Practical Middleware for Massively Multiplayer Online Games

A massively multiplayer online game (MMOG) lets thousands of players interact simultaneously within a virtual world via the Internet. Middleware plays an important role in the development of next-generation MMOGs, which must be built on platforms that address not only the service aspect, but also code maintainability and development for programmers. The authors' compact, high-performance message-oriented middleware has a code-generation programming model that is designed to address many of these problems.

Tsun-Yu Hsiao and Shyan-Ming Yuan
National Chiao Tung University

Massively multiplayer online games (MMOGs) let thousands of players (between 6,000 and 10,000, according to general reports from game companies) simultaneously interact in persistent, online, multiplayer-only worlds. The most popular examples are Sony's EverQuest, NC Soft's Lineage, and Blizzard's World of Warcraft (see www.mmogchart.com for the analysis of subscription growth). The MMOG sector generates the majority of online gaming revenue, especially in the Asia/Pacific region, which has the largest market worldwide. According to IDC's Asia/Pacific Online Gaming report, the market has grown over the past several years, commanding US$1 billion in subscription revenue in 2004, and will more than double by 2009 (www.idc.com/getdoc.jsp?containerId=pr2005_08_04_110417).

This rapidly growing MMOG market signifies entertainment computing technologies' importance, and has forced game makers to step up competitively. However, several challenges exist for vendors with regard to development, service, and security. Given that these companies' ultimate goal is to increase profits, they need a low-cost, robust, scalable, secure, attractive MMOG service to improve development and decrease time to market.

Middleware helps programmers manage the complexity and heterogeneity of distributed computing environments.[1,2] To address the challenges that vendors (and even players) face when developing and maintaining MMOGs, we've developed the Distributed-organized Information Terra (DoIT) middleware platform. We began this project in 2002 at National Chiao

Tung University's (NCTU) Department of Computer and Information Science. This article explains our experience and outlines the components essential to crafting a practical MMOG middleware.

MMOG Development Challenges

Creating a MMOG typically takes two to three times as long as creating and launching a traditional single-player game (two to three years versus 9 to 12 months). A MMOG project from a major PC games studio can require a five-year timetable and hundreds of people's efforts prior to release. (For example, Electronic Arts/Westwood Studio's Earth & Beyond MMOG didn't appear until 2003, even though its blueprint stage began in 1998 — and EA shut it down in September 2004 to focus resources on new games.) Westwood and other game vendors produced single-player or matching games (network games involving between 16 and 64 players, usually implemented via peer-to-peer technology) throughout the nineties, but competing in today's MMOG market requires such vendors to utilize distributed technologies and concepts and familiarize themselves with network issues. (Korean vendors' efforts in this arena have led the country to its current position as the preeminent exporter of MMOGs worldwide [www.idc.com/getdoc.jsp?containerId=IDC_P6396].)

Once a MMOG becomes available to gamers, vendors often encounter problems with versioning (fixing bugs, adding content, and so on). Furthermore, most MMOGs are client-server programs; game companies generally host their virtual worlds on dedicated servers or ISPs' Internet data centers in chosen collocations (in which they rent a secure space with reliable high-speed network connectivity in data centers while maintaining their own equipment and services). Once a game begins online operation, service providers must make its content attractive enough to induce users to stick with it. Thus, developing the content is only the beginning — continuity and profit depend on quality of service (QoS) and attractiveness to end users over time. Service reliability, availability, restorability, and scalability are vital issues for success.

Security is also critical because MMOGs are just as vulnerable to hackers as any other Internet services. Fortunately, industry-standard firewall and intrusion-detection technologies can reduce the effects of most attacks. In-game (application-level) cheating and attacks on content protocols (the players' command formats and their contents) are troubling issues, however, because fake and illegal protocols affect QoS and the games' fairness. In the Asia/Pacific region, the most common security problem comes from users' running illegal plug-ins or cheat programs (such as artificial intelligence robot programs) to play MMOGs for easy money or experience points. This is both unfair to other gamers and degrades QoS for vendors, who must do more server-side checks to prevent cheating. Yet, counterattacking these programs is no simple task, given that attackers have the client program, and can thus access the execution code and restore and analyze the protocol's format. Widespread culprits in China have become such a significant nuisance that vendors can't simply ignore them because MMOG players are likely to quit rather than continue with an unfair or vulnerable MMOG service.

MMOG Platform Architecture

By using middleware to build MMOGs, programmers can manage the complexity and heterogeneity of distributed computing environments, and possibly address the development, service, and security issues we've outlined here. A virtual-world platform solution based on distributed technologies can help vendors build MMOGs quickly, without having to worry about network-transmission issues or collaboration between servers or clients.

Using middleware to support a MMOG isn't a new idea. Researchers have devoted much effort to building frameworks to support MMOG development,[3–5] generally as distributed systems with *n*-tier architectures. Figure 1 shows an example of such a system with a 4-tier architecture comprising a client, proxy/gateway, cell server, and database:

- Gamers control the client applications, which deploy and run graphics, the user interface, and network communication.
- The proxy/gateway is important not only in forwarding packets but also in providing security functionalities (as it can work like a firewall or run the protocol-checking procedure) for cell servers and cooperating with portals or billing systems.
- The cell server houses, maintains, and executes the virtual world. It also receives players' control commands and verifies, computes, updates, and then forwards new player states to all gamers who will be affected by these commands (for example, a successful `move` command will generate a player's new position — player state — on the virtual-world map.

- The database stores periodically updated player states to ensure that players continue to exist in the virtual world.

Most commercial products use this generic architecture because the proxy's security functionalities facilitate antihacking efforts in both MMOG and enterprise-computing environments and because it provides a suitable mechanism for billing players.

Other research and open-source middleware projects for MMOGs have proposed similar architectures. Mauve and colleagues' work indicates that the proxy can take over certain server functions to help distribute the processing load, prevent cheating,[4] and — using a hardware solution to speed up processing — improve scalability.[5] We can also view the proxy as an extension of the server that can provide flexibility during game design and deployment. Moreover, proxies can help assure cell servers that they can trust information coming from the proxy. (For more information on other MMOG middleware technologies, see the "Related Work in MMOG Middleware Projects" sidebar.)

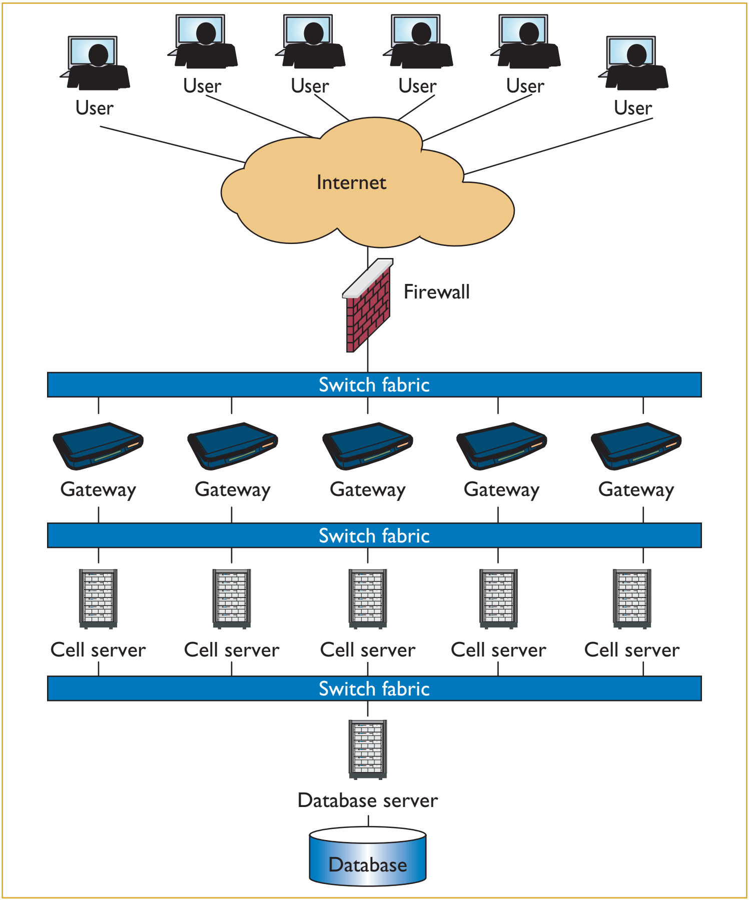


Figure 1. Generic 4-tiered MMOG architecture. Gamers (users) control the client, which runs graphics and the user interface. The proxy/gateway acts as security for the cell servers, which house, maintain, and execute the virtual world. The database periodically stores player states.

Practical Next-Generation MMOG Middleware

Current MMOG technologies focus on providing a scalable, reliable, fault-tolerant, low-cost, load-balancing, single-sign-on, secure framework for building seamless virtual worlds — no shards (identical copies of online games on different server clusters), specific servers, or zones, so that all players essentially exist in the same game world. However, only a few new MMOGs have been based on such frameworks, and fewer are currently in operation. For this reason, rather than developing solutions themselves, game vendors who are unfamiliar with distributed technologies would benefit from easy-to-use MMOG middleware that addresses QoS and helps provide code that's easy to develop and maintain. A MMOG platform should satisfy four essential "ease of" requirements, in addition to the aforementioned gateway, functionality, performance, and security needs.

Ease of Development

The MMOG platform should

- provide easier and faster ways to develop content,
- hide the underlying network programming from the content programmer (which simplifies actual game programming), and
- provide an API that's simple and flexible enough to let developers focus on content and presentation.

Interface definition language-based programming paradigms with automatic code-generation mechanisms would be a good choice to satisfy these requirements.

Ease of Deployment

Most MMOG vendors frequently update their game content because doing so provides better, more attractive service. The middleware platform's architecture should provide both servers and clients with straightforward ways to deploy content, especially in collocation environments, in which all content must be updated remotely.

Related Work in MMOG Middleware Projects

Several significant middleware technologies aim to support MMOG application; most of them are commercial products.

The Butterfly Grid

Unlike traditional MMOGs, which rely on a rigid centralized-server approach, Butterfly.net (www.butterfly.net) is a self-managed, fully meshed network that shifts processing to idle resources as needed. To better compete in the MMOG market, Butterfly.net built the Butterfly Grid — based on the grid computing model — which consists of two clusters of roughly 50 IBM eServer xSeries servers. The grid uses the DB2 universal database, the WebSphere application server, and IBM's global services for implementation and hosting. At a high level, Butterfly.net uses the Globus Toolkit to integrate these components into the grid's fabric.

BigWorld Technology

Microforté spent more than three years and US$8 million to research and develop a complete MMOG middleware solution. Its BigWorld Technology (www.bigworldtech.com) includes a scalable, reliable, customizable, and fault-tolerant server infrastructure that can handle millions of players; a client 3D engine; and a set of tools, such as a model viewer and particle system editor, for building and managing MMOG worlds. The BigWorld server has adaptive load management, on-the-fly reconfiguration, and an optimized RPC mechanism. General consensus seems to be that BigWorld is the only vendor to cover everything from client 3D to server back-end development, providing a complete solution.

The Internet Communications Engine

ZeroC Internet Communications Engine (ICE; www.zeroc.com)[1] is a highly efficient middleware platform that's as powerful as Corba, but without its flaws. In addition to avoiding Corba's complicated core, ICE supports more features that aid MMOG development and maintenance, such as object-state versioning, the distribution of software updates, efficiency protocols, definitions of the persistent support setting in the Slice interface definition language, and multilanguage support.

Massiv

Massiv is an open-source distributed game middleware that aims to simplify the development of distributed persistent MMOGs (http://massiv.objectweb.org/). Like ICE, its programming paradigms are IDL-based. Moreover, Massiv is designed to run on multiple servers located throughout the world. Therefore, to keep a consistent status between servers worldwide, it addresses issues of security, latency, and time synchronization.

Reference

1. M. Henning, "Massively Multiplayer Middleware," *ACM Queue*, vol. 1, no. 10, 2004, pp. 40–45.

Ease of Maintenance

Placing a MMOG online is only the beginning of the service challenge: easy maintenance and monitoring are subsequent requirements that vendors must heed to support and retain customers at low cost. For example, to ensure fairness in the virtual world, a tracker program that could automatically report uncommon changes in players' states would benefit game vendors.

Ease of Change

Many popular games encounter security problems on one hand and service issues (such as server overload) on the other. Making changes to the content protocol to fix bugs or update content is common, but even if no new content is added, changing protocols can make hacking them more difficult. A MMOG middleware should make it easy for vendors to make changes to content protocol.

Performance and Load Balancing

Building a seamless game world requires a tight, dynamic collaboration between cell servers. Game-interaction processing – boundary updates and user-state migration, for example – is spread across a network of server farms. Most current models are built on shards, but a middleware that can support a seamless game world is very important for next-generation MMOGs to help maintain QoS whenever the number of online players increases. It also facilitates scalability by allowing vendors to add more servers to a cluster. Given that the geography of player states is essential in an online world, a good dynamic load-balancing scheme is also desirable.

The Distributed-organized Information Terra Platform

We began developing the DoIT platform at NCTU's distributed computing system laboratory in October 2002 in an attempt to build a practical, efficient middleware for MMOGs that meets most of the requirements outlined in the previous section. We implemented it in Java, using a 4-tier middleware architecture like the one in Figure 1, and the code-generation engine supports both Java and C++. We further chose to use message-oriented middleware (MOM) technology rather than an RPC-based architecture.

The first reason for this decision is that MMOG behavior is generally event-driven — real players send commands to control virtual players, and the changes one virtual player makes affect what other players view on their monitors. MOM works well with event-driven models, and can therefore support this scenario directly. Second, in most cases, the client can execute multiple codes in parallel. For example, when sending commands to the server, clients can execute appropriate prediction algorithms to give MMOG players a smoother gaming experience rather than blocking and waiting for the corresponding updates from the server. Furthermore, MOM technology lets the gaming platform decouple the versioning relationship for both clients and servers, in contrast with RPC, in which any interface change requires recompiling the whole program. The MOM-based architecture makes it easy for clients following the protocol definition to connect to a MMOGs built on it.

DoIT has two main features that differentiate it from other frameworks or solutions. First, its customizable, message-based network engine reduces complexity for MMOG application programmers. We tried to stick to the idea that "simple is better." Our network engine helps developers focus on creating simple, high-impact protocol descriptions without worrying about network programming.

We also introduced a code-generator model to help generate corresponding content protocols. The content developer begins by describing the message format in an XML file. Our code-generation engine then parses the protocol description and generates corresponding message-factory and handler codes for both clients and servers. Next, the developer implements the detailed content code within these pregenerated handler classes — defining how to handle a client's `PLAYER_MOVE` message, for example. Finally, both the client and the server can deploy the codes.

Figure 2 (next page) shows an example of a protocol defined in this platform. In Figure 2a, the content protocol (number 1) is `LoginMessage`, and the parameters include a long type id and a string-type password. Figure 2b shows the protocol that our code generator outputs. We believe that defining the protocol precisely makes the generated protocol handlers more compact, thus increasing network-transmission performance without as much overhead as we'd encounter in Corba.

To address security and content-updating issues, our platform models the content-oriented protocols as sets of message fields; the code-generator engine randomly shuffles the fields, thus increasing protection against hacking message-oriented protocols and faking messages. Game vendors undergo this automatic updating process periodically and easily, and it shouldn't give them, or players, too much trouble. Although this won't provide total protection from hacking, it makes attacks more difficult. Moreover, including both messaging protocol and encryption algorithms, such as Secure Sockets Layer (SSL), can help prevent attacks.

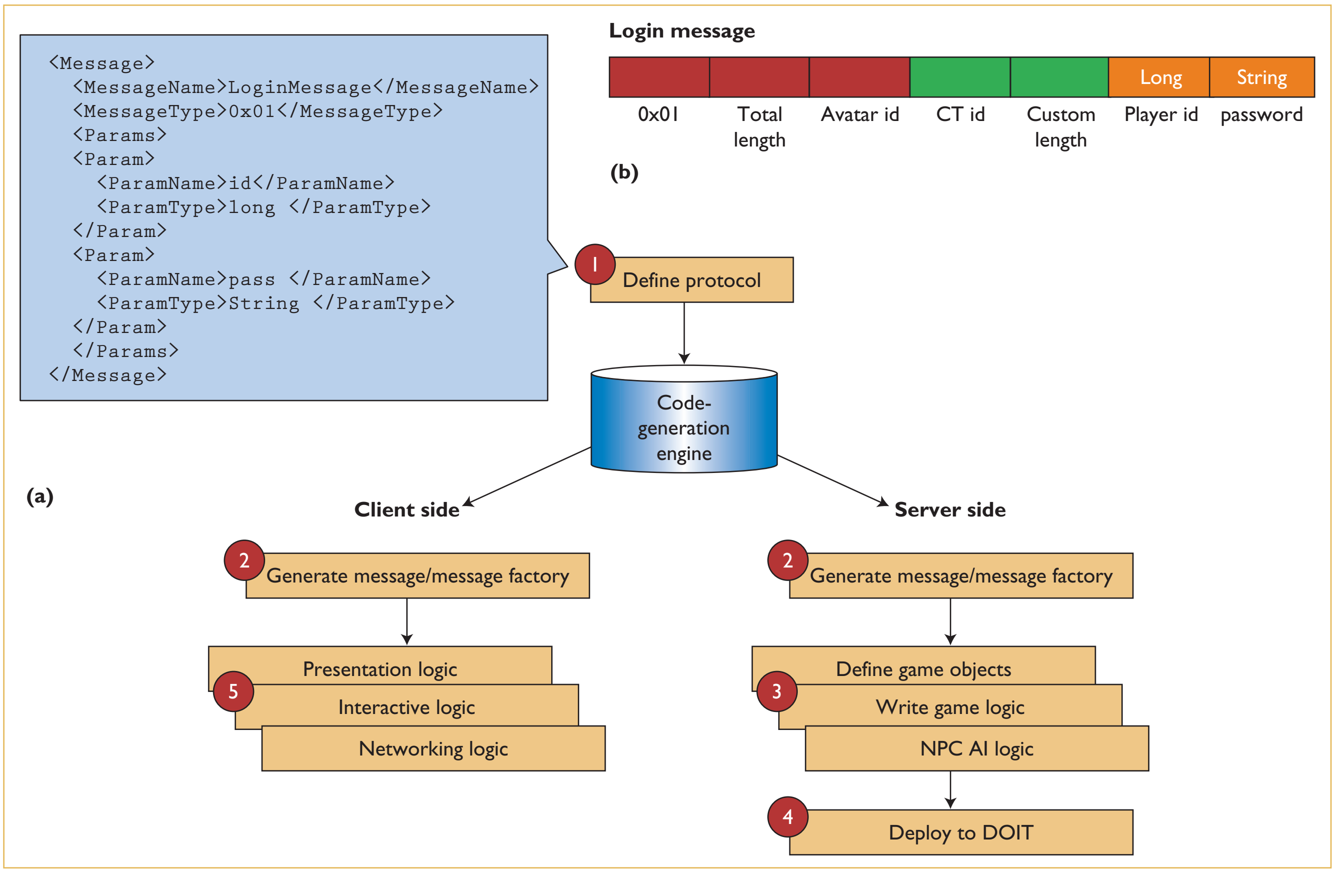
The second unique feature DoIT includes is a lightweight, real-time virtual-world logic (VWLogic) adapter, which simplifies content development, deployment, and revision. The network engine receives control messages from the game's client program and then asynchronously puts the messages into the VWLogic adapter, which demultiplexes the messages for each corresponding VWLogic component. For example, gamer A sends a `move` control message through the client program's network engine to the server's network engine, which then asynchronously puts this message into the VWLogic adapter. The adapter will find move-related VWLogic (that is, handler classes) to process the control message. After running the computation, the VWLogic sends an update request, such as "move player A to new location (x,y)" to the virtual-world container. Finally, the container sends the update to the clients through the network engine.

The MMOG developer creates the VWLogic, which can be plugged into or removed from the adapter at runtime. Developers can change the class, for instance, by replacing bug-containing VWLogic with a bug-free class, even when the entire MMOG application is running. Asynchronous-adapter design makes such "hot swaps" possible on the DoIT platform and more maintainable once the service is online.

Simulation Results

We conducted simulation testing in June 2003 and June 2004, using the 39 nodes of Pentium-III-level computers in the department's computer room. All computers were in the same subnet with Fast Ethernet (100 Mbps), and were isolated to the Internet.

It's difficult to quantitatively evaluate metrics such as ease of development or change, so we focus primarily on evaluating scalability issues. To do this, we use a robot program that simulates real player action. The virtual player's `movement` action in the virtual world is quite basic and important to evaluate because it involves state updates, the replacement of states in a map, and inter-server



```
<Message>
  <MessageName>LoginMessage</MessageName>
  <MessageType>0x01</MessageType>
  <Params>
  <Param>
    <ParamName>id</ParamName>
    <ParamType>long </ParamType>
  </Param>
  <Param>
    <ParamName>pass </ParamName>
    <ParamType>String </ParamType>
  </Param>
  </Params>
</Message>
```

Figure 2. Code-generator model showing (a) the content protocol and (b) the code generator's output. The content programmer (1) writes the message description file that gamers and servers will use to communicate. The Distributed-organized Information Terra system development kit (DoIT SDK) can parse this XML file and (2) generate the skeleton code content programmers use to (3) and (5) implement the game rules. After (4) deploying the server game rules , the MMOG service is ready to start up.

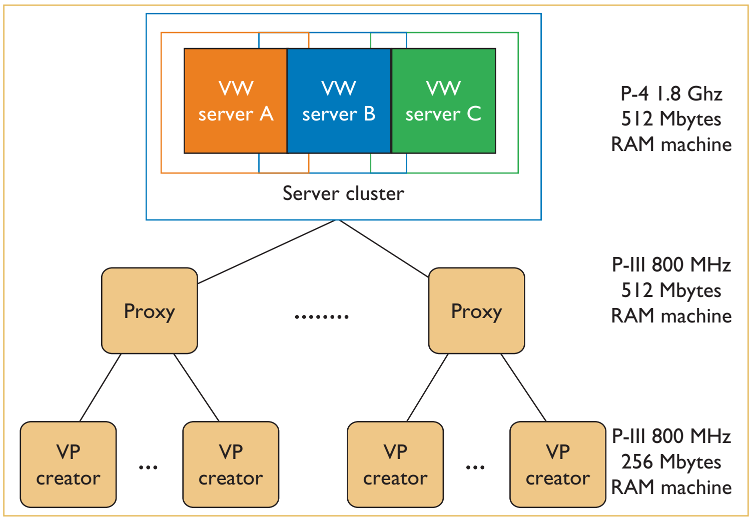


Figure 3. Performance evaluation environment for DoIT. Our testbed used three Pentium III, 1.8 GHz machines as the server cluster, and 36 P-III, 800 MHz machines as proxies and clients. All clients (virtual player creators) were connected to the virtual-world server by a proxy.

state migration. The program's robots thus send `movement` commands in random directions and receive updates from the virtual-world server, which uses the corresponding movement VWLogic.

The frequency of control commands that players send depends on a given MMOG's design, but many current models let players send only one command within a one-to-three-second period. This prevents players from flooding the server with commands and allows enough time for the server to reply. Thus, we designed the robots to send one control command per second. Additionally, the primary variable for evaluating the server's scalability is the total number of players online and issuing commands simultaneously. To measure this, we use the response times for the control messages.

Figure 3 shows our evaluation environment:

- all the computers (including clients, proxies,

and the server) are in the same subnet;
- a single server's map size is 1,000 × 1,000, with all objects on the map falling at coordinates within this range;
- virtual players' (VPs) initial map locations are randomly assigned by the server; and
- each VP's nimbus (area of interest) is 5 × 5 — meaning that any update to a virtual-world object will affect the view of objects within those coordinates.

Our three goals in evaluating servers are to find the maximum number of VPs that:

- a proxy can support without inducing too much delay (which would inhibit a smooth gaming experience);
- a single VW server can support; and
- multiple VW servers can support.

We determine the maximum number using the average response time that is acceptable for most real-time MMOGs (generally below 150 milliseconds).

Table 1 shows the test results for the number of proxies used in three instances. Increasing this number dramatically decreases the average response time. After the first test, we adjusted the allocation of machines to 25 machines running 120 VP creators (that is, robot programs) each and 10 machines with proxies. In this condition, the maximum number of VPs evaluated in a single server was 3,000; the average response time was 38.6 ms and the jitter was between 0 and approximately 3,645 ms.

Figure 4 shows the scalability test results. As we mentioned earlier, one MMOG server cluster (comprising roughly 20 computers, including the proxy/gateway and server) can support between 6,000 and 10,000 players simultaneously. Therefore, the results for our platform's scalability (3,000 players on a single server) were good.

One particularly interesting result emerged from our gateway performance evaluation: After examining the experimental results and the gateway's CPU and I/O usage, we discovered that gateways are the performance bottleneck in generic 4-tier architectures. The first row of Table 1 shows a large I/O load, with connections exceeding 600, which results in poor response time. We can generally state that if a server cluster must simultaneously handle 10,000 players successfully, it needs a total of 18 nodes (15 gateways and 3 cell servers). In 2004, we did the same experiment by deploying

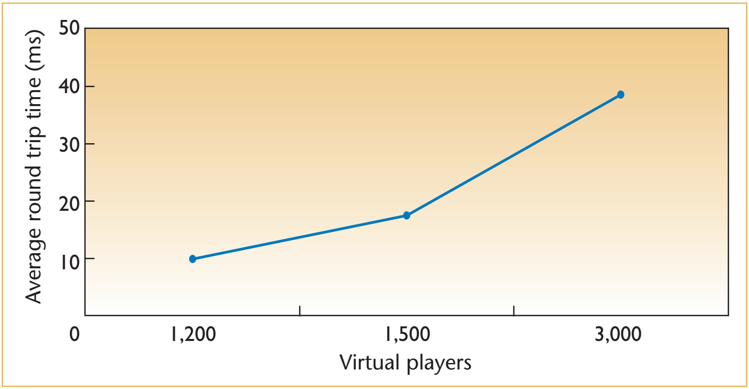


Figure 4. Scalability test results. We found that a single server node can support 3,000 players simultaneously with good average response time (38.6 milliseconds).

Table 1. Evaluation results for various numbers of proxies with 1,800 virtual players.

Proxies	Average response time
2	5,512 ms
3	57 ms
10	6 ms

a gateway program on a Pentium-4, 2.4 GHz machine with 512 MBytes of RAM. Our testing results showed that gateway performance drops dramatically when the gateway needs to process more than 6,000 messages per second (a single gateway handles 1,500 clients). If we improved gateway performance, we could greatly decrease the number of gateways necessary, and lower hardware costs.

Middleware will play an important role in the development of next-generation MMOGs, but building such a middleware is a great challenge. MMOG middleware should serve the varying needs of game developers. Some middleware provides PC client libraries for interacting with servers, whereas other middleware is aimed at game-console (PlayStation 2, Xbox, and so on) or mobile-device clients. Even when client libraries exist, they can't always meet development demands (for instance, at least two next-generation game consoles will ship within the next year). A practical middleware should also give developers a variety of API layers, such as a server-level API for use with a system timer or a thread pool, or a virtual-world level API for creating game objects in the virtual world. Furthermore, it must provide a good plug-in framework to support future content.

In addition to the functionalities we discuss in this article, we've built a persistent object-relation mapping library to help content programmers store virtual worlds' object states without having to program using SQL-commands. To make DoIT more practical, we've formed a student team to craft a real MMOG based on our DoIT platform. Once the MMOG client program is released, we will evaluate the results. Moreover, we intend to build facilities to make mobile devices compatible with more traditional clients such as PCs and Sony's Playstation 2, so that all gamers can play in the same seamless virtual world.

Acknowledgments

The Ministry of Education of the Republic of China partially supported this work under grant nos. NSC93-2752-E001-004-PAE (Advanced Technologies and Applications for Next-Generation Information Networks (II)) and NSC93-2213-E009-079 (A Research on Next-Generation Massive Multiplayer Virtual Environment Platform). We thank the anonymous reviewers and editors for many useful comments.

References

1. A.T. Campbell, G. Coulson, and M.E. Kounavis, "Managing Complexity: Middleware Explained," *IT Professional*, vol. 1, no. 5, 1999, pp. 22–28.
2. K. Geihs, "Middleware Challenges Ahead," *Computer*, vol. 34, no. 6, 2001, pp. 24–31.
3. B. Knutsson et al., "Peer-to-Peer Support for Massively Multiplayer Games," *Proc. Infocom*, vol. 1, IEEE Press, 2004, pp. 96–107.
4. M. Mauve, S. Fischer, and J. Widmer, "A Generic Proxy System for Networked Computer Games," *Proc. 1st Workshop Network and System Support for Games* (NetGames 2002), ACM Press, 2002, pp. 25–28.
5. D. Bauer, S. Rooney, and P. Scotton, "Network Infrastructure for Massively Distributed Games," *Proc. 1st Workshop Network and System Support for Games* (NetGames 2002), ACM Press, 2002, pp. 36–43.

Tsun-Yu Hsiao is a PhD candidate in computer and information science at National Chiao-Tung University, Taiwan. His research interests are in distributed systems, Internet technologies, and middleware. He has a BS and an MS in information engineering and computer science, respectively, from Feng-Chia University. Contact him at tyhsiao@cis.nctu.edu.tw.

Shyan-Ming Yuan is a professor in the Department of Computer and Information Science at National Chiao-Tung University. His research interests include distributed objects, Internet technologies, middleware, and e-learning. Yuan has a BSEE from National Taiwan University and an MS and a PhD from the University of Maryland, all in computer science. Contact him at smyuan@cis.nctu.edu.tw.