Report

Yating Li(yl657), Rui Yang(ry70)

Test Method:
We open 512 threads in the client. Each thread keeps sending and receiving requests until the desired number has been sent and received. Then, the time from sending the first one to receiving the last response is the total time.
Each result is the average of 10 experiments.

Test1:
Conditions: fix bucket size = 512;
fix delay range = small(1, 3).
Changes: test method: per-thread, pre-thread;
the number of cores: 1, 2, 4.
Performance: the number of requests that could be handled per second

#core	#request	per-time/ms	performance	pre-time/ms	performance
1	10	3044	2.9069	4960	2.0161
	50	3307	15.1194	7832	6.3841
	200	5372	37.2301	8214	24.3487
	500	5789	86.3707	19709	25.3691
	1000	8641	115.7273	19076	52.4219
2	10	3022	3.309	5295	1.8889
	50	3362	14.8721	5555	9.0010
	200	4551	43.9464	6171	29.6428
	500	6732	74.2721	13250	37.7359
	1000	9726	102.81712	20789	48.1024
4	10	3079	3.2478	5017	1.9932

	50	3296	15.1699	6238	8.0154
	200	6791	29.4507	7171	27.8901
	500	12580	39.7456	7725	64.7249
	1000	18280	54.7046	18134	55.1450

1, As the number of cores increases, the performance increases. Also, the extent of the performance boost will increase when the number of requests increases.
2, The pre-created threading method is not as efficient as per-created threading when the number of cores is small or the number of threads is small. This is reasonable since pre-thread needs to check for available threads. When the number of cores is large, it is fast to do those check work. When the number of requests is large, the fraction of time spent on distributing the requests to specific threads decreases.

Test2:
Conditions: fix the bucket size = 512;
fix the number of cores = 4;
Changes: test method: per-thread, pre-thread;
delay range: small(1-3), large(1-20).

delay range	#request	per-time/ ms	performance	pre-time/ ms	performance
small	10	3079	3.2478	5017	1.9932
	50	3296	15.1699	6238	8.0154
	200	6791	29.4507	7171	27.8901
	500	12580	39.7456	7725	64.7249
	1000	18280	54.7046	18134	55.1450

large	10	17058	0.5862	18701	0.5347
	50	20410	2.4498	20848	2.3983
	200	22029	9.079	21141	9.4603
	500	27334	19.2922	22561	22.1621
	1000	41237	24.2501	44658	22.3924

When the delay time is large, the pre-thread will have a better performance. For the per-thread method, it will create a lot of new threads which is very time-consuming. While pre-thread will balance the work and assign the new request to the available thread.

Test3:
Conditions: fix the number of cores = 4;
fix the delay range = small(1-3).
Changes: test method: per-thread, pre-thread;
bucket size: 32, 128, 512, 2048.

#bucket	#request	per-time/ms	performance	pre-time/ms	performance
32	10	3034	3.2960	4276	2.3386
	50	3657	13.6724	4328	11.5527
	200	5641	35.4547	6273	31.8827
	500	7671	65.1806	8258	60.5473
	1000	12225	81.7996	19020	52.5762
128	10	3031	3.2992	4916	2.0342
	50	3514	14.2288	4959	10.0827
	200	5084	39.3391	6331	31.5906

	500	9373	53.3447	8143	61.4024
	1000	11281	88.6446	20944	47.7364
512	10	3018	3.1334	5017	1.9932
	50	4147	12.0569	6238	8.0153
	200	6087	29.1121	7171	27.8901
	500	16658	30.0156	7725	64.7249
	1000	15885	62.9525	20944	47.7463
2048	10	3033	3.2971	6380	1.5673
	50	3192	15.6641	4864	10.2796
	200	5329	37.5305	5400	37.0370
	500	10845	46.1041	8850	56.4971
	1000	19085	52.3971	17893	55.8878

If the ratio of the number of requests over the bucket size is large, it means that there are lots of requests need to check the same bucket. However, when threads access the same bucket, we need to use the lock to prevent a race condition, which leads to the serialization of the whole process.

Reflection:

1, The result shows that the number of experiments is not large enough to decline the influence of random. We should have started earlier and tested more to calculate the average.

2, We should have managed the resources such as memory allocations better so that our server could handle more requests at the same time and run for a longer period.