39 UTC 2019
Finished:	Fri May 31 10:37:39 UTC 2019
Elapsed:	5mins, 59sec
Diagnostics:	
Average Map Time	3mins, 15sec
Average Shuffle Time	4mins, 53sec
Average Merge Time	0sec
Average Reduce Time	1sec

Map Reduce Word Count를 수행해 주는데 약 6분이 걸렸음을 알 수있다.

```
words :7595750
processing time: 43.3030149937s
```

Spark를 이용한 경우 43초 정도로 Spark가 훨씬 빠름을 알 수 있다.

속도 비교 : Spark > MapReduce

그 이유는 데이터 프로세싱 방법의 차이가 있기 때문이다. 하둡은 단계별 데이터 처리방식을 취하는 맵 리듀스를 활용하지만 스파크는 전체 데이터 셋을 한 번에 다루기 때문에 훨씬 빠르다고 합니다.

추가적으로 궁금한 점이 들어서 검색을 통해 알아 보았습니다.

Map Reduce의 실행 과정은, 클러스터에서 데이터를 읽어낸 뒤, 동작을 실행하고, 결과를 클러스터에 기록한 다음, 또 다시 업데이트된 데이터를 클러스터로부터 읽어내고 다음 동작을 실행한 후 결과를 클러스터에 입력하는 방법이고, 스파크는 모든 데이터 운영을 메모리 내에서 실시간에 가깝게 처리할 수 있게 클러스터로부터 데이터를 읽어 들이고 필요한 모든 과정을 수행하고 결과물을 클러스터에 입력하는 전 과정이 동시에 진행되는 것 이라고 한다.