

# Building a simple yet powerful MMO game architecture, Part 3: Capabilities and limitations

Hyun Sung Chu

April 08, 2009

Massive multiplayer online (MMO) virtual-world games offer tantalizing new ways to learn, entertain, collaborate, socialize, visualize information, and do business. In this series, learn about an architecture based upon the first 3D MMO game from IBM®, PowerUp. You can use a Web and database back end to provide MMO game functions. In this article, explore how the MMO game architecture described in this series meets the high-level architectural goals of: scalability, flexibility in deployment, flexibility in gaming design, performance, functions, and security.

[View more content in this series](#)

## Introduction

This article is the final part in the series about an MMO game architecture based upon IBM's first 3D MMO game, PowerUp. [Part 1](#) outlines the architecture, specifications, and intended functions of the architecture. You learned about the high-level and detailed architecture design. [Part 2](#) examines the underlying Web use cases and database tables that comprise the architecture.

This article evaluates the architecture's capabilities and limitations. It assesses how the architecture meets the high level goals of: scalability, flexibility in deployment, flexibility in gaming design, performance, functions, and security.

## High-level architectural goals

[Part 1](#) outlined the high level goals, in Table 1 below, that an MMO game architecture should address. This article examines each goal in detail, and assesses how successfully the architecture met the goals.

**Table 1. Specifications and goals**

| For...                      | The architecture should...   |
|-----------------------------|--|
| <a href="#">Scalability</a> | Allow for any potential number of concurrent users. It should also scale with relative ease. |

|                                     |   |
|-------------------------------------|---|
| <b>Flexibility in deployment</b>    | Allow for a wide range of gaming server configurations and updates. Ideally, the server configurations should be relatively easy to reconfigure. Updates should be transparent to the end user. |
| <b>Flexibility in gaming design</b> | Minimize constraints on game designers, and facilitate the ability to design an expansive, integrated gaming world.   |
| <b>Performance</b>                  | Perform smoothly and quickly. Provide performance-related functions, such as game-server load balancing and monitoring.   |
| <b>Functions</b>                    | Provide a means to perform functions for an MMO game. A feature might be the ability to persist and retrieve game data, such as player and game-related data.                                   |
| <b>Security</b>                     | Be secure and not expose any security risks.  |

## Scalability

Adding new game servers is straightforward. All you need to do is:

1. Add a new server entry in the GAME\_SERVER table.
2. Deploy and start a corresponding game server.

In principal, the number of game servers that can be accommodated by the architecture is limitless.

It's trivial to add new cloned, or sharded, game servers to immediately scale to almost any number of users. The only limit is the number of physical servers. Alternatively, the game can also scale by adding completely new game servers, or worlds, missions and so on. This does not allow the game to immediately scale to any number of users like adding a cloned server; you're limited by how fast the game developers can create new content. However, for a small MMO game that grows relatively slowly over time, this should suffice.

Web server load balancing and database clustering are beyond the scope of this article.

As the number of gamer servers and clients grows, the Web application server and database will have to scale accordingly. By no means is it a cinch, but all enterprise Web application servers and databases provide mechanisms for scalability.

With the simple and loose coupling between the gaming client components and the Web and database back end (with HTTP GET requests), both the game components and Web back-end components can easily scale. The complexity of Web and database scalability is transparent to the gaming components. The Web and database back end behave exactly like your typical Web site, without any HTML, CSS, or JavaScript overhead, and with the gaming client and server taking the place of a Web browser.

## Flexibility in deployment

Removing game servers is as easy as adding them. To remove a game server, simply:

1. Remove the server entry in the GAME\_SERVER table.
2. Stop the corresponding game server.

With PowerUp, which consists of about a dozen physical servers running 60 game servers, it was straightforward to manually manage all the servers. You can shift game servers to different physical servers, deploy updated server code, take game servers offline, and add new missions or worlds with ease.

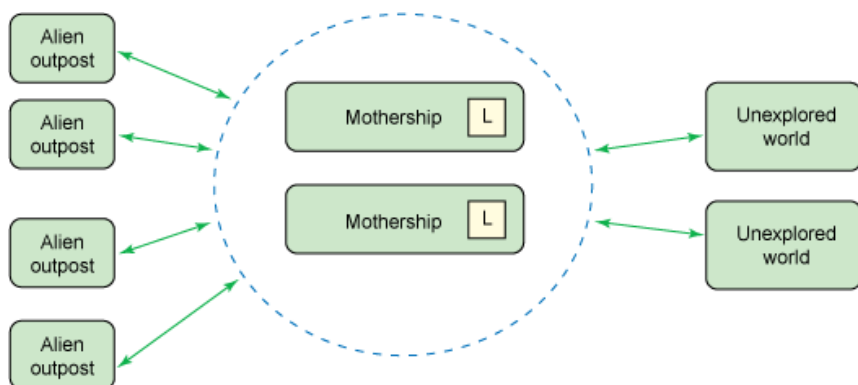
In a truly large MMO game server environment, which might have hundreds of game servers, it would be advisable to use other technologies to manage the deployment of the servers. Options include network virtualization, or rolling your own game server deployment batch or shell scripting files.


Another major benefit of the architecture in this series is that all the server updates can be made with complete transparency to the user. In the whole system there are only two hard coded IP addresses or URLs: the public WebSphere® (or Web application) server IP address in the gaming client, and the secure WebSphere IP address in the gaming server. These two IP addresses can remain unchanged, no matter how many servers are added to the system.

## Flexibility in game design

[Part 2](#) covered the player server flow of Starship IBM, as shown in Figure 1.

**Figure 1. Starship IBM player server flow**



 Indicates a server that the player initially logs in to when starting the game

The architecture does not limit a game to this hub-and-spoke player server flow pattern, where a player initially logs into a central Mothership and then travels to other missions and worlds. The architecture allows game designers the flexibility to create expansive game worlds, with significantly different game server configurations, as shown below.

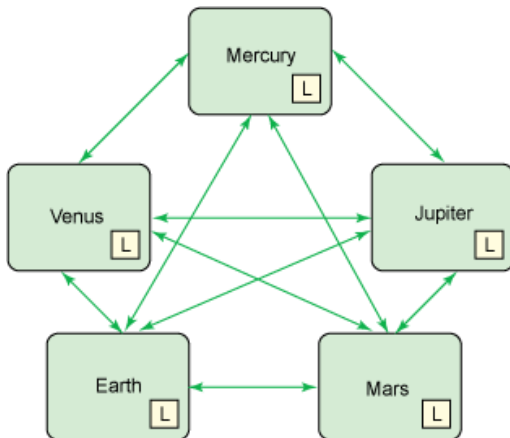
**Figure 2. Starship IBM alternate player server flow**

Figure 2 shows the player server flow of a game without a central Mothership server. Instead, the game can track which server the player was last on, and automatically log that player onto that server. A typical sequence showing how a player goes through the game, similar to the use cases described in Part 2, follows.

1. The player enters a username and password, and selects Login to start the game. This will initiate a use case very similar to "Get available Mothership server." For the game in [Figure 2](#), "Get available Mothership server" will instead return the IP address of the server the player was last on -- defaulting to, for example, the Earth game server if this is the very first time the player logs in.  
You need to track the IP address of the game server that the player was on. To track this IP address, a column can be added to the PLAYER table. Using the IP address, "Login to server" will be invoked, which will authenticate the player and update the database to indicate the player is currently on the specified game server.
2. The player brings up an interface that retrieves a list of available planet game servers, which will invoke "Get a list of destination server," which will return a list of the other game servers.
3. The player selects one of the destination planet servers and starts that mission. "Check if server is full" is performed, then "Login to server," using the IP address and port from the selected destination game server.

This action can be performed repeatedly to travel to the various planet game servers.

With PowerUp, the ability to create distinct worlds running on different servers provided tremendous flexibility in game design. Originally, the team thought of PowerUp as a typical Multiplayer Online Game (MOG), with essentially one game world cloned on multiple servers. Or, in Starship IBM terms, the original intention was to have the Mothership, Alien Outpost, and Unexplored World all in one world. This stymied the team from a game design perspective, as there were intractable geographical and world design constraints. Deciding on a more MMO game style allowed the designers greater flexibility, and let them create multiple distinct worlds. In PowerUp, one world was overcast, temperate, and in an ocean setting, while another was sunny and set in a desert. This would not have been feasible without the ability to have multiple distinct worlds.

## Limitations of the architecture

The architecture described in this series is a sharded MMO game architecture. This is the de-facto standard method for providing scalability for an MMO game. Almost all MMOs use this style of architecture, including the current leading MMO *World of Warcraft*. However, there are drawbacks to using a sharded architecture.

With a sharded architecture, scalability is provided by distributing servers onto multiple physical servers.

The main design issue is that each shard is isolated, so a player on one server cannot interact with a player on another server. If the game incorporates cloned servers with a transparent game server load balancing mechanism, it can lead to odd user experiences. For example, two friends login to the Mothership and naturally expect to interact with each other. But, each could be placed on a different Mothership server by the load balancing function. Or, if a non-transparent load balancing mechanism is used, the player is allowed to choose a Mothership server to log into and now the game presents a myriad of Mothership servers to the user. Users might wonder why there are multiple Starship IBMs, which breaks the illusion of the game design.

There are some solutions that can partially address and work around these limitations. Cross server communication can be added to address server isolation (discussed further in [Functions](#)). Creative game design can also address issues such as the multiple Starship IBM problem. For example, each Mothership server can be slightly different from another, with slightly different art content for each server. Each server represents a different ship in a fleet -- one is Starship IBM, the other is Starship AIX, Starship Deep Blue, and so forth. In a medieval or earth-based game, with a city in lieu of a Mothership, each server can represent a neighborhood and have a different configuration of buildings and structures. This would not place an excess burden on artists and programmers if the programmers create a tool that can automatically generate starships or neighborhoods, similar in principle to existing tools in gaming that generate random dungeons, caves, or buildings.

Each Torque Game Engine Advanced (TGEA) server can handle approximately 200 concurrent users. At first this might seem rather low for an MMO, but it is a conservative estimate. It's difficult to provide an exact upper limit, simply because the limit depends on several factors. It's possible to get significantly higher numbers by changing these factors, such as improved hardware. Improved coding can help. For example, TGEA is at heart a First Person Shooter engine. This genre typically has a limit of 64 players per server at most, since it requires a higher degree of networking fidelity than a turn based style Role Playing Game (RPG). Theoretically, this networking code can be toned down for a turn based style game and increase the number of concurrent players. Even game world design is a considerable factor. For example, the number of objects in the world factors into the performance of the game.

Alas, these methods would achieve only a finite increase, and you would still have the need for sharded servers. For a shard to scale indefinitely, clustered gaming servers are needed. [Eve Online](#), which uses clustered gaming servers, is unique in that it can accommodate all of its 40,000+ concurrent users in a single shard. Eve Online is a highly customized game architecture built on top of what essentially is a clustered super computer. Game server clustering is not

a typical feature for most gaming engines. Game server clustering requires either extensive customization of the gaming engine, or integrating a generic gaming networking product designed for game server clustering (which is not trivial).

## Performance

The performance of the integrated gaming servers and clients with the Web back end should be sufficient for almost any MMO game. Using HTTP requests, with a simple scheme of returning lists of key value pairs, is a good balance of ease of use and development against performance. All the HTTP requests described in this series should take a few split seconds to complete. And, the number of Web requests made would typically be light in most MMO games. It's difficult to quantify this number, since it largely depends on how the game and Web APIs are designed. A handful of Web requests every few minutes for each user would be a reasonable ballpark estimate.

Using a simple game like Starship IBM as an example, the Web request use cases related to traveling from one server to another ("Get available mothership," "Login to server," "Get a list of destination servers," "Check if server is full," and "Remove player from server") are called every time a player travels to a server. In a typical gaming session this would occur every few minutes at most. Similarly, "Retrieve player points" and "Update player points" can also be invoked once each time a player travels to a server. "Retrieve player points" can occur at the beginning of a mission, with the gaming server tracking the player points, and "Update player points" can be called when the player leaves the server.

You could gain extra performance using a different protocol and binary data for the gaming and Web integration, but that would compromise ease of use and development. HTTP is understood by virtually all developers and is widely implemented. For example, the Torque Gaming Engine has HTTP functions out of the box, and of course a Web application server is in essence an HTTP server.

It's also possible to connect the game servers directly to the database, essentially cutting out the middle-man, or middleware, for additional performance gains. However, there are significant drawbacks to this scheme:

- There's no sensible way for gaming clients to directly connect to the database. Without the ability to directly connect to the back end from the client, the game servers would be tasked to mindlessly traffic data from the back end to the gaming clients.  
It's crucial to avoid unnecessarily burdening the gaming servers, which is often the most performance sensitive element in a multiplayer game.
- This scheme is less scalable and robust, since database connections will have to be maintained and updated for each gaming server. That's a significant consideration for any large MMO game comprised of dozens or hundreds of gaming servers.  
With the architecture described in this series, a Web server should handle all the Web requests for multiple game servers. It's also easier to maintain this handful of database connections from the Web servers.
- A three-tiered architecture allows for a separation of duties. Web and database developers don't have to know a lot about game development, and the game developers don't have to

know a lot about Web and database development; the components are integrated with a simple HTTP request API. There aren't many experienced gaming *and* database developers.

- A Web server provides the ability to offload functions from the game server. Without the middleware, back-end functions would have to be incorporated in the gaming servers. If this logic is complex enough, the gaming servers will be unnecessarily burdened.
- Enterprise middleware greatly eases development for difficult features, such as managing transactions that maintain ACID (atomicity, concurrency, isolation, durability) properties.

Another drawback, and annoyance, with a sharded architecture is the load times as players connect from one server to another. Gaming clients are required to load new resources and art content into memory. However, for most gaming engines load time is a function of resource and art content. A sharded architecture allows for the "spreading out" of art content onto different servers. Load time can also be spread out, avoiding the loading of gaming content that the user might not be interested in. With PowerUp, it was originally conceived that the game was a typical MOG game, with all the content in one world. Naturally, the team ran into issues of long load times. However, by selecting a sharded MMO game architecture they had more flexibility with the load times because they could place art content on different servers.

## Functions

The architecture can readily provide functions related to player and game persistence, game server travel, authentication, application integration, and game server administration. A benefit of middleware is that functions can be offloaded onto the Web server, away from the performance-sensitive game servers. Aside from the humdrum task of data persistence, there is potential for other interesting functions.

One intriguing possibility is to offload potentially computationally-intensive functions, such as AI, onto the Web server. This does not mean the "tactical" level AI of non-player characters (NPC) immediately responding to players, which should remain in the game server since it requires split second response time, but the more strategic level AI of how NPCs react to game conditions at a high level.

Cross game server communication is another potentially important function, and you can do it a few different ways. One way is to set up the game servers to listen for messages. For example, in Torque there's a default function to start an HTTP Web server in the game server. So, given another player username, it should be straightforward to retrieve the IP address the player is on from the Web and database back end and use the IP address to directly message the player on the other game server from the current game server. The main drawback to this scheme is that the game servers will also have to act as a Web server, which increases the complexity of the architecture and introduces more "moving parts" that can break.

You can also leave the architecture as is and achieve cross game server communication with some additional Web back-end calls. Basically you'd use the Web server as a messaging bus. The Web server needs to act as a bus because the game servers can instantiate calls to the Web server and not the other way around. Messages can be sent from the game server to the Web server at any time, but all of the game servers will have to periodically poll the Web server

to check for and grab any messages that is intended for it. The drawback is that messages won't be received in real time. This issue can be mitigated with a heuristic polling frequency algorithm, such as polling more frequently depending on the number of players on the server. If there's only a single Web server for the game, implementation would be straightforward. The single Web server can track the entire messaging queue. However, if the game requires more than one Web server, it becomes more of a challenge to implement this messaging queue. Either the Web servers will have to be synchronized, or the messaging data will have to be stored in the centralized database, which then raises the issue of storing transient data in a database.

The drawback with any cross game server communication is the dramatic increase in communication and transactions in the back end -- more so than almost any other back-end function. This is a big consideration when planning features that likely call for more Web and database firepower to handle the increased traffic.

## Security

The architecture does not introduce any new security risks. The private WebSphere server is completely hidden from the public and is known only to the game servers. The public WebSphere server is directly accessible from the gaming client, so this server should only provide secure functions (primarily views). All updates to the database should be handled by the gaming server in conjunction with the private WebSphere server.

The use cases in this series did not mention one important function: registering a player. Registering a player should be handled through the Web, and not the gaming client, especially if there is sensitive data involved. Many MMO games require a player to register first through a Web site mainly because of security concerns. Secure HTTPS Web transactions through a Web browser are well established and battle tested. If the gaming client handles the registration, it raises concerns about the security of the gaming client. This is not to say a Web browser is absolutely secure and a gaming client is not. However, it's easier to use a well established security mechanism such as HTTPS through a Web browser instead of ensuring that a gaming client and registration process are secure. Developers can breathe easier knowing they're using the same security techniques as large commercial Web sites such as Amazon.com or Bank of America.

## Summary

In this series, you learned how to create a simple MMO game architecture using a typical multiplayer gaming engine in conjunction with a straightforward Web application server and database back end. This final article examined the limitations and capabilities of the architecture. The primary benefit of the architecture is its simplicity, making it quick and easy to implement and maintain while still providing all of the features of a typical MMO game. The primary drawbacks of the architecture are the limitations associated with a sharded MMO game.

If you're new to MMO game development, this series provided an introduction and detailed inside look at an actual MMO game architecture. For you more experienced MMO game developers, hopefully you can compare notes and take away some useful tricks and techniques. Everyone is invited to contact the author regarding the article or gaming in general.



## Acknowledgement

Thanks to George Dolbier from IBM Global Sales and Distribution for freely and generously donating his time and advice on gaming and production game server hosting environments for PowerUp.

Thanks to the core PowerUp gaming developers and artists, Mark Laff and Shari Trewin from IBM Research and John Dingler, Jessica Marceau and Colin Freeman from IBM Global Business Services. Colin was also the one who suggested the idea of "side missions" for PowerUp, and so was an influence on the architecture.

Thanks to Peter Rodriguez from IBM Enterprise Initiatives. Peter was PowerUp's overall project and technical lead and worked with the author on high level technical strategy, including architecture.

## Related topics

- Check out the other parts of this series:
  - [Part 1](#) shows you how to build a flexible and powerful MMO game architecture that is quick and easy to implement.
  - [Part 2](#) explores technical details of the architecture, including the functions, and calls for integrating game clients and servers with back-end systems.
- Read about [MMOs](#) on Wikipedia.
- [GarageGames](#) specializes in gaming engines targeted for small and independent game developers.
- [Torque Game Engine Advanced \(TGEA\)](#) is GarageGame's flagship 3D game engine.
- [Torque Game Engine \(TGE\)](#), another game engine from GarageGames, has less advanced graphical capabilities than TGEA. It cannot render shaders (out of the box), but its networking capabilities are very similar to TGEA. The architecture described in this article does not use any particular feature that is TGEA specific and not part of TGE.
- The GarageGames forum covers [Torque client/server networking](#).
- Read about how [Torque interacts with a Web server](#).
- For comparison purposes, [Unity](#) is a 3D gaming engine that has features comparable to Torque.
- Check out [Unity's game networking](#), particularly the section on "WWW functions" to learn how Unity talks with Web servers.
- Read an overview of [Eve Online's](#) cutting edge architecture, which is unique in that it doesn't use a sharded architecture. There are links that provides more details on the architecture, including some of its challenges. IBM was also involved with *Eve Online*, particularly in hardware.
- [Raknet](#), [Project Darkstar](#), and [NetDog](#) are just a few of the many game networking products available that provide MMO game server clustering. Each product has pluses and minuses regarding licensing costs, maturity, and features.
- Learn more about the various IBM products described in this article, including [IBM WebSphere](#) , [IBM DB2](#) , and [IBM BladeCenter](#)® servers.
- Get the [RSS feed](#) for this series.

© Copyright IBM Corporation 2009

([www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml))

[Trademarks](#)

([www.ibm.com/developerworks/ibm/trademarks/](http://www.ibm.com/developerworks/ibm/trademarks/))