Parallel/Distributed Computing (CSEG414/CSE5414) Assignment #1

120220172 박규리

1.(1)

**(a)** Amdahl’s Law; S(p) = 1/(f+(1-f)/p)

Num of processors: 4 => S(4) = 1/(0.2+0.8/4) = 2.5

Num of processors: 8 => S(8) = 1/(0.2+0.8/8) = 3.33

Num of processors: Infinite => S(∞) = 1/(0.2+0.8/∞) = 5

**(b)** Gustafson’s Law; Ss(p) = ⍺ + p(1-⍺)

Num of processors: 2 => Ss(2) = 0.2+2\*0.8 = 1.8

Num of processors: 4 => Ss(4) = 0.2+4\*0.8 = 3.4

Num of processors: 8 => Ss(8) = 0.2+8\*0.8 = 6.6

**(c)** Efficiency = S(p) / p

0.8 = 1/(f+(1-f)/p) \* 1/p. 즉, f = 0

0.8 = ⍺ + p(1-⍺) \* 1/p. 즉, s = 0.2

(2)

Processors p를 k배 => problem size n이 m배. 이는 와 같은 식 구할 수 있다.

이를 정리하면

Processors 개수 8개에서 16개로 늘리면 p=8, k=2

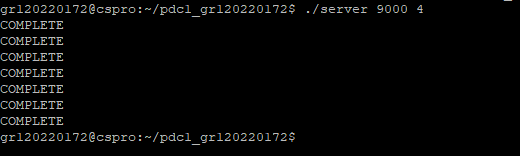
이 된다.

또한, problem size가 늘어난 비율이 processor가 늘어난 2배보다 크므로 scalable 하지 않다.

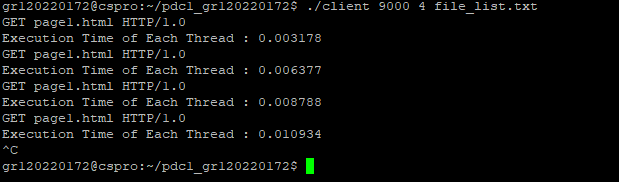
2. < Requirements for the performance evaluation and the report >

1. Test your web server using your client program.

-server



실행방법: ./server [port number] [thread number]



실행방법: ./client [port number] [thread number] file\_list.txt

서버와 클라이언트에서 쓰레드를 각각 네개씩 생성하여 실행시킨 결과이다. 클라이언트에서 서버로 요청을 보내게 되면, Get [filename] HTTP/1.0 이라는 문구가 클라이언트 단에서 출력이 되고, 이 요청이 성공적으로 끝났을 경우 서버단에서 COMPLETE 문구를 출력해주고, client에서 각 thread의 실행시간을 출력해주게 된다. 프로그램 종료는 ctrl+c로 종료시킨다.

2. Vary the number of threads in both client and server and report your observations as well as the performance of the server.

Server thread 8 fixed, client thread 4~1024 varied

|  |  |
| --- | --- |
| Number of thread | Execution time |
| 4 | 0.004466 |
| 8 | 0.005647 |
| 16 | 0.006369 |
| 32 | 0.006433 |
| 64 | 0.007146 |
| 128 | 0.010480 |
| 256 | 0.005444 |
| 512 | 0.008399 |
| 1024 | 0.004863 |

Client thread 개수 변화에 따른 그래프를 보면, client thread의 개수가 증가함에 따라 수행시간이 늘어나는 것을 확인할 수 있다. Thread의 개수가 증가할수록 http request 개수 또한 같이 증가한다. 그래프를 보면 thread가 64개까지 증가했을 경우에는 천천히 증가하다가 thread의 개수가 128개가 되었을 경우 급격하게 증가하는 것을 볼 수 있다. 그러다 갑자기 수행시간이 줄어드는 것을 볼 수 있지만, 다시 증가하는 추세를 보인다. 급격한 증가를 보이는 구간은 client에서 요청한 thread가 많아질수록 server에서 요청 처리를 하는데 생기는 overhead 때문에 생기는 것으로 보인다. client에서 요청한 thread가 많아질수록 socket overhead, transfer overhead 등이 점점 증가하기 때문이다.

Client thread 8 fixed, server thread 4~1024 varied

|  |  |
| --- | --- |
| Number of thread | Execution time |
| 4 | 0.006726 |
| 8 | 0.008272 |
| 16 | 0.007217 |
| 32 | 0.006944 |
| 64 | 0.006635 |
| 128 | 0.007331 |
| 256 | 0.006803 |
| 512 | 0.007225 |
| 1024 | 0.007009 |

server thread 개수 변화에 따른 그래프를 보면, server thread 개수가 증가할수록 server thread 하나 당 client thread 처리 개수가 줄어들 것이라고 예상했던 것과 다른 결과를 볼 수 있다. Server의 thread가 8개까지는 실행 시간이 늘어나다가, 그 이후로는 조금씩 줄어드는 경향을 보인다. 그러나 64개에서 128개로 thread의 개수가 증가할 때는 실행 시간도 같이 증가하는 것을 볼 수 있다. 그 후에는 다시 감소하지만 비슷하게 증가했다가 감소했다가를 반복한다. 예상한 결과와 다른 결과를 보이는 이유는 아마 프로그램이 돌아가는 server의 코어 수 등을 고려하지 않고 thread의 개수를 생성하였기 때문인 것 같다. 또한 Thread 개수가 증가할수록 context switch 등과 같은 overhead가 증가할 수 있고, 각 실험 마다 실행 시간이 다른 것을 보아 실험을 하는 환경에 따른 오차가 발생하는 것처럼 보인다.

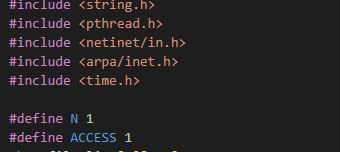
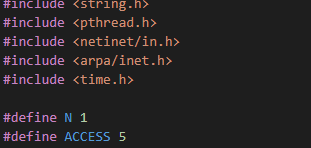
3. Vary also other parameters (e.g., size of file, interval, etc) and report your observations as well as the performance of the server.

-file size 변화. Client/Server thread 각각 8로 고정, access 횟수 1로 고정

|  |  |
| --- | --- |
| File size(bytes) | Execution time |
| 292823 | 0.006726 |
| 584377 | 0.008272 |
| 1749949 | 0.015217 |
| 5249668 | 0.022944 |
| 10499245 | 0.030635 |

파일 사이즈를 증가시키면서 각 파일 사이즈에 대한 실행 시간을 실험한 결과이다. 파일 사이즈가 증가함에 따라 실행 시간도 증가하는 것을 볼 수 있다. 이는 파일 사이즈가 증가하면 전송 시간이 길어지므로 생기는overhead 때문에 생기는 당연한 결과로 볼 수 있다.

-Access 횟수 변화. Client/Server thread 각각 8로 고정, access 횟수 1로 고정

 -> 

client.c 파일에서 ACCESS 로 define 해주었고, 숫자를 1에서부터 5까지 변경하면서 실험을 진행하였다.

|  |  |
| --- | --- |
| Access | Execution time |
| 1 | 0.003797 |
| 2 | 0.006716 |
| 3 | 0.009303 |
| 4 | 0.011931 |
| 5 | 0.013784 |

접근 횟수가 증가함에 따라 실행시간 또한 증가하는 것을 볼 수 있다. 파일 사이즈 별 실행 시간과 비슷하게 이 결과 또한 당연한 결과로 보인다. 접근 횟수 별 실행시간의 증가는 linear하게 증가하는 양상을 보인다. 접근 횟수의 증가는 server로의 http request 증가와 전체 data 전송 양 증가를 의미하고, 따라서 요청에 대한 overhead와 data 전송에 대한 overhead로 실행시간이 증가하는 것을 볼 수 있다.

4. Try to modify the client and server codes so that they maintain the connections (persistent connections) as long as each client has something to send to the server. In this case, the server should be modified to follow peer model with thread pool and each worker uses epoll mechanism (https://en.wikipedia.org/wiki/Epoll) to monitor multiple connections. Compare the performance with original implementation by varying various parameters and report the results as well as how you have implemented client/server codes.

- Server thread 8 fixed, client thread 4~1024 varied

|  |  |
| --- | --- |
| Number of thread | Execution time |
| 4 | 0.003399 |
| 8 | 0.003013 |
| 16 | 0.003239 |
| 32 | 0.004259 |
| 64 | 0.003908 |
| 128 | 0.004830 |
| 256 | 0.003250 |
| 512 | 0.004106 |
| 1024 | 0.004295 |

2번에서 수행했던 실험을 epoll server에도 적용해본 결과이다. 실행시간 양상은 master-worker server와 비슷한 양상을 띄는 것을 볼 수 있다. 이를 2번에서 실험한 그래프와 비교해보면 아래와 같다.

Master-worker server와 epoll server 모두 client thread가 증가할 때 실행 시간이 증가하는 비슷한 양상을 보인다. epoll server 같은 경우, thread간 경쟁이 일어나지 않기 때문에 위와 같은 결과가 나온 것으로 보인다. 또한 epoll server가 master-worker server보다 성능이 더 좋게 나왔는데, 이는 client가 요청을 보내고 그 client의 한 thread에서 같은 connection을 계속 이용하여 요청을 보냈기 때문에 더 좋은 성능을 나타낸 것으로 보인다.