School of Information Technologies
Faculty of Engineering & IT

ASSIGNMENT/PROJECT COVERSHEET - GROUP ASSESSMENT

Unit of Study: COMP5349 Cloud Computing

Assignment name: **Hadoop MapReduce Programming**

Tutorial time: **Thurday 16：00-17:00** **Tutor name:** **Andrian**

DECLARATION
We the undersigned declare that we have read and understood the *University of Sydney Academic Dishonesty and Plagiarism in Coursework Policy*, an, and except where specifically acknowledged, the work contained in this assignment/project is our own work, and has not been copied from other sources or been previously submitted for award or assessment.

We understand that failure to comply with the *Academic Dishonesty and Plagiarism in Coursework Policy* can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

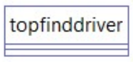
We realise that we may be asked to identify those portions of the work contributed by each of us and required to demonstrate our individual knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Project team members				
Student name	**Student ID**	**Participated**	**Agree to share**	**Signature**
1.Wensi Tang	450489549	Yes	Yes	汤文思
2.Geng Yuan	450621024	Yes	Yes	袁庚
3.		Yes / No	Yes / No	
4.		Yes / No	Yes / No	
5.		Yes / No	Yes / No	
6.		Yes / No	Yes / No	
7.		Yes / No	Yes / No	
8.		Yes / No	Yes / No	
9.		Yes / No	Yes / No	
10.		Yes / No	Yes / No	

Task1：

Code structure:

Two map-reduce were used in task 1.
The Driver of the two map-reduce is called topfinddriver.

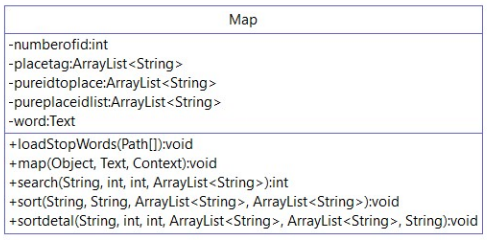


The Driver of the first map-reduce is called Driver.

1. Map function has two function. First is joining the place and input file. Second is filter useless information (including name and year tags) and reorganize useful information.

   Join function based on override about setup() function in map. It reads place.txt into memory of Map task sorts and saves it in ArrayList of String (Compared with hash map ArrayList are better at save large information. Although, for this 40MB file it does not matter, it would be easier to upgrade).

   The output of Map will be like this:

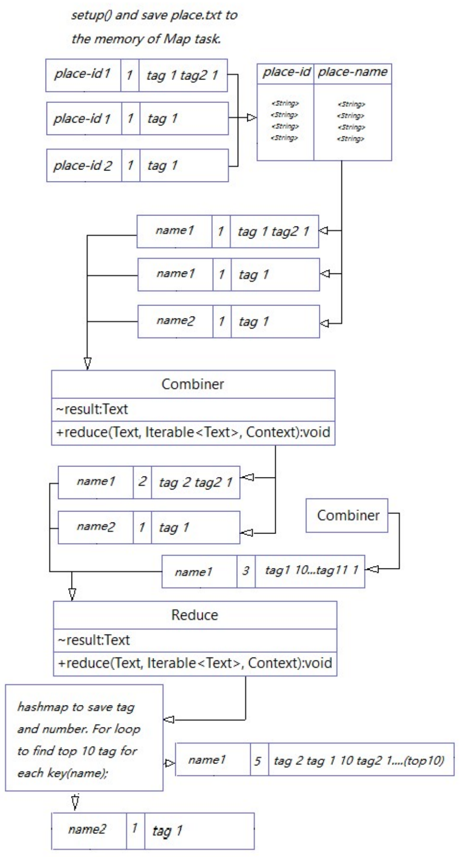

Map
-numberofid:int
-placetag:ArrayList<String>
-pureidtoplace:ArrayList<String>
-pureplaceidlist:ArrayList<String>
-word:Text
+loadStopWords(Path[]):void
+map(Object, Text, Context):void
+search(String, int, int, ArrayList<String>):int
+sort(String, String, ArrayList<String>, ArrayList<String>):void
+sortdetal(String, int, int, ArrayList<String>, ArrayList<String>, String):void

Key (/country/ locality)   value (1 tag 1   tag2 1…)

3. Combiner will combine value by key.

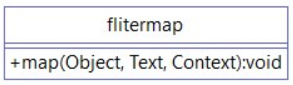
Key (/country/locality)   value (23   tag 2   tag2   15…)

4. Reduce will sum information from combiner and find top 10 tag with its frequency. Besides it will also recognize in the structure we want to output.


setup() and save place.txt to the memory of Map task.
place-id 1 | 1 | tag 1 tag2 1
place-id 1 | 1 | tag 1
place-id 2 | 1 | tag 1
place-id | place-name
<String> <String>
name1 | 1 | tag 1 tag2 1
name1 | 1 | tag 1
name2 | 1 | tag 1
Combiner
~result:Text
+reduce(Text, Iterable<Text>, Context):void
name1 | 2 | tag 2 tag2 1
name2 | 1 | tag 1
Combiner
name1 | 3 | tag1 10...tag11 1
Reduce
~result:Text
+reduce(Text, Iterable<Text>, Context):void
hashmap to save tag and number. For loop to find top 10 tag for each key(name);
name1 | 5 | tag 2 tag 1 10 tag2 1....(top10)
name2 | 1 | tag 1

The second map reduce will find the top 50 cities.

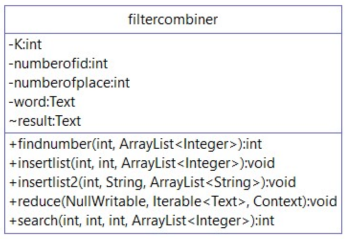
1. The Driver of the first map-reduce is called filterdriver.

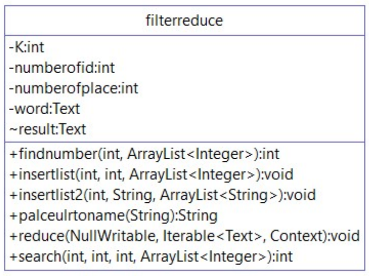


2. Mapper will read each line and give a NullWritable key to them.

3. The combiner which is executed in map task will find the top 50 records of its mapper.

4. For the output of combiner is in small number. We only need one reduce to sum and find the top50 cities.


filtercombiner
-K:int
-numberofid:int
-numberofplace:int
-word:Text
~result:Text
+findnumber(int, ArrayList<Integer>):int
+insertlist(int, int, ArrayList<Integer>):void
+insertlist2(int, String, ArrayList<String>):void
+reduce(NullWritable, Iterable<Text>, Context):void
+search(int, int, int, ArrayList<Integer>):int


filterreduce
-K:int
-numberofid:int
-numberofplace:int
-word:Text
~result:Text
+findnumber(int, ArrayList<Integer>):int
+insertlist(int, int, ArrayList<Integer>):void
+insertlist2(int, String, ArrayList<String>):void
+palceulrtoname(String):String
+reduce(NullWritable, Iterable<Text>, Context):void
+search(int, int, int, ArrayList<Integer>):int

Run result

Job	map input bytes	map output bytes	shuffled bytes	reduce output bytes	total execution time	number of map tasks	avg map time	longest map time	number of reduce tasks	avg reduce time	longest reduce time
1	10G	7.9G	900MB	36MB	1min, 43sec	88	1min, 2 sec	1min 14s	3	13s	14s
2	36MB	35MB	32KB	10KB	17sec	3	6sec	7sec	1	7sec	7sec
1	1G	800MB	72MB	7MB	51sec	9	33sec	45sec	3	17sec	18sec
2	7MB	7MB	31KB	10KB	10sec	3	3sec	3sec	1	2sec	2sec

Filter of tags

Besides filter year, it could filter local name in tags. No matter it is combined (sanfrancisco), separate (san francisco), or written in upper or lower case (San Francisco).

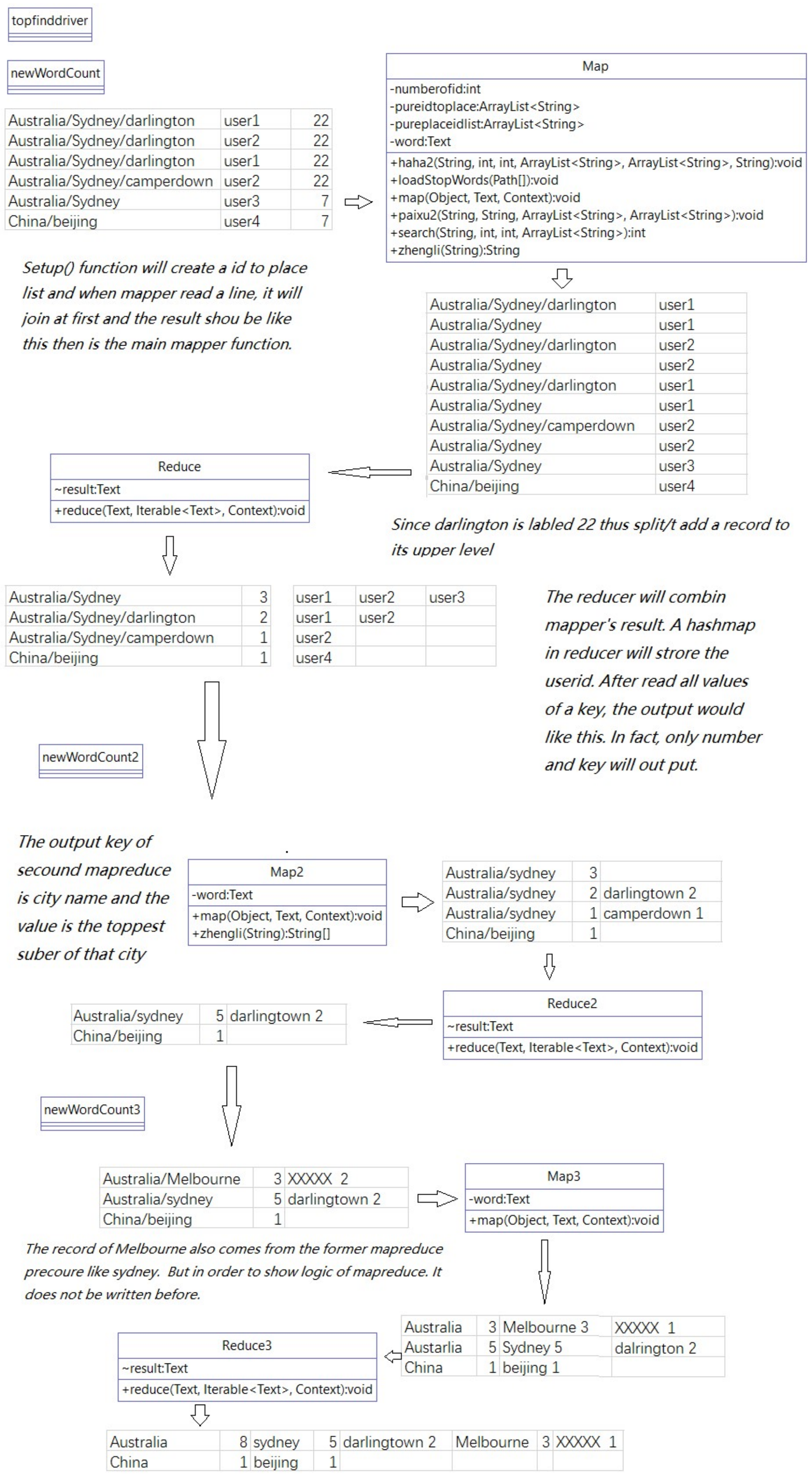
Run commend and result file

hadoop jar userTag.jar task1.topfinddriver /share/place.txt /share/photo wtan4210task1output.txt

two file are saved in "/user/wtan4210/task1-10g-output.txt" and "/user/wtan4210/task1-1g-output.txt"

Task2:

Code structure:



```
topfinddriver
```

```
newWordCount
```

Australia/Sydney/darlington	user1	22
Australia/Sydney/darlington	user2	22
Australia/Sydney/darlington	user1	22
Australia/Sydney/camperdown	user2	22
Australia/Sydney	user3	7
China/beijing	user4	7

```
Map
-numberofid:int
-pureidtoplace:ArrayList<String>
-pureplaceidlist:ArrayList<String>
-word:Text
+haha2(String, int, int, ArrayList<String>, ArrayList<String>, String):void
+loadStopWords(Path[]):void
+map(Object, Text, Context):void
+paixu2(String, String, ArrayList<String>, ArrayList<String>):void
+search(String, int, int, ArrayList<String>):int
+zhengli(String):String
```

Setup() function will create a id to place list and when mapper read a line, it will join at first and the result shou be like this then is the main mapper function.

Australia/Sydney/darlington	user1
Australia/Sydney	user1
Australia/Sydney/darlington	user2
Australia/Sydney	user2
Australia/Sydney/darlington	user1
Australia/Sydney	user1
Australia/Sydney/camperdown	user2
Australia/Sydney	user2
Australia/Sydney	user3
China/beijing	user4

Since darlington is labled 22 thus split/t add a record to its upper level

```
Reduce
~result:Text
+reduce(Text, Iterable<Text>, Context):void
```

Australia/Sydney	3
Australia/Sydney/darlington	2
Australia/Sydney/camperdown	1
China/beijing	1

user1	user2	user3
user1	user2	
user2		
user4		

The reducer will combin mapper's result. A hashmap in reducer will strore the userid. After read all values of a key, the output would like this. In fact, only number and key will out put.

```
newWordCount2
```

The output key of secound mapreduce is city name and the value is the toppest suber of that city

```
Map2
-word:Text
+map(Object, Text, Context):void
+zhengli(String):String[]
```

Australia/sydney	3	
Australia/sydney	2	darlingtown 2
Australia/sydney	1	camperdown 1
China/beijing	1	

```
Reduce2
~result:Text
+reduce(Text, Iterable<Text>, Context):void
```

Australia/sydney	5	darlingtown 2
China/beijing	1	

```
newWordCount3
```

Australia/Melbourne	3	XXXXX 2
Australia/sydney	5	darlingtown 2
China/beijing	1	

```
Map3
-word:Text
+map(Object, Text, Context):void
```

The record of Melbourne also comes from the former mapreduce precoure like sydney. But in order to show logic of mapreduce. It does not be written before.

Australia	3	Melbourne 3	XXXXX 1
Austarlia	5	Sydney 5	dalrington 2
China	1	beijing 1	

```
Reduce3
~result:Text
+reduce(Text, Iterable<Text>, Context):void
```

Australia	8	sydney	5	darlingtown 2	Melbourne	3	XXXXX 1
China	1	beijing	1				

Run result

Job	map input bytes	map output bytes	shuffled bytes	reduce output bytes	total execution time	number of map tasks	avg map time	longest map time	number of reduce tasks	avg reduce time	longest reduce time
1	10G	4.7G	4.8G	10MB	1min, 57sec	88	43sec	51sec	3	33sec	35sec
2	10Mb	9.9Mb	10.5MB	7MB	12sec	3	3sec	3sec	3	3sec	4sec
3	7MB	5MB	5.4MB	43KB	11sec	3	3sec	3sec	1	3sec	2sec
1	1G	483MB	503MB	4.8MB	47sec	9	32sec	38sec	3	3sec	3sec
2	4.8MB	2.4MB	2.5MB	2MB	11sec	3	3sec	3sec	3	0	0
3	2MB	1Mb	1.2MB	32.3KB	10sec	3	2sec	2sec	1	0	0

Run commend and result file

hadoop jar userTag.jar task1.topfinddriver /share/place.txt /share/photo wtan4210task2output.txt
two file are saved in "/user/wtan4210/task2-10g-output.txt" and "/user/wtan4210/task2-1g-output.txt"

Analyse：

Why Shuffle byte is different from mapout byte.
mapout is the sum of the size of all context output. But the size it on disk is "Map output materialized bytes". Shuffle process reads from disk, thus it equals to Map output materialized bytes. So, to the task1, I have combiner in it. Thus the data what write on the disk is much smaller than map output. In the secound task, I did not write combiner in it, thus it is a little larger than map output.

why there is big variation in reduce times.
Usually, the reason that happening is that different key has different size of values. Sometime, this different might be extremely large. What's more a certain key must be executed in one reduce task. So a key of large size of value need much time. The way to solve this problem is use partitioner.