# Towards Cloud Data Management for MMORPGs

Ziqiang Diao and Eike Schallehn
*Institute of Technical and Business Information Systems, Otto-von-Guericke University Magdeburg, Universitaetsplatz 2, Magdeburg, Germany*

Keywords: MMORPG, Data Management, Cloud Storage System.

Abstract: Massively multiplayer online role-playing games (MMORPG) provide a persistent and collaborative world for millions of players. With increasing numbers of players and growing volumes of data, architectures based on conventional RDBMS limit the development of MMORPGs, because issues related to the availability and scalability of the storage system become a big challenge. These properties are typically well supported by Cloud data storage systems, while other typical requirements of MMOPRGs are not or not yet supported, as we will show in this paper. Furthermore, in a current project we assess the usability of Cassandra and possibly extend its functionality for MMORPGs. In this paper, we will classify data based on its management requirements, highlight limitations of the existing architecture and identify potentials and issues of Cassandra in the management of diverse data in MMORPGs.



## 1 INTRODUCTION

Worldwide revenues for Massively multiplayer online role-playing games (MMORPG) like *World of Warcraft* and *Star Wars: The Old Republic* have increased to billions of dollars each year. Unfortunately, the complex architecture of MMORPGs makes them hard to maintain, resulting in considerable costs and development risks. In this paper, we propose to use and extend Cloud storage systems to address these issues.

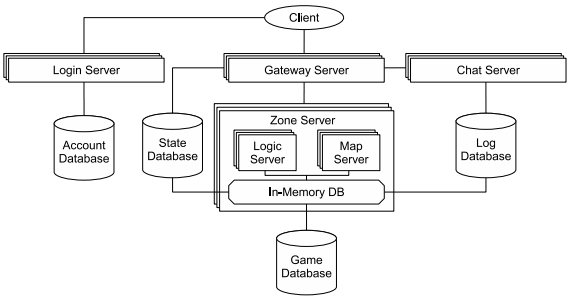


Figure 1: A simplified architecture of MMORPGs.

In order to support player interaction and ensure system security (Zhang et al., 2008) the MMORPG typically employs a Client-Server model. Figure 1 shows a simplified architecture of an MMORPG (White et al., 2007; MuchDifferent, 2013). The login server is responsible for determining the validity of players login requests. It cooperates with an account database that stores user account information. If the validation is passed, then players communicate with the gateway server, which maintains the connection state of a player, monitors the player's requests and transmits them to an appropriate server. In order to reduce the response time, servers are usually deployed in parallel in several data centers (zone servers from the perspective of players), which locate around the world. However, according to the choice made by a player, only one zone server provides services. The zone server is responsible for maintaining a virtual game world for players, executing the game logic, simulating interactions among characters, and synchronizing the simulation results to the relevant players. In order to balance the server workload, a zone server is composed by multiple map servers and logic servers. The zone server reads game scripts and rules from a game database, accesses state data of characters from a state database, and processes data in an in-memory database in real-time. Moreover, all player operations and sometimes also the chat history are monitored for data mining and intrusion detection.

Note that the data size grows significantly with the continuation of time. Furthermore, MMORPGs usually have a large amount of concurrent players, for example, World of Warcraft has millions of concurrent players, which also exacerbates the burden of managing data. A qualified database system for data persistence in MMORPGs must guarantee the data consistency, and also be efficient and scalable (Zhang et al.,

Diao Z. and Schallehn E..
Towards Cloud Data Management for MMORPGs.
DOI: 10.5220/0004404403030308
In *Proceedings of the 3rd International Conference on Cloud Computing and Services Science* (CLOSER-2013), pages 303-308
ISBN: 978-989-8565-52-5

2008). However, the existing RDBMS cannot fully satisfy all these requirements simultaneously (White et al., 2007). Therefore, with the increasing data volume, the storage system becomes a bottleneck in MMORPGs, and solving scalability and availability issues become a major cost factor and development risk.

As a Cloud storage system, Cassandra (Apache, 2013), which has the ability to support highly concurrent data accesses and huge storage, may become a solution. Unfortunately, in contrast with the conventional DBMS, Cassandra is designed for Web applications that have different access characteristics and require lower or different consistency levels. In this paper, we try to apply Cassandra to MMORPGs in order to assess the usability of Cassandra for MMORPGs, open issues, and possible solutions. The rest of the paper is structured as follows. In Section 2, we classify data of MMORPGs into four groups and discuss their requirements. In Section 3, we analyze potentials of Cloud-based data management for MMORPGs. In Section 4, we discuss the usability and challenges of applying Cassandra to MMORPGs, and, finally, conclude and summarize this paper in Section 5.

## 2 DATA MANAGEMENT REQUIREMENTS OF MMORPGs

According to data management requirements, for our following considerations we classify data slightly different into four data sets because these different classes should be managed according to their requirements.

- **Account Data:** this category of data include user account information, such as user ID, password, recharge records, and account balance. This data is usually only used when players log in or log out of a game or for accounting purposes.
- **Game Data:** data such as world geometry and appearance, object and NPC (Non Player Character) metadata (name, race, appearance, etc.), system logs, configuration and game rules/scripts in an MMORPG are generally only modified by game developers. Some significant part of the game data is often stored on the client side to minimize network traffic for unchangeable data.
- **State Data:** as we discussed above, PC (Player Character) metadata, position and state of characters and objects, and inventory in MMORPGs are modified constantly. Currently, the change of state data is executed by an in-memory database in real-time and recorded in a disk resident database periodically.
- **Log Data:** analyzing user chat history and operation logs in an MMORPG is the most objective and direct way for game providers to evaluate a game, find out the operating habits of players, explore the game development trends, and even supervise the financial system of the game world.

For these classes we summarize data management requirements in Table 1 and discuss them in the following.

**Support for Different Levels of Consistency:** in a collaborative game players interact with each other. Changes of state data must be synchronously propagated to the relevant players within an accepted period of time. For this purpose we need a continuous consistency model in MMORPGs (Li et al., 2004). The state data and account data changes must be recorded in the database. It is intolerable if players are surprised to find that their last game records are lost when they log in the game again. As a result, a strong or at least a Read Your Writes consistency (Vogels, 2008) is required for such data. However, strong consistency is not necessary for log data and game data, for example, the existence of a tree in the map, the synchronization of a bird animation, or the clothing style of a game character can be different in the client side. Log data is generally not analyzed immediately. Hence, eventual consistency (Vogels, 2008) is sufficient for these two classes of data.

**Performance/Real-time:** state data is modified constantly by millions of concurrent players in MMORPGs, which brings a large amount of data throughput to game servers and thousands of concurrent database connection operations. It has become a challenge to the database performance. Such changes must be executed in real-time and persisted on disk efficiently.

**Availability:** as an Internet/Web application, an MMORPG system should be able to respond to each user request within a certain period of time. The system also needs to have the ability to tolerate data loss. This can be achieved by increasing data redundancy and setting up fail-over servers.

**Scalability:** typically, online games start with a small or medium number of users. If the game is successful, the number can grow extremely. To avoid problems of a system laid out for too few users or the

Table 1: Data classification and requirements analysis.

| Data | Consistency | Performance | Availability | Scalability | Partitioning | Flexible model | Simplified processing | Security |
|---|---|---|---|---|---|---|---|---|
| **Account Data** | ☆ ☆ ☆ | ☆ ☆ | ☆ ☆ ☆ | ☆ | ☆ ☆ | ☆ | ☆ | ☆ ☆ ☆ |
| **Game Data** | ☆ | ☆ | ☆ ☆ ☆ | ☆ | ☆ ☆ | ☆ | ☆ ☆ | ☆ |
| **State Data** | ☆ ☆ ☆ | ☆ ☆ ☆ | ☆ ☆ ☆ | ☆ ☆ ☆ | ☆ ☆ ☆ | ☆ ☆ ☆ | ☆ ☆ ☆ | ☆ ☆ |
| **Log Data** | ☆ ☆ | ☆ ☆ | ☆ | ☆ ☆ ☆ | ☆ ☆ | ☆ | ☆ ☆ ☆ | ☆ ☆ ☆ |

costs of a system initially laid out for too many users, the data management needs to be extremely scalable (Gupta et al., 2008). Furthermore, log data will be appended continually and retained in the database statically for a long time (White et al., 2007). The expansion of data scale should not affect the database performance. Hence, the database should have the ability to accommodate the growth by adding new hardware (Iimura et al., 2004).

**Data Partitioning:** performing all operations on one node can simplify the integrity control, but that may cause a system bottleneck. Therefore, data must be partitioned into multiple nodes in order to balance the workload, process operations in parallel and reduce processing costs. Current partitioning schemes are most often based on application logic, such as partial maps (map servers). This does not easily integrate with the requirement of scalability, i.e. re-partitioning is not trivial when new servers are added. Accordingly, suitable partitioning schemes are a major research issue.

**Flexible Data Model:** part of the state data does not have a fixed schema, for example PCs have dynamic skills, tasks, and inventory and MMORPGs are typically bug-fixed and extended during their runtime. Therefore, it is difficult to adopt the relational model to manage such data. A flexible data model without a fixed schema is more suitable.

**Simplified Data Processing:** in MMORPGs only changes of account data and a part of state data must be executed in the form of transactions. In addition, the transaction processing in the database of a game is very different from in a business database. For example, in MMORPGs, there are many transactions, but most of the small size. Parallel operations with conflicts occur rarely. Strong consistency is not always necessary. Hence, a simplified data processing mechanism is possible.

**Security:** game developers always have to be concerned about data security. User-specific data, such as account data and chat logs, must be strongly protected. Furthermore, it must be possible to recover data after being maliciously modified.

**Ease of Use, Composability, and Re-usability:** the data management method should be easy for developers to use, and can be applied to other MMORPGs. Companies developing and maintaining MMORPGs should be able to re-use or easily adapt existing data management solutions to new games, similar to the idea separating the *game engine* from the *game content* currently widely applied. This requirement is independent of the class of data.

## 3 DATA MANAGEMENT FOR MMORPGs

**State of the Art:** when a player starts an MMORPG, all character state data of this player must be read from the state database at one time. This data is cached and managed in an in-memory database in real-time, and written to the disk periodically. When the player quits the game, this state data is persisted to the state database, and deleted from the in-memory database.

A disk resident database is not directly used because that it cannot cope with the heavy I/O workload of MMORPGs, and executing transactions in it will inevitably pause the game. This pause may occur at any time, which cannot be tolerated by a real-time system. However, an in-memory database cannot guarantee the data persistence when it fails, and its storage cost is high. Therefore, game providers still employ the disk resident database to backup data (White et al., 2007). In order to reduce the database I/O workload, data is asynchronously (5 to 10 minutes) written to the disk in a batch mode.

Currently, MMOPRGs apply distributed RDBS for data persistence, which can commit complex transactions and are proven to be stable. As an example, consider MySQL Cluster(Oracle, 2013) and it's characteristics. MySQL Cluster adopts a shared nothing architecture to ensure the system scalability. It automatically partitions data within a table based on the primary key across all nodes. Each node can help clients to access correct shards to satisfy a query or commit a transaction. For the purpose of guarantee-

ing availability, data is replicated to multiple nodes. MySQL Cluster applies a two-phase commit (2PC) mechanism to propagate data changes to the primary replica and one secondary replica synchronously, and then modifies other replicas asynchronously. In this case, at least one secondary replica has the consistent and redundant data, which can be used as a fail-over server while the primary server fails. When MySQL Cluster maintains tables in memory, it can support real-time responses. The result is that it can also be used as an in-memory DBS in MMORPGs.

**The Case for Cloud-based Data Management:** unfortunately, RDBMS cannot fulfill some data management requirements well, or there must be some provisos(Cattell, 2010). In contrast to RDBMS, Cloud-based storage systems have inherent advantages regarding their major characteristics like scalability and availability. Nevertheless, some typical requirements are neither fully supported by any of the two alternatives. Accordingly, we discuss key requirements and the suitability of the RDMS vs. Cloud-based storage systems.

- **High Performance:** traditional RDBMS are not designed for managing data with a large number of attributes, such as PC state data. If we create a wide table, i.e. with hundreds of columns, RDBMS often fail to manage it well. If we partition a wide table into several tables, the massive join operations would bring a great impact on the system performance. Different from RDBMS, Cloud-storage systems can manage all attributes by applying a simplified data model as well as data redundancy. Although this solution increases the storage burden on the system, it improves the query performance.
- **Scalability:** for large-scale Internet/Web applications, a single or a few servers cannot satisfy the user demand for data access. The only solution is scaling out. Although some RDBMS (e.g. MySQL Cluster, Oracle RAC, etc.) also have used the shared-nothing architecture, the system scalability is limited by its complex schema. Furthermore, data set volume growth has a significant impact on the system performance (Franke et al., 2011; Cattell, 2010). With their simplified data model, Cloud storage systems are proven to have a great potential for scalability (Franke et al., 2011).
- **Flexible Data Model:** the relational model of RDBMS is good at normalizing table schema and removing data redundancy, but not at adapting to a dynamic schema and processing big data. Cloud storage systems typically adopt a flexible data model, such as a key-value data model. There is no fixed schema for items. Each item consists of a key and a dynamic set of attributes.
- **Simplified Data Processing:** RDMS usually follow a strict transaction mechanism, such as a table-level or a row-level atomicity, multi-version concurrency control mechanism, transaction isolation, and rollback. As we discussed above, this is not necessary for all data in MMORPGs. Cloud systems are designed for Web applications, in which strong consistency is not as necessary as it in business applications. For this reason, they generally do not support transaction processing.

# 4 USING CASSANDRA FOR MMORPGs

For the above reasons, in this paper, we try to assess the usability of a Cloud storage system for MMORPGs. The existing RDBMS mainly has trouble in the management of state data and log data. Therefore, we tend to choose a Cloud storage system based on the access characteristics of these data.

State data is backed up to disk in frequent intervals, so the state database in MMORPGs should be write-intensive and have the ability to access large amounts of data in parallel. Cassandra has a similar application scenario with state database. It is designed for Web applications that perform more equal shares of concurrent write than read operations, such as a social networking website. Additionally, Cassandra is always writable, which makes it more suitable for managing state data than other Cloud storage systems. Therefore, we choose Cassandra as the object of the assessment.

In the following, we evaluate and predict potentials and open issues of Cassandra in the management of diverse data in MMORPGs by analysis of characteristics of Cassandra.

**Management of State Data:** state data must be persisted to the disk. In this process, a high performance for concurrent data access, a flexible data model, simplified data processing, and system scalability are required.

Cassandra has a decentralized structure (Lakshman, 2010), i.e., each node is identical and is able to initiate reads and writes independently. Data are automatically replicated to multiple nodes. Therefore, there is no network bottleneck and single points of failure, which can satisfy the write performance requirements of state data. This is differ-

ent with RDBMS (e.g., MySQL Cluster) and some other Cloud storage systems (e.g., Google Bigtable (Chang et al., 2006)), which usually adopt a primary/secondary model (the primary node may become a system bottleneck).

Cassandra provides a column family based data model, which is richer than a simple key-value store. Every row in a super column family in Cassandra consists of a row key and a dynamical set of super columns, each of which maintains a different number of columns. In this way, we can manage the PC data of one player in a single row, and partition data based on row key across multiple nodes in the cluster. Hence, there is no more join operation during reading data, and the read performance can be increased.

Cassandra adopts a shared-nothing architecture and a simplified data model as we mentioned above. As a result, it can scale out easily by adding new hardware, and reach a linearly increasing read and write throughput.

Another advantage is that Cassandra provides a quorum based data replication mechanism. That means as long as a write can receive a quorum responses, it can complete successfully. In this way, Cassandra ensures availability and fault tolerance. Additionally, by controlling the number of replicas that must respond to a read request, Cassandra offers a tunable data consistency.

On the other hand, there are still some open issues that cannot be well solved by Cassandra directly.

Real-time: as discussed above, managing the state data in real-time is a big challenge for a disk resident RDBMS. Unfortunately, Cassandra with its shared nothing architecture is not intended to provide real-time support. Although Cassandra firstly manages data in Memtable in memory, we cannot limit all related data to one node. The changes of data must be propagated to other nodes in the form of messages, which will increase the response time. Therefore, we have to continue applying an in-memory database in each zone server.

Data consistency: Cassandra employs Read Repair to guarantee data consistency. It means that all replicas must be compared in order to return the up-to-date data to users. In MMORPGs, state data may have hundreds of attributes and are distributed in multiple data centers. Hence, such a feature will significantly reduce the read performance and increase the network traffic. A common method for solving this problem is to limit the data consistency in the application layer of the system(Gropengießer et al., 2011). Note that Cassandra records timestamps in each column, and uses it as version identification. Therefore, a possible solution is that we record the timestamps from a global server in both server side and columns in Cassandra. When we read state data from Cassandra, the timestamp recorded on the server side will be sent with the read request. In this way, we can find out the most recent data easily. We set all columns in a single row with the same timestamp, so that only one row key and one timestamp are stored for a single row on the server side. In addition, these timestamps are partitioned and managed by servers in parallel. For these two reasons, accessing these timestamps in the server side will be not a challenge.

Customized functions: we need to develop some new functions based on features of MMORPGs. There are two examples: the mainly task of state database is data backup. Each value that has been written in the database must be persistent until it is out of date. Therefore, if an update operation fails in the committing process, values that have been recorded in the database should be applied and other values must try to update again. To comply with this rule, a column-level atomicity is sufficient (Cassandra offers an atomicity at the column family level.); note that writes and queries in games are relatively fixed. Hence, we propose to add a stored procedure to each node to optimize the system performance. The stored procedure is also responsible for splitting a write operation into column-level, which is convenient for offering a column-level atomicity.

Data partitioning: in order to get a high performance of accessing the entire state data of a character or object, we manage these data in a single row and partition them based on row key horizontally. However, this method increases the processing costs of querying data across characters. We can partly relieve this problem by creating indexes. Unfortunately, it increases the response time of write. Cassandra cannot solve this problem autocratically, so a more efficient data partitioning method based on semantics (e.g., map zones) or a mix of both has to be proposed.

**Management of Log Data:** log data has a large scale. Once it has been written to disk, it does not need to be modified. In addition, log data is usually analyzed after a long time, so strong consistency is not important. Generally speaking, in addition to the demand for scalability, management requirements of log data are not difficult to meet. Cassandra provides scalability, availability, tunable data consistency and high fault tolerance, as we mentioned. Cooperating with Hadoop MapReduce, it can also process and analyze large data set in parallel. As a result, Cassandra can manage log data well.

If there is an update conflict on two nodes in Cassandra, the last write wins. However, update of Log

data is append operation. That means, all values need to be persisted. Therefore, the conflict values need to be sorted by timestamps and combined. When two write operations with different timestamps modify a same log data concurrently, the write operation with lower timestamp should win, and another needs to put in the queue. This is convenient for sorting.

**Management of Account Data:** account data must be processed carefully because that an operation error may cause an economic dispute or a player's information disclosure. As a result, each operation needs to be executed in the form of a transaction. Unfortunately, Cassandra cannot perform transactions. Furthermore, account data has no strong requirements for scalability and high performance, so RDBMS as a service is more suitable for it.

**Management of Game Data:** game data exists typically in the form of files. However, Cassandra belongs to the Cloud structured data system. Moreover, game data processing does not pose a challenge to the existing solutions. Unless it is stored on the client side, we can manage it in a Cloud-based file system, such as HDFS (Shvachko et al., 2010).

## 5 CONCLUSIONS

In this paper, we have shown that a single storage system cannot meet management requirements of all data sets of MMORPGs. Although RDBMS are intended to provide a high-level of consistency, it falls short of fulfilling requirement regarding scalability and availability. In this paper, we proposed how to apply Cloud storage systems, and specifically Cassandra to MMORPGs. By matching characteristics of Cassandra with application requirements, we found that Cassandra basically can meet the persistence requirement of state data and log data. However, a Cloud storage system specifically designed for these data sets still needs to be developed. Additionally, Cassandra fails to manage account data and game data well, and process state data in real-time. Accordingly, we believe that a set of co-operating services for MMORPGs, such as RDBMS as a Service and Cloud-based file storage systems need to be developed.

## REFERENCES

Apache (2013). Cassandra. http://cassandra.apache.org/.

Cattell, R. (2010). Scalable SQL and NoSQL Data Stores. *ACM Special Interest Group on Management of Data (SIGMOD)*, 39(4):12–27.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2006). Bigtable: A distributed storage system for structured data. In *Proceedings of 7th Symposium on Operating System Design and Implementation(OSDI)*, pages 205–218.

Franke, C., Morin, S., Chebotko, A., Abraham, J., and Brazier, P. (2011). Distributed Semantic Web Data Management in HBase and MySQL Cluster. In *IEEE International Conference on Cloud Computing, CLOUD 2011*, pages 105–112.

Gropengieß er, F., Baumann, S., and Sattler, K.-U. (2011). Cloudy transactions cooperative xml authoring on amazon s3. In *Datenbanksysteme fr Business, Technologie und Web (BTW)*, pages 307–326.

Gupta, N., Demers, A., and Gehrke, J. (2008). SEMMO : A Scalable Engine for Massively Multiplayer Online Games [ Demonstration Paper ]. In *ACM SIGMOD Conference 2008*, pages 1234–1238.

Iimura, T., Hazeyama, H., and Kadobayashi, Y. (2004). Zoned Federation of Game Servers : a Peer-to-peer Approach to Scalable Multi-player Online Games. In *Proceedings of the 3rd Workshop on Network and System Support for Games, NETGAMES 2004*, pages 116–120.

Lakshman, A. (2010). Cassandra - A Decentralized Structured Storage System. *Operating Systems Review*, 44(2):35–40.

Li, F. W., Li, L. W., and Lau, R. W. (2004). Supporting continuous consistency in multiplayer online games. In *12. ACM Multimedia 2004*, pages 388–391.

MuchDifferent (2013). Mmo architecture. http://www.muchdifferent.com/?page=game-unitypark-architecture-mmo.

Oracle (2013). Mysql cluster overview. http://dev.mysql.com/doc/refman/5.5/en/mysql-cluster-overview.html.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The Hadoop Distributed File System. In *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10.

Vogels, W. (2008). Eventually consistent. *ACM Queue*, 6(6):14–19.

White, W., Koch, C., Gupta, N., Gehrke, J., and Demers, A. (2007). Database research opportunities in computer games. *ACM Special Interest Group on Management of Data (SIGMOD)*, 36(3):7–13.

Zhang, K., Kemme, B., and Denault, A. (2008). Persistence in massively multiplayer online games. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games, NETGAMES 2008*, pages 53–58.