Cloud-based Persistence Services for MMORPGs

Ziqiang DIAO[1], Shuo WANG, and Eike SCHALLEHN
Institute of Technical and Business Information Systems,
Otto-von-Guericke University Magdeburg,
39106 Magdeburg, Germany

Abstract. Massive Multiplayer Online Role-Playing Games represent very complex applications, and their growing popularity makes them an important commercial factor in the entertainment area. However, the need to fulfil very high-level technical and user requirements makes the development a risky task. We address these issues from a data management perspective with services to support the management of a persistent game world along with other data sets. The main focus of this contribution is to investigate how scalability and a sufficient level of consistency can be achieved by using and extending Cloud-based technology.

Keywords. Cloud storage system, online game, data persistence

Introduction

Massively Multiplayer Online Games (MMOGs) and the popular sub-class of *Massively Multiplayer Online Role-Playing Games* (MMORPGs) do have a significantly growing importance as a form of entertainment and, accordingly, as an economic factor[2]. The indisputable attractiveness of these kinds of games, from a technical point of view, is based on

- complex architectures of game clients accessing partitioned game server clusters (or *shards*)
- supporting a complex application logic for competitive and co-operative gameplay
- of a possibly very high but not fixed or easily foreseeable number of concurrent users
- in a continually existing virtual reality (persistent world) with
- high requirements from a user's perspective regarding availability, consistency, and real-time interaction.

These characteristics go far beyond traditional games or even simple distributed or multiplayer games and make the development of such games more expensive tasks with high risks. Common problems, especially during early days of a new game going online, are failures to scale up to an unexpectedly high number of users (even the

[1] Corresponding Author: Ziqiang Diao, Otto-von-Guericke University Magdeburg, B. 29, Universitaetsplatz 2, 39106 Magdeburg, Germany; E-mail: diao@iti.cs.uni-magdeburg.de.

[2] http://www.gamesindustry.biz/articles/mmo-subscription-revenue-to-hit-USD2-billion-by-2013 (accessed 20.02.2014).

currently most popular MMORPG *World of Warcraft* faced these problems, like most others) and inconsistencies of game data (e.g. *GTA V Online*).

The idea to use cloud-based or cloud-inspired technology to address issues of scalability and availability is a current topic[3], most of all in industrial research and development[1], [2]. Nevertheless, the complex set of requirements cannot easily be fulfilled by a singular or generic solution. As an example, because of the CAP Theorem[3] we know that consistency is limited for a distributed system providing tolerance for partitioning and availability. Though (or because of) using a cloud-based storage system to deal with the high load, *GTA V Online* faced problems related to consistency.

To support developers of MMOGs, resolve the issues described above, and as a result, decrease the risk of project failure, we see our research within an envisioned set of cloud-based services sketched in Figure 1. Similar to traditional game engines, which provide a more high-level interface to typical gaming aspects such as graphics, physics, and generic gameplay issues, MMOGs should be based on reusable functionality offered as services configurable according to a given MMOGs requirements. Our current work focuses on persistence services to deal with various types of game data, by using and extending Cloud storage systems such as Cassandra as well as more conventional technology like RDBMS in prototypical MMORPG environments. The results presented as a contribution in this paper are as follows:

1. we analyse MMOGs regarding different classes of data and various requirements attached to these classes to come up with a minimal set of services required for game data storage
2. based on a prototypical implementation we evaluate trade-offs triggered by varying degrees of scalability and availability for access patterns typical in MMOGs to support the decision for a specific storage service and its configuration.
3. we analyse game-specific requirements of consistency, which are different from those in conventional database applications, and describe an extension for Cloud-based Storage systems to fulfil these requirements.

1. A Cloud-based Service Infrastructure for MMORPGs

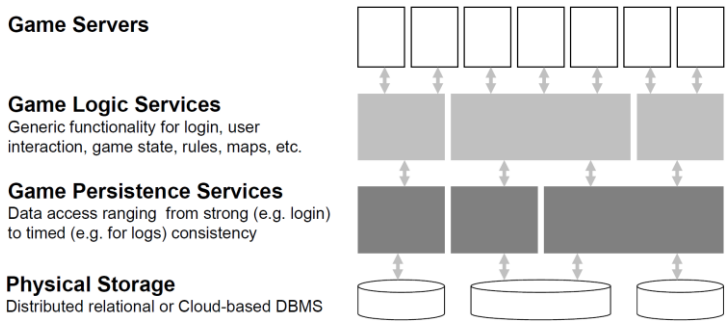


Figure 1. Architectural Framework of Cloud-based services for MMORPGs

[3] http://www.tesoelite.com/2013/04/elder-scrolls-online-mega-server-explained/ (accessed 20.02.2014).

In this section we will analyse requirements of MMORPGs and, based on this, sketch an infrastructure that serves as a frame for our research activities described in more detail in the following chapters. To point out data management requirements, for our following considerations we classify data into four data sets. These different classes vary widely regarding their requirements and have to be managed accordingly. A more detailed version of this analysis can be found in [4].

Account data: this category of data includes user account information, such as user ID, password, recharge records, and account balance. These data are only used when players log in to or log out of a game for identification and accounting purposes.

Game data: data such as world geometry and appearance, object and NPC (Non Player Character) metadata (name, race, appearance, etc.), system logs, server configurations, 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 and is often not subject for the server side data management.

State data: player character (PC) metadata, position as well as state of characters and objects, and inventory in MMORPGs are modified constantly. Modifications of state data are currently executed by an in-memory database in real-time and backed up to the disk resident database periodically.

Log data: analysing user chat history and operation logs in an MMORPG is the most objective and direct way for game providers to evaluate the game, find out the operating habits of players, explore the game development trends, and even supervise the financial system of the game world.

To support the management of such heterogeneous data sets, there are numerous requirements that need to be fulfilled, e.g. regarding performance, availability, flexibility, security, etc. For our research described within the context of this paper, we will focus on the following requirements:

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 [5]. Changes of state data and account data must be recorded in the database. It is intolerable that players find that their last game records are lost when they log in to the game again. As a result, a strong or at least a Read-Your-Writes consistency [6] 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, or the clothing style of a game character is allowed to be different among client sides. Log data are generally not analysed immediately. Hence, eventual consistency [6] is sufficient for these two classes of data.

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 costs of a system initially laid out for too many users, the data management needs to be extremely scalable [7]. Furthermore, log data will be appended continually and retained in the database statically for a long time [8]. The expansion of data scale should not affect the database performance. Hence, database should have the ability to accommodate the growth by adding hardware [9].

Ease of use, Composability, and Re-usability: The data management system should be easy to use for developers, and it should be easy to apply it to various 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.

2. Supporting Game Scalability

In order to provide a proof of concept for Cloud-based online game, as well as to evaluate the scalability and performance of it, we have designed and implemented a prototypical game platform by porting an open source MMORPGs test environment to Cassandra. This platform consists of game clients and servers. The server side includes some simplified game logic, such as receiving commands from clients and sending character states (e.g. characters' movements and chatting) back to clients. The client side is scripted to support experimental setups of thousand of players. We have to point out that physical resources for the experiments were limited as described below, so the focus is most of all on scaling the number of clients versus a small set of up to five Cassandra servers. Nevertheless, these experiments returned some interesting results.

Test Environment: For running this prototype, we have applied 8 virtual machines with Ubuntu operating system, each of which configures 2.40 GHz CPU, 8 GB memory (28.3%~34.5% used during the experiment) and 91GB hard disk (20GB used). For security reasons, these virtual machines cannot be visited from outside directly. A Client needs to connect a stepping stone server through the security shell (SSH) protocol firstly and then get access to the virtual machines via it indirectly. These virtual machines are used to deploy Cassandra cluster and game servers. Figure 2 shows a detailed structure of the prototype.

In our prototype, we have adopted Cassandra[4] (version 1.2.5) to manage data. Cassandra is an open source Cloud database and is designed to provide high performance and scalability. Furthermore, it is proposed to be a relatively appropriate Cloud database for MMORPG [4]. On the server side, we have applied a high level Java API (Hector[5]), which makes it possible to access Cassandra through RPC (Remote Procedure Call) serialization mechanism. We have used Darkstar[6] to realize the communication between the client and the server. Darkstar is an Apache project, which is specifically designed for MMOG network architecture.

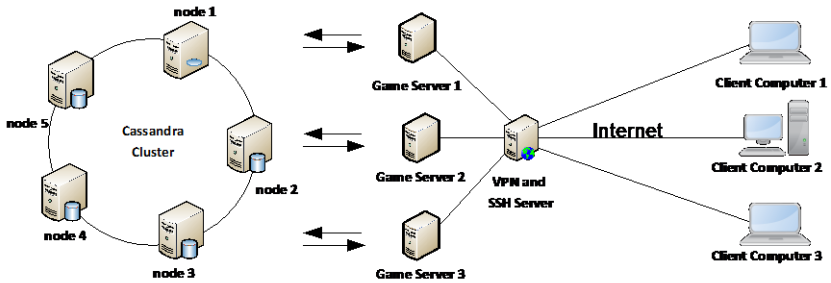
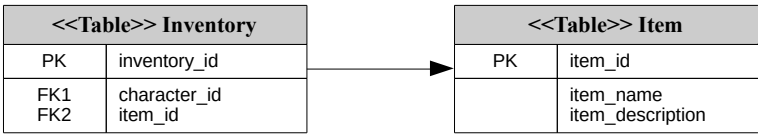
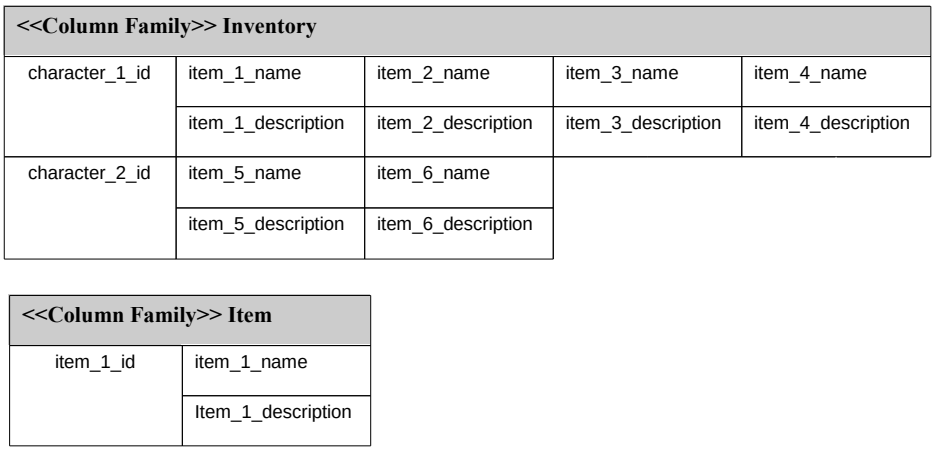


Figure 2. The infrastructure of the prototype

[4] Cassandra website: http://cassandra.apache.org/ (accessed 20.02.2014).
[5] Hector website: http://hector-client.github.io/hector/build/html/index.html (accessed 20.02.2014).
[6] DarkStar website: http://sourceforge.net/apps/trac/reddwarf/ (accessed 20.02.2014).



(a) Inventory and item table design in RDBMS



(b) Inventory and item column family design in Cassandra

Figure 3. Comparison of schema design

2.1. Game Database Schema Design with Cassandra

The process of building a new data-driven application running on top of a RDBMS is typically like this: firstly we need to analyse the domain; and then model it to some normalized tables; finally set foreign keys to refer to the related columns in other tables. However, the design concept between Cassandra and RDBMS is totally different.

Cassandra doesn't support a powerful query language and cannot perform join operations. In other words, there are only limited query operations supported by Cassandra. In order to avoid join operation and improve the efficiency of query, denormalization in Cassandra is common. That means, we add data redundancy in column family so that all necessary data could be obtained from one row. In this case, there is no more join operation in a query.

For this reason, a primary criterion for Cassandra schema design is starting with queries. Namely, it is necessary to determine first what queries does the application need; think about query patterns up front; and then organize the data around the queries; design column families accordingly at last.

An example to highlight the difference is showed in Figure 3. In RDBMS, we only store character ID and item ID in an inventory table to reduce the data redundancy. We have to use a foreign key (item ID) to join inventory and item tables when we query the inventory of a character. However, in Cassandra all information could be stored in a single row of the inventory column family, so there is no more expensive joins across cluster. The reason is that Cassandra is a partitioned row store, which has no fixed data model as RDBMS and distinguished by row key. Each row could have different number of columns. Hence, we can store an item name as column name and an item

description as the column value in a row if the item name is unique. Although we still need an item column family in our prototype, it is only queried by the game engine during the game and has no relation with inventory column family.

2.2. Evaluation

We have implemented a simplified Command-Line game client for the experiment because it consumes less system resources and works like the GUI client. Our benchmark is a player's normal behaviour, such as moving and trading. From data management perspective, the essence of these operations is performing update/select to the database. For these reason, for each character we have created one row in the column family, each of which insists of 20 columns. The row size is 540s bytes. The client randomly generates a writing/reading command to one column, and then sends it to the game server. Meanwhile, the response time of each command has been recorded.

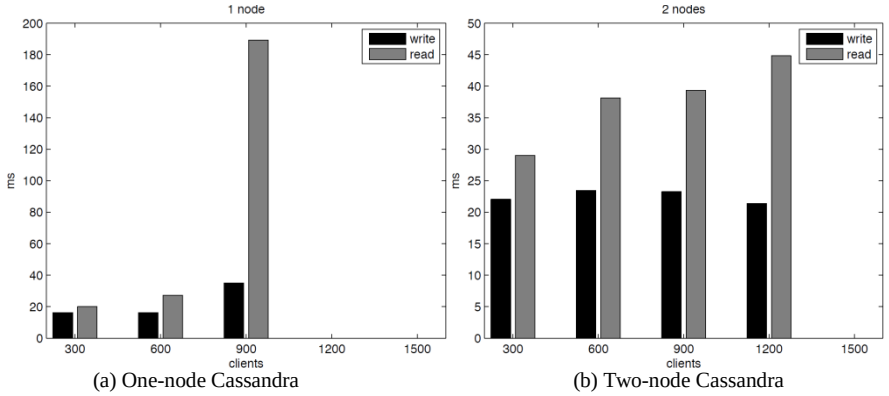
This evaluation focuses on the potential scalability and performance of our prototype in the case of multiplayer concurrent accessing. For guaranteeing data consistency as well as avoiding the effect of replication on the reading and writing performance, we set the replication factor of the Cassandra cluster as one. That means, the cluster has only one replica for each column family and a read/write operation will success once one node responds to it.

Scalability of a database is reflected by its ability that through increasing the number of database nodes to linear improve database performance. Hence, the goal of this experiment is to evaluate the influence of the increasing player number on the database response time. For this purpose, we have fixed the number of game servers to three, each of which is connected by 100, 200, 300, 400, and 500 clients in turn (500 is the maximum number that a single game sever in our prototype can concurrently support [10]). That means, the Cassandra cluster handles 300, 600, 900, 1200, 1500 clients separately. Every client sends 500 reading or writing commands. Therefore, Cassandra cluster needs to handle 150000, 300000, 450000, 6000000, 750000 commands in turn. The corresponding response time of each command is recorded and afterwards the mean response time is calculated. We have carried out five groups of experiments in total. The number of nodes in Cassandra cluster is set from one to five.

From Figure 4 (a), we can find that, a high performance of one-node Cassandra is got under 300 and 600 clients. When the number of clients reaches 900, the response time of read operation increases sharply over 180ms, which is unexpected. If we start 1200 clients, Cassandra cluster will not respond the write and read request normally. Many clients report a connection time out exception because of limitation of Cassandra I/O. So we terminate the first-round experiment and conclude that one-node Cassandra can only support up to 600 clients under our experimental environment.

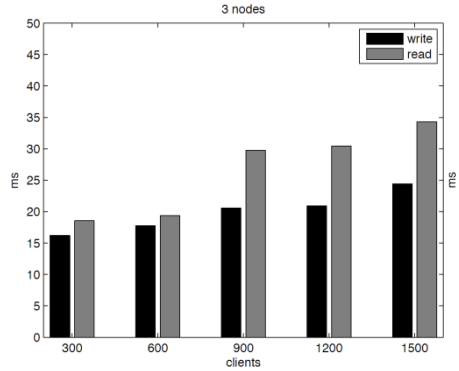
Figure 4 (b) shows that the maximum number of clients reaches 1200 when there are two nodes in Cassandra cluster. In the case of 1500 concurrent connections, the issue of timeout appears again. Therefore, we believe that two-node Cassandra cluster can support about 1200 clients by using our prototype.

Figure 4 (c), (d) and (e) present that, when the number of nodes in Cassandra cluster is more than three, our prototype can support at least 1500 concurrent players.

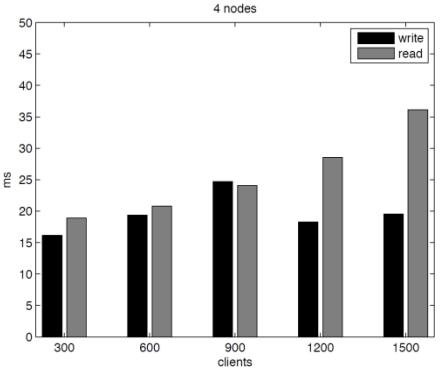


(a) One-node Cassandra

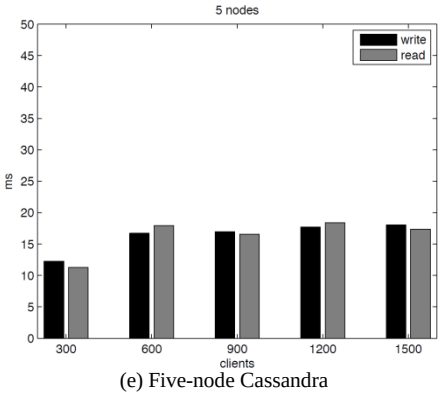
(b) Two-node Cassandra



(c) Three-node Cassandra



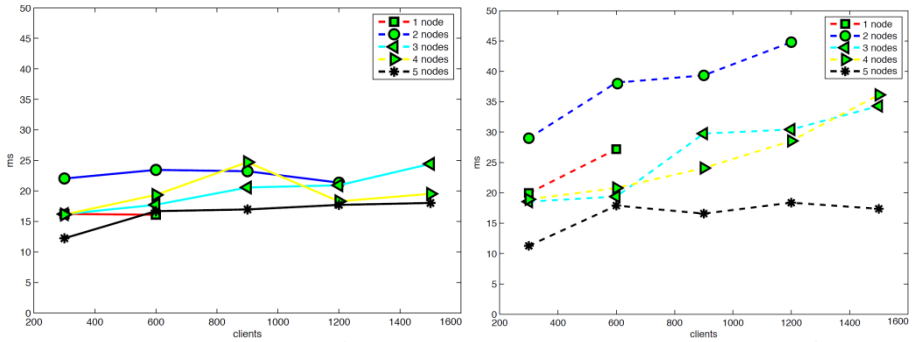
(d) Four-node Cassandra



(e) Five-node Cassandra

Figure 4. Performance of Cassandra cluster

In order to observe the different results in Figure 4, we plot the writing and reading response time in Figure 5 (a) and (b). The reading response time of one-node Cassandra under 900 clients is unexpected, so we have removed it from the new figure.



(a) Average response time of writes (b) Average response time of reads

Figure 5. Comparison of response time of writes and reads while applying from one to five nodes in Cassandra cluster

According to the experimental result, we can observe the following tendency:

1. The number of concurrent players supported by our prototype can be increased (from 600 to 1500) by adding more nodes into the Cassandra cluster;
2. Cassandra presents a satisfactory writing performance (around 20ms), which is relatively better than the reading performance. Furthermore, the change in the number of nodes has little influence on writing performance (concentrated between 15ms and 25ms). In contrast, reading performance is obviously improved by adding nodes. In the case of five-node Cassandra cluster, reading and writing performance tends to become similar;
3. Five-node Cassandra cluster exhibits the best and most stable performance in all range of clients' number. With increasing number of clients, there is no obviously variation of reading and writing response time. Both of them fluctuate around 15ms;
4. Generally, the system performance has been improved by scaling out Cassandra cluster. For example, five-node has the best performance; three-node and four-node cluster are observably better than two-node cluster. However, there are still some exceptions. An example is that the performance of three-node and four-node Cassandra is similar. Theoretically, four-node Cassandra should be better. However, our experimental data shows some contrary results, such as reading response time at 1500 clients and writing response time at 900 clients. It may be caused by network latency, system configurations, or even some internal processing mechanism of Cassandra. Unfortunately, our prototype cannot detect it and explain this tendency;
5. One-node Cassandra shows a better performance in the case of 300 or 600 clients. The reason could be that the advantage of multi-node Cassandra cluster is not outstanding when the concurrent players are relatively less. In addition, the communication between nodes also consumes some time since data are distributed on different nodes.

Base on the analysis above, we can conclude that Cassandra exhibits a satisfactory scalability under our test environment. With increasing numbers of clients, the database performance encounters a bottleneck. However, the database throughput as well as response time can be improved easily by scaling out the cluster; Cassandra shows a high performance in the experiment. The response time of writing and reading typically fluctuates between 10ms and 40 ms, which fulfils the requirement of an MMOG [11];

Cassandra is a write-intensive database. The experiment results show that its writing performance is stable and excellent. This feature makes it suitable to perform a backend database of a multiplayer online game, which needs to handle more write requirements.

3. Supporting Game Consistency

Though using a Cloud storage system can solve the system scalability issue, it also introduces new challenges, such as guarantee of game consistency. It is noteworthy that we are not going to discuss the data synchronization issue among players in this paper but data persistence in the storage system.

3.1. Game Data Persistence Issue in the Cloud

Inconsistent data in a database may lead to system anomalies and/or even economic losses of users. Therefore, RDBMS adopts various measures to ensure data consistency. In contrast, a Cloud storage system sometimes deliberately tolerates this situation. There are two main factors causing data inconsistency in the Cloud:

1. According to the CAP theorem, a distributed system can only have two of the three properties: consistency, availability, and partition tolerance. Cloud storage system typically sacrifices data consistency to get high availability. For example, although in Cassandra we can set the replication factor to determine the consistency level of each read/write operation, data consistency is confined to a row level. Hence, updates across rows can only be propagated asynchronously (eventual consistency) in Cassandra.
2. Concerns of query efficiency, convenience, data protection, and communication cost lead to exploit data redundancy in a distributed database. Unfortunately, data anomalies may occur after partially updating the redundant data. Particularly in a Cloud storage system like Cassandra, which does not support transaction and relies on denormalization, to ensure data consistency becomes more difficult.

The typical application scenario of a Cloud storage system like SNS usually does not care about the timeliness of data, so inconsistent data will not affect the user experience. However, it is sometimes unacceptable in online games. For this reason, data inconsistency becomes a new issue that has to be addressed. Particularly, how to solve it in a concurrent environment becomes a challenge.

3.2. Classification of Data Consistency in Online Game

In order to take advantages of Cloud storage system, ensure system availability, and keep normal operation of an online game, we need to analyse game-specific requirements of consistency at first. In this paper, we classify them into four groups:

Strong consistency. In a distributed database system, it means that each replica should hold the same view on data values at any time. Strong consistency is required by account data. Anomalies of such data will bring troubles to players as well as the game provider, or even lead to an economic or legal dispute.

Read-your-writes consistency. Changes of state data must be returned timely when a player reads them again. However, for game engine and other players, timeliness of these data is not so critical. Hence, data inconsistency among replicas is

allowed. We only need to take some measures to ensure the owner of a character to get the up-to-date view. Hence, a read-your-writes consistency is acceptable [6].

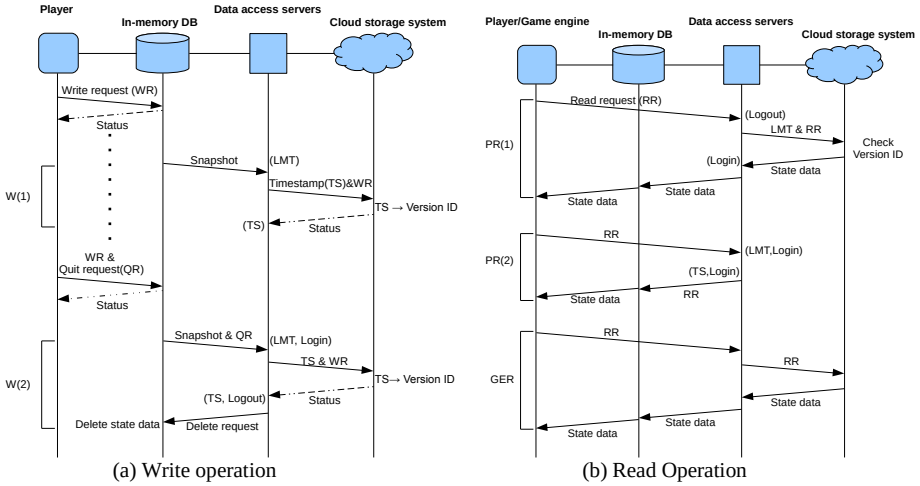
Causal consistency. Some of the game data may have a local replica on the client side, or have been replicated on all game servers; some of them (e.g. item information in Figure 3b) may be redundant in character's state information. Only game developers have permission to modify game data. In addition, changes of these data cannot be propagated synchronously to all replicas. We can only ensure that these data are consistent when a player/game server is online. Other offline players/game servers will continue to retain the outdated data. In this paper, we state this kind of consistency as causal consistency [6], [12].

Timed consistency. Log data have a big volume in common, which will consume large amounts of network bandwidth and affect the system performance while propagating them. Moreover, log data are generally analysed after a long time. Therefore, asynchronous propagation is feasible when database is idle. We propose a deadline-based consistency model like timed consistency for log data [13], [14]. In this paper, timed consistency represents that updated values must be propagated to all replicas within a time bounded.

3.3. Extension for Cloud Storage System

In order to fulfil all requirements of consistency in a Cloud-based online game efficiently, the game server is needed to spread the database workload.

Support of strong consistency. Cassandra is capable of ensuring strong consistency in a row level. So if only consider this aspect, Cassandra is qualified for managing account data. The proviso is that the replication factor of writes is set to *ALL*.



(a) Write operation (b) Read Operation

Figure 6. Executions of Write/ Read Operations: in Figure (a), W(1) describes a general backup operation; W(2) shows the process of data persistence when a player quits the game; in Figure (b), PR(1) shows a general read request from the player; In the case of PR(2), the backup operation is not yet completed when the read request arrives; GER presents the execution of a read operation from the game engine.[15]

Trading among players involves a transaction across rows in database. In this case, a centralized in-memory relational database is required on the server side to take over the responsibility of Cassandra. That means, during the game, players only access data from a memory-resident database. There are three main tasks of Cassandra cluster in the new architecture: sending character's state data to the player and game server when a player starts the game; backing up state data from in-memory database and store log data periodically during the game; persisting character's state data after a player quits the game. (The character's state data will be then deleted from the in-memory database.)

The optimized architecture makes it possible to take advantages of RDBMS and Cloud storage system. The in-memory database can support transaction processing (strong consistency), powerful queries, and real-time response, which is much faster than Cassandra cluster (a distributed disk-resident database). The retrieval functionality of Cassandra is simplified so that it can focus on game scalability. Furthermore, persisting data in Cassandra keeps data of the in-memory database in a small-scale.

Support of read-your-writes consistency. The most convenient way is to set the sum of replication factor number of writes and reads greater than the replica number in Cassandra cluster. In other words, both writes and reads need to get responses from multiple replicas. Obviously, this strategy causes unnecessary data propagation, thereby effecting on database performance and availability. We propose to introduce a data access server on the server side to assist with ensuring data consistency [15]. Data access server is responsible for data exchange among players, game servers, and Cassandra cluster. It maintains a timestamp table inside, which records the last time (timestamp) of a character's state data backed up to Cassandra (see Figure 6 (a)). This timestamp is used here as version identification, which is also stored with the state data in Cassandra in parallel. Figure 6 (b) shows that when a player starts a game, the read request will be sent with relative timestamp from data access server to Cassandra. By comparing the timestamp, Cassandra can accurately find out the latest record. Our proposal can significantly reduce the amount of data propagation inside Cassandra cluster. That is because state data of a character need only be propagated to most/all replicas when the player quits the game. Moreover, a read request normally only needs to obtain the response from one replica. Queries from game engine do not need to compare the timestamp and thereby may get outdated value.

Support of causal consistency. Inconsistent data caused by data redundancy cannot be detected and fixed by Cassandra itself. Game server has to play a role of arbitrator, when data conflict occurs during the game. The latest data will be then updated to Cassandra. For example, the item information is redundant in all characters' state data (see Figure 3), so changes of that cannot be synchronized to all replicas. For this reason, an item column family is required in Cassandra, which keeps the up-to-date information of items and is retrieved by game server to reconcile conflicts.

Support of timed consistency. While storing log data, data access server will be used as a global counter, which generates a monotonically increasing timestamp. The timestamp will be exploited in the row key in Cassandra. Writes of log data will be propagated to a quorum of replicas at first. The internal consistency strategy of Cassandra, such as *Read Repair* and *Anti-Entropy*, will then ensure that the log data are eventual consistent and ordered.

4. Conclusion

In this paper we have shown how Cloud-based data management solutions can be applied to implement scalable and re-usable services providing levels of consistency suitable for the application specific requirements of MMORPGs. Scalability was investigated within an experimental setup that, first of all, provided a proof of concept for using Cloud data management for MMORPGs. Secondly, despite the limits regarding the number of servers tested with the prototype, the experiments have shown that the scalability is satisfactory and makes this a viable alternative to the typically used RDBMS. Nevertheless, the consistency levels provided by Cloud-DBMS may not be sufficient for all MMORPG data sets. We provided an according data set classification and derived suitable consistency levels and how these could be achieved by extending existing Cloud data management solutions. In our future work we will focus on the evaluation of scalability (continued) and consistency, as well as a clean separation of concerns by providing minimal interfaces for persistency services.

References

[1] Das S., Agrawal D., Abbadi A. E. G-store: a scalable data store for transactional multi key access in the cloud. In: Hellerstein J. M., Chaudhuri S., Rosenblum M. (eds.) *Proceedings of the 1st ACM symposium on Cloud computing, 10- 11 June, Indianapolis, Indiana, USA.* NY: ACM, 2010. 163–174.

[2] Muhammad Y. Evaluation and Implementation of Distributed NoSQL Database for MMO Gaming Environment [dissertation]. Uppsala: Uppsala University press, 2011. 51p.

[3] Gilbert S., Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM Special Interest Group on Algorithms and Computation Theory*, 2002, 33(2), 51–59.

[4] Diao Z., Schallehn E., Wang S., Mohammad S. Cloud Data Management for Online Games: Potentials and Open Issues. *Datenbank-Spektrum*, 2013, 13(3), 179–188.

[5] Li F. W. B., L. Li W. F., Lau R. W. H. Supporting continuous consistency in multiplayer online games. In: Schulzrinne H., Dimitrova N., Sasse M. A., Moon S. B., Lienhart R. (eds.) *Proceedings of the 12th annual ACM international conference on Multimedia, 10-16 October, New York, NY, USA.* NY: ACM, 2004, 388–391.

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

[7] Gupta N., Demers A., Gehrke J. SEMMO : A Scalable Engine for Massively Multiplayer Online Games. In: Wang J. T. (ed.) *ACM SIGMOD Conference, 10-12 June, Vancouver, BC, Canada.* NY: ACM, 2008. 1234–1238.

[8] White W., Koch C., Gupta N., Gehrke J., Demers A. Database research opportunities in Computer Games. *ACM Special Interest Group on Management of Data*, 2007, 36(3), 7–13.

[9] Iimura T., Hazeyama H., Kadobayashi Y. Zoned Federation of Game Servers : a Peer-to-peer Approach to Scalable Multi-player Online Games. In: Feng W. (ed.) *Proceedings of the 3rd Workshop on Network and System Support for Games*, 30 August, *Portland, Oregon, USA.* NY: ACM, 2004. 116–120.

[10] Wang S. Towards Cloud Data Management for Online Games - A Prototype Platform [dissertation]. Magdeburg: OVGU press, 2013. 95p.

[11] Chen K., Huang P., Huang C., Lei C. Game Traffic Analysis: An MMORPG Perspective. *Computer Networks*, 2006, 50(16), 3002–3023.

[12] Lamport L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 1978, 21(7), 558–565.

[13] Liu H., Bowman M., Chang F. Survey of state melding in virtual worlds. *ACM Computing Surveys*, 2012, 44(4), 1–25.

[14] Torres-Rojas F. J., Ahamad M., Raynal M. Timed consistency for shared distributed objects. In: Coan B. A., Welch J. L. (eds.) *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing, 3-6 May, Atlanta, Georgia, USA.* NY: ACM, 1999. 163–172.

[15] Diao Z. Consistency Models for Cloud-based Online Games : the Storage System's Perspective. In: Sattler K., Baumann S., Beier F., Betz H., Gropengießer F., Hagedorn S. (eds.) *25rd GI-Workshop on Foundations of Database, 28- 31 May, Ilmenau, Germany*. Aachen: CEUR-WS, 2013. 16–21.