ECE 1747 : PARALLEL PROGRAMMING - ASSIGNMENT 1

CHEN SUN, STUDENT NUMBER: 1001418799
JIAYUE ZHANG, STUDENT NUMBER: 1003555146

1 BACKGROUND INTRODUCTION

The current common issue for servers of on-line games is the load management problem. The problem is that given a game consisting of M regions assigned to N servers, we need to find a proper way to relocate regions to servers such that we can balance the server load and decrease inter-sever communication. This report is based on the algorithm comparison between 'static' and 'dynamic' schemes in several scenarios. The problem here is that we want to balance server load by replicating existing game world partitions across several servers and decrease inter-server communication by maintaining locality of adjacent regions.

One existing solutions for this problem is static partitioning for example, row or column based partition. The static partitioning limits cross-server interactions between players, and exposes the division of the world to players. We have also dynamic load balancing algorithms. The dynamic load management algorithm enables us to better handle transient crowding by adaptively dispersing or aggregating regions from servers in response to quality of service violations.

In this report, we use SimMud, a massively multiplayer game simulator to implement and evaluate the Dynamic Uniform Load Spread - "Spread" method. We analyze the performance of the static vs. dynamic schemes in the following two scenarios:

(a) There is no active quest and the players are moving more or less randomly,

(b) There is an active quest and as a result all the players are concentrated in one area the map.

The rest of this report is organized as follows: Section 2 introduces the related background of SimMud and implementation. In section 3, we analyzes the simulation results and compare the performance of the static vs. spread scheme. Section 4 concludes this report.

2 SIMMUD IMPLEMENTATION

We use the SimMud provided and add some test variables in the three-staged loop. In "src/server/WorldUpdateModule.cpp", we have implemented test variables to examine the following interested terms for each thread:

- Number of client requests
- Time spent on processing clients' requests

- Number of client updates
- Time sending updates

For the implementation of Simmud, we have listed several simulation parameters in Table 1. Some parameters of the Map in the config file are changed, such as map size, region size (in number of client-areas-of-interest), blocks and resources. These enlarged parameters may make the synchronization more difficult and make the experiment results distinct.

Table 1: Some of the Simulation Parameters

Map Size	Block	Resources	Region Size
48x48	100	50	8x8

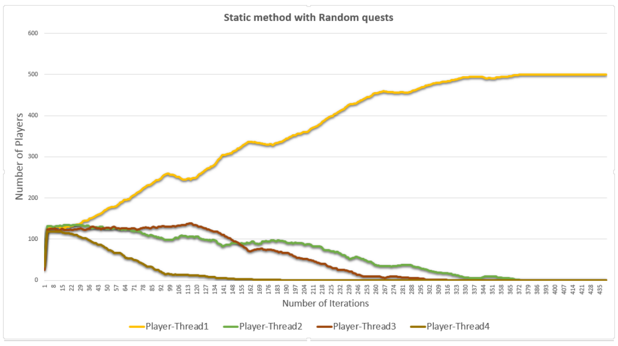
3 SIMULATION RESULT

In this section, we compared the performance between static and spread method in two different scenarios.

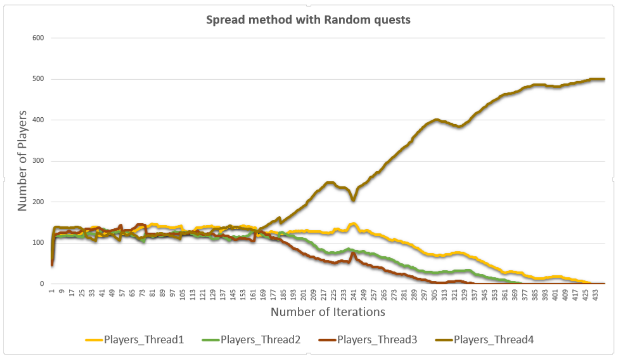
3.1 *No Active Quest*

In this part, we stick to the case where the quest is random, hence the flocking effect should be less.

3.1.1 *Number of players*



(a) Static partition

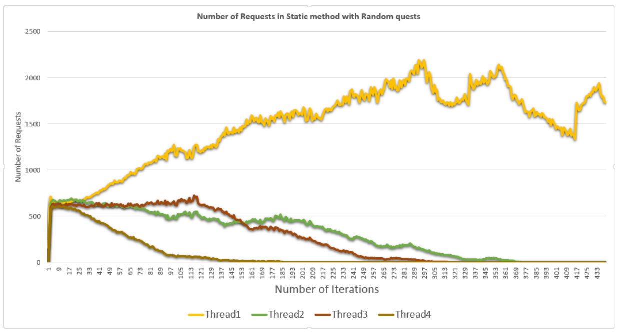


(b) Spread method

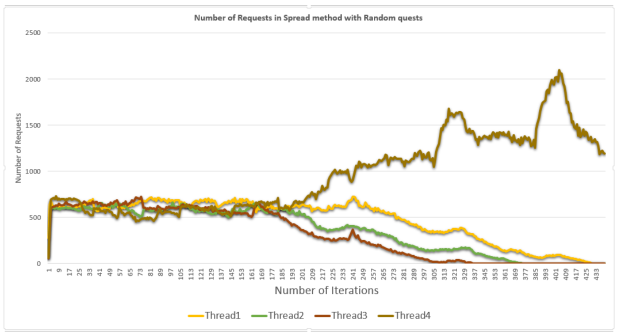
Figure 1: Number of Players for each thread in the random quests scenario

Figure 1 (a) and (b) showed player distribution on each thread for Static and Spread method. From the figure, we can see that Static has uneven distribution of players, especially after 36th iteration and almost all players are concentrated to one Thread after 288th iteration. There is no adjustment after for static method. On the other hand, Spread has players allocated in each thread uniformly for most of the time. After 193th iteration, distribution of players in spread method started to have uneven behavior. This is much better than static. Further more, there is obvious attempt of adjustment at 241th and 329th iteration. Unfortunately, after a certain period of time of an active quest, players eventually are gathered into a single thread. We concluded such behavior is due to players are moving to a single region which cannot be transferred to another thread. This is may be due to the simulation region size is not large enough. The spread method have better performance according to the player numbers on each thread.

3.1.2 *Number of requests*



(a) Static partition

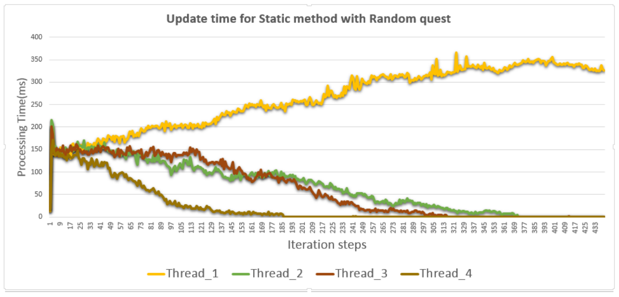


(b) Spread method

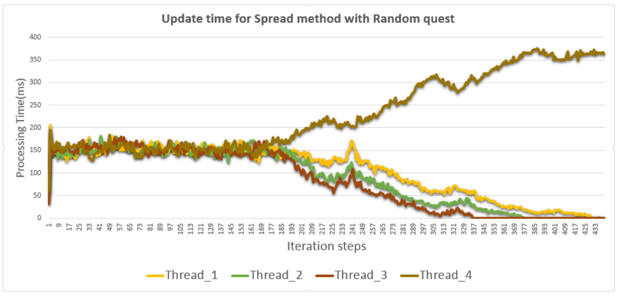
Figure 2: Number of requests for each thread in the random quests scenario

Figure 2 (a) and (b) showed the requests per second in each thread for Static and Spread. They have shown a strong correlation between the player distribution and request distribution. A quest requires the players to move to another region. All the threads in these two schemes deal with approximately 500 requests in no quest phase and after the quests come in, larger amount of requests are sent to Thread 1 in Static and Thread 4 in Spread, which are the destinations.

3.1.3 *Update time comparison*



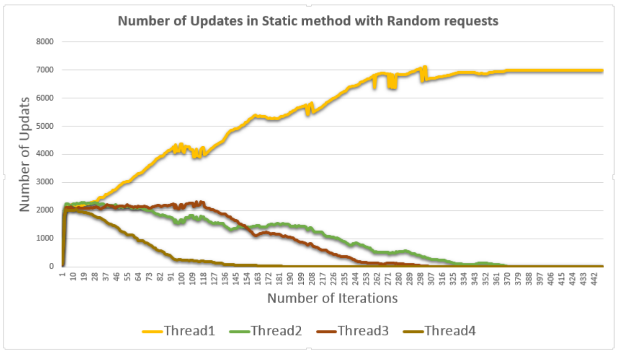
(a) Static partition



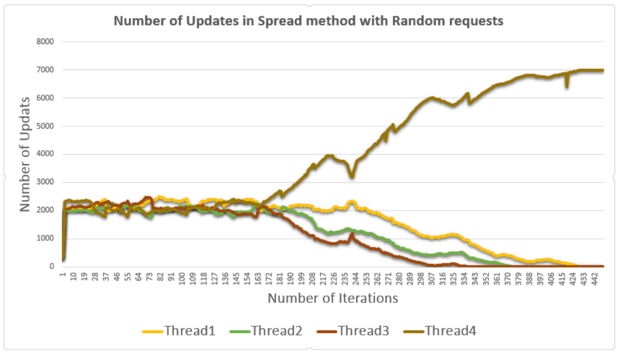
(b) Spread method

Figure 3: Update time for each thread in the random quests scenario

Similarly, Figure 3 (a) and (b) showed the update time for each thread in the random quests scenario. The processing time for phase 3 overall, spread method wins over static.



(a) Static partition



(b) Spread method

Figure 4: Number of updates for each thread in the random quests scenario

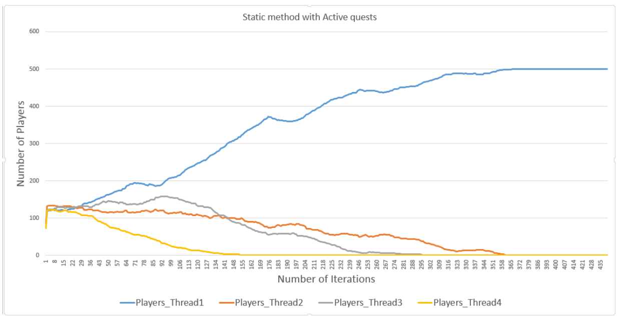
3.1.4 *Number of Updates*

Figure 4 (a) and (b) showed the number of client updates per second in each thread in Static and Spread. In Static, more client updates are sent in one thread and it reaches 7k in 279 iterations (28 - 307). But in Spread, the clients updates to Thread 4 reaches 6k in only 198(181 - 379) iteration. Server reaches the third stage in the 3-staged loop that each thread sends back world updates to clients in shorter time in Spread, which proves that Spread beats Static in processing the quests.

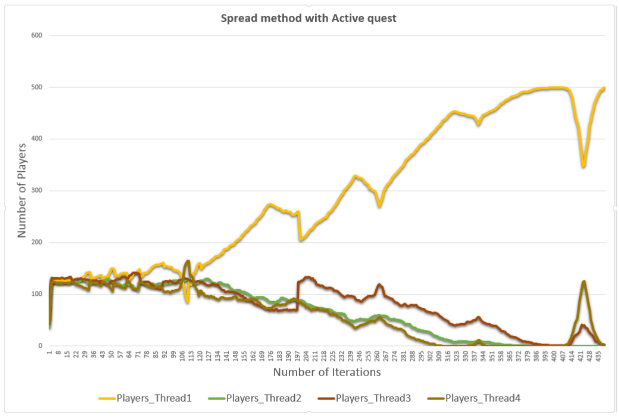
3.2 *Active Quest*

Now we look into the performances in active quest scenario, where the flocking effect would happen way more faster than random quest scenario.

3.2.1 *Number of players*



(a) Static partition

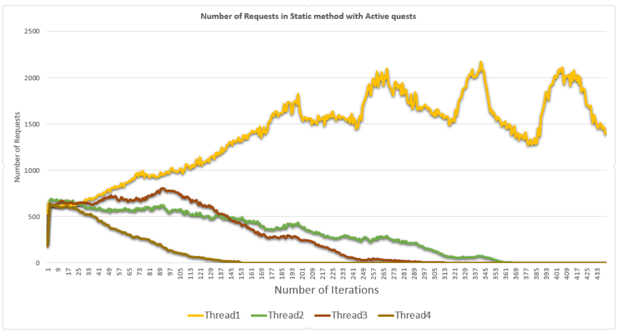


(b) Spread method

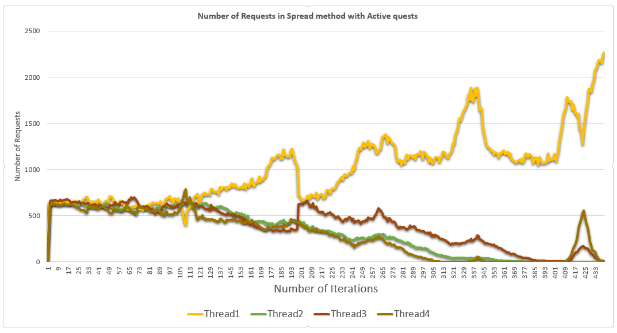
Figure 5: Number of Players for each thread in the active quests scenario

Figure 5 (a) and (b) compared the player distribution on each thread for Static and Spread method in active quest scenario. As we expected, in active quest scenario, the player flocking behavior happens faster. We can see that Static has uneven distribution of players, especially after 29th iteration and almost all players are concentrated to one Thread after 200th iteration. There is no adjustment after for static method. On the other hand, for Spread method, player in each thread started to have uneven behavior after 120th iteration, this is quicker than the other scenario. However, there is obvious frequently attempt of adjustment after since in active quest scenario, the player movement is predictable. We should expect that if the map is large enough and have many regions, the performance of Spread method will be further enhanced.

3.2.2 *Number of requests*



(a) Static partition

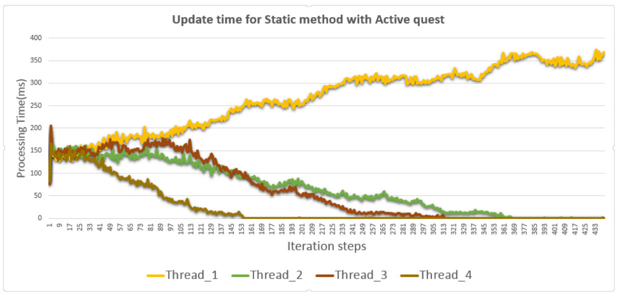


(b) Spread method

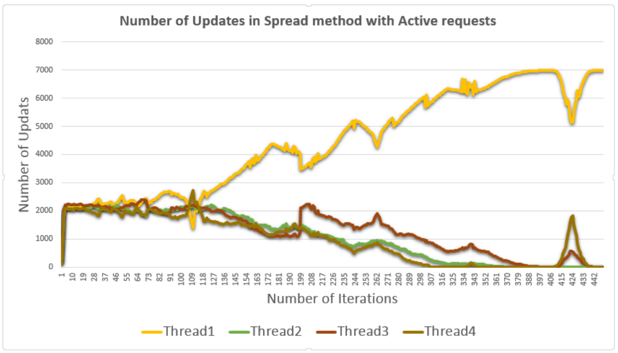
Figure 6: Number of requests for each thread in the active quests scenario

Similar to the random quest scenario, Figure 6 (a) and (b) showed the correlation between player distribution and request distribution. There is fluctuations in the distribution because of the active quest.

3.2.3 *Update time comparison*



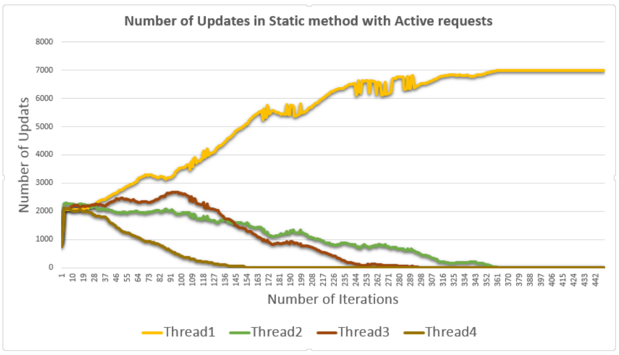
(a) Static partition



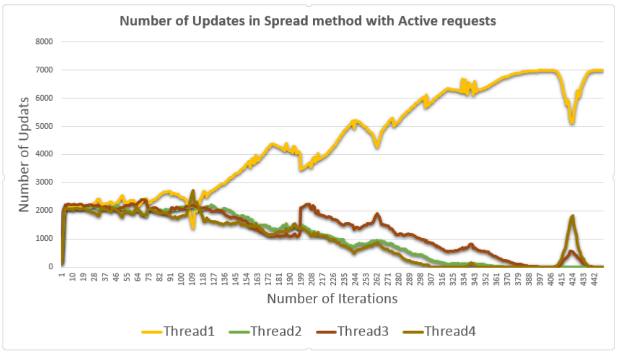
(b) Spread method

Figure 7: Update time for each thread in the active quests scenario

Figure 3 (a) and (b) showed the update time for each thread in the active quests scenario. The processing time for phase 3 overall, spread method wins over static.



(a) Static partition



(b) Spread method

Figure 8: Number of updates for each thread in the active quests scenario

3.2.4 *Number of Updates*

Figure 8 (a) and (b) showed the number of client updates per second in each thread in Static and Spread in the case where there is active quest. In Static, more client updates are sent in one thread and it reaches 7k in 270 iterations (28 - 298). But in Spread, the clients updates to Thread 4 reaches 6k in only 207 (127 - 334) iteration. Server reaches the third stage in the 3-staged loop that each thread sends back world updates to clients in shorter time in Spread, which proves that Spread beats Static in processing the quests. However, in the comparison between the random quest scenario, the difference is smaller.

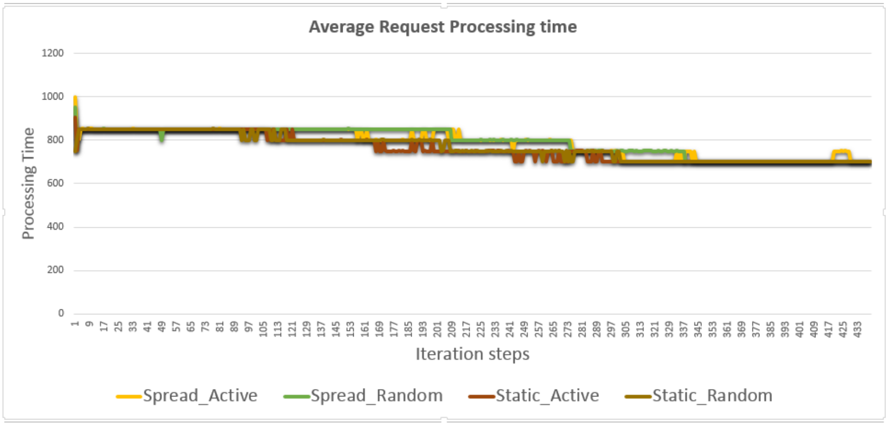


Figure 9: The comparison of average request processing time

Figure 9 shows the comparison in the average actual processing time of client requests between Static and Spread algorithm among the two scenarios. This time is actually only a very small percentage in the entire processing time, it is reasonable that it is quite similar for those two algorithms.

4 CONCLUSION

Overall, the performance of spread algorithm outweighs static algorithm from many aspects. First of all, the Spread method has better performance than Static in the thread/CPU utilization. The load shedding can provide certain adjustment instead static partition cannot. Further more, Spread algorithm processes more requests/updates than Static in a certain time, hence it is faster. However, there is one concern in the experiment, the region size. The size will influence load shedding according to which scenario we are in. If the size of the region is too big, then it is hard to assign it to any thread. If the size is too small, fluctuation happens due to frequent re-assigning overheads. It is quite tricky to choose appropriate region size in order to have better comparison result.