Distributed Programming Assignment #2

By 20141500 권태국

가. RPC Programming

구현하였다.

나. Multithreaded Programming

나는 simple하게 reader-writer lock을 구현해보았고, 동시에 simple버전을 최적화하여 optimized version을 구현해보았다. 이 것들을 바탕으로 실험을 수행하였다.

처음에 INSERT한 KEY의 개수는 항상 1만개로 고정하였다.

1. Thread 개수를 변화하면서 실험했을 때,

	op개수 = 3만개, op비율 = 7 : 2 : 1 (search : insert : delete)							
thread개수	1	2	4	8	16	32	64	128
pthread	2.03	2.2	2.23	2.32	2.37	2.38	2.42	2.44
simple	2.13	1.8	1.92	2.05	2.1	2.12	2.17	2.2
optimized	2.04	1.7	1.89	2.01	2.05	2.05	2.12	2.14

	op개수 = 3만개, op비율 = 4 : 4 : 2 (search : insert : delete)							
thread개수	1	2	4	8	16	32	64	128
pthread	2.56	2.69	2.76	3	3.31	3.26	3.34	3.4
simple	2.49	2.62	2.79	3	3.2	3.31	3.31	3.47
optimized	2.56	2.62	2.74	2.97	3.18	3.24	3.3	3.38

나는 위와 같은 실험 결과를 얻었다. (단위는 seconds이다.)

보다시피 thread의 개수가 많아질수록, lock에 대한 경쟁이 심해져서 수행 시간이 증가함을 알 수 있다. 그러나 thread의 개수가 증가하는 것에 비해서 수행 시간의 증가 폭의 점점 작아지는 것을 알 수 있다. 이 것은 다음과 같은 이유 때문으로 생각된다. Thread들은 모두 reader일 경우에 동시에 동작할 수 있다. 그러나 한 thread만 writer여도 오직 동시에 1가지 thread만 동작할 수 있다. 즉, thread 개수가 적을 때는 thread 개수를 조금만 늘려도 thread들이 모두 reader인 경우가 크게 줄어들어 수행시간이 크게 증가하게 되는 것이다. 반면에 thread 개수가 많을 때는 thread 개

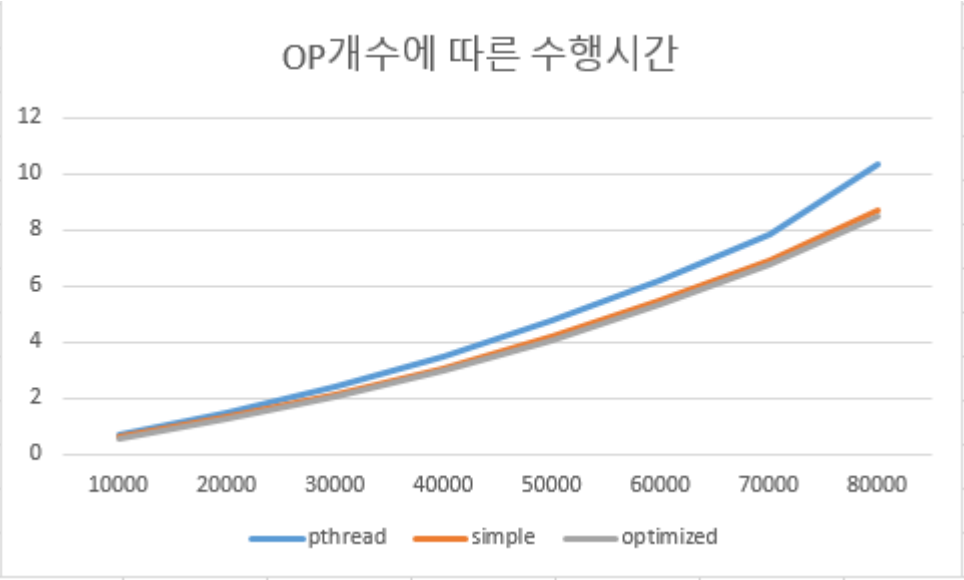
수를 늘려도 thread들이 모두 reader인 경우가 별로 줄어들지 않으므로 수행시간의 증가폭이 작은 것이다.

또한, op의 비율이 7:2:1일 때보다 4:4:2일 때 수행시간의 증가 폭이 훨씬 크다. 이 이유도 마찬가지로 비율이 7:2:1일 때는 thread의 개수를 증가시켜도 모두 reader인 경우가 별로 줄어들지 않는데, 4:4:2일 경우는 thread의 개수를 증가시키면 writer의 수가 상대적으로 굉장히 많이 늘어나므로, 모두 reader인 경우가 크게 줄어든다. 따라서 4:4:2일 때 수행시간의 증가 폭이 훨씬 큰 것으로 생각된다.

그리고 pthread와 simple과 optimized의 성능 비교를 해봤을 때, 어떤 구현체가 더 뛰어난지 판단하는데 있어서 thread의 개수는 중요하지 않다는 것을 알 수 있다. 단, 비율이 7:2:1일 때는 pthread > simple > optimized 순으로 수행시간이 적고, 4:4:2일 때는, 3개의 구현체가 거의 비슷함을 알 수 있다.

2. OP의 개수를 변화하면서 실험했을 때,

	thread 개수 = 16개, op비율 = 7 : 2 : 1 (search : insert : delete)							
op개수	10000	20000	30000	40000	50000	60000	70000	80000
pthread	0.67	1.46	2.38	3.47	4.77	6.18	7.86	10.37
simple	0.59	1.3	2.11	3.04	4.2	5.5	6.93	8.68
optimized	0.577	1.28	2.03	2.97	4.07	5.34	6.76	8.47



위는 thread의 개수와 op비율을 고정시키고 OP의 개수를 변화시키며 실험했을 때의 결과이다. (단위는 seconds이다.)

실험 결과에서, op의 개수와 수행시간은 거의 비례하는 관계를 가지고 있음을 알 수 있다.

그리고, op의 개수는 어떤 구현체가 뛰어난지 판단하는 데 있어서 영향을 미치지

않는 다는 것을 알 수 있다. 여기에서는 pthread >> simple > optimized 순으로 수행 시간이 적음을 알 수 있다.

3. OP비율을 변화시키면서 실험했을 때,

thread 개수 = 16개, op개수 = 3만개

op비율	8 : 1 : 1	6 : 2 : 2	4 : 3 : 3	2 : 4 : 4	0 : 5 : 5	4 : 1 : 5	4 : 5 : 1	6 : 4 : 0
pthread	2.05	2.36	2.77	3.18	3.65	2.13	3.71	3.19
simple	1.72	2.22	2.78	3.61	3.83	2.17	3.88	2.94
optimized	1.67	2.16	2.76	3.33	3.85	2.13	3.64	2.9

위 실험 결과로 미루어 보았을 때, thread중에 reader의 개수가 많을 때는 simple과 optimized가 수행 속도가 더 빠르고, writer의 개수가 많을 때는 pthread의 구현체가 더 빠르다는 것을 알 수 있다. 구체적으로는 reader와 writer의 비율이 4 : 6에서 writer의 비율이 증가하는 순간부터 pthread의 구현체가 더 빨라 짐을 알 수 있다. 그 전까지는 내 구현체인 simple과 optimized가 더 빠름을 알 수 있다. 그리고 그 중에 optimized가 더 빠름을 알 수 있다.

이로서, 어떤 구현체가 더 좋은 것인지 판단할 때, 영향을 주는 조건은 thread의 개수나 OP의 개수가 아닌 OP의 비율이라는 것을 알 수 있다.

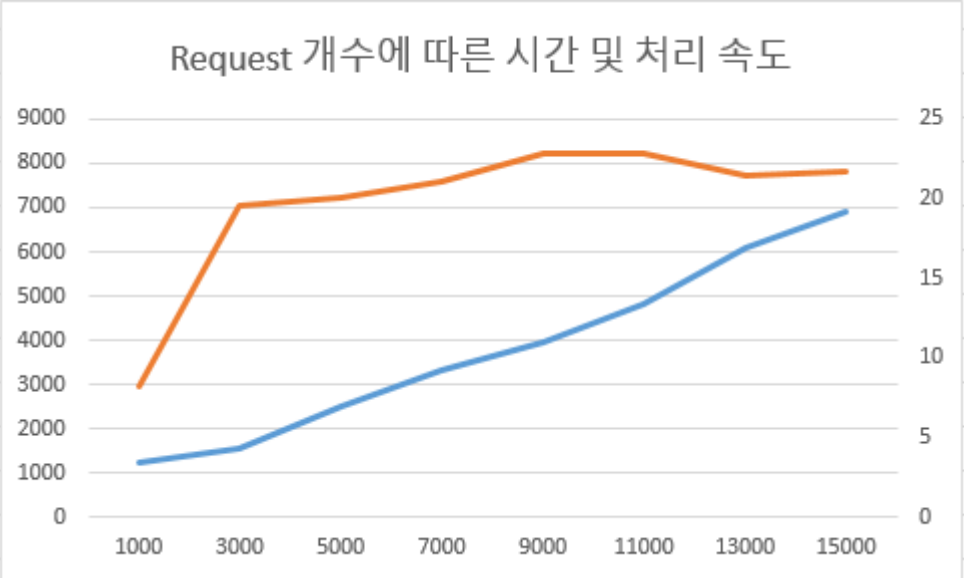
결론적으로 어떤 구현체가 더 좋은 지 결정을 내려보겠다. 애초에 reader-writer lock을 쓰는 경우는 reader가 writer에 비해서 많은 경우이다. 따라서 위 결과로 미루어 보았을 때 pthread 보다는 simple. simple보다는 optimized가 더 좋다고 판단을 내릴 수 있을 것이다.

다. Multithreaded Server

1. Request 개수에 따른 수행 시간 및 처리 속도

Server worker thread : 4개, File : various set, client 개수 : 10개

request 개수	1000	3000	5000	7000	9000	11000	13000	15000
시간	3.37	4.25	6.92	9.21	10.95	13.39	16.86	19.23
request/sec	2967.359	7058.824	7225.434	7600.434	8219.178	8215.086	7710.558	7800.312



위는 server worker thread의 개수를 4개로, client 개수를 10개로 고정한 뒤, client당 request 개수를 다양하게 바꿔가며 실험한 결과이다.

위 그래프에서 보여주듯이, request 개수가 증가함에 따라 처리 시간도 비례해서 증가함을 알 수 있다. 그러나 초당 request 처리 수도 그래프 초반을 제외하면 거의 비슷한 수치를 유지하는 것을 알 수 있다.

2. Client 개수에 따른 시간 및 처리 속도

Server worker thread : 4개, File : various set, 총 request 개수 : 90000개

client 개수	1	2	4	5	10	20	30	50	100
시간	31.97	20.4	13.11	11.81	11	11.7	11.33	16.42	25.19
request/sec	2815.139	4411.765	6864.989	7620.66	8181.818	7692.308	7943.513	5481.121	3572.846

위는 server worker thread의 개수를 4개로, 총 request 개수는 9만개로 고정한 뒤, client의 개수를 변화하면서 실험한 결과이다.

보다시피 Client 개수가 적거나 많으면 처리 시간이 오래 걸리고, 적당할 때 가장 최적의 속도를 보인다. 이는 client 개수가 매우 적은 수에서 적당한 수로 증가시킬 경우에는 client에서 server로 request가 충분히 가게 되어서 처리 시간이 줄어드는 것으로 보인다. 그러나 오히려 client 개수가 적당한 수에서 많은 수로 증가될 경우에는 이번 과제에서 제작한 client simulator가 단일 컴퓨터에서 동작되므로 이로 인한 context switching등의 비용으로 인해 simulator 자체의 속도가 느려져서 처리시간이 느린 것으로 측정되는 것으로 보인다.

Server worker thread : 8개, File : various set, 총 request 개수 : 9000개

client 개수	1	2	4	6	8
시간	3.32	2.01	1.36	1.22	1.3
request/sec	2710.843	4477.612	6617.647	7377.049	6923.077

Server worker thread를 8개로 증가시켜서 실험을 해본 결과, server worker thread의 개수와 적절한 client 개수 사이의 상관관계가 없음을 알 수 있었다.

즉, 이 말은 아까 말했듯이 client 개수에 따라 처리 시간이 달라졌던 것은 단순히 client simulator에서 server로 얼마나 효율적으로 request가 보내졌는지가 측정된 것이지, 서버의 performance가 측정됐다고는 할 수 없는 것이다. 따라서 서버의 performance를 정확하게 측정하려고 한다면, 이번 과제에서 요구한 simulator를 이용해서는 불가능할 것이다. 이번 과제에서 요구한 simulator는 multi-threaded 방식이나 이러한 방식으로는 simulation이 정확하게 할 수 없을 것 같고, epoll등을 이용해서 효과적으로 simulator를 제작해야 제대로 실험을 진행 할 수 있을 것이다.

3. File의 크기에 따른 처리 시간

	Server worker thread : 4개, client 개수 : 10개, 총 request 개수 : 30000개					
file종류	invalid	very_small_40	small_180	medium_500	big_1500	very_big_10000
시간	3.77	3.94	3.95	3.97	4.07	6.4
request/sec	7957.56	7614.213198	7594.93671	7556.675063	7371.00737	4687.5

위는 server worker thread의 개수를 4개, client의 개수를 10개, 총 request 개수를 3만개로 고정하고, file의 크기를 다양하게 하며 실험한 결과이다.

보다시피, file의 크기가 증가할수록 처리 시간이 증가함을 알 수 있다.

4. Thread 개수에 따른 처리 시간

	File : very_big_10000, client 수 : 100개, client당 request 개수 : 10개				
server thread수	1	2	4	8	16
시간	9.03	9.23	9.01	9.22	9.21
	File : various set, client 수 : 100개, client당 request 개수 : 50개				
server thread수	1	2	4	8	16
시간	12.23	12.04	12.04	12.24	12.23
	File : various set, client 수 : 4개, client당 request 개수 : 2500개				
server thread수	1	2	4	8	16
시간	1.35	1.39	1.46	1.52	1.56

위는 다양한 조건하에서 server thread의 개수를 변경시키면서 처리 시간을 측정한 실험 결과이다. 보면 thread 개수가 증가해서 처리 시간이 감소하지 않고 오히려 증가하는 경우도 있음을 볼 수 있다. 이는 실험이 정확하지 않기 때문으로 추측된다. 위의 2번 실험 항목에서 보았듯이 client simulator가 단순 multithreaded로 구현되어 정확하게 simulation이 수행되지 않았을 것으로 생각된다.