Chapter 6 Experiments and Evaluations

6.1 Assumption

The evaluation in this dissertation mainly focuses on the scalability issues of building a virtual world platform for MMOGs in 4-tiered architecture. Experiments that is done by Chen [29] and Kim [22] are mainly focus on global traffic characteristics and analysis. However, we measure the average response time of client's commands, which is the mostly concern of MMOG players.

The experiments include performance of the NetEngine, performance of the core framework, and performance of real action simulations. The frequency of control commands sent by players depends on the design of MMOG. However, many current models (MMORPG type) allow players to send only one command within any period of one to three seconds. This feature prevents players flooding the server with too many commands, and allows enough time for a reply from the server. Thus, we design the robots to send control commands one per second. Next, the primary variable for evaluating the server's scalability is the number of players. The measurement information is the response times for the control messages.

In experiments round 3, we evaluate scalability by a robot program that simulates real player action. The *movement* action in the virtual world is quite basic and critical for evaluation, because it involves state updates, the replacement of states in a map, and inter-server state migrations. The robots in the program thus send move controls in random directions and receive updates from the VW server, which, in addition, is using corresponding VWLogic. In general, the average response time bellows to

250ms is acceptable result for most real-time MMOGs in Q4 of 2005. All of the experiment is done with JDK version version 1.4.2_01-b06 on Windows XP with service pack 1.

6.2 Experiments

6.2.1 Performance Evaluation of Message-oriented Network Engine

The goal of experiment 1 here is trying to measure the performance of our message-oriented network engine, that is, we try to find out the performance limit of our network engine.

For the hardware configuration, we use one machine as the server and 10 machines as clients. Table 6-1 describes the detail configuration.

Table 6-1. hardware configuration of experiments round 1

Usage	Number	Configuration
Server	1	P4 2.4GHz CPU with 1GB RAM
Client	10	P4 1.6~2.4GHz CPU with 512MB RAM

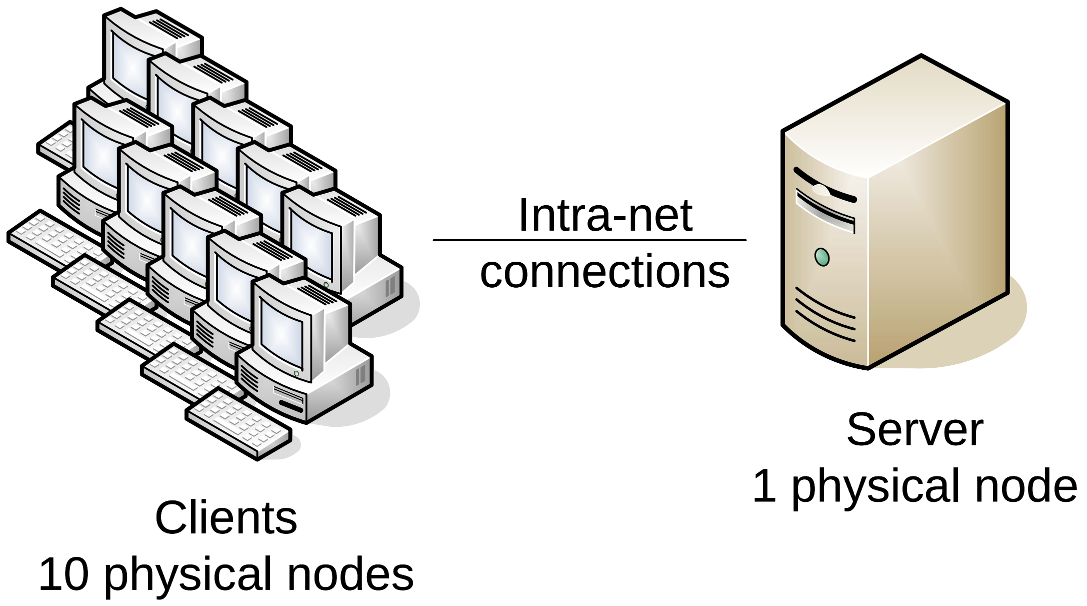

Figure 6-1. Communication architecture of round 1

We conducted the simulation testing by a virtual client generator program to simulate multiple players in a single machine. We run this program in 10 machines simultaneously, and we increase the number of clients by 50 for each run each machine(i.e. the number of client is increased by 500 for each run). Experiments includes 8 rounds, and the client number is increasing from 500 to 4000. Each client send one message per second for 10 minutes. Clients and the server run on the same LAN and all of them are connected with 100 Mbit Ethernet (Fast-Ethernet).

The network engine of server side performs simple reply action. When the server receive a message, it simply send it back to the client. This echo message contains a 4-bytes serial number field, which is used to identify where should the message is sent back to. For each round, we run for 10 minutes and collect the data in the medium 8 minutes as the effective data. Table 6-2 shows the number of clients, the average response time and standard deviation for each test.

Table 6-2. Experiment result of round 1

Client number	Average Response Time (ms)	Standard Deviation
500	0.33	3.00
1000	0.87	5.00
1500	3.03	19.00
2000	4.06	26.00
2500	7.12	31.00
3000	9.34	29.00
3500	26.70	79.00
4000	50.31	516.00

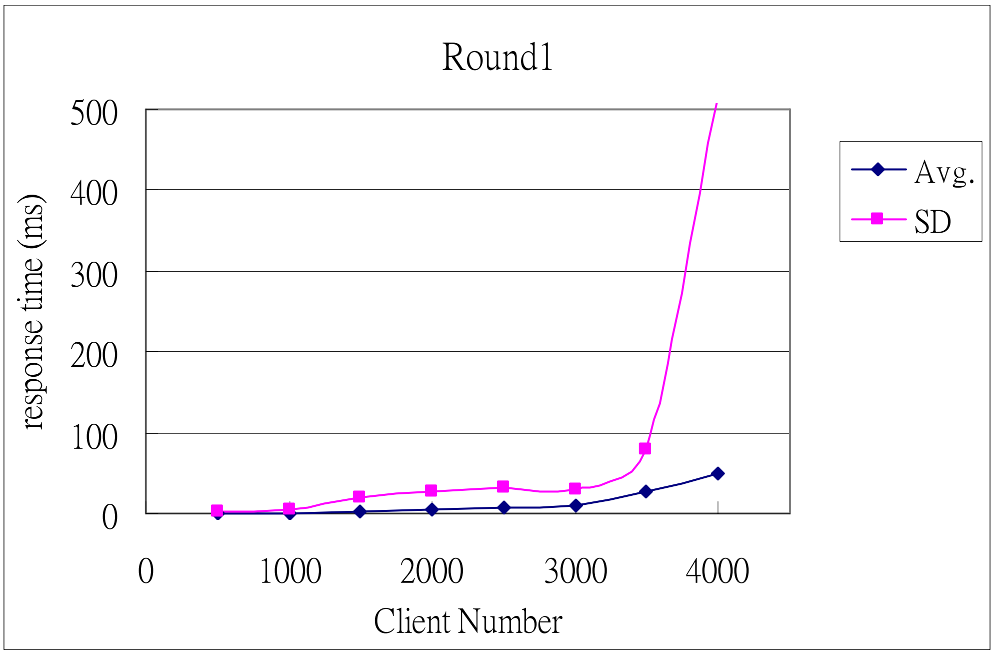

Figure 6-2. Experiment result of round1

As the result shows, the network engine part gets excellent performance under 3000 clients concurrently. The average response time is less than 10 ms and the standard deviation is less than 30ms. It begins to unstable when a server handles over 4000 clients.

6.2.2 Performance Evaluation of DoIT client-gateway-server architecture design

The goal of experiment 2 here is trying to measure the performance of our DoIT client-gateway-server architecture, that is, we try to find out the performance limit of our client-gateway-server design.

During evaluation of experiment 3, it includes all the essential component of DoIT "network" components and object adapters. This evaluation will help us to realize the performance baseline under client-gateway-server architecture. In this round, we also use the simple echo program (echo message is generated by a virtual world game logic). In this round, we prepared one more machine to run as a gateway. This machine has the same configuration as the server.

Table 6-3. Hardware configuration. of round 2

Usage	Number	Configuration
Server	1	P4 2.4GHz CPU with 1GB RAM
Gateway	1	P4 2.4GHz CPU with 1GB RAM

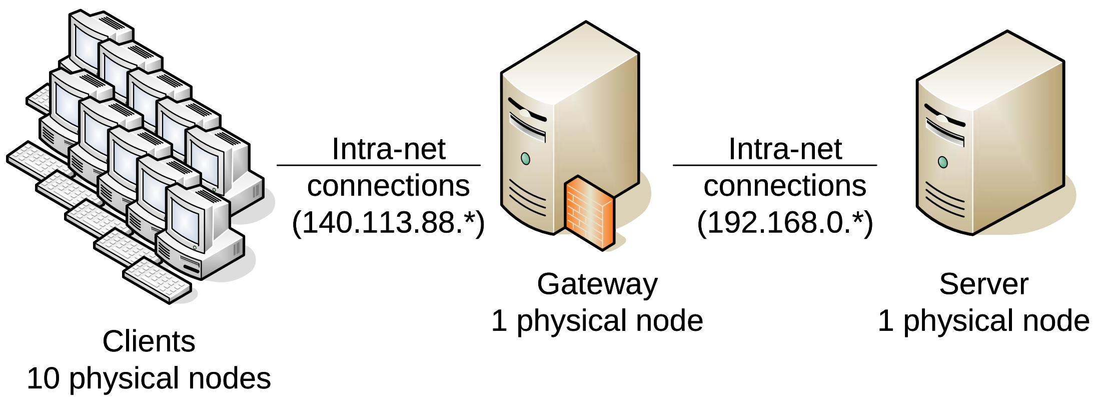

Figure 6-3. Communication architecture of round 2

Also, we use the same client generator program to perform the experiment. The client number also increase from 500 to 4000 each round (total 8 rounds). The most different point is that the test environment was separated into 2 LAN. Clients connected to the gateway in a subnet and the gateway connected to server in another one (see Figure **6-3**). Therefore, the traffic from client to gateways didn't inference the one between the gateway and the server.

Concerning the test program, we designed an echo server as well. The difference is that the echo program was implemented and deployed on DoIT platform as a virtual world logic component. Thus, this program can be considered as the simplest game logic and the performance can be considered as the performance baseline. Similarly, there were 8 rounds test and each run for ten minutes.

Table 6-4. Experiment result of round 2

Client number	Average Response Time (ms)	Standard Deviation
500	12.87	33.33
1000	13.19	25.30
1500	11.46	31.21
2000	19.34	34.91
2500	17.83	35.27
3000	20.17	44.22
3500	25.26	78.29
4000	32.72	122.09

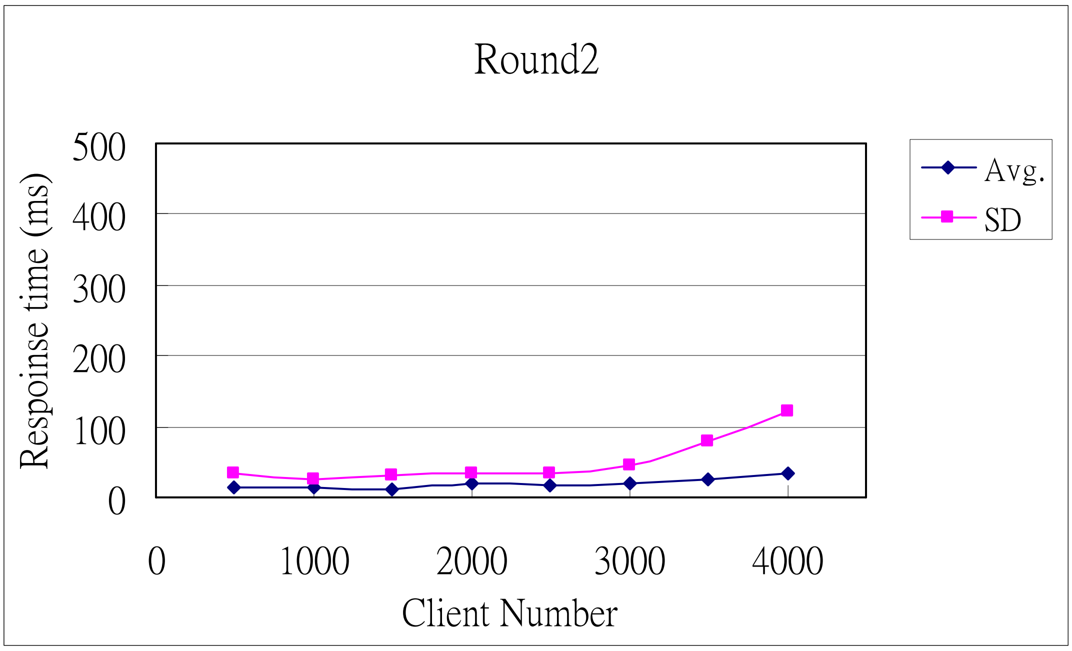

Figure 6-4. Experiment result of round 2

We observe that the result was very similar to the result of round 1. Although the average response was slightly larger than that of round 1, it was also less than 100ms even if the client number reached to 4000. The standard deviation also had the level near 100 ms. It is interesting to note that the average response time in 500 clients was higher than that in 1000 clients. The reason could be caused that it fully exploited the benefit of message aggregation in test of 1000 clients such that it got the better performance than that in 500 clients. The similar result also appeared in round 3.

6.2.3 Performance Evaluation of a virtual world simulation

In the 3rd experiment, we try to simulate a simple real game environment and observe the performance. We used two computers for servers, two for gateways, ten for clients. The detail configuration is list as follow.

Table 6-5. Hardware Configuration of round 3

Usage	Number	Configuration
Server	2	P4 2.4GHz CPU with 1GB RAM
Gateway	2	P4 2.4GHz CPU with 1GB RAM
Client	10	P4 1.6~2.4GHz CPU with 512MB RAM

To simulate the real game world, we implemented the most significant game logic "*movement*" in the silmuation program. However, the different implementation of the move logic will affect the result obviously. So we should make more effort to describe the implementation detail of move logic. We should first define the term AOI (Area of Interest) as the area in which all events are interesting to a given game object. This game object is interested in all the game objects withn AOI. Assume that one avatar moves right, we should send the player update message to all the game objects inside the AOI plus 1 unit. This is because the game objects in the AOI of new location should be interested in the new state of this avatar. The game objects in the leftmost of the AOI of the old location should receive the player update to indicate that this avatar was move out of their AOI. In addition to send updates to the surrounding objects, the avatar also needs to extend his eye sight. So the states of the game objects in the oblique line area should also be sent back to avatar. The AOI can be set dynamically in package mmog.doit.gamespaceutil.

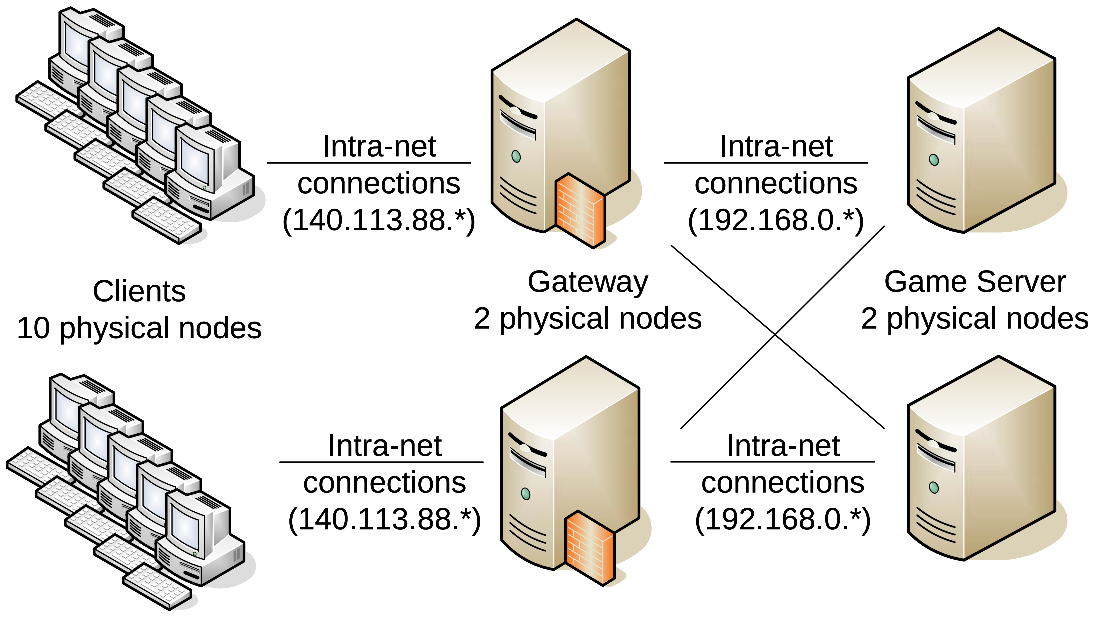
Intra-net connections (140.113.88.*)
Intra-net connections (192.168.0.*)
Clients 10 physical nodes
Gateway 2 physical nodes
Game Server 2 physical nodes
Intra-net connections (140.113.88.*)
Intra-net connections (192.168.0.*)

Figure 6-5. Communication architecture of round 3

The Figure 6-5 illustrates the communication architecture of this round. The virtual world was divided into 4 equal size regions plus a login region. They were deployed to 2 game servers. In addition, we provided two gateways for clients to connect in. Similarly, we put the two servers in the private network. The gateway was responsible for routing messages between internal and external network. Due to multiple regions, it is possible that one avatar will migrate from one region to another, the avatar migration also be implemented in this round.

Concerning the test program, there were also 10 machines running the client generator program. We separated them into two groups. Machines in the same group connected to the same gateway. There were 8 runs in this round. The number of clients is increase with the number of 50 in each client generator program. Therefore, in the extreme case, the gateway had 2000 clients connected concurrent at most, and the game platform had 4000 clients in the virtual world concurrently. Furthermore, we performed different tests for different map size and AOI size. The map had 500 x 500 and 1000 x 1000 two different sizes, and AOI had 9 x 9 and 16 x 16 two different sizes. We hope to realize the performance in different environment. Besides, in each test, we pick the updates with avatar migration to do further analysis. The following figure and table (figure 6-6 to 6-9, table 6-6 to 6-12) shows the statistic of our raw data.

Table 6-6. Average Response Time – Total

Average Response Time - Total				
	500x500 9x9	1000x1000 9x9	500x500 16x16	1000x1000 16x16
23.74	23.26	23.16	27.82	23.74
21.80	23.27	24.56	20.39	21.80
27.21	17.95	24.94	22.26	27.21
24.83	12.24	34.36	18.92	24.83
22.74	15.83	54.66	31.25	22.74
30.86	23.22	240.14	24.02	30.86
66.95	21.98	491.99	46.49	66.95
125.65	29.22	4188.32	55.79	125.65

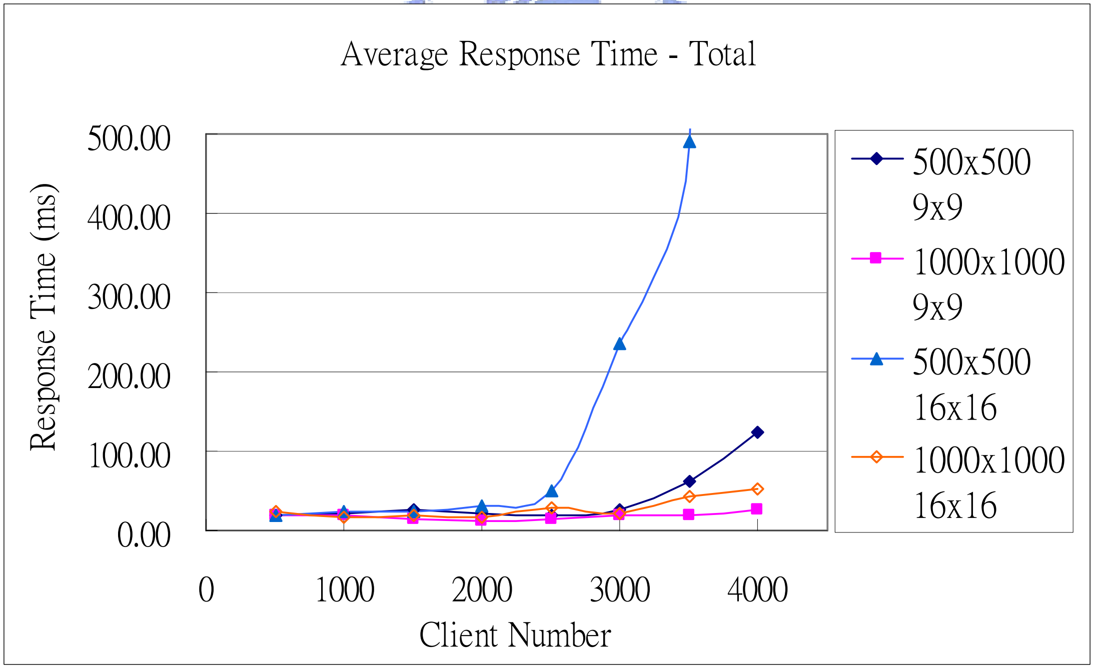

Figure 6-6. Average response time – total

Table 6-8. standard deviation – total

Standard Deviation - Total				
	500x500 9x9	1000x1000 9x9	500x500 16x16	1000x1000 16x16
500	56.43	49.85	52.68	62.79
1000	56.50	43.44	60.12	42.03
1500	62.46	41.13	58.00	53.07
2000	52.98	35.05	74.65	41.18
2500	56.68	40.04	175.69	54.88
3000	78.40	46.01	834.17	97.16
3500	271.01	56.03	1013.94	141.07
4000	591.70	102.10	5712.77	250.44

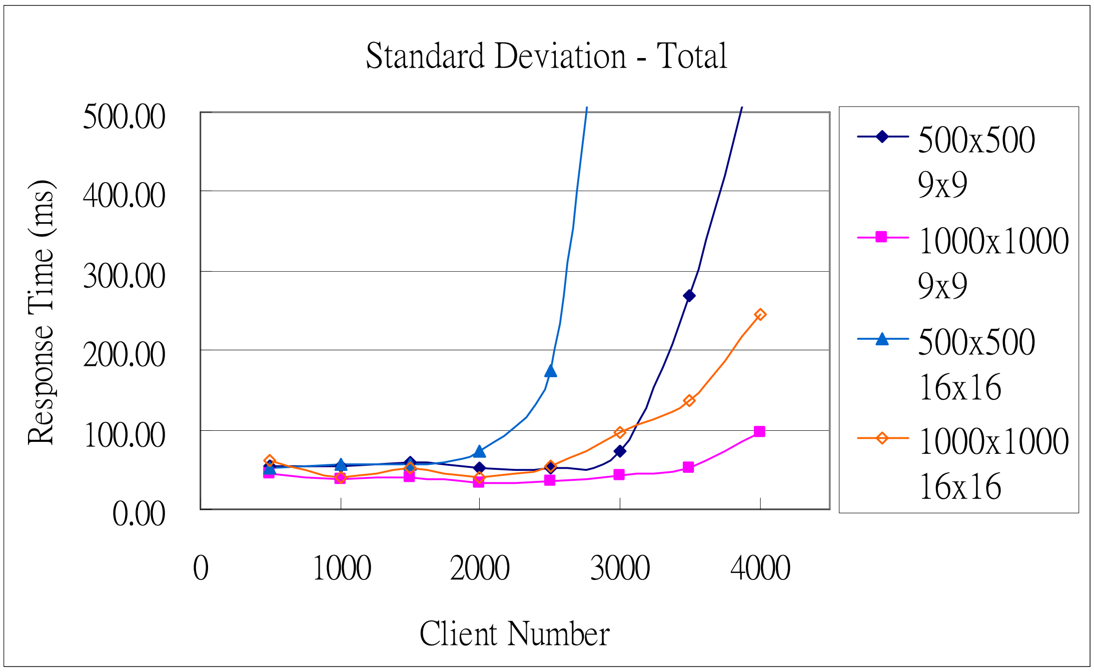

Figure 6-7. Standard deviation total

Table 6-9. Average response time – migration

Average Response Time - Migration				
	500x500 9x9	1000x1000 9x9	500x500 16x16	1000x1000 16x16
500	45.08	30.11	37.23	51.67
1000	31.16	33.68	42.17	35.15
1500	29.49	30.70	46.43	23.94
2000	38.05	23.31	40.89	33.80
2500	31.76	25.71	78.27	43.65
3000	38.50	34.33	201.92	33.16
3500	74.61	31.57	628.70	63.66
4000	119.16	36.90	5187.59	63.68

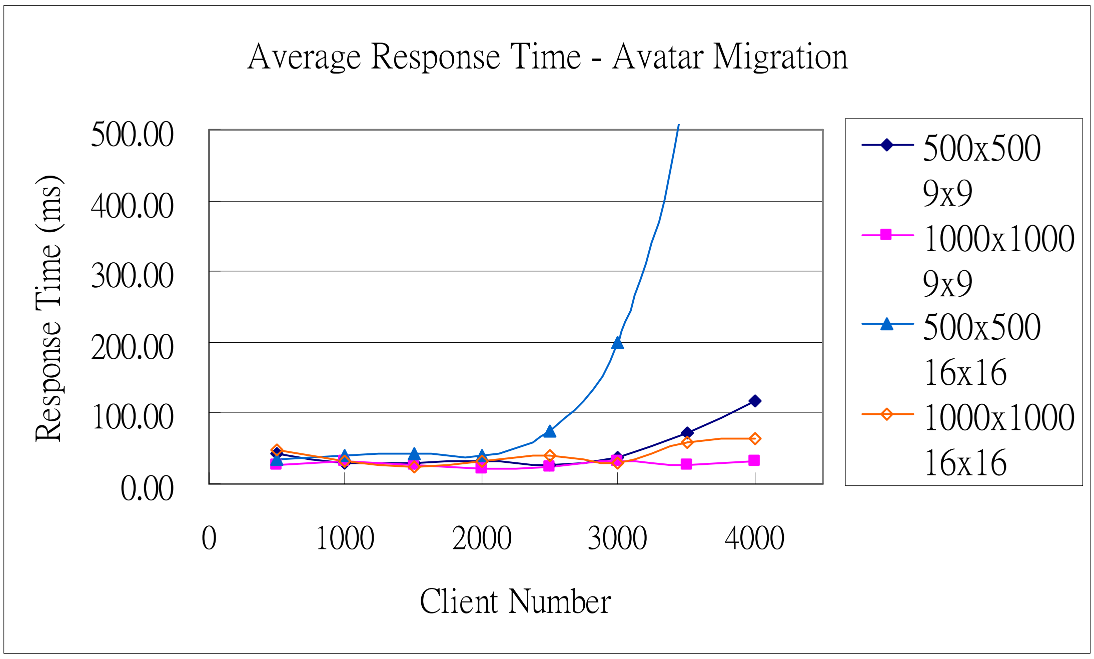

Figure 6-8. Average response time – migration

Table 6-10. Standard Deviation – Migration

Standard Deviation - Migration				
	500x500 9x9	1000x1000 9x9	500x500 16x16	1000x1000 16x16
500	77.70	49.53	69.24	86.87
1000	60.79	52.61	73.01	62.75
1500	60.76	56.85	82.84	56.46
2000	69.03	55.47	82.88	63.86
2500	59.35	51.17	380.69	66.56
3000	76.13	63.01	913.79	100.30
3500	198.03	53.26	919.04	144.63
4000	483.43	94.72	6280.69	211.38

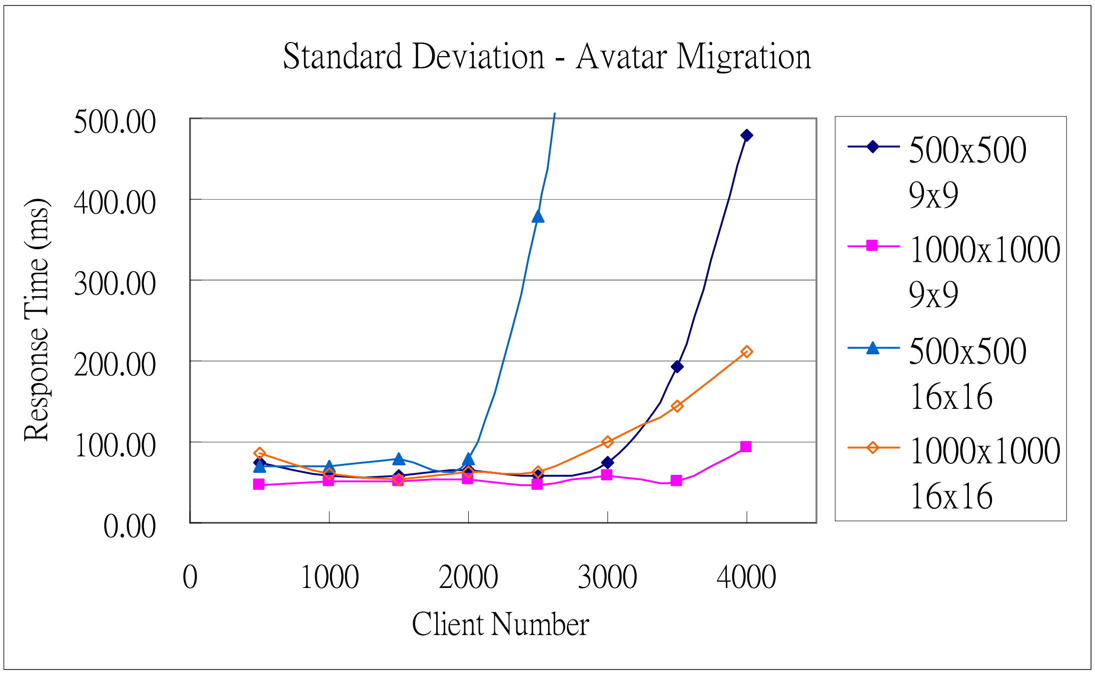

Figure 6-9. Standard Deviation – Migration

Table 6-11. CPU load

	500x500 9x9		1000x1000 9x9		500x500 16x16		1000x1000 16x16	
	Gateway	Server	Gateway	Server	Gateway	Server	Gateway	Server
500	7	3	5	3	4	5	7	4
1000	24	4	14	4	6	8	14	4
1500	36	12	19	3	11	7	24	8
2000	38	10	24	12	22	12	26	10
2500	40	11	40	11	52	17	43	12
3000	79	11	59	12	92	21	69	18
3500	96	16	87	16	98	27	89	25
4000	99	37	89	17	101	33	92	29

Table 6-12. The total amount of messages forwarded per second in a gateway

	500x500 9x9	1000x1000 9x9	500x500 16x16	1000x1000 16x16
500	629	553	653	592
1000	1473	1098	1639	1180
1500	2463	1692	2867	1844
2000	2983	2300	4399	2694
2500	4115	2936	6124	3439
3000	5178	3573	7941	4325
3500	6466	4299	9610	5318
4000	7815	5059	12841	6315

6.3 Discussion

In this section, our major discussion focuses on the phenomenon encountered in experiments.

Aggregation design of gateway gains better scalability.

From the result of experiment 1 and 2, we can say that, the message aggregation mechanism and reduction of connection between server and gateway helps DoIT serves more clients with better performance. In Table 6-2, when a network engine of server handles 4000 connections from 4000 clients, then standard deviation is 543ms, and the average response time is 55.11ms. However, in DoIT client-gateway-server architecture, in Table 6-4, the standard deviation reduces to 135ms, and the average response time reduces to 33.22ms. Although the average response time is a little getting higher by aggregation delay of gateway relay mechanism, it is an acceptable improvement.

The bottleneck of Gateway performance

In Figure 6-7 and Figure 6-8 we can observe that different map size and different AOI size lead to different result. In all the tests, as we can expect, the data of map size 500 x 500 and AOI 16 x 16 got the worst result. In this configuration, the average response time grows steadily under 2500 clients concurrently. But it starts to have dramatic growth after 2500 clients. Concerning the two configuration of 1000 x 1000, 16 x 16 and 500 x 500, 9 x 9, there is no obvious growth until the client number reaches to 3500 clients. But the amount of growth is steady and slow. Concerning the configuration of 1000 x 1000, 9 x 9, the average responses time is almost under good result throughout the test.

If we observe the percentage of CPU usage in Table 6-11, we can find that the performance bottleneck is on the gateway. The CPU usage is always under 50% in the servers. We take a future look. We recorded the update sent per second in the gateway at the Table 6-12. We consider it have tight relation to the performance. The main game logic is *movement* message. According to the *movement* game logic described above, the number of update messages should be a function of total clients, map size, AOI size, and clients in the gateway. We first consider how many update messages are sent when one move message is process. First of all, the update message should be sent back to the avatar himself.

We calculate the total messages per second processed by a gateway by the given update messages per second plus the command message sent by clients. According the result we get above and map it to this diagram, we can find that a gateway can bear 5000~6000 messages per second with excellent performance.

Finally, we observe the impact of avatar migration in Figure 6-9 and Figure 6-10. The trend of updates of avatar migration is very similar to the general message updates. The average response time is 1.5 times larger than general message. But it is acceptable because the avatar migration does not happen often.

According the result we get above and map it to this diagram. The results show that gateway performance drops dramatically when the gateway needs to process over 6000 messages per second (the number of clients handled by a single gateway is 1500).We can generally state that if a server cluster is required to successfully handle simultaneously about 10,000 players, we need a total of 10 nodes of a server cluster (6 gateways and 4 cell servers).

If we could improve the performance of the gateways, we could greatly decrease the number of gateways and lower the cost. Gateways in MMOG service now play an important role. Not only do they forward packets, but also provide security. Moreover, MMOGs with mobile devices support might soon be available. As can be seen, gateways carry a heavy responsibility and arranging their performance and functionalities is an important task.

Overall performance evaluation conclusion

According to information received at Q4 of 2005 from the games industry, one server cluster (consisting of about twenty computers, including proxy/gateway and server) can serve about 9000 players simultaneously. Therefore, the results for the scalability of our DoIT platform were good. We can achieve the same simultaneous online players by using only 10 PC-based machines.