Journal of Korea Multimedia Society Vol. 12, No. 6, JUNE 2009(pp. 876-881)

Prediction-based Dynamic Thread Pool System for Massively Multi-player Online Game Server

Woosuk Ju†, Choongjae Im††

ABSTRACT

Online game servers usually has been using the static thread pool system. But this system is not fit for huge online game server because the overhead is always up-and-down. Therefore, in this paper, we suggest the new algorithm for huge online game server. This algorithm is based on the prediction-based dynamic thread pool system. But it was developed for web servers and every o.1 seconds the system prediction the needed numbers of threads and determine the thread pool size. Some experimental results show that the check time of 0.4 seconds is the best one for online game server and if the number of worker threads do not excess or lack to the given threshold then we do not predict and keep the current state. Otherwise we apply the prediction algorithm and change the number of threads. Some experimental results shows that this proposed algorithm reduce the overhead massively and make the performance of huge online game server improved in comparison to the static thread pool system.

Key words: massively multi-player online game server, thread pool, prediction-based dynamic thread pool system

1. INTRODUCTION

Computer online game is a kind of game that includes hundreds of players distributed over WAN scale networks. Now, there have been two fundamental server architectures proposed Peer-to-Peer (P2P) and Client-Server (C/S) for communicating each other. We concentrate on the research based on C/S model. To construct C/S online game, we need make a server program to do with players' requests and send related data to specific clients. For designing this program we need first know some characteristics about online game. First, every player requests game packets in the time of o.1seconds constantly until the game is ended and its processing time must be very short. Second, a huge of requests is transferred to game server from hundreds of player simultaneity. Third, requests need to be response and the response packet are sent to related clients by game server in time but its size is extremely small in contrast with web server. Finally, normally, the number of player is below 1000 in the same game server and take the static thread pool system.

In this paper, we experiment with below 1000 users based on prediction-based dynamic thread pool system. But we find that the system was not fit for huge online game server because overhead is up-and-down constantly and the addition and reduction process happened frequently. So the amount of processed packets is reduced. Therefore, we propose to set the threshold for the number of working thread and find the fact that the check time must be increased by some experiment. Under these condition, we apply the prediction-based dynamic thread pool system to huge online game server.

※ Corresponding Author : Choongjae Im, Address : (705-701) Department of Game & Mobile Contents, Keimyung University, Deagu, Korea, TEL : +82-53-620-2185, FAX : +82-53-620-2175, E-mail : dooly@gw.kmu.ac.kr
Receipt date : June 8, 2009, Approval date : June 23, 2009
† Dept. of Game, Dongseo University
(E-mail : savrang@dongseo.ac.kr)
†† Dept. of Game & Mobile Contents, Keimyung University
※ The present research has been conducted by the Bisa Research Grant of Keimyung University

Some experimental results show that our proposed algorithm can make the overhead reduced and the performance of online game server increased.

This paper is composed as followings. We first introduce some characteristics of current online game server and the theories of stati c/prediction-based dynamic thread pool system in Section 2. We suggest the our proposed algorithm for huge online game server in Section 3. Finally, we compare the performance of the proposed online game server with static thread pool system and give some results in Section 4. Conclusions are presented in Section 5.

2. PREVIOUS WORKS

The client's requests transferred by the TCP/UDP layer are processed in worker thread. Fig 1 shows the overall architecture of thread pool system.

There are two methods to determine the number of worker threads, called static thread pool system and dynamic tread pool system. In this section, we will introduce the static thread pool system for online game server and prediction-based dynamic thread pool system for online game server.

2.1 Static thread pool system for online game server

This system use the fixed worker thread pool size as follows.

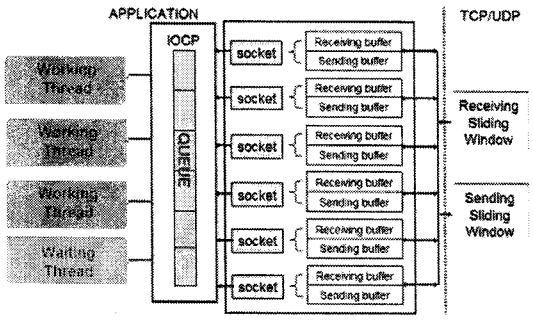

Fig. 1. Architecture of thread pool system

worker thread pool size =
the number of CPU X 2

This means that the size of thread pool should be two times of the number of CPUs on the host machine [1]. Normally, it is considered as a start point and then is tuned according as the ability of processing packets and overhead to game system in order to further optimize the size of thread pool. Because there is not any time-waste on creating and destroying threads during the runtime, it reduces the response time. And it also avoids consuming the resource of server such as CPU usage and memory, when it creates overfull threads in game server. However, from the fixed the pool size, it can not use the system resource effectively, thus it can make lower the game server performance lower [2].

To solve this problem, it is desirable to have a kind of thread pool which can configure itself based on the current status of thread pool. That is dynamic thread pool system. D. Xu and B. Bode suggested the dynamic thread pool system using the heuristic algorithm. J.H. Jung et al. [3] suggested that the worker thread pool size is determined by the prediction-based dynamic thread pool system.

2.2 Prediction-based Dynamic Thread Pool System

It is a combination of a worker thread pool and a pool manager thread. Pool manager thread can tune the size of worker thread pool based on the current status of worker thread pool so as to deal with more requests through using the rest of game server's resource.

Fig 2 shows about internal operation of Pool Manager Thread. Pool Manager Thread can tune the size of Worker Thread Pool.

It is unique and works through checking the status of worker thread pool (Busy, Idle and Normal) at intervals of a check time. If the status of worker

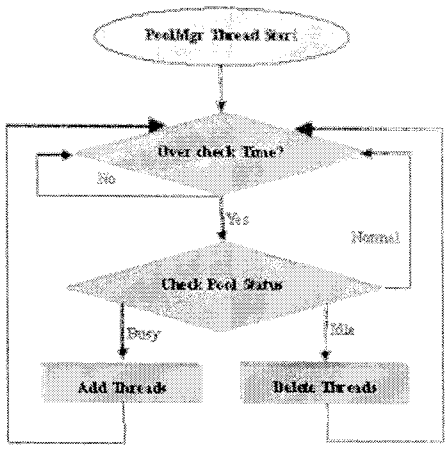

Fig. 2. Flow chart of pool manager thread operation

thread pool is busy, pool manager thread adds some new worker threads. If the status of worker thread pool is idle, pool manager deletes some free worker threads. If status is normal, there is not any change in worker thread pool until next check. Dynamic Thread Pool avoids the management overhead of too many threads because it can delete extra worker threads by itself. And it can increase the performance because it can add several worker threads when most of worker threads are busy to work and there are still many requests waiting for processing. However, because its variance of thread pool size is fixed and the amount of clients' requests changes constantly, though worker threads added into the dynamic thread pool can increase the server performance of processing requests, sometimes they are not enough. To solve this problem, exponential average can be used to predict the amount of worker threads in next time.

If we can predict how many extra worker threads is needed in next time, we can save creating time and avoid process those packets until the idle worker thread is created when there are big amount of packets arrived. And we can also reduce wasting in system resource because thread waste a part of memory and CPU scheduling time when too many idle worker threads still exist. Today, predictable dynamic thread pool has been advanced.

Before tuning the thread pool size, game server program first predicts the amount of threads needed in the next time through using exponential average idea [4,5] in favor of increasing the server performance and saving the extra system resource waste. Exponential average approach [5] is used in the CPU scheduling problem for the prediction of the idle period. In the CPU scheduling problem, operating systems need to predict the length of the next CPU burst in order to make appropriate process scheduling. In general, the next CPU burst is predicted as an accumulative average of the measured lengths of previous CPU bursts. Similarly, we can predict the next idle period by the accumulative average of the previous idle periods. Exponential average formula is

$$\Gamma_{n+1} = \alpha t_n + (1-\alpha)\Gamma_n \ (0 \le \alpha \le 1)$$

The parameter α controls the relative weight of recent and past history in the prediction. If $\alpha = 0$, then $\Gamma_{n+1} = C$. In other words, the recent history has no effect. On the other hand, if $\alpha = 1$, then $\Gamma_n = t_n$. In this case, the prediction only takes into account the most recent running threads but ignores the previous predictions. In our implementation, α is set to be 0.5 so that the recent history and past history are equally weighted.

3. PROPOSED SYSTEM AND EXPERIMENTAL RESULTS

3.1 Proposed System

Fig 3 shows internal operation of pool manager thread in addition to the function of predicting how many worker threads need be added/deleted based on the amount of worker threads in the next time when the status of worker thread pool is busy or idle.

This figure is full operation of pool manager thread in our proposed system. There are two problems to realize proposed system to online game

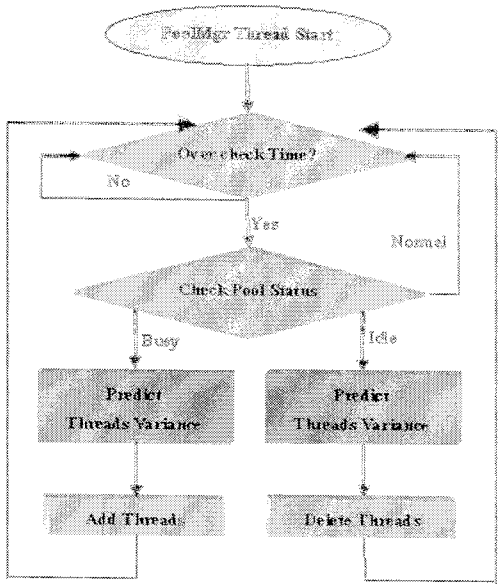

Fig 3. Flow Chart of Pool Manager Thread operation with predicting variance of worker threads

server programming. Problem 1 is whether proposed system must be better than static thread pool for any amount of users or not. Problem 2 is how to determine check time. That is to say the frequency of checking the status of worker thread pool. If check time is too small, game server program has to add/delete some threads very frequently. It brings so much Creation/Destroy of thread as to spend much time on creating/destroying thread instead of processing game packets. The benefit of adding worker threads is canceled by the overhead of threads' creation. If check time is too big, the Pool Manager Thread can not monitor the status the dynamic thread pool effectively. To judge that dynamic thread pool fits what kind of online game server and describe the effect of check time for improving the performance of game server program, I do several groups of experimentations with different user amount and tune the check time value in every group.

The game server operating system is Windows 2000 Server running on two 2GHz Intel(R) Xeon (TM) processors. The physical memory size is 2G. Game server's I/O model is IOCP. And clients run

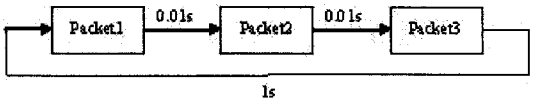

Fig 4. Iteration of Packets sent by Client

on one 2.8GHz Intel(R) Pentium(R) 4 processor and physical memory size is 512MB.

They are 800 users and 1000 users. We run 200 threads (every thread simulates one client) on one client PC at best. A new thread will be created every one second. Every client sent 3 kinds of different packets. Sending method is as follows. The whole testing time is 3 minutes.

3.2 Experimental results for problem 1

We did 10 tests about proposed system with different check times (they are from 100ms to 1000ms) and one test about static thread pool. These following 3 figures are comparison diagrams of game server with dynamic and static thread pool as follows.

Fig 5 shows the experimental result that the proposed system does not have difference with static system.

Fig 6 shows that the effect of static thread pool and dynamic thread pool is different. We can see the performance of dynamic thread pool is much better than the one of static thread pool when check time equals 0.4s.

Fig 7 also shows that the effect of static thread pool and dynamic thread pool is different. This time we also can find the performance of dynamic

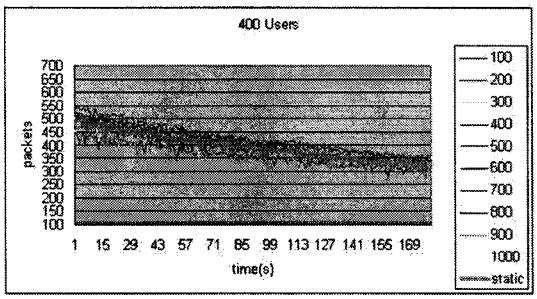

Fig 5. Proposed/Static Thread Pool for 400 Users

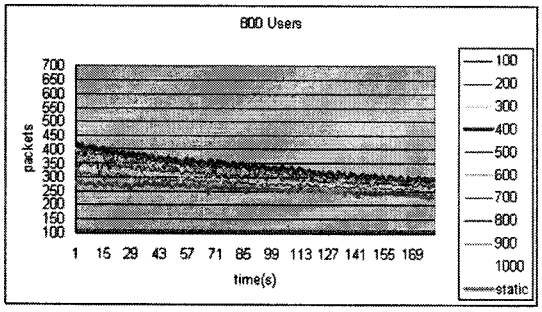

Fig 6. Proposed/Static Thread Pool for 800 Users

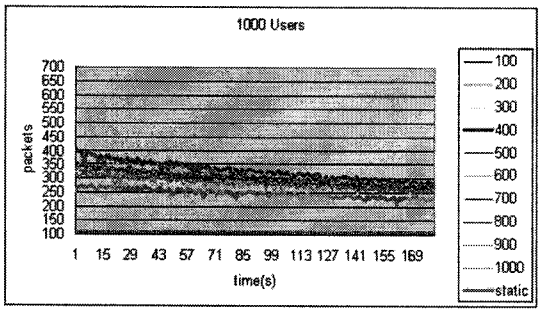

Fig 7. Proposed/Static Thread Pool for 1000 Users

thread pool is much better than the one of static thread pool when check time equals 0.4s again.

Through these experimentations above, we find that the dynamic thread pool doesn't make much difference to the performance of game server using static and proposed system when the amount of users is 200, 400 and 600. However, the dynamic thread pool starts to make much difference to the performance of game server when the amount of game users is 800 and 1000. And when check time of dynamic thread pool is modified properly, the effect of dynamic thread pool is the best.

3.3 Experimental results for problem 2

Because the check time of dynamic thread pool give the effect to the performance of game server using dynamic thread pool, I listed the average packets processed by dynamic thread pool when the amount of users is 800 and 1000.

I find that the performance of game server is the best when check time equals 400ms in the Fig 8

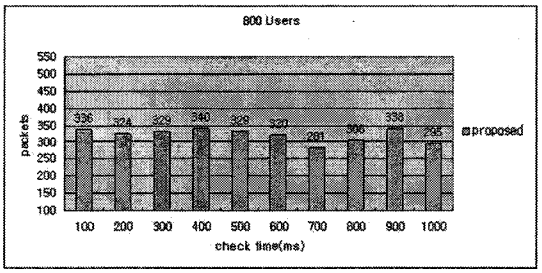

Fig 8. Average Packets processed by proposed system for 800 Users

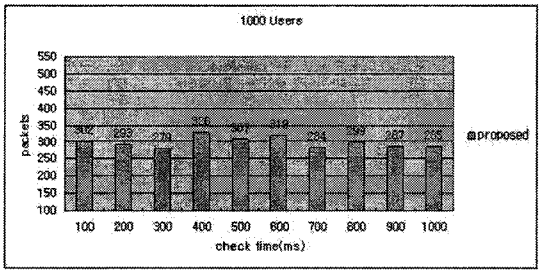

Fig 9. Average Packets processed by proposed system for 1000 Users

and Fig 9 above.

4. CONCLUSIONS

In this paper, I built a dynamic thread pool system for online game server program based on exponential average. It can check the status of worker thread pool at intervals and predict how many worker threads are needed in the next time and add/delete proper worker threads for improving the performance of game server program and saving the game server's resource. From experimental results, the dynamic thread pool system can increase the throughput of game server only when the amount of game user is huge. If there are not so huge game users, it is possible in using static thread pool or dynamic thread pool. In reverse, it is better in using dynamic thread pool in game server program when the scale of online game is big. And when we use dynamic thread pool system in online game program, we need to tune the check time in order to obtain the best performance for

game server program.

REFERENCES

[1] Richter J., *Advanced Windows, 3rd Edition*, Microsoft Press, 1996.

[2] Jones, A., and Ohlund, J., *Network programming for Microsoft windows, 2nd Edition*, Microsoft Press, 1995.

[3] Jung, J. J., Han, S. Y., and Park, S. Y. "Prediction-based dynamic thread pool model for efficient resource usage," *Computer Systems and Theory*, Vol.31, No.4, pp. 213-223, 2004.

[4] Ling, Y., Mullen, T., and Lin, X., "Analysis of Optimal Thread Pool Size," *ACM SIGOPS Operating System Review*, Vol.34, No.2, pp. 42-55, 2000.

[5] Peterson, J. L., and Silberschatz, A., *Operating System Concepts, 2nd Edition*, Addison-Wesley Publishing Co. Inc., 1985.

Woosuk Ju

He received his B.S. degree from Kyungnam University, Masan, Korea, in 1998 and his M.S. degrees from Dongseo University, Busan, Korea, in 2005. He is currently a Ph.D. degree student of the Pusan National University, Korea. He is currently a professor of the Dept. of Game in Dongseo University, Pusan, Korea. His research interests include online game development and image processing.

Choongjae Im

He received his B.S. and M.S. degrees from Chungnam National University, Deajeon, Korea, in 1991 and 1993, respectively. He is currently a professor of the Dept. of Game & Mobile Contents in Keimyung University, Daegu, Korea. His research interests include multi-platform game development and computer graphics.