Hadoop and Hive

Submission

Due Date: 12 May

For this practical, there are 4 assessed elements.

- Question on page 5
- Task on Page 8
- Task on Page 11
- Question on Page 11

The files created by the tasks on pages 8 and 11 must be uploaded to blackboard and shown to your tutor to be graded upon the section. The answers to the questions on pages 5 and 11 must be answered in class to your tutor.

Learning objectives:

1. Learn the principle of MapReduce;
2. Learn how to use Hive;
3. Learn how to use Hadoop;
4. Get familiar with basic *nix command lines.

Notation in this practical:

- $HIVE_HOME: the home directory of your Hive project, or known as Hive home
- #: for comment
- *nix means: linux/unix
- $: the starting point of the console/terminal
- Be careful about the case-sensitive when you type *nix command

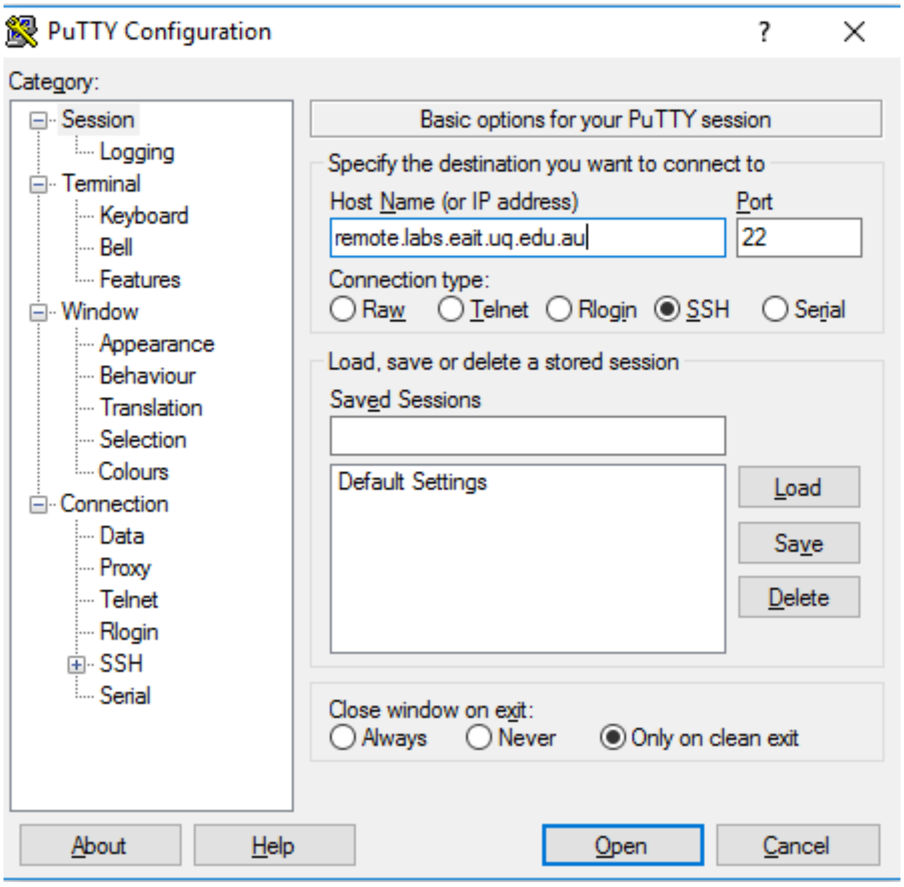
Useful Linux Commands:

• cd aaa	go to folder aaa
• cd ..	go back to the upper folder
• cd ~	go to your home folder
○ where your Desktop,Document,Download folders are	
• pwd	show the full directory address of the current folder
• ls	list the files in the current folder
• cat aaa	show the content in file aaa
• vim aaa	open the file aaa (this command is optional)
○ a	append new content
○ esc then :q	exit
○ esc then :wq	save and exit
• “Tab”	auto complete the current file name

Introduction

Hive is a Hadoop-based data warehouse tool that enables easy data summarization, ad-hoc querying and analysis of large volumes of data. Hive defines a simple SQL-like query language, called HQL, that enables users familiar with SQL to query the data.[1]

In our lab, we can access the ITEE moss server via putty using SSH. Putty is located on the start menu for lab computers, or is available for free download on personal devices. Note that Putty is a Windows only tool, however MacOS computers already have an SSH client preinstalled, please ask your tutor for assistance if you have any issues.


PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): remote.labs.eait.uq.edu.au

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

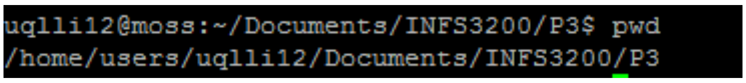
Saved Sessions: Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

Fig 1

Type in the host name of Moss Server addr: remote.labs.eait.uq.edu.au
Then press Open.
Then type your student number and password, it may allow you to login. (student number should start with s, e.g. s4000888)

After login, you can see the contents on your desktop(they share the same files). You can use the commands above to find the places that you put your codes and data, and run pwd to find out the actual location on MOSS.



```
uqlli12@moss:~/Documents/INFS3200/P3$ pwd
/home/users/uqlli12/Documents/INFS3200/P3
```

Fig 2

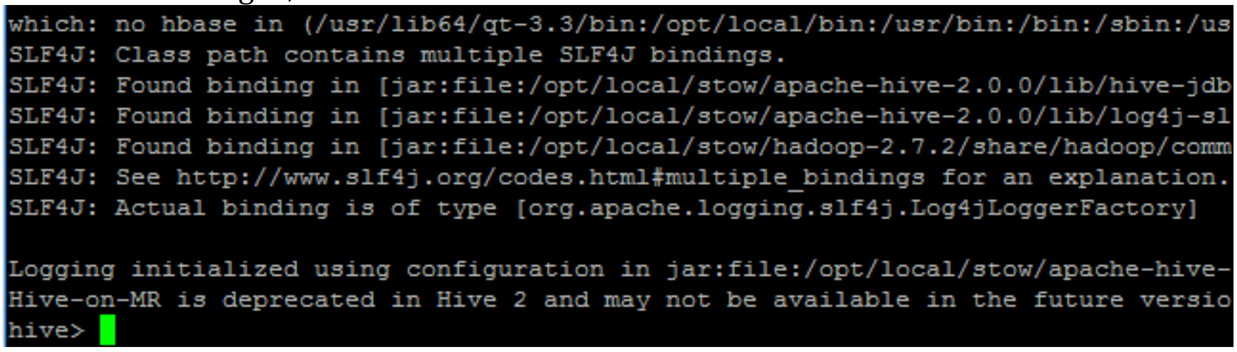
Task 1 Hive

1. Run hive

Hive should be running before we do any operations. First, go to the hive folder:

```
cd /opt/local/stow/apache-hive-2.0.0/bin
./hive
```

After a successful log in, the command line looks like:



```
which: no hbase in (/usr/lib64/qt-3.3/bin:/opt/local/bin:/usr/bin:/bin:/sbin:/us
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/local/stow/apache-hive-2.0.0/lib/hive-jdb
SLF4J: Found binding in [jar:file:/opt/local/stow/apache-hive-2.0.0/lib/log4j-sl
SLF4J: Found binding in [jar:file:/opt/local/stow/hadoop-2.7.2/share/hadoop/comm
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in jar:file:/opt/local/stow/apache-hive-
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versio
hive>
```

2. Create a database in Hive.

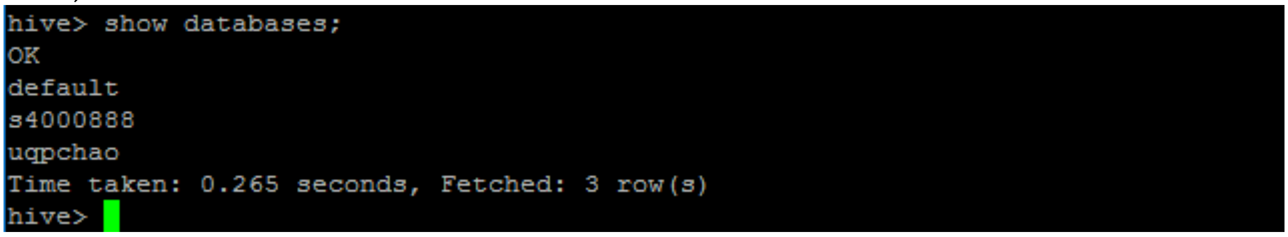
Run the following command to display the current database you are using.

```
set hive.cli.print.current.db=true;
```

Before starting your tasks, you need to create your own workspace, which is called "database" in Hive. Use your student number (s4000888 for example) to create a new database:

```
create database s4000888;
```

Then, check whether it is created:



```
hive> show databases;
OK
default
s4000888
uqpchao
Time taken: 0.265 seconds, Fetched: 3 row(s)
hive>
```

Now, you can start to use your database:

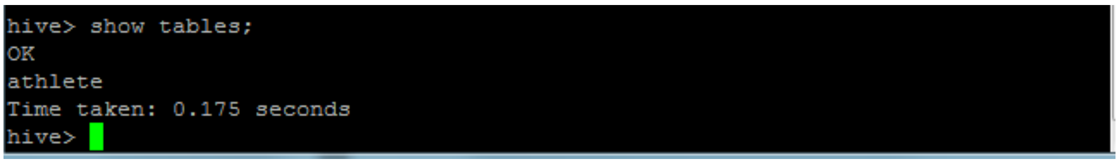
```
use s4000888;
```

3. Create a table in Hive.

you need to create a table under your database in Hive with the following command line:

```
create table athlete (
  AthleteID int,
  FirstName string,
  LastName string,
  DOB string,
  Gender string,
  Country string)
  row format delimited
  Fields terminated by ','
  Stored as textfile;
```

After the table is created, check whether it is successfully built using the command line as shown in Figure 3.



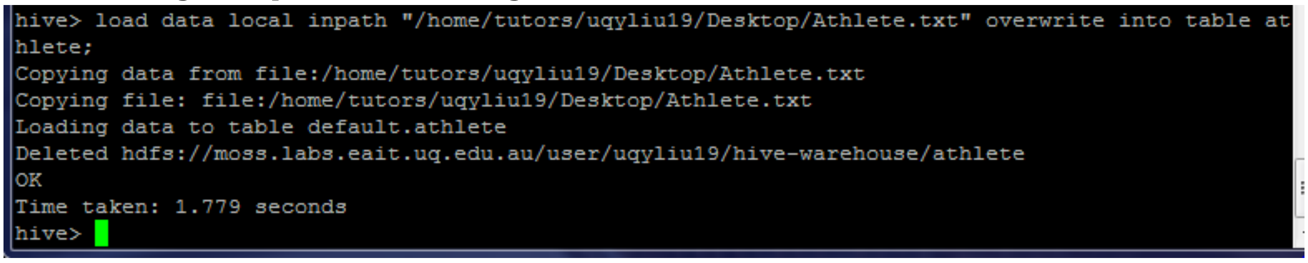
```
hive> show tables;
OK
athlete
Time taken: 0.175 seconds
hive>
```

Fig 3

As long as you have created a table in Hive, you need to import the data from an external file named Athletes.txt, which is provided in the package this practical. Using the following command to import Athletes.txt to Hive:

```
load data local inpath "your-Athletes.txt-location" overwrite into table athlete;
```

The running example is shown in Figure 4



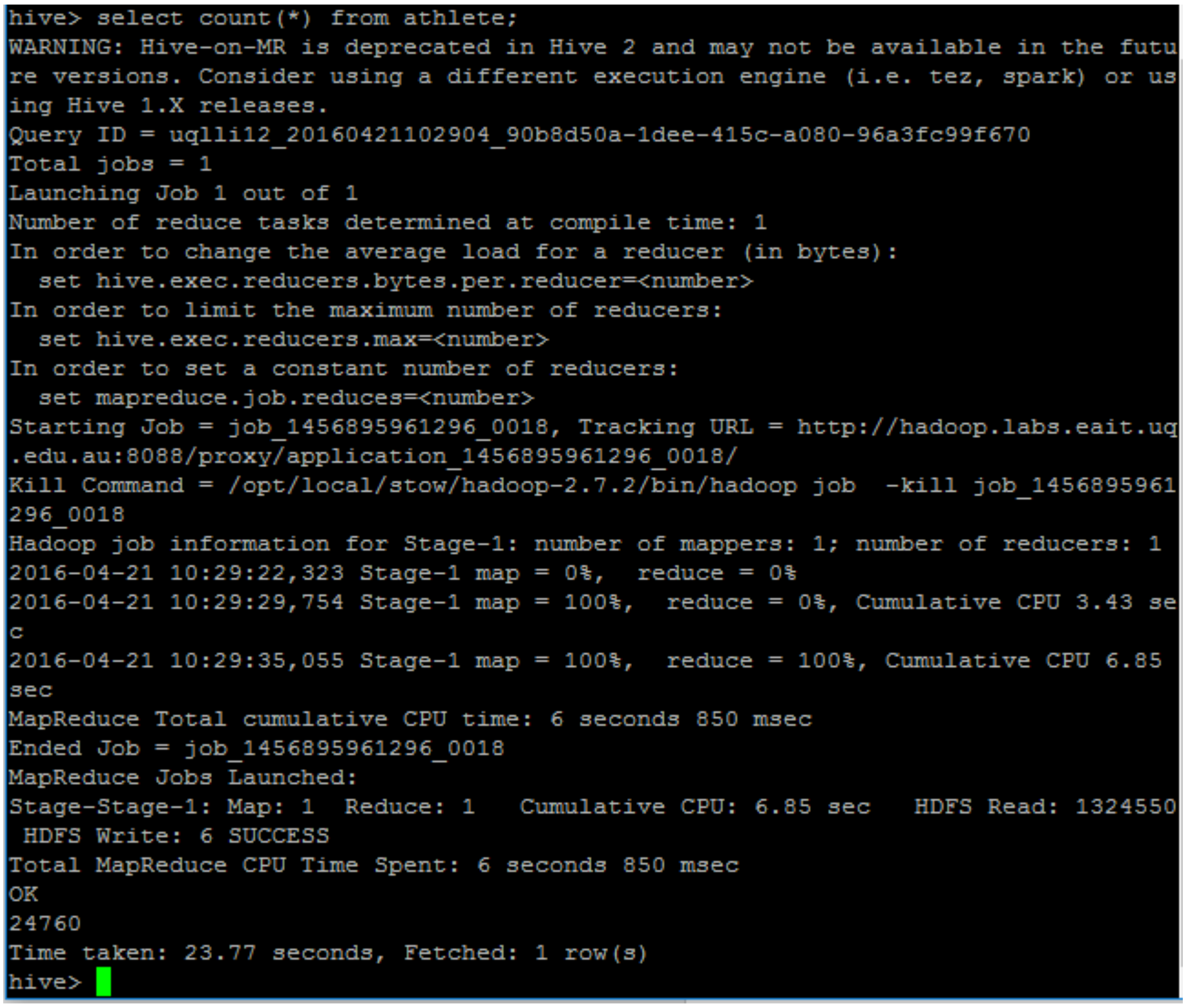
```
hive> load data local inpath "/home/tutors/uqyliu19/Desktop/Athlete.txt" overwrite into table at
hlete;
Copying data from file:/home/tutors/uqyliu19/Desktop/Athlete.txt
Copying file: file:/home/tutors/uqyliu19/Desktop/Athlete.txt
Loading data to table default.athlete
Deleted hdfs://moss.labs.eait.uq.edu.au/user/uqyliu19/hive-warehouse/athlete
OK
Time taken: 1.779 seconds
hive>
```

Fig 4

4. Count the number of rows in table athlete:

```
Select count(*) from athlete;
```

Hive will use MapReduce to get the number of the rows. See Figure 5.



```
hive> select count(*) from athlete;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the futu
re versions. Consider using a different execution engine (i.e. tez, spark) or us
ing Hive 1.X releases.
Query ID = uqlli12_20160421102904_90b8d50a-1dee-415c-a080-96a3fc99f670
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1456895961296_0018, Tracking URL = http://hadoop.labs.eait.uq
.edu.au:8088/proxy/application_1456895961296_0018/
Kill Command = /opt/local/stow/hadoop-2.7.2/bin/hadoop job  -kill job_1456895961
296_0018
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2016-04-21 10:29:22,323 Stage-1 map = 0%,  reduce = 0%
2016-04-21 10:29:29,754 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.43 se
c
2016-04-21 10:29:35,055 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.85
sec
MapReduce Total cumulative CPU time: 6 seconds 850 msec
Ended Job = job_1456895961296_0018
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.85 sec   HDFS Read: 1324550
 HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 850 msec
OK
24760
Time taken: 23.77 seconds, Fetched: 1 row(s)
hive>
```

Fig 5

QUESTION:

In this query there was only 1 mapper and 1 reducer used for the entire counting process, which means that there was no parallelism used. Can a counting job like this be distributed across several nodes and in what scenario would this distribution occur?

5. Quit Hive

quit;

Task 2 Hadoop

Some useful Hadoop commands:

- HDFS:
    - hadoop fs -ls list the files in HDFS
    - hadoop fs -rm aaa remove file aaa
    - hadoop fs -mv aaa bbb move file aaa to bbb
    - hadoop fs -put src dst put file to HDFS from src to dst
    - hadoop fs -get src dst get file from HDFS from src to dst
    - hadoop fs -cat aaa show file aaa's content
- Compile the Java Code
    - hadoop com.sun.tools.javac.Main Your_Code.java
- Wrap a jar:
    - jar cf wc.jar Your_Code*.class
- Run a job:
    - hadoop jar Your_Program.jar YourCode input output

2.1 Inverted List

In computer science, an inverted index (also referred to as postings file or inverted file) is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents (named in contrast to a Forward Index, which maps from documents to content). The purpose of an inverted index is to allow fast full text searches, at a cost of increased processing when a document is added to the database. The inverted file may be the database file itself, rather than its index. It is the most popular data structure used in document retrieval systems.

Figure 6 shows an example of how to computed an inverted list in Hadoop using MapReduce paradigm. Each mapper is responsible for an input document. It splits the document into words and output an intermediate pair <word, documentID>. For example, <cat, doc1.txt> is one of the output of mapper 1. In the shuffle phase, the intermediate data are sent to different reducers based on the key, i.e., the word itself. For example, all the intermediate pair with key "cat"(<cat, doc1.txt>,<cat,doc2.txt>) are all sent to reducer 1, while all the intermediate pair with key "sat" are all sent to reducer 2. After the reduce receives all the intermediate data, it organized them into the output pair like based on the intermediate keys <cat： doc1.txt,doc2.txt>.

The code file is invertedIndex.java file. Let's run this example step by step.

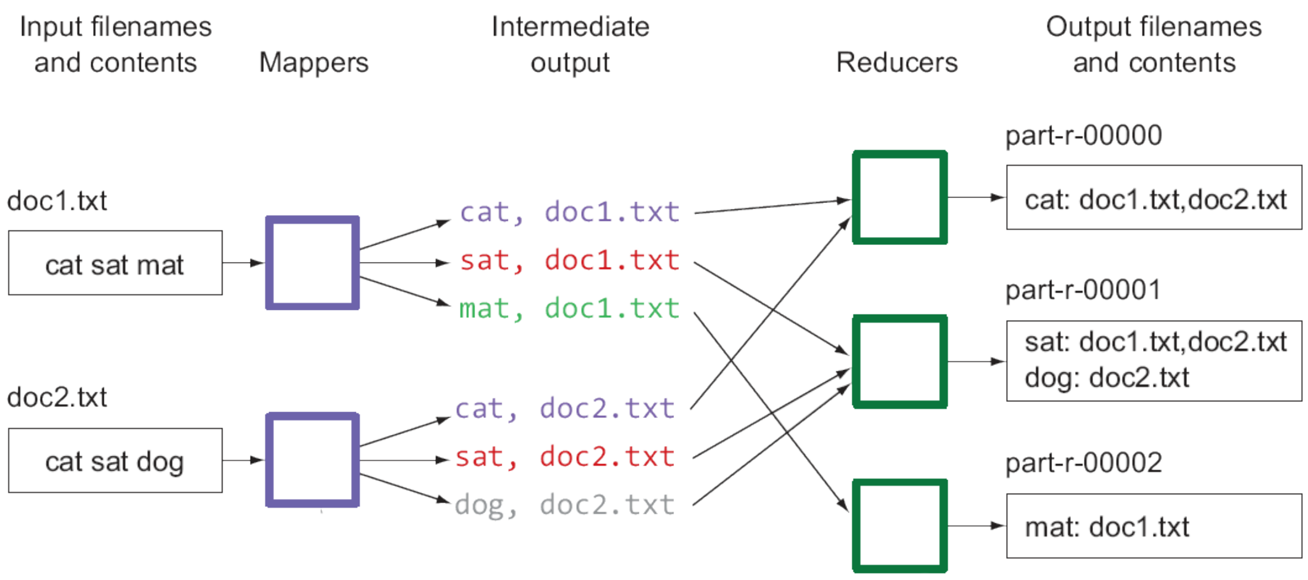


Fig 6

Steps:

1. **Set up environment**

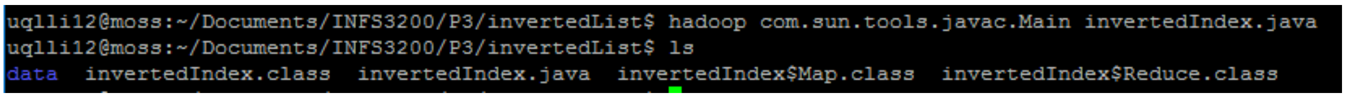
   Run command: module load hadoop/2.7.2

2. **Compile the Java source code**

   Run command: hadoop com.sun.tools.javac.Main invertedIndex.java

   This will produce three class files:

   invertedIndex.class, invertedIndex$Map.class, invertedIndex$Reduce.class



```
uqlli12@moss:~/Documents/INFS3200/P3/invertedList$ hadoop com.sun.tools.javac.Main invertedIndex.java
uqlli12@moss:~/Documents/INFS3200/P3/invertedList$ ls
data  invertedIndex.class  invertedIndex.java  invertedIndex$Map.class  invertedIndex$Reduce.class
```

Fig 7

3. **Wrap it into a jar package**

   Run command: jar cf i.jar invertedIndex*.class

   This will produce a jar package i.jar

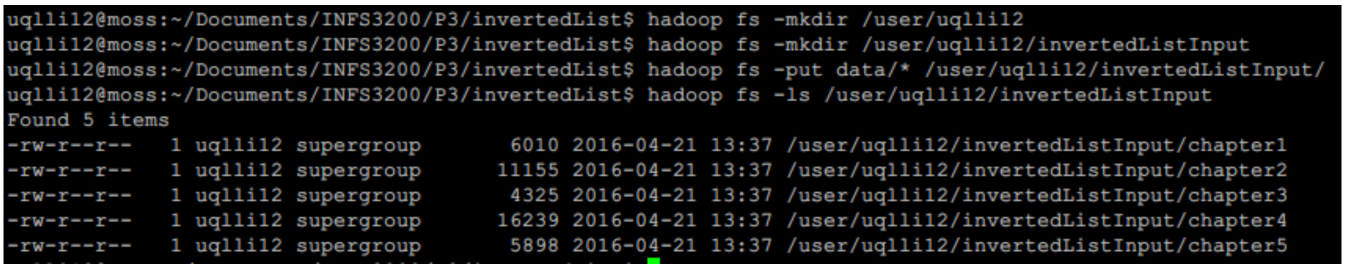
4. **Make an input folder in HDFS**

   Run command: hadoop fs -mkdir -p /user/your-student-number/invertedListInput

   This will make a folder in HDFS: /user/your-student-number/invertedListInput

5. **Put the documents in the input folder**

   Run command: hadoop fs -put chapter* /user/your-student-number/invertedListInput/



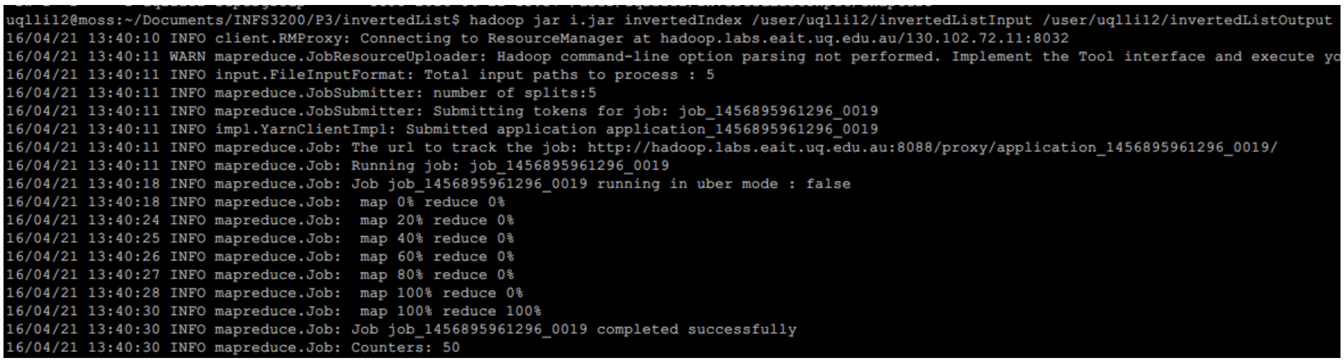
```
uqllii2@moss:~/Documents/INFS3200/P3/invertedList$ hadoop fs -mkdir /user/uqllii2
uqllii2@moss:~/Documents/INFS3200/P3/invertedList$ hadoop fs -mkdir /user/uqllii2/invertedListInput
uqllii2@moss:~/Documents/INFS3200/P3/invertedList$ hadoop fs -put data/* /user/uqllii2/invertedListInput/
uqllii2@moss:~/Documents/INFS3200/P3/invertedList$ hadoop fs -ls /user/uqllii2/invertedListInput
Found 5 items
-rw-r--r--   1 uqllii2 supergroup       6010 2016-04-21 13:37 /user/uqllii2/invertedListInput/chapter1
-rw-r--r--   1 uqllii2 supergroup      11155 2016-04-21 13:37 /user/uqllii2/invertedListInput/chapter2
-rw-r--r--   1 uqllii2 supergroup       4325 2016-04-21 13:37 /user/uqllii2/invertedListInput/chapter3
-rw-r--r--   1 uqllii2 supergroup      16239 2016-04-21 13:37 /user/uqllii2/invertedListInput/chapter4
-rw-r--r--   1 uqllii2 supergroup       5898 2016-04-21 13:37 /user/uqllii2/invertedListInput/chapter5
```

Fig 8

6. **Run the jar**

   Run command: hadoop jar i.jar invertedIndex /user/y-s-n/invertedListInput /user/y-s-n/invertedListOutput

   If this step fails, it may because the output folder has already exist. So make sure the output folder does not exist before running this command. Just use a new output folder name or you can delete the existing one by command "hadoop fs -rm -r output" to remove a folder named "invertedListOutput" in HDFS.



```
uqllii2@moss:~/Documents/INFS3200/P3/invertedList$ hadoop jar i.jar invertedIndex /user/uqllii2/invertedListInput /user/uqllii2/invertedListOutput
16/04/21 13:40:10 INFO client.RMProxy: Connecting to ResourceManager at hadoop.labs.eait.uq.edu.au/130.102.72.11:8032
16/04/21 13:40:11 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute yo
16/04/21 13:40:11 INFO input.FileInputFormat: Total input paths to process : 5
16/04/21 13:40:11 INFO mapreduce.JobSubmitter: number of splits:5
16/04/21 13:40:11 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1456895961296_0019
16/04/21 13:40:11 INFO impl.YarnClientImpl: Submitted application application_1456895961296_0019
16/04/21 13:40:11 INFO mapreduce.Job: The url to track the job: http://hadoop.labs.eait.uq.edu.au:8088/proxy/application_1456895961296_0019/
16/04/21 13:40:11 INFO mapreduce.Job: Running job: job_1456895961296_0019
16/04/21 13:40:18 INFO mapreduce.Job: Job job_1456895961296_0019 running in uber mode : false
16/04/21 13:40:18 INFO mapreduce.Job:  map 0% reduce 0%
16/04/21 13:40:24 INFO mapreduce.Job:  map 20% reduce 0%
16/04/21 13:40:25 INFO mapreduce.Job:  map 40% reduce 0%
16/04/21 13:40:26 INFO mapreduce.Job:  map 60% reduce 0%
16/04/21 13:40:27 INFO mapreduce.Job:  map 80% reduce 0%
16/04/21 13:40:28 INFO mapreduce.Job:  map 100% reduce 0%
16/04/21 13:40:30 INFO mapreduce.Job:  map 100% reduce 100%
16/04/21 13:40:30 INFO mapreduce.Job: Job job_1456895961296_0019 completed successfully
16/04/21 13:40:30 INFO mapreduce.Job: Counters: 50
```

Fig 9

7. **Retrieve the output from HDFS**

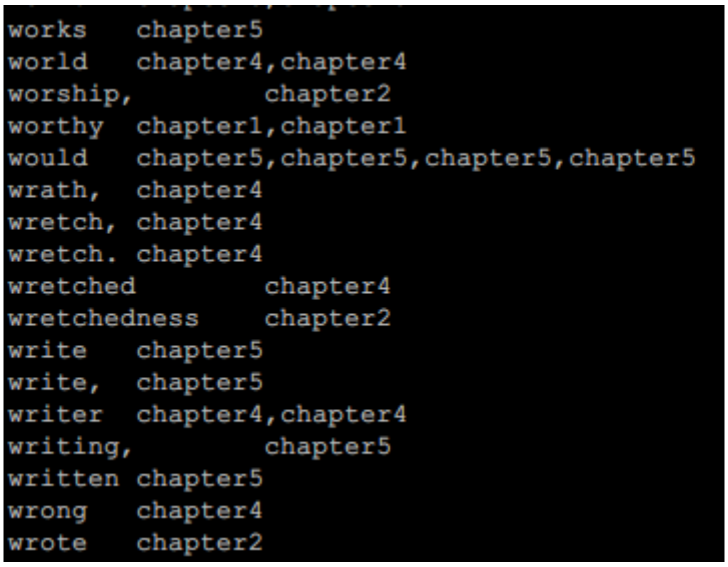
   Run command: hadoop fs -get /user/y-s-n/invertedListOutput/* ./

   Then you can open the output (named as part-r-00000) in your local operating system.

   Run command: cat part-r-00000
   The result is shown in Fig 10.

SUBMISSION:

Show the part-r-00000 file to your tutor and upload it to Blackboard. You may need to rename it to something else before uploading.



```
works   chapter5
world   chapter4,chapter4
worship,        chapter2
worthy  chapter1,chapter1
would   chapter5,chapter5,chapter5,chapter5
wrath,  chapter4
wretch, chapter4
wretch. chapter4
wretched        chapter4
wretchedness    chapter2
write   chapter5
write,  chapter5
writer  chapter4,chapter4
writing,        chapter5
written chapter5
wrong   chapter4
wrote   chapter2
```

Fig 10

2.2 Matrix Multiplication

Multiplication of two matrices is defined if and only if the number of columns of the left matrix is the same as the number of rows of the right matrix. If A is an m-by-n matrix and B is an n-by-p matrix, then their matrix product AB is the m-by-p matrix whose entries are given by dot product of the corresponding row of A and the corresponding column of B:

$$[\mathbf{AB}]_{i,j} = A_{i,1}B_{1,j} + A_{i,2}B_{2,j} + \cdots + A_{i,n}B_{n,j} = \sum_{r=1}^{n} A_{i,r}B_{r,j}$$

where $1 \le i \le m$ and $1 \le j \le p$.

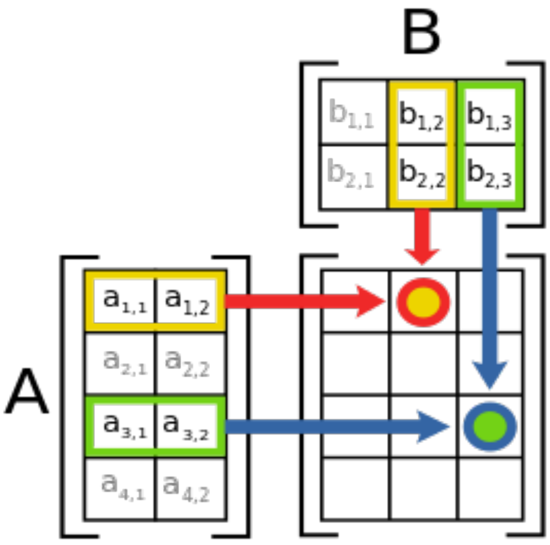


Fig 11

For example, the underlined entry 2340 in the product is calculated as $(2 \times 1000) + (3 \times 100) + (4 \times 10) = 2340$:

$$\begin{bmatrix} \underline{2} & \underline{3} & \underline{4} \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & \underline{1000} \\ 1 & \underline{100} \\ 0 & \underline{10} \end{bmatrix} = \begin{bmatrix} 3 & \underline{2340} \\ 0 & 1000 \end{bmatrix}.$$

Figure 12 shows an example of how to run matrix multiplication in Hadoop using MapReduce.

The mapper is responsible for emitting the corresponding values from A and B to AB. For example, A[0][0] will be used in AB[0][0] and AB[0][1], so the mapper should send A[0][0] to all A[0]. The intermediate key-value pairs mapped from A[0][0] are ((0,0), A[0][0]) and ((0,1),A[0][0]). The keys (0,0) and (0,1) denotes the positions in the result matrix AB. Similarly in Matrix B, for example the value B[2][1] should be emitted to AB[1], so the intermediate key-value pairs should be ((0,1),B[2][1]),((1,1),B[2][1]) and ((2,1),B[2][1]). The mapper's logical function is shown below:

- For each (i, j) of A, emit($(i, k), A[i, j]$,) for $1 \leq k \leq p$
- For each (j, k) of B, emit($(j, k), B[j, k]$,) for $1 \leq i \leq m$

The reducer is responsible for collecting the intermediate data sent from the mappers. Each value in the result has one reducer. So in our example, it should have 6 reducers. Reducer [0][0] will compute the values of AB[0][0] with the intermediate data with the key (0,0). In this case, is should be ((0,0),A[0][0]), ((0,0),A[0][1]), ((0,0),A[0][2]), ((0,0),A[0][3]), ((0,0),A[0][4]) and ((0,0),B[0][0]), ((0,0),B[1][0]), ((0,0),B[2][0]), ((0,0),B[3][0]), ((0,0),B[4][0]). Then it computes the result using the corresponding values: A[0][0] * B[0][0] + A[0][1] * B[1][0] + A[0][2] * B[2][0] + A[0][3] * B[3][0] + A[0][4] * B[4][0]. So the reducer's logical function is shown below:

- Key= (i, k)
- Value=$Sum_j(A[i, j] * B[j, k])$

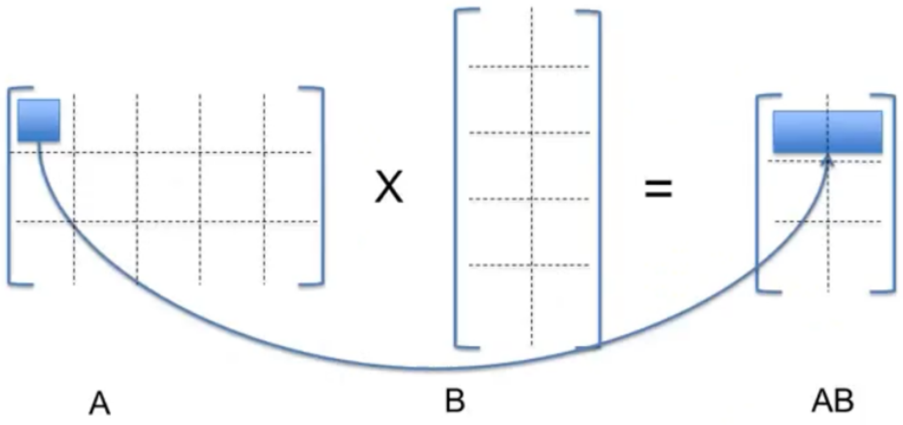


Fig 12

Steps: first go the folder that contains the matrix source code and data

1. **Compile the Java Source code**

   Run command: hadoop com.sun.tools.javac.Main -Xlint:deprecation Matrix.java

This will produce three class files:

Matrix.class, Matrix$Map.class, Matrix$Reduce.class

2. **Wrap it into a jar package**

   Run command: `jar cf m.jar Matrix*.class`

   This will produce a jar package m.jar

3. **Make an input folder in HDFS**

   Run command: `hadoop fs -mkdir –p /user/your-student-number/matrixInput`

   This will make a folder in HDFS: /user/your-student-number/matrixInput

4. **Put the documents in the input folder**

   Run command:

   ```
   hadoop fs -put m /user/your-student-number/matrixInput/
   ```

5. **Run the jar**

   Run command:

   ```
   hadoop jar m.jar Matrix /user/y-s-n/matrixInput /user/y-s-n/matrixOutput
   ```

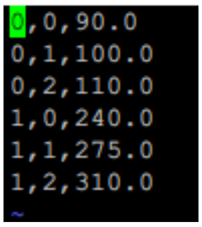
6. **Retrieve the output from HDFS**

   Run command: `hadoop fs -get /user/y-s-n/matrixOutput/* ./`

   Then you can open the output (named as part-r-00000) in your local operating system.

   Run command: `cat part-r-00000`

   The result is shown in Fig 13



```
0,0,90.0
0,1,100.0
0,2,110.0
1,0,240.0
1,1,275.0
1,2,310.0
~
```

Fig 13

SUBMISSION:

Show the part-r-00000 file to your tutor and upload it to Blackboard. You may need to rename it to something else before uploading.

QUESTION:

In the prior 2 tasks, the execution time was lengthy for the execution. Explain how on a larger dataset the execution times for a MapReduce algorithm could be considered to be more acceptable. Some things to be considered in your answer are the use of parallelization and a comparing the start time to the execution time.