Ashok Arun
TextProvider Project

OVERVIEW

There are 6 main directories in this project. Each will be described in depth below. I didn't read any blogs when working on this project. The libraries I used include:

- java.net
- java.io
- java.util.StringTokenizer,
- java.sql

The main portion of each version of code includes creating a server on port 8080 using a ServerSocket. A client link is then connected the first time the command <curl localhost:8080...> is entered. Instead of creating a client java program which the user would then have to compile and run I decided to use a rest server which the user can curl into. Then 2 variables are created, one which stands for the path to write to the client and the other stands for the path to save what the client enters. What is entered by the client is then parsed to either get the line number, see if shutdown is entered, or return saying there was an error in the client message. Finally the client connection is closed and the server waits for it to open again.

RUNNING THE CODE (runs the threadedVersion)

- Enter **./build.sh**
- Enter **./run.sh**
- In a different shell enter **curl localhost:8080/get/<n>** to get the text corresponding to line <n>

simpleVersion

This is the most basic version of the project. When getting the line the code goes through all the lines in the txt file until it reaches the line number given or the end of the file. The larger the line number the slower the response will be. The size of the file doesn't matter. If a file is 1GB or 100GB and you ask for line 5 it will take the same time. If the number of clients/second increases everything will slow down. There is no threading in this simple version so each client will have to wait until the request before is done.

databaseVersion

This version will be very hard to test. In the folder the sql script is included on how to create a table and add all the lines in a txt file to that table to postgreSQL. The table has 2 columns, lineNum and lineText. lineNum starts at 1 and increments by 1 until the last line is added to the table. In the java code a postgreSQL driver is used to access the table. The line we are looking for in the table is specified by the WHERE command. This code is better than simpleVersion

because the line number does not affect the response time. Line 5 and 1,000,000 will take the same time to respond too. Threading is still not implemented in this version.

manyFileVersion
After creating the databaseVersion, I thought a better way to get the files would be to split the txt file into a bunch of single line txt files, each named numerically.txt. Then after the java code parses the line number, the code simply returns <n>.txt. Once again in this version the line number does not affect the response time and threading is not yet implemented.

threadedVersion
In this version I built off of the manyFileVersion to get threading to work. In this case increasing the requests/second slows down the overall response rate but many requests can run at once.

seekVersion
Uses seek functionality and goes based on byte values in the file.

multipleServerVersion
This version is built off of the threadedVersion. I started this version to implement load balancing but I reached the 10 hour limit before I could finish. Right now in this version the user can choose which port to start the server. I was curious to see if running on multiple servers was faster than running on multiple threads in my local machine. In the version the ./run.sh script takes 2 arguments. First the file location, and second the port number.

testing
There are 3 scripts in the testing folder which I used to formulate the following tests. My computer is not very powerful and some of these tests were affected because of that. But overall I could notice that the 1 line files were faster than the database, and the multiple servers on the same machine were faster than threading. I could also notice that the times in (ms) when doing 10 tests were much less volatile when on the database compared to the file system. The threading and 2 server tests were also really slow but once again I believe that is because my computer is not very powerful.

Test	Text Format	size	# of Threads	# of servers	# of requests per thread
1	Files	1 G	1	1	1,000
2	Files	1 G	1	1	5,000
3	Files	1 G	1	1	10,000
4	Files	1 G	2	1	1,000

5	Files	1 G	2	1	5,000
6	Files	1 G	1	2	1,000
7	Files	1 G	1	2	5,000
8	Database	1 G	1	1	1,000
9	Database	1 G	1	1	5,000

Test	**AVERAGE (ms)**	MIN (ms)	MAX (ms)	Requests/Second
1	**614**	579	671	1,628
2	**3,189**	2,991	3,444	1,567
3	**7,851**	6,275	11,064	1,273
4	**788**	737	869	2,538
5	**27,037**	5,324	39,787	369
6	**934**	845	1,291	2,141
7	**17,269**	4,116	41,813	579
8	**7,591**	6,825	12,839	132
9	**38,556**	32,275	46,291	129

Test	run 1 (ms)	run 2 (ms)	run 3 (ms)	run 4 (ms)	run 5 (ms)	run 6 (ms)	run 7 (ms)	run 8 (ms)	run 9 (ms)	run 10 (ms)
1	646	619	607	671	579	608	607	598	601	606
2	2,991	3,370	3,055	3,375	3,197	3,178	3,214	3,005	3,058	3,444
3	8,000	8,400	6,275	8,040	7,640	7,912	6,890	7,039	7,245	11,064
4	767	782	793	789	737	794	737	869	799	812
5	39,787	33,703	5,324	39,002	25,471	30,095	12,683	33,046	33,182	18,075
6	1,291	941	942	940	950	855	867	845	858	855
7	4,737	36,570	4,190	41,813	5,814	32,728	4,116	5,871	32,080	4,773
8	7,187	6,923	6,865	6,825	6,866	12,839	7,059	6,916	7,368	7,059
9	32,275	39,661	39,496	40,168	39,650	34,137	40,444	46,291	33,928	39,505