CS 342 - Operating Systems

Project 2

Enes Keleş 21501348

Buğra Aydın 21501555

1. General Information

In the experiments, we decided to use a constant keyword parameter sized 5. Then, we decided on the parameters that supposed to chance. Those parameters were line amount in the input.txt file, client amount and the percentage of the keyword occuring in the lines of the input.txt file. Our dependent variable is the time is takes to complete the process in miliseconds. We created a table for each line amount variable. Then we observed the change of the of time it takes to complete the process in a data table as the occurrence percentage changed.

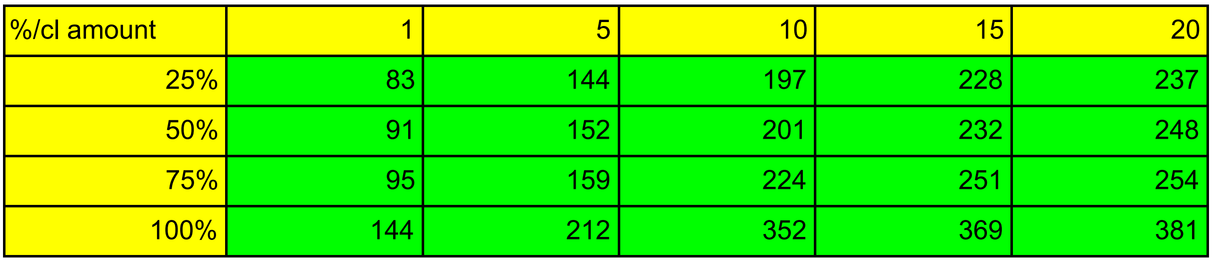
Input.txt Line Size = {100, 1000, 10000}
Client Amount = {1, 5, 10, 15, 20}
Occurence percentage = {25%, 50%, 75%, 100%}

2. Environment and Setup

In the experiments we ran our codes on an Ubuntu 16.04.03 64-Bit virtual machine that is working on Oracle VirtualBox. The virtual machine has 2GB's of RAM and one CPU core. For the testing, we prepared shell scripts that run 1, 5, 10, 15 and 20 clients in a concurrent and sequential manner.

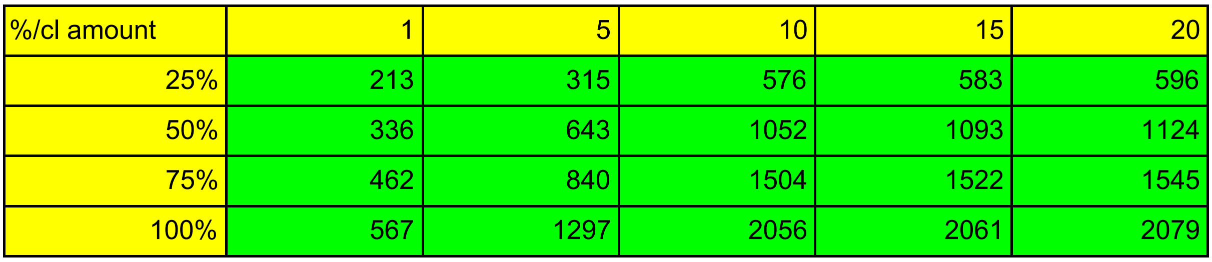
3. Results of Experiments and Analysis

For every case, we concluded 5 experiments and calculated the avarage value.



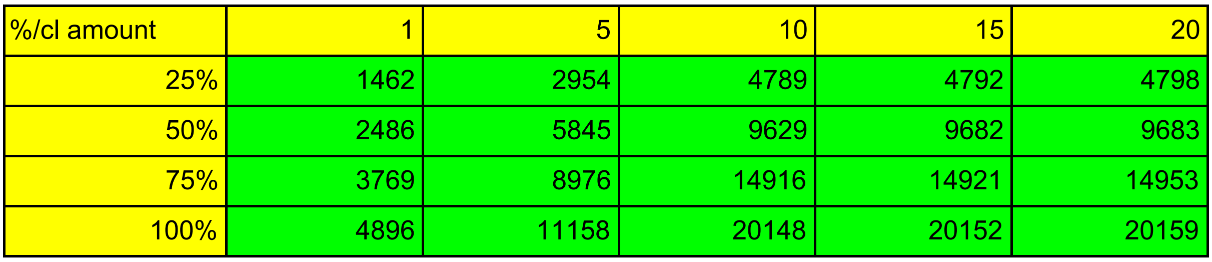
%/cl amount	1	5	10	15	20
25%	83	144	197	228	237
50%	91	152	201	232	248
75%	95	159	224	251	254
100%	144	212	352	369	381

Figure 1: Table of time in miliseconds, line amount 100, Concurrent run



%/cl amount	1	5	10	15	20
25%	213	315	576	583	596
50%	336	643	1052	1093	1124
75%	462	840	1504	1522	1545
100%	567	1297	2056	2061	2079

Figure 2: Table of time in miliseconds, line amount 1000, Concurrent run



%/cl amount	1	5	10	15	20
25%	1462	2954	4789	4792	4798
50%	2486	5845	9629	9682	9683
75%	3769	8976	14916	14921	14953
100%	4896	11158	20148	20152	20159

Figure 3: Table of time in miliseconds, line amount 10000, Concurrent run

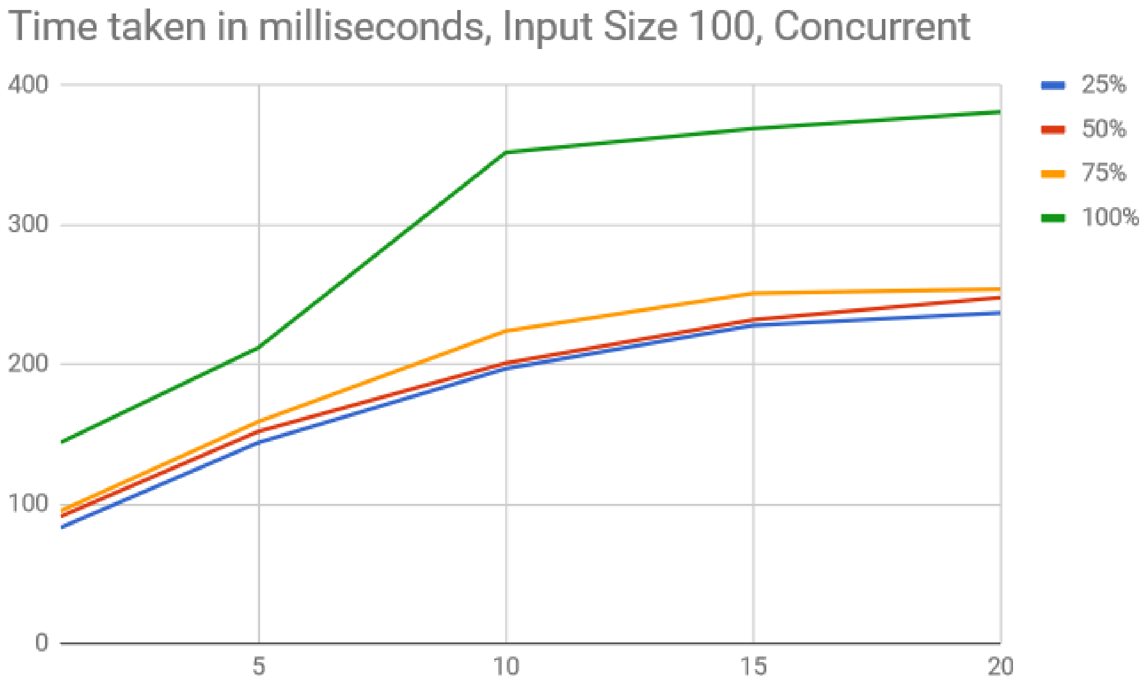


Figure 4: Line graph of time, line amount 100, Concurrent run, x axis is client count

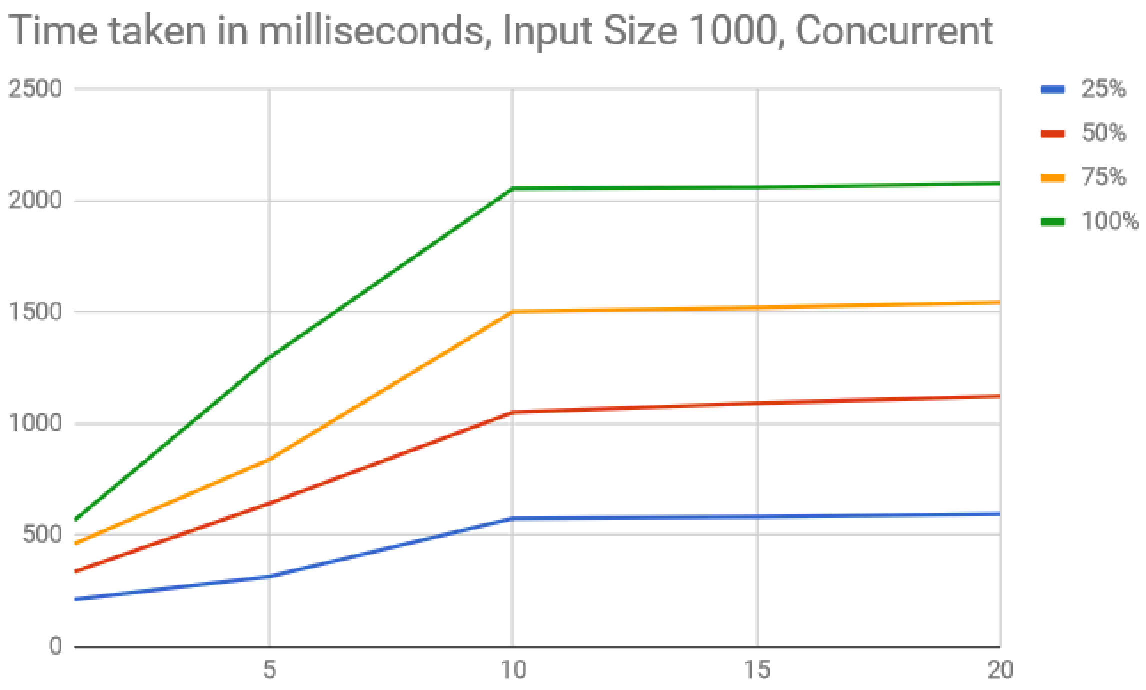


Figure 5: Line graph of time, line amount 1000, Concurrent run, x axis is client count

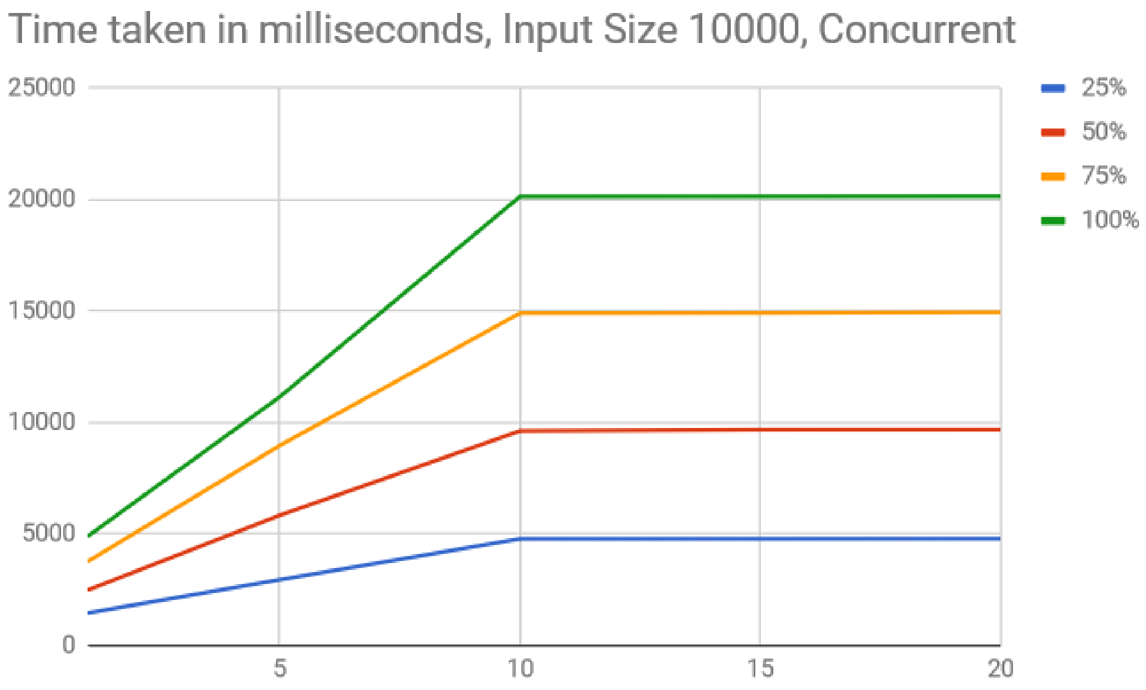


Figure 6: Line graph of time, line amount 10000, Concurrent run, x axis is client count

As it can be observed from the Figures 4-5-6, time it takes to complete the concurrent runs increases as the client amount increases. But since the maximum concurrent client amount is 10, it stops growing rapidly after 10 clients.

%/cl amount	1	5	10	15	20
25%	92	455	952	1603	1973
50%	95	532	1096	1737	2207
75%	101	672	1205	1659	2355
100%	143	711	1528	2171	3092

Figure 7: Table of time in miliseconds, line amount 100, Sequential run

%/cl amount	1	5	10	15	20
25%	223	1263	2512	3436	4872
50%	329	1488	2916	4652	6136
75%	467	2387	4832	7155	10312
100%	582	3092	6128	8565	11984

Figure 8: Table of time in miliseconds, line amount 1000, Sequential run

No experiments were done for line amount 10000, since the sequential runs takes a long time to comple.

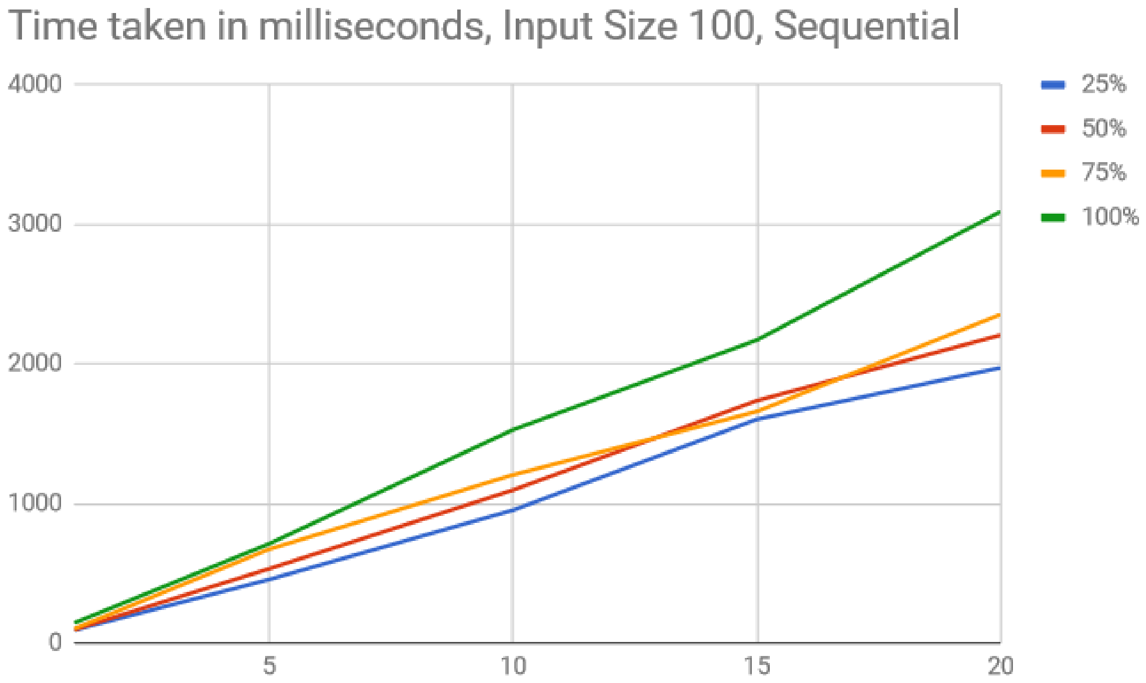


Figure 9: Line graph for line amount 100, Sequential run. x axis is client amount

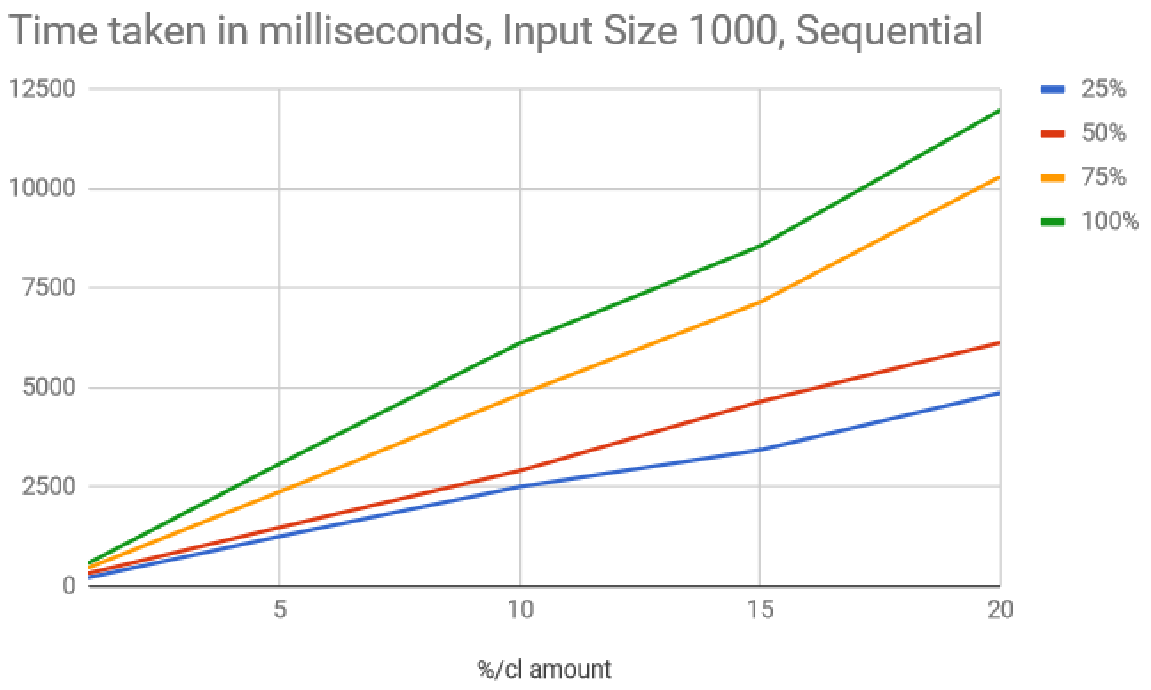


Figure 10: Line graph for line amount 100, Sequential run. x axis is client amount

As it can be seen from the Figures 9-10, increasing the client amount increases the time in a linear manner. Since our test keyword was the same for every client and in sequential every client starts after other one is run, this is expected.

%/cl amount	1	5	10
25%	9	311	755
50%	4	380	895
75%	6	513	981
100%	-1	499	1176

Figure 11: Table of time differences, sequencial - concurrent, input size 100

%/cl amount	1	5	10
25%	10	948	1936
50%	-7	845	1864
75%	5	1547	3328
100%	15	1795	4072

Figure 12: Table of time differences, sequencial - concurrent, input size 1000

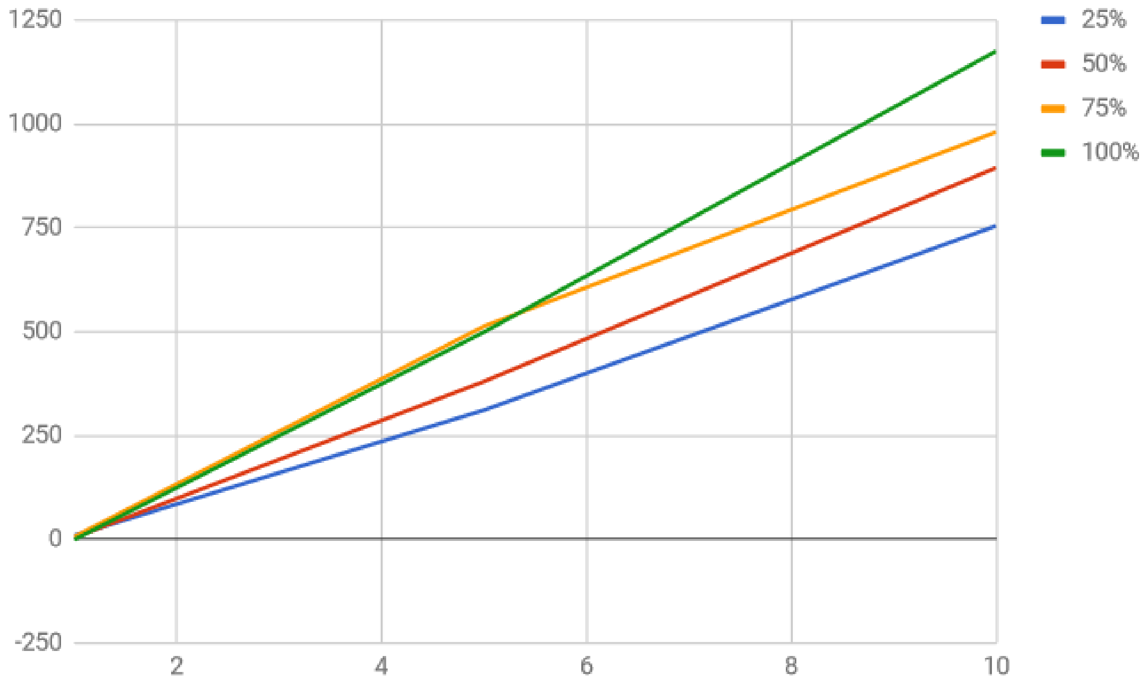


Figure 13: Line graph of time differences, sequencial - concurrent, input size 100

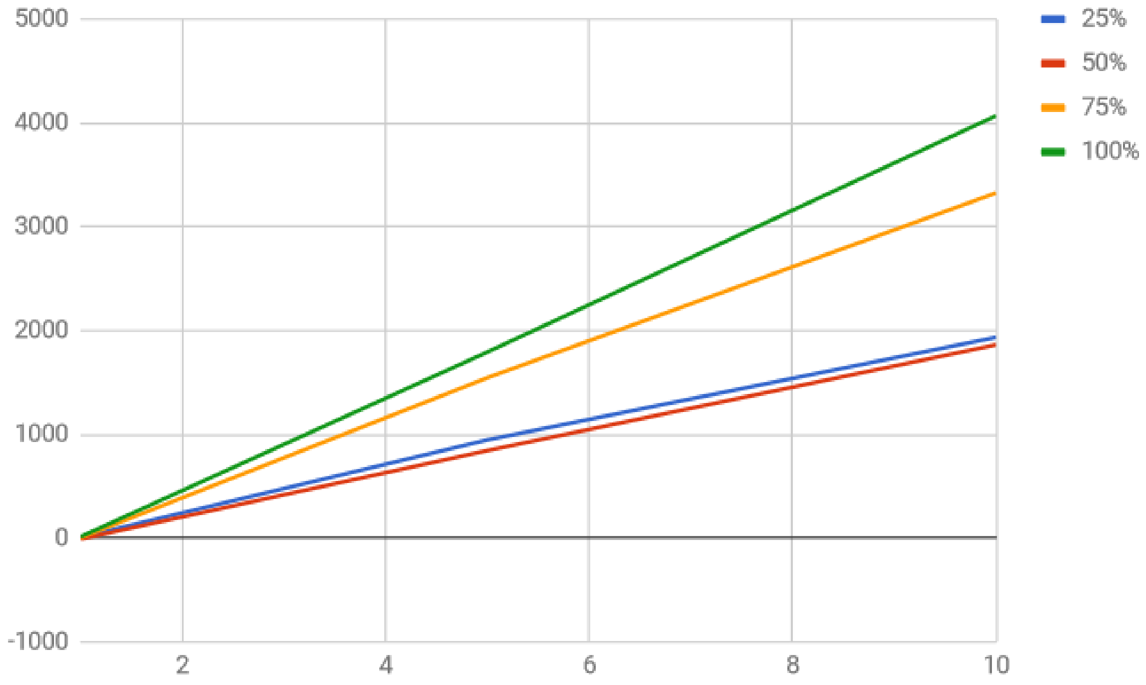


Figure 14: Line graph of time differences, sequencial - concurrent, input size 1000

The time is takes increases is highest on 10 in both runs since multithreading in concurrent runs decreases the time drastically compared to the sequential ones.

4. Discussion and Conclusion

<u>Concurrent runs:</u>

For concurrent runs, we saw that the execution times increases as the percentage of lines containing the keyword increases. Our string search algorithm has a complexity of O(NxM) where N = length of the text and M = length of the pattern. As the occurrence of the keyword increases, this algorithm is executed more frequently therefore the execution times are higher.

We observed that as the number of clients gets greater, the execution time also gets greater. The increasement is small because of concurrency. If the number of clients exceed the number result queues which is 10 in this case, there is no drastic change in the execution time because excess clients see that all result queues are taken and they terminate.

The last variant is number of lines in the input file. As the number of lines increase the execution time also increase because of I/O operations and time complexity of our search algorithm.

<u>Sequential Runs:</u>

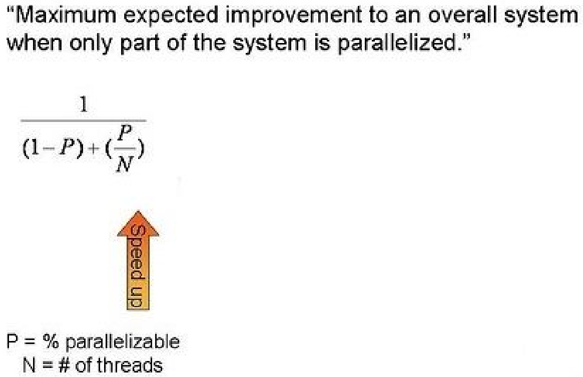
Also for sequential runs, the execution time increases as the percentage of lines containing the keyword increases. The reason is specified above in the concurrent runs part.

We observed that the execution time is proportionate to number of clients for sequential runs.

Number of lines in the input file has the same effect for sequential runs. The increasement is not linear in both cases because elapsed time for I/O operations depend mostly on file open and close operations.

<u>Comparison:</u>

Obviously, sequential runs takes much more time. We calculated the time saved when running clients concurrently and saw that the results comply with Amdahl’s Law which states:



“Maximum expected improvement to an overall system when only part of the system is parallelized.”

$$\frac{1}{(1-P)+(\frac{P}{N})}$$

P = % parallelizable
N = # of threads