1. tinyGoole

TinyGoogle will support 2 operations, searching and indexing. We think people who are qualified to insert index data into database are different from people who would search all the time in the database. So we separate index-client from search-client. You can either start an index-client to index new files or start a search-client to search for keywords. When you start a server, you have to specify several parameters, such as IP address, port number, location where you want to store index file, number of helpers that will connect to server initially. Then you need to start corresponding number of helper to connect to server in order for server to go ahead. If you start a index-client, you need to type in server IP address and port number for connection, then simply type in the path of file that you want to index. We design it to support concurrency of searching and indexing. The underlying mechanism is that when an index request comes in, the index master will first make a copy of each sub index file. Then each index helper will update those new copies of index files first. And the master will inform the server that index file names have been updated once all index helpers have done their jobs. The server therefore will change its local information of the paths of the index files. And those search requests that come in when the system is indexing will be performed on the original index files which is at most one version older than the latest version of index files.

For the experimentation, we choose to measure response time, we will change number of keywords, size of input file, number of helpers.

Conclusion, since we need to synchronizing each index helper when they are trying to update main index file, we actually do not benefit from more index helpers. We even need to wait more time for the writing. Additionally, index time increases as files sizes increase, but since the synchronizing time is almost the same, the increasing rate of index time is slower than the increasing rate of files sizes.

Search:

Keyword	response time	size of files that are indexed
a	1005	11
a in	1002	.....

README

1.start server

```
C:\eclipse\eclipse-workspace\os2510project2\src>java tinyGoogle.Server 0.0.0.0 9999 2 C:\eclipse\index 10000
```

Java FileName “IP Address of server” “port number of server” “number of index helpers you want to create at beginning”(you can always add new helper later) “file path of main index file”(the location where you want to store main index file) “port number of server master”

2. start index helper

```
C:\eclipse\eclipse-workspace\os2510project2\src>java tinyGoogle.IndexHelper 0.0.0.0 9999
```

Java FileName “IP Address of server” “port number of server”

Every helper is a single process, start as many helpers as you specified when you start server.

3. start index client

```
C:\eclipse\eclipse-workspace\os2510project2\src>java tinyGoogle.IndexClient 0.0.0.0 9999
```

Java FileName “IP Address of server” “port number of server”

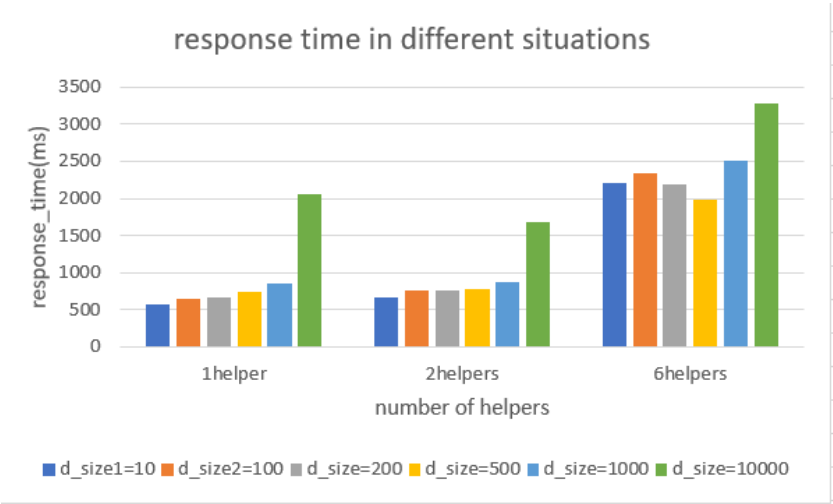
You can start as many index clients as you want

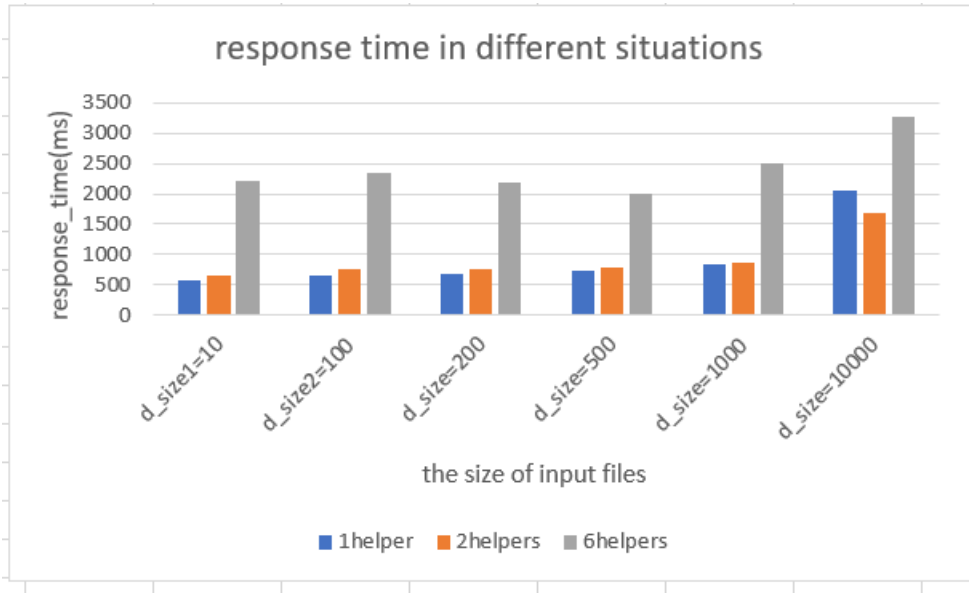
4. start search client

```
C:\eclipse\eclipse-workspace\os2510project2\src>java tinyGoogle.SearchClient 0.0.0.0 10000
```

Java FileName “IP Address of server” “port number of search master”

You can start as many search clients as you want, and they can run at the same time.





Part 2:

Part2 is has the same functions as part1, we can run the program on a Hadoop server to build the index depending on the input files and then search the keywords according to the index.

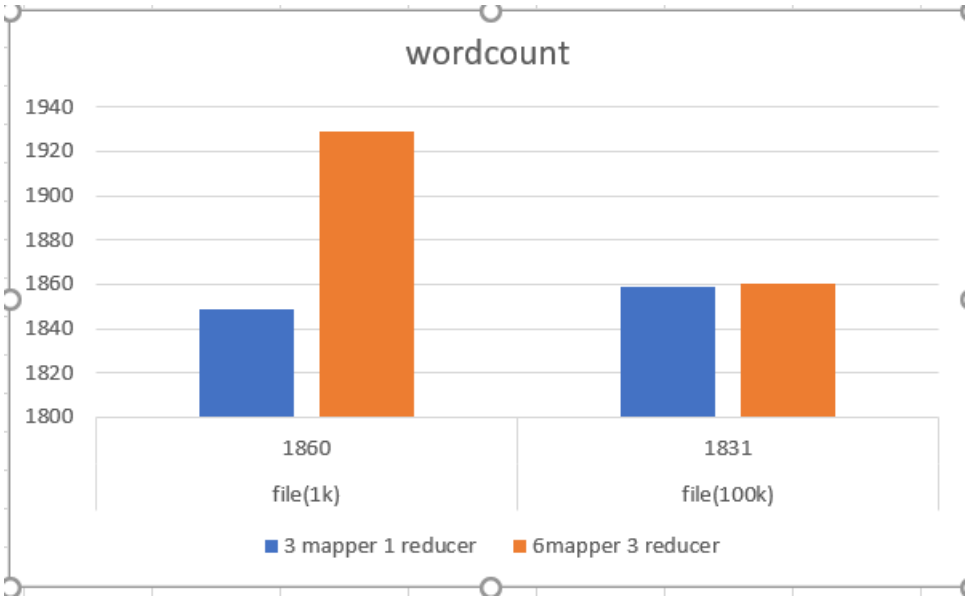
Firstly we can run wordcount.java to build the index, we upload the input files to the Hadoop server and then tell the program the input path and output path, and we can get a temporal index file. Next we run indexBuild.java to do more reduce, which means we can get all the info of a keyword in one single line. Finally we can do the search part, the user send a parameter to tell the program which key to search, and the mapper of the program try to match the keywords in the index, then the reducer just combine the correct result together and output a list of files where the input keywords show up.

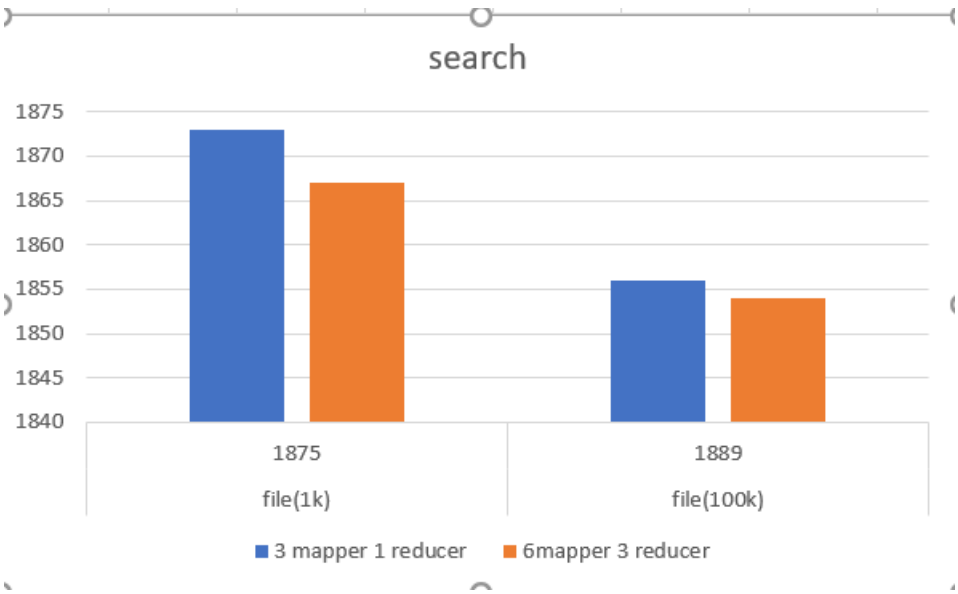
For part 2, we can also change the number of mappers and reducers according to the user's parameters. Depending on different size of the input files, we can either increase the number of mappers or reducer to make the program run faster. According to the experiment we can know in some time increase the number of mappers or reducers may increase the time.

```
    Job Counters
            Launched map tasks=3
            Launched reduce tasks=1
            Rack-local map tasks=3
            Total time spent by all maps in occupied slots (ms)=4789
            Total time spent by all reduces in occupied slots (ms)=1752
            Total time spent by all map tasks (ms)=4789
            Total time spent by all reduce tasks (ms)=1752
            Total vcore-milliseconds taken by all map tasks=4789
            Total vcore-milliseconds taken by all reduce tasks=1752
            Total megabyte-milliseconds taken by all map tasks=4903936
            Total megabyte-milliseconds taken by all reduce tasks=1794048
```

```
        HDFS: Number of write operations=2
Job Counters
        Launched map tasks=2
        Launched reduce tasks=1
        Rack-local map tasks=2
        Total time spent by all maps in occupied slots (ms)=3263
        Total time spent by all reduces in occupied slots (ms)=1776
        Total time spent by all map tasks (ms)=3263
        Total time spent by all reduce tasks (ms)=1776
        Total vcore-milliseconds taken by all map tasks=3263
        Total vcore-milliseconds taken by all reduce tasks=1776
        Total megabyte-milliseconds taken by all map tasks=3341312
        Total megabyte-milliseconds taken by all reduce tasks=1818624
Map-Reduce Framework
```

```
Job Counters
        Launched map tasks=7
        Launched reduce tasks=3
        Data-local map tasks=4
        Rack-local map tasks=3
        Total time spent by all maps in occupied slots (ms)=15866
        Total time spent by all reduces in occupied slots (ms)=5845
        Total time spent by all map tasks (ms)=15866
        Total time spent by all reduce tasks (ms)=5845
        Total vcore-milliseconds taken by all map tasks=15866
        Total vcore-milliseconds taken by all reduce tasks=5845
        Total megabyte-milliseconds taken by all map tasks=16246784
        Total megabyte-milliseconds taken by all reduce tasks=5985280
Map-Reduce Framework
```




search
1875
1870
1865
1860
1855
1850
1845
1840
1875
1889
file(1k)
file(100k)
3 mapper 1 reducer
6mapper 3 reducer