

Databases in Online (Social) Gaming

Designing a scalable data infrastructure

Contributed by Joshua Butcher
January 2012

Table of Contents

1	Introduction.....	3
2	Business and Technical Challenges.....	4
3	Data Modeling and Sharding	5
3.1	Game Accounts (Legally Liable).....	7
3.2	User Profiles (Legally Liable)	7
3.3	Friends Connections	8
3.4	Messaging (Ingame) (Legally Liable)	9
3.5	Billing (Legally Liable)	10
3.6	Entitlements	10
3.7	Sessions.....	10
3.8	Inventory.....	11
3.9	Leaderboards.....	11
3.10	Achievements/Trophies	12
4	Database recommendation.....	13
4.1	Replication limitations.....	13
	About Joshua Butcher.....	14
	About Severalnines.....	14
	Contacting Severalnines	14

1 Introduction

Massively Multiplayer Online Games (MMOG) are multiplayer video games capable of supporting thousands of players simultaneously. They are played on the Internet and players connect via personal computers, game consoles or even smart phones. The game allows players from around the world to interact, cooperate and compete with each other on a large scale.

With the arrival of social network platforms, the industry has seen an explosion in casual gaming, called social gaming. MMOGs have effectively moved from the hardcore gamer community to the mainstream. For instance, Happy Farm has 228 million active users and 23 million daily users, mostly in Asia. World of Warcraft has over 11 million monthly subscribers worldwide.

Caprica, the prequel to the hugely popular and cult-like TV series Battle Star Galactica explores the future of online gaming and its potential. What seems like fiction however is not that far away from today's gaming reality. Sony's Playstation Home is a virtual world where users evolve in an elaborate online world with millions of people from across the globe. Users have their own avatar, their own apartments that they can furnish by purchasing virtual furniture. They go to the cinema to watch the latest movies from the "real" world – within Playstation Home. It seems that we are only one step away from Caprica's parallel virtual worlds and consciousness-gaining robots.

In today's online gaming world, powerful database systems are needed however to power all this virtual activity.

MMOGs host a large number of players and all these players can interact with each other at any given time. Because of scale, a game universe can 'be sharded' in several ways. One way may be where each shard is a smaller universe deployed on a server. Another way is to have the shard hold a certain number of simultaneous players on a complete universe or playing field. Other games might feature a single universe, which is divided and placed on different servers. Players who log into a particular server will be in one part of the universe; they will need to switch servers to go to another part of the game universe.

MMOGs are data-driven applications, and databases used in these applications have to satisfy stringent requirements in terms of performance, availability and scalability.

This paper discusses the importance of databases for the gaming industry, what its requirements are in terms of database technology as well as a discussion on why MySQL is or should be the database of choice for anyone wanting to develop online social games that are reliable and stable in all their aspects.

In addition, the paper also discusses the different types of data that need to be stored by a social MMOG with recommendations on how to handle these.

2 Business and Technical Challenges

Social gaming has exploded in the last few years, and there are several possible explanations to this:

- Easy to learn and play
- Games adopted/played by the mainstream, rather than a hardcore gamer community
- Viral social networks make a game easily promoted across audiences
- Take advantage of the social graph for a shared, fun experience with real friends
- Use of social connections and communications APIs to make the game seem more compelling
- Audience: kids are now the biggest consumers of games, be it offline or online. They play on their portable devices (Nintendo DS / mobile phones or their parents' iPads & tablets ...). They then grow into adult gamers who continue playing; what changes are the types of games they play, but the community keeps on growing.

Building an infrastructure to handle a social MMOG is challenging, and includes:

- On-demand scalability and elasticity: It is hard to predict the success of a new game, and a successful game can attract millions of users.
- High Availability: players are from all over the world, playing at any time of the day. Therefore, the system has to function 24*7. The data also needs to be distributed worldwide.
- Predictable latency: avoiding database lag spikes that introduce 'item lag'.
- Manageability: A COTS server typically might handle about 10,000 players. This means 100 servers for each million users.
- Cost: Online gaming is a competitive industry, with everything from free games to monthly subscriptions of \$15 per player. Efficient operations are an important business goal. This is now also moving to Freemium: the games are free, but elements of the game (e.g., equipment for a character etc that can be purchased to increase / enhance the gaming experience).

Web driven, social game companies also have to deal with a diversity of data along with simultaneous access. The company may have hundreds of games and titles, which presents an interesting data modeling challenge to the game developer. Despite the high scale of users, a devops team would want to run as few databases as possible.

3 Data Modeling and Sharding

MMOGs have very interesting usage/storage patterns. The database server will go through rather long periods of being idle followed by sudden jumps in concurrent reads and writes. Why? Because the MMOG server tends to cache user data into its own memory bank, and will then batch-write the data back to the database server like a cron job would, every X minutes. So the database server will sit idle for a while and see sudden surges in activity.

For a massive game, the database layer will need to address the following requirements:

- Up to 30,000 tps per million concurrent users
- Scale from a few thousand to 2-3 millions concurrent users
- High availability
- Synchronous replication for session data
- No data loss for non-session/leader related data
- Log archival (keeping logs) for legally liable data for up to 7 years.
- Sharded data for extreme speed where needed
- Predictable latencies for a smooth gameplay experience

Criteria for data management will vary for different types of data stored for different functions.

No data loss is essential for legal compliance when it comes to:

- Billing
- Inventory
- Character Entitlements
- Account Information

No data loss is essential for professional reputation when it comes to:

- Trophy data
- Messaging

Speed is essential for:

- Session data
- Character Data

- Inventory
- Leaderboards

Data Integrity is optional for:

- Sessions
- Leaderboards

Leaderboard integrity is optional, because there should be persistent data that can be used to rebuild the leaderboard, should it be lost.

Game developers also tend to want to store the data in the DB as a JSON or some other type of serialized C/C++ object. This is to have the application read it in as one BLOB and not have to convert DB fields into C/C++/Java fields. This is a practice that has to be discouraged. For instance, the game developer might need to use the database server to query on specific parts of data inside the JSON/BLOB.

Another thing to keep in mind is that an MMOG is a world that companies want to bill for over and over again. This means it's an actual world, and everything inside the world is "real" to the user. This means the game company can never lose any of it.

The following sections are a recommendation on how to break a schema down with each portion stored in its own master/slave replication cluster. A typical replication setup is one master and a minimum of two slaves in each data center. The redundancy allows for a devops person to take one slave offline for maintenance, backups, creation of new masters/slaves, etc. while still having a slave serving live traffic.

3.1 Game Accounts (Legally Liable)

NOTE: The company is legally liable for this data.

For instance, if there are children playing its games who are under 13 years of age, the company needs to be aware of COPPA¹ laws and what other users are saying to under-13 year olds on its platform.

Game accounts are not user profiles. This is very important these days. A single game account can have multiple game profiles associated to it.

This data is typically shared by a web site and allows the user to manage their account, as well as the game itself. It would do well to have a REST service sitting in front of it.

Account information is going to be some of the smallest and least accessed data. Once the game has authenticated the player, it tends to act on the user profile(s) inside the game server's memory. Most MMOG engineers like to use the DB to hold data until it is read into memory. Once it is in the game server's memory, a good portion of that data will stay there until it is flushed back to the server for a new persistent state only. Game server engineers will try to avoid going back to the database.

The replication latency across data centers is fine, because users should be connected to only one data center at any one time and access services from that data center - unless there is a failure. Any updates to a master server should be available as quickly as possible to its local read slaves however.

A typical hardware configuration might look as follows:

Master: 16G RAM, 4 Disk RAID-10

Slave: 3x16G of RAM, 4 Disk RAID-10

3.2 User Profiles (Legally Liable)

NOTE: The company is legally liable for this data.

A user profile is a character created in an MMOG. These days, MMOGs will let one game account have 5+ characters.

¹ Children's Online Privacy Protection Act (COPPA) is a US federal law. It applies to the online collection of personal information by persons or entities under US jurisdiction from children under 13 years of age. It details what a website operator must include in a privacy policy, when and how to seek verifiable consent from a parent, and what responsibilities an operator has to protect children's privacy and safety online.

This data is typically shared by a web site that allows the user to manage their user profiles, as well as the game itself. It would do well to have a REST service sitting in front of it.

This is going to be some of the smallest and least accessed data. Once the game has authenticated the player, it will act on your user profile(s). As with the account data, once the game server has read this information into memory, it will probably be accessed there exclusively until the game server flushes a new copy back to the DB for a new persistent state. The user profile might also include character information, which includes the player's items and assigned skills. This is managed directly in memory to guarantee real-time operations (e.g. a player has killed a creature and gained some status, and the nearest 100 players need to find out about it ASAP)

Additionally, just as for account information, the replication latency across data centers is fine since users are only connected to one data center at any one time.

This data will usually be cached by the MMOG server on a game shard. It will not be read very often once it is loaded. The game server will checkpoint its cached state of the updated user profile data typically every X minutes. This is fine as long as the game server shard doesn't crash. Caching is great as it takes away some of the load on the database server. However, the cache is lost in the case of a crash of the game server. Losing a cache means potential loss of updates to the profile.

A typical hardware configuration might look as follows:

Master: 16G RAM, 4 Disk RAID-10

Slave: 3x16G of RAM 4 Disk RAID-10

3.3 Friends Connections

This data is shared by the game's website. For example, in Ultima Online, you can access your in-game character information from the web site, and in some cases modify it. It is also a way for someone to say "Hey, look at my character" to friends and on social websites by linking to a character webpage, without having to make everyone go into the game to see the avatar/character. It would do well to have a REST service sitting in front of it.

This is going to be some of the smallest and least accessed data. Once the game has authenticated the player, it will act on the user profile(s) as described above in section 3.2. This information is likely to be cached by the game server itself.

Just as for account and user profile information, the replication latency across data centers is fine since users are only connected to one data center at any one time.

It might be a good idea to limit the number of friends on a user profile, so as to more easily predict how this data should scale. This data should be readily available to read

as quickly as possible from the DB. This data can be cached by the local game server, although it is better to use a caching layer like Memcached or Redis.

A typical hardware configuration might look as follows:

Master: 16G RAM, 6 Disk RAID-10

Slave: 3x16G of RAM 6 Disk RAID-10

3.4 Messaging (Ingame) (Legally Liable)

The company is legally liable for messaging data, e.g., to know what other users are saying to under-13 year olds on the platform.

Players can send messages to each other, but do not have to be online simultaneously in order to socialize. They can leave automated messages for each other in order to keep the game going.

This data is shared by a web site that allows the user to view and manage their friends, as well as the game itself. The company and game moderators will look for abusive usage patterns in messaging, and may get access to this information from an intranet, or a secure webpage. Moderators may be working from home. It would do well to have a REST service sitting in front of it.

Just as for account and user profile information, the replication latency across data centers is OK since users are only connected to one data center at any one time. However, messaging data should be available as quickly as possible to its local read slaves. Otherwise, a page reload might see messages *disappear* if the previous page loaded its data from an up-to-date master, but the reload actually reads the data from a slave that has not yet received the update.

The message data will grow and the life of a message could be limited to 30 days for example, or when a person's account has been deemed inactive for 6 months. Message usage patterns will vary, some games use a lot of messaging while others make casual, light use of this. In some cases, the game developer may want to use caching. Memcached is a good recommendation here.

A typical hardware configuration will have a more powerful master server and more disks in order to handle the high insert load:

Master: 32G RAM, 6 Disk RAID-10

Slave: 3x16G of RAM 6 Disk RAID-10

3.5 Billing (Legally Liable)

When deciding to bill, the game developer has to decide whether or not to store credit cards, or sensitive personal information that could be used to hack the credit card info for a user. If the company is storing credit card data, it must be fully PCI compliant. If not, then the database needs to store only minimal information for entitlements.

This information does not need to be shared with the game server, and will be one of the least accessed pieces of data.

This data needs to be absolutely 100% ACID, and available. It can be placed on a single powerful machine acting as a master, with a replication slave solely as a backup. This means that all reads and writes only happen on the master.

A recommended hardware configuration might look as follows:

Master (Reads/Writes): 16G RAM, 8 Disk RAID-10

Slave (Backup Only): 1x16G of RAM 4 Disk RAID-10

3.6 Entitlements

Entitlements are what rights/products the game account has access to.

This determines whether or not a user has access to a particular game. When a credit card does not auto-bill for example, the user's entitlements are disabled, not the user's account.

This data is typically shared by a web site that allows the user to see their entitlements, as well as the game itself. The game developer may allow the user to view and/or modify their entitlements on a secure webpage, whereas the game will also consume the information so it knows what users have access to. It would do well to have a REST service sitting in front of it.

This is going to be some of the smallest and least accessed data. Entitlements are typically read from the database by the game server at login only, so the game server can know what a user has access to.

The replication latency across data centers is fine since users are only connected to one data center at any one time.

This product tends to live on the User Accounts database hardware.

3.7 Sessions

Sessions are one of the most highly accessed types of data in the system. It is also the most volatile. User sessions can be reconstituted even if the users were on a game server that crashed.

Session objects are a very good fit for a geo-replicated main memory database. This will provide worldwide access to the data, and data can be served at a very high transaction volume.

A recommended hardware configuration might look as follows:

Multi-Masters: Up to 20 x 32G RAM, 6 Disk RAID-10

3.8 Inventory

Inventory data is as precious as billing data for the user. This describes how the player interacts with the gaming universe. Inventory data are virtual belongings that uniquely present a player's digital lifestyle in a game.

Game developers tend to cache this in the server's memory bank. The disks need to be generous in speed and size, because a user's inventory can grow quite large, and the checkpoint every X minutes for multiple users can be a strain on the disk IO.

This data is typically going to be cached by the MMOG server on a game shard. It will not be read very often once it is loaded. The game server will checkpoint its cached state of the inventory data every X minutes. This is fine as long as the game server shard doesn't crash.

The replication latency across data centers is fine since users are only connected to one data center at any one time. Updates to inventory data should however be propagated as quickly as possible to its local read slaves

A recommended hardware configuration might look as follows:

Master: 16G RAM, 6 Disk RAID-10

Slave: 3x16G of RAM 6 Disk RAID-10

3.9 Leaderboards

Leader boards at the end of the day are volatile. People want to know how they are doing against someone else, but leader boards change all the time and as soon as one is published, it is no longer valid. These can be reconstituted if they crash.

For speed and performance's sake, keeping this data in memory is handy, because leaderboards are often real time or near real time. Players today want to know how they are doing against one another. Keeping the data in memory will allow the fastest possible way to continuously recalculate new data. Using the same architecture (and most likely the same platform deployment as sessions) will do nicely here.

3.10 Achievements/Trophies

These are as precious to a player as inventory is. This tells the rest of the world how awesome they are as a player. This is the second most important reason as to why they are accessing the game.

Luckily however, game developers tend to cache this in the server's memory bank. The disks need to be generous in speed and size however, because a user's inventory can grow quite large and the checkpoint every X minutes for multiple users can be a strain on the disk IO.

The replication latency across data centers is fine since users are only connected to one data center at any one time.

A recommended hardware configuration might look as follows:

Master: 16G RAM, 6 Disk RAID-10

Slave: 3x16G of RAM 6 Disk RAID-10

4 Database recommendation

With so many NoSQL databases now available, one might wonder why MySQL would be a good database choice for the gaming industry.

In fact, MySQL is extremely general purpose and has proven itself in large-scale infrastructures.

The NoSQL movement for instance was started with niche in mind, and although there are a range of products available, they all address different needs.

Having a general purpose product such as MySQL helps reduce the number of different products used in an infrastructure. Reusing the same components simplifies the architecture and makes it more maintainable in the long run.

Storing information that the company is legally liable for puts a strong emphasis on reliability and data integrity. NoSQL is great for performance, but is not as mature and stable as MySQL. For instance, the game developer would want to use a well-proven transactional database for billing data.

In terms of the vast array of NoSQL databases available, it is likely that the industry will rally around one or two database products that are well supported by professional support organizations. MongoDB is gaining a lot of traction and looks very promising, although it is unclear how easy it is for a devops team to manage a scalable, clustered database architecture.

4.1 Replication limitations

It is possible to make MySQL redundant and scalable, but it is not an out of the box functionality as for some of the NoSQL databases (with the exception of MySQL Cluster). For instance, MySQL replication is simple to set up and used by thousands of websites. However, it is somewhat fragile². Fail-over is not automatic and it is possible for slave servers to end up with different data from the master. Fortunately, there are products available that help address this. E.g. the Severalnines Replication Configurator can automatically set up a Replication cluster, with support for automatic fail-over and re-synchronization. MySQL Replication is asynchronous and works well when data needs to be distributed in different data centers to enable worldwide access with reasonable performance.

It might also be wise not to have too many slaves connected to a master, as it drags down the IO of the master. With the above sharding strategy, we divided a schema into product categories and each category could be placed on a dedicated replication cluster. This works better than the generally accepted way of sharding the entire schema by user ranges.

² See <http://www.severalnines.com/resources/clustercontrol™-mysql-replication-tutorial> to read more about MySQL Replication limitations

About Joshua Butcher

Before sinking his teeth into large scale databases for the gaming industry, Joshua Cannon Butcher started technical life programming on personal projects at the age of 16 with self-taught Basic, Pascal, Assembler and C++. He then went onto professional end user support for organizations like Coca Cola, owned a consulting business offering services ranging from support to systems, network, web and desktop application architecture. He has written commercial backup applications for Windows, written indie role playing games, but most importantly, gained extensive experience designing large scale database systems for high volume online marketing, online social gaming platforms and high concurrency online games.

LinkedIn Profile: <http://www.linkedin.com/in/drsq1>

About Severalnines

Severalnines provides software for fast, scalable and highly available cloud database platforms. This enables organizations to more efficiently build mission-critical clustered database environments, whether deployed on premise or in a cloud infrastructure, hence lowering capital expenditures as well as increasing productivity and innovation. Severalnines is recognized by thousands of organizations globally, with its products being used in different applications in a number of industries spanning from online gaming to telecom and finance.

Severalnines's flagship product, ClusterControl™, enable customers to Deploy, Manage, Monitor and Scale their clustered database platforms, free from the complexity and learning curves associated with database clusters.

Contacting Severalnines

Phone +46 70 267 18 62
Email sales@severalnines.com
Mail Severalnines AB
Box 1263, Isafjordsgatan 22
164 29 Stockholm
Sweden

© Copyright 2011 Severalnines AB. All rights reserved. Severalnines and the Severalnines logo(s) are trademarks of Severalnines AB. MySQL is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. © 2011 Severalnines AB.