TP 1

CAO Anh Quan
M2 D&K Big Data Processing

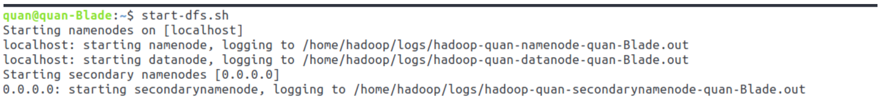
September 27, 2018

1 Submission

I submitted the lab using Github link but Dr. Jacob Montiel told me that we cannot submit using Github. Therefore, I deleted my Github repository and resubmit the lab using Dropbox.

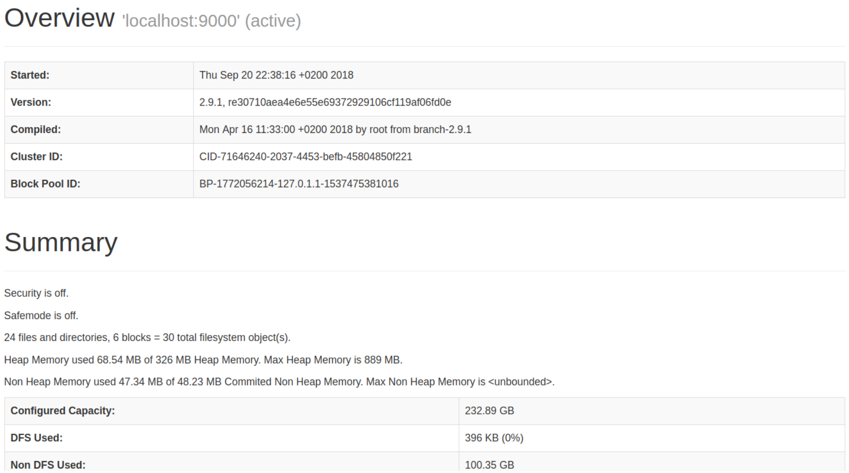
2 Intall Hadoop with Pseudo-Distributed Mode

In this tp, I successfully install Hadoop in Pseudo-Distributed Operation mode and YARN. Following is the images of HDFS and YARN.



```
quan@quan-Blade:~$ start-dfs.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/hadoop/logs/hadoop-quan-namenode-quan-Blade.out
localhost: starting datanode, logging to /home/hadoop/logs/hadoop-quan-datanode-quan-Blade.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/hadoop/logs/hadoop-quan-secondarynamenode-quan-Blade.out
```

Figure 1: Run the start-dfs.sh script to run the HDFS



Overview 'localhost:9000' (active)

Started:	Thu Sep 20 22:38:16 +0200 2018
Version:	2.9.1, re30710aea4e6e55e69372929106cf119af06fd0e
Compiled:	Mon Apr 16 11:33:00 +0200 2018 by root from branch-2.9.1
Cluster ID:	CID-71646240-2037-4453-befb-45804850f221
Block Pool ID:	BP-1772056214-127.0.1.1-1537475381016

Summary

Security is off.

Safemode is off.

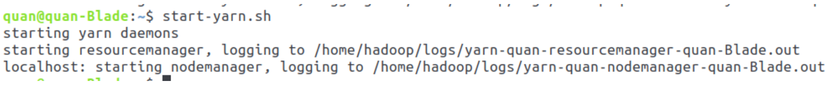
24 files and directories, 6 blocks = 30 total filesystem object(s).

Heap Memory used 68.54 MB of 326 MB Heap Memory. Max Heap Memory is 889 MB.

Non Heap Memory used 47.34 MB of 48.23 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

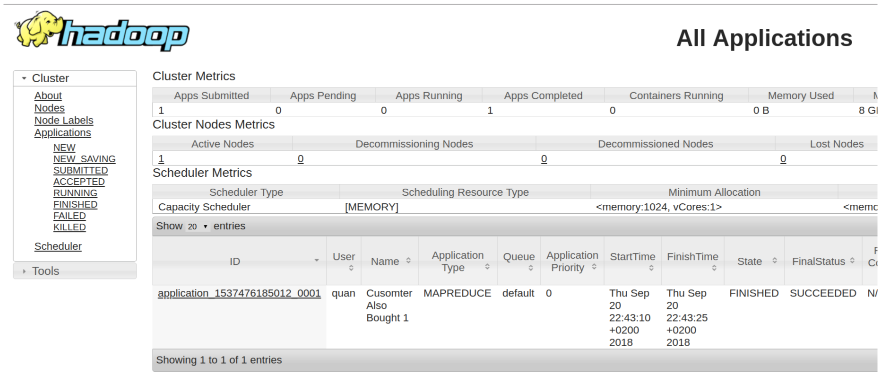
Configured Capacity:	232.89 GB
DFS Used:	396 KB (0%)
Non DFS Used:	100.35 GB

Figure 2: HDFS's web interface



```
quan@quan-Blade:~$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /home/hadoop/logs/yarn-quan-resourcemanager-quan-Blade.out
localhost: starting nodemanager, logging to /home/hadoop/logs/yarn-quan-nodemanager-quan-Blade.out
```

Figure 3: Run the start-yarn.sh script to run the YARN



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	M
1	0	0	1	0	0 B	8 GI

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes
1	0	0	0

Scheduler Metrics

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

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	F Co
application_1537476185012_0001	quan	Cusomter Also Bought 1	MAPREDUCE	default	0	Thu Sep 20 22:43:10 +0200 2018	Thu Sep 20 22:43:25 +0200 2018	FINISHED	SUCCEEDED	N/

Showing 1 to 1 of 1 entries

Figure 4: YARN's web interface

3 Solving "Customers who bought this item also bought" exercise using Hadoop

3.1 MapReduce Solution

I implement the exercise with **2 solutions**.

3.1.1 Solution 1

I output all the pairs of items in the mapper. Then, in the reducer, I output each item as the key and the value is the number of each item bought with an item in the key.

Input and output example for Mapper, Reducer and Combiner

- **Mapper:**
  - Input

    ```
    book12  book34  cd12
    book32  book34  dvd32
    book32  book34
    ```

  - Output

    ```
    <book12 ,  book34> <book34 ,  book12> <book12 ,  cd12> <cd12 ,  book12>
    <book34 ,  cd12> <cd12 ,  book34>

    <book32 ,  book34> <book34 ,  book32> <book32 ,  dvd32> <dvd32 ,  book32>
    <book34 ,  dvd32> <dvd32 ,  book34>
    ```

```
<book32 , book34> <book34 , book32>
```

- **Reducer:**

  - Input

    ```
    <book12 , {book34 , cd12}>
    <book34 , {book12 , cd12 , book32 , dvd32 , book32}>
    <book32 , {book34 , dvd32 , book34}>
    <cd12 , {book12 , book34}>
    <dvd32 , {book32 , book34}>
    ```

  - Output:

    ```
    book12 (book34 , 1) (cd12 , 1)
    book34 (book32 , 2) (book12 , 1) (cd12 , 1)  (dvd32 , 1)
    book32 (book34 , 2) (dvd32 , 1)
    cd12 (book12 , 1) (book34 , 1)
    dvd32 (book32 , 1) (book34 , 1)
    ```

- **Combiner:** (In the version with combiner) I output as the string ”book32,1 book34,1” so I can parse it in the reducer.

  Because *combiner* must have the **same input and output data type** with the **output data type** of the *mapper*, I have to format the output data of the combiner to the same output datatype of the mapper.

  - Input

    ```
    <book12 , {book34 , cd12}>
    <book34 , {book12 , cd12 , book32 , dvd32 , book32}>
    <book32 , {book34 , dvd32 , book34}>
    <cd12 , {book12 , book34}>
    <dvd32 , {book32 , book34}>
    ```

  - Output:

    ```
    <book12 , ”book34 ,1  cd12 ,1” >
    <book34 , ”book32 ,2  book12 ,1  cd12 ,1  dvd32 ,1” >
    <book32 , ”book34 ,2  dvd32 ,1” >
    <cd12 , ”book12 ,1  book34 ,1” >
    <dvd32 , ”book32 ,1  book34 ,1” >
    ```

3.1.2 Solution 1 implementation in Hadoop MapReduce

- Hadoop application with *Mapper*, *Reducer*:

  - **Map function**:

```
function MAP(docid a, doc d)
    items $\leftarrow$ split d into list of item
    for all item1 $\in$ items do
        for all item2 $\in$items do
            if item1 $! =$ item2 then
                Emit(item1, item2)
            end if
        end for
    end for
```

- **Reducer function**:

```
function REDUCE(item item, list of item items)
    map $\leftarrow$ put items into a map of key-value pairs: <item, number of occurences>
    SortedMap $\leftarrow$ Sort the map by value (number of occurents) in descending order
    Emit(item, SortedMap)
```

- Hadoop application with *Mapper*, *Combiner* and *Reducer*
  - **Mapper** is similar.
  - **Combiner** is the Reducer function without sorting the *map* and the output of *Combiner* is formated as "book34,1 cd12,1"
  - **Reducer** Reducer is almost similar. I have to parse the value from *Combiner*

3.1.3 Solution 2

In this solution, in the **mapper**, instead of emitting the pairs of all items, I emit the key as an item, and the value is the **map** contains the items and number of that item which are bought together with the key item.

The **Reducer** and **Combiner** only need to take the **map** from **Mapper** and combines them together.

Input and output example for Mapper, Reducer and Combiner

- **Mapper:**
  - Input

```
book12 book34 cd12 book34
book32 book34 dvd32
book32 book34
```

  - Output

```
book12 (book34 , 2) (cd12 , 1)
book34 (cd12 , 1) (book12 , 1)
book34 (cd12 , 1) (book12 , 1)
cd12 (book34 , 1) (book12 , 1)
```

```
book32 (book34, 1) (dvd32, 1)
book34 (book32, 1) (dvd32, 1)
dvd32 (book34, 1) (book32, 1)

book32 (book34, 1)
book34 (book32, 1)
```

- **Reducer:**
  - Input

    ```
    <book12, {(book34, 2), (cd12, 1)}>
    <book34, {(book12, 1), (cd12, 1), (book32, 1),
    (dvd32, 1), (book32, 1), (cd12, 1), (book12, 1)}>

    <book32, {(book34, 1), (dvd32, 1), (book34, 1)}>
    <cd12, {(book12, 1), (book34, 1)}>
    <dvd32, {(book34, 1) (book32, 1)}>
    ```

  - Output:

    ```
    book12 (book34, 2) (cd12, 1)
    book34 (book32, 2) (book12, 2) (cd12, 2)  (dvd32, 1)
    book32 (book34, 2) (dvd32, 1)
    cd12 (book12, 1) (book34, 1)
    dvd32 (book32, 1) (book34, 1)
    ```

- **Combiner:** The combiner in this version is the same as Reducer.

3.1.4 Solution 2 implementation in Hadoop MapReduce

- Hadoop application with *Mapper*, *Reducer*:

  - **Map function**:

    ```
    function MAP(docid a, doc d)
        items ← split d into list of item
        for all item ∈ items do
            map ← put items in the same transaction that are bought together with item into a map of key-value pairs: <item, number of occurences>
            Emit(item, map)
        end for
    ```

  - **Reducer function**:

    ```
    function REDUCE(item item, list of maps maps)
        map ← Sum all the maps by key <item, number of occurences>
    ```

```
    SortedMap ← Sort the map by value (number of occurents) in descending order
    Emit(item, SortedMap)
```

- **Combiner:** is similar as the *Reducer*

3.2 Code files structure

- folder **src**: contains the Java codes
  - *Solution1CABCombiner.java*: Hadoop MapReduce implementation with *Combiner* of **Solution 1**
  - *Solution1CABNoCombiner.java*: Hadoop MapReduce implementation without *Combiner* of **Solution 1**
  - *Solution2CAB.java*: Hadoop MapReduce implementation with *Combiner* of **Solution 2**
  - *RandomData.java*: Generate *input* data for the hadoop program
- folder **test result**: Contains some example results of the Hadoop application including the input from slide and one random generated input.
- file **cab.jar**: jar file for submitting to the hadoop cluster.

3.3 How to run the program

1. Run the DFS (Distributed File System) and YARN:

```
start-dfs.sh
start-yarn.sh
```

2. Put the input file into the HDFS:

```
hdfs dfs -put input input
```

3. Run the Hadoop application **Solution 1 without Combiner**

```
hadoop jar cab.jar Solution1CABNoCombiner input output
```

4. Run the Hadoop application **Solution 1 with Combiner**

```
hadoop jar cab.jar Solution1CABCombiner input output
```

5. Run the Hadoop application **Solution 2**

```
hadoop jar cab.jar Solution2CAB input output
```

6. Download the output from HDFS:

```
hdfs dfs -get output
```

3.4 Result

Some test results:

1. Input from slides:

   - input:

     ```
     book12 book34 cd12 cd42 dvd32
     book32 book34 dvd32
     ```

   - output:

     ```
     book12    (cd42, 1) (book34, 1) (dvd32, 1) (cd12, 1)
     book32    (book34, 1) (dvd32, 1)
     book34    (dvd32, 2) (cd42, 1) (book12, 1) (book32, 1) (cd12, 1)
     cd12    (book34, 1) (cd42, 1) (book12, 1) (dvd32, 1)
     cd42    (book34, 1) (book12, 1) (dvd32, 1) (cd12, 1)
     dvd32    (book34, 2) (cd42, 1) (book12, 1) (book32, 1) (cd12, 1)
     ```

2. A random generated input:

   - input:

     ```
     book0 dvd3 hat4 car1 cd2 cake5 hat4
     car1 book0 cd2 cake5 car1 car1 hat4
     hat4 hat4 cake5 dvd3 car1 car1
     hat4 cd2 cd2 cd2 cake5 hat4 book0
     dvd3 dvd3 hat4 car1 dvd3
     hat4 cake5 cd2 hat4 cake5 cd2 hat4
     cd2 car1 cake5 car1 car1
     hat4 cd2 hat4 cake5 car1 cake5 cd2
     dvd3 cake5 cake5 cd2 dvd3
     cd2 car1 car1 cake5 car1
     hat4 car1 car1 book0 car1 cd2 dvd3 cake5
     hat4 cd2 hat4 cd2 cake5
     hat4 dvd3 car1 cake5 cd2 car1 cake5
     cd2 cake5 hat4 cake5 cd2
     cd2 car1 dvd3 hat4 hat4 hat4 hat4
     ```

   - output:

     ```
     book0    (car1, 7) (cd2, 6) (hat4, 6) (cake5, 4) (dvd3, 2)
     cake5    (cd2, 26) (hat4, 24) (car1, 21) (dvd3, 9) (book0, 4)
     car1    (hat4, 21) (cake5, 21) (cd2, 18) (dvd3, 12) (book0, 7)
     cd2    (hat4, 31) (cake5, 26) (car1, 18) (dvd3, 6) (book0, 6)
     dvd3    (hat4, 13) (car1, 12) (cake5, 9) (cd2, 6) (book0, 2)
     hat4    (cd2, 31) (cake5, 24) (car1, 21) (dvd3, 13) (book0, 6)
     ```