Databases and Information Systems VIII
H.-M. Haav et al. (Eds.)
© 2014 The authors and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License.
doi:10.3233/978-1-61499-458-9-71

CloudCraft: Cloud-based Data Management for MMORPGs

Ziqiang DIAO [1], Shuo WANG, Eike SCHALLEHN and Gunter SAAKE

Institute of Technical and Business Information Systems,
Otto-von-Guericke University Magdeburg,
39106 Magdeburg, Germany

Abstract. Massively Multiplayer Online Role-Playing Games (MMORPGs) are very sophisticated applications, which have significantly grown in popularity since their early days in the mid-90s. Along with growing numbers of users the requirements on these systems have reached a point where technical problems become a severe risk for the commercial success. Within the CloudCraft project we investigate how Cloud-based architectures and data management can help to solve some of the most critical problems regarding scalability and consistency. In this article, we describe an implemented working environment based on the Cassandra DBMS and some of the key findings outlining its advantages and shortcomings for the given application scenario.

Keywords. Cloud storage system, online game, data persistence

Introduction

Massively Multiplayer Online Role-Playing Games (MMORPGs) have popularized the term Persistent World [10], which describes a virtual environment which continuously exists and changes, no matter whether millions of users, only few users, or even none at all interact with it [20]. From a computer science perspective, these *Persistent Worlds* represent very complex information systems consisting of multi-tiered architectures of game clients [18], game logic, and game data management which typically implement application-specific patterns of partitioning/sharding, replication, and load-balancing to fulfill the high requirements regarding performance, scalability, and availability [9,19]. On the other hand, MMORPGs, because of their popularity, have become common applications with a huge economic importance[2].

To support the development of such complex applications, recently developed Cloud DBMS like Cassandra, HBase, Riak, etc. seem like a perfect match for these requirements [23]. Nevertheless, a closer look shows some problems, mainly in two fields:

[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)

Table 1. Data classification and game consistency.

Data Sets	Description	Consistency
Account Data	player account and account balance	strong consistency
Game Data	game world's geometry and appearance, metadata of object and NPC , system logs, server configurations, and game rules/scripts	Causal consistency
State Data	metadata of avatar, position information, state of avatars and objects, and inventory	Read-your-writes consistency
Log Data	player's chat history and operation logs	Timed consistency

Consistency and availability: MMORPGs require a very high consistency of some data sets, e.g. for account and game state data. In accordance with the CAP Theorem [3] consistency in Cloud data management is either very loosely defined (e.g. eventual consistency) or must be traded versus availability and/or performance. The latter may not be an option for MMORPGs.

Scalability and performance: the advantages of an easily scalable data management solution for the development and maintenance of an MMORPG are obvious, but some specific requirements like partitioned game logic servers, real-time requirements, and very specific workloads (e.g. write intensive phases of checkpoints) do not easily fit with what these systems were developed for and raise the question of how the systems could be used optimally.

Within the CloudCraft project we address the question of how to take advantage of the capabilities of Cloud data management solutions while finding ways to address their shortcomings for this class of applications [22]. For this purpose, we implemented an environment based on the Darkstar MMORPG open source project [21] that we ported to run on Cassandra, one of the most popular Cloud DBMS. Furthermore, we developed an environment to run simulated, scripted interactions of many clients with many game logic servers as well as Cassandra nodes. Based on observations made within this environment, we illustrate approaches to efficiently use the scaling capabilities and how to achieve the consistency-level required for some data sets.

1. A Cloud-based Service Infrastructure for MMORPGs

In this section we will analyze requirements of MMORPGs and, based on this, sketch an infrastructure that serves as a frame for our research activities within the CloudCraft project. To point out data management requirements, for our following considerations we classify data into four data sets (see Table 1). These different classes vary widely regarding their requirements and have to be managed accordingly. A more detailed version of this classification can be found in our analysis in [5].

Account data: this category of data includes user account information, such as user ID, password, recharge records, and account balance. These data are mostly 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 (also called avatar) metadata, position as well as state of avatars and objects, and inventory in MMORPGs are modified constantly. Typically also within the CloudCraft project, modifications of state data are currently executed by an in-memory database in real-time [26,25] and backed up to the disk resident database periodically [24,25].

Log data: analyzing 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 [6]. 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 [7] 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 is allowed to be different among client sides. Log data are generally not analyzed immediately. Hence, eventual consistency [7] 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 [8]. Furthermore, log data will be appended continuously and retained in the database statically for a long time [9]. 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 of separating the game engine from the game content applied for conventional games.

2. Design and Implementation of the CloudCraft MMORPG Environment

In order to provide a proof of concept for Cloud-based online game, as well as to investigate the scalability and performance of it, we have designed and implemented a prototypical game platform. For this purpose we applied an open source MMORPGs test environment and ported it to Cassandra [12].

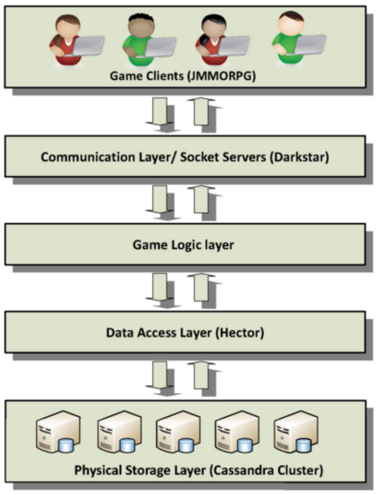

Game Clients (JMMORPG)
Communication Layer/ Socket Servers (Darkstar)
Game Logic layer
Data Access Layer (Hector)
Physical Storage Layer (Cassandra Cluster)

Figure 1. Architecture of the CloudCraft environment

2.1. *Prototype Architecture*

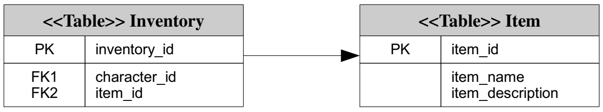
Figure 1 shows the architecture of our game prototype, which consists of a client side and a server side. The client side can be scripted to support experimental setups of thousands of players; the server side is responsible for handling requests from game clients and managing the various data sets in the game. There are four layers at the server side, namely, the communication layer, the game logic layer, the data access layer, and the physical storage layer. The game client and the game server communicate via a socket server, which we named the communication layer; the game logic layer is responsible for handling commands sent by players and dealing with game logics; the data access layer is used for communication between the logic layer and the storage layer; the physical storage layer performs data accessing operations and hosts data in the game.

In our prototype, we have adopted a Cassandra[3] cluster on the physical storage layer. Cassandra is an open source Cloud DBMS, and designed to provide high performance and scalability, and an appropriate choice for an MMORPG for reasons outlined in [5].

2.2. *Game Database Schema Design with Cassandra*

The process of building a new data-driven application running on top of an RDBMS is typically designing tables based on an application scenario and then normalizing it by applying primary/foreign key relationships. However, the design concepts applied for a wide column store like Cassandra differ from RDBMS [4]. Cassandra does not support a powerful query language and cannot perform join operations. In order to avoid join operations, denormalization is common in Cassandra [4]. That means, we add data redundancy in column families 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, schema design for Cassandra often starts with analyzing queries/use cases. Namely, it is necessary to

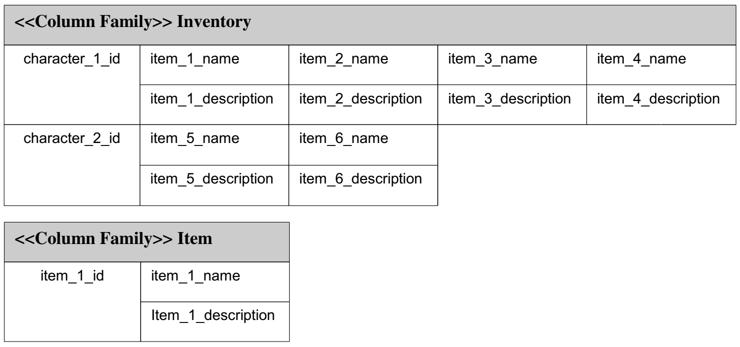
[3]Cassandra website: http://cassandra.apache.org/ (accessed 20.02.2014).



<<Table>> Inventory	
PK	inventory_id
FK1 FK2	character_id item_id

<<Table>> Item	
PK	item_id
	item_name item_description

(a) Inventory and item table design in RDBMS



<<Column Family>> Inventory				
character_1_id	item_1_name	item_2_name	item_3_name	item_4_name
	item_1_description	item_2_description	item_3_description	item_4_description
character_2_id	item_5_name	item_6_name		
	item_5_description	item_6_description		

<<Column Family>> Item	
item_1_id	item_1_name
	Item_1_description

(b) Inventory and item column family design in Cassandra

Figure 2. Comparison of schema design

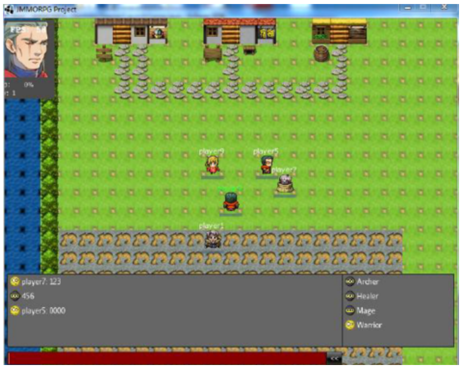
determine first what queries an application does use, to think about query patterns up front, then organize the data around the queries, and design column families accordingly.

An example to highlight the difference is shown in Figure 2. In an RDBMS, only the character ID and item ID is stored in an inventory table to reduce the data redundancy. A foreign key (item ID) is used to join inventory and item tables when querying the inventory of an avatar. However, in Cassandra all information should be stored in a single row of the inventory column family, so there is no expensive join across clusters required. Cassandra is a partitioned row store, which has no fixed schema as required by RDBMS. Each row could have different 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 the inventory column family.

2.3. Implementation of the MMORPG Environment

Our research focuses on analyzing the influence of using a Cloud storage system for MMOGs rather than designing a real and complex online game. Therefore, a simplified but robust game client and game server supporting basic game logics suffice to fulfill our experimental requirements. We have implemented a game prototype based on an open source project JMMORPG[4], which consists of a simple Java game client and a game server running on an RDBMS. We have used the architecture and the client GUI (Graphical User Interface) of it, such as avatar figures and maps (see Figure 3). Based on this, we have redesigned a new game server so that it can run on the Cassandra cluster.

[4]JMMORPG project:http://sourceforge.net/projects/jmmorpg/ (accessed 20.02.2014).



player7: 123
456
player5: 0000

Figure 3. Game GUI

The communication layer in the prototype is based on the Darkstar project[5], which provides a convenient functions library, which helps developers to deal with the challenging aspects of networked game development [21].

In the logic layer, we have simulated some basic game logics, such as responding to commands ordered by clients (e.g., players' login and avatars' movements) and supporting interactions among players (e.g., chatting and trading), all of which involve querying the database. We have applied a high-level Java API (Hector[6]) for the data access layer, which makes it possible to access Cassandra through an RPC (Remote Procedure Call) serialization mechanism. Furthermore, we have created the previously outlined database schema with according column families for accounts, avatars, NPCs, logs, maps, inventories, and items in the Cassandra cluster.

3. Supporting Game Scalability

We have to point out that physical resources for the experiments were limited as described below, so the focus is mostly on scaling the number of clients versus a small set of up to five Cassandra servers. Nevertheless, we got some interesting results.

3.1. Experimental Setup

For running this prototype, we have applied 8 virtual machines with an 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 the Cassandra cluster (version 1.2.5) and game servers. (See Figure 4)

We have implemented a simplified command-line game client for the experiments because it consumes less system resources and works like the GUI client. Our benchmark is a player's normal behavior, such as moving and trading. From data management

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

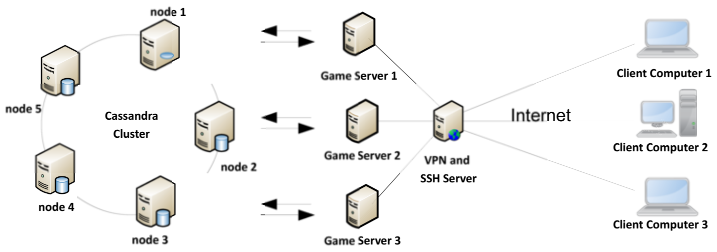
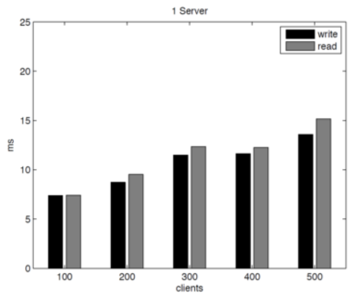
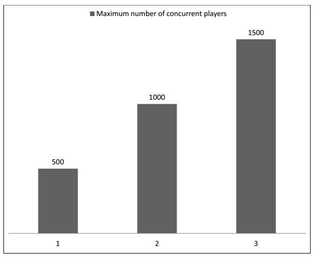


Figure 4. Infrastructure of the prototype



(a) Concurrent clients on one game server



(b) Maximum number of clients

Figure 5. Scalability of game server

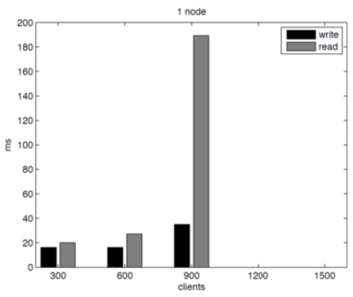
perspective, the essence of these operations is performing writes/reads to the database. We have created one row for each avatar in the avatar column family to host its state data, each of which consists of 20 columns and has 540 bytes (row size). The game client randomly orders a write/read command regarding one of those columns, and then sends it to the game server. Meanwhile, the response time of each command has been recorded.

The evaluation focuses on the potential scalability and performance of our prototype in the case of multi-player concurrent accesses. For guaranteeing data consistency as well as avoiding the effect of replication on the reading/writing performance, we set the replication factor of the Cassandra cluster as one. That means, the cluster has only one copy for each row and a read/write operation will succeed once a node responds to it.

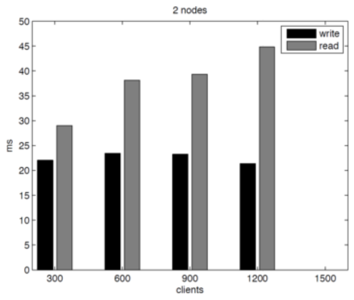
3.2. *Scalability of the Game Server*

From this experiment, we intend to get the maximum number of concurrent clients that our game server can support. Therefore, we have fixed the number of nodes in the Cassandra cluster to five, and added one to three game servers during the experiment. The number of concurrent clients connecting to the server is increased from 100 to 1500. Each client randomly sends 500 read/write commands.

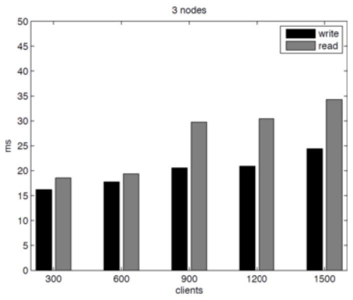
We present the experimental results with a single game server in Figure 5(a). When the client number is not more than 500, the response time for each read/write command is under 15ms. That means, 500 concurrent clients put little pressure on the game server as well as the 5-node Cassandra cluster. However, when the client number is up to 600, the game server throws many “time-out” exceptions, which block the acceptance of sub-



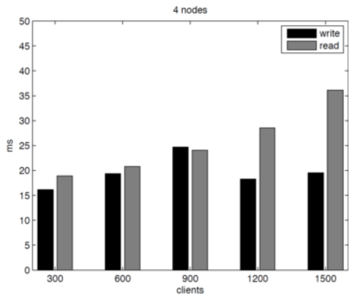
(a) One-node Cassandra



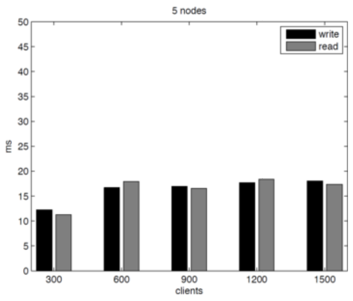
(b) Two-node Cassandra



(c) Three-node Cassandra



(d) Four-node Cassandra



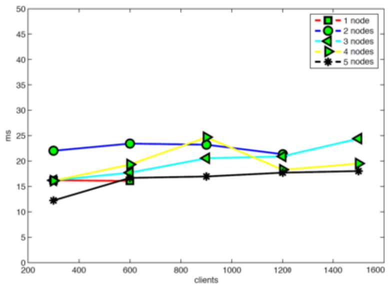
(e) Five-node Cassandra

Figure 6. Performance of Cassandra cluster

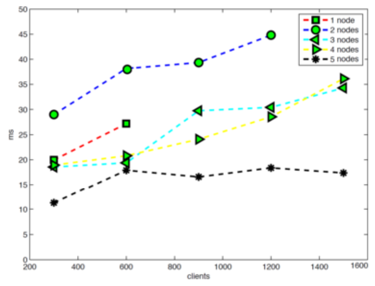
sequent commands. So the maximum number of concurrent clients in the case of single game server is around 500. Similarly, we found that the client number is directly proportional to the growth of the number of game servers (see Figure 5(b)). Therefore, we came to the conclusion that the total amount of clients is limited by the concurrent processing capability of the game server, whereas it could be raised easily by adding more servers.

3.3. Potential Scalability of Cloud Database in MMORPG

Scalability of a database is reflected by its ability that by increasing the number of database nodes to improve database performance. Hence, this time we have fixed the number of game servers to three, and set the node number in the Cassandra cluster from one to five. Each game server is connected by 100, 200, 300, 400, and 500 clients in turn. That means, the Cassandra cluster handles 300, 600, 900, 1200, 1500 clients separately.



(a) Average response time of writes



(b) Average response time of reads

Figure 7. Comparison of response time for write and read commands

Every client sends 500 read or write commands. The corresponding response time of each command is recorded and afterwards the mean response time is calculated.

From Figure 6(a) we can find that, a high performance of one-node Cassandra is achieved for less than 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, the Cassandra cluster will not respond to 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 in our experimental environment.

Figure 6(b) shows that the maximum number of clients reaches 1200 when there are two nodes in the Cassandra cluster. In the case of 1500 concurrent connections, the issue of timeout appears again. Therefore, we conclude that a two-node Cassandra cluster can support about 1200 clients by using our prototype.

Figure 6(c), 6(d) and 6(e) present that, when the number of nodes in Cassandra cluster is more than three, our prototype can support at least 1500 concurrent players.

In order to observe the different results in Figure 6, we plot the writing and reading response time in Figure 7(a) and 7(b).

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. The 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 obvious variation of reading and writing response time. Both of them fluctuate around 15ms;
4. Generally, the system performance has been improved by scaling out the 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 experiment 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 reveal the reason.
5. One-node Cassandra shows a better performance in the case of 300 or 600 clients. The reason could be that the advantage of a multi-node Cassandra cluster is not outstanding when the number of concurrent players is relatively small. In addition, the communication between nodes also consumes some time since data are distributed on different nodes.

Based on the analysis above, we can conclude that Cassandra exhibits a satisfactory scalability for typical MMORPG requirements. 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 40ms, which fulfills the requirement of an MMOG [13]; Cassandra is a write-intensive database. The experimental results show that its writing performance is stable and excellent. This feature makes it suitable to perform a backend database of a multi-player online game, which needs to handle more write requirements.

4. Supporting Game Consistency

Though, as shown in the previous section, using a Cloud storage system can efficiently solve the system scalability issue, it also introduces new challenges, such as the guarantee of a consistency level suitable for MMORPGs. Corresponding requirements and possible solutions are discussed in this section. Note that neither in this paper nor within the project we are going to discuss the data synchronization issue among game clients [11], for which established solutions on the upper levels of the game architecture exist.

4.1. Game Data Persistence Issue in the Cloud

Inconsistent data in a database may lead to system anomalies and/or even have economic consequences triggered by unsatisfied user requirements. Therefore, RDBMSs adopt various measures to ensure data consistency. In contrast, Cloud storage systems deliberately tolerate this situation. There are two main factors causing data inconsistent in the Cloud:

1. According to the CAP theorem [3], 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 transactions and relies on denormalization to ensure data consistency becomes more difficult.

Typical application scenarios of Cloud storage systems like social networking services usually have no strong requirements regarding 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.

4.2. Classification of Data Consistency in Online Game

In order to take advantages of a Cloud storage system, ensure system availability, and maintain the normal operation of a game, we need to analyze game-specific requirements of consistency. Within the CloudCraft project, we classify them as follows (see Table 1):

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 cause problems for 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 an avatar to get the up-to-date view. Hence, a read-your-writes consistency is acceptable [7].

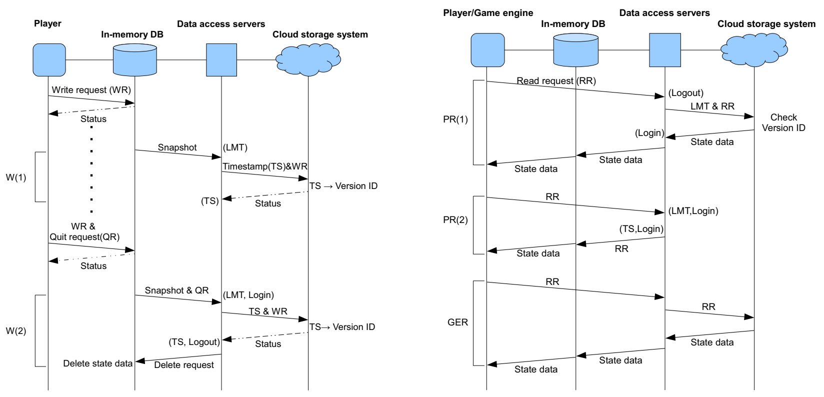
Causal consistency. Some of the game data have a local replica on the client side, or have been replicated on all game servers. Some of them (e.g. item information in Figure 2(b)) may be redundant in the avatar's state information. Only game developers have permission to modify them. In addition, changes of these data cannot be propagated synchronously to all replicas. These data could be eventually 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 [7,14].

Timed consistency. Logs usually cause big data volumes, which consume large amounts of network bandwidth and affect the system performance while propagating them. Note that log data are generally analyzed after a long time. Therefore, asynchronous propagation is feasible when the database is idle. We propose a deadline-based consistency model (timed consistency) for log data [15,16]. Here, timed consistency represents that updated values must be propagated to all replicas within a bounded time.

4.3. Extension for Cloud Storage System

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

Support of strong consistency. Cassandra is capable of ensuring strong consistency at a row level. So only considering this aspect, Cassandra is qualified for managing account data. The proviso is that the replication factor of writes is set to *ALL*. Trading among players involves a transaction across operation sequences on several rows. For this purpose, a centralized in-memory relational database is applied on the server side to take over the responsibility of Cassandra. That means, during the game, players only query data from a memory-resident database. There are three main tasks of the Cassandra



(a) Write operation (b) Read Operation

Figure 8. Executions of Write/Read Operations: in Figure 8(a), W(1) describes a general backup operation; W(2) shows the process of data persistence when a player quits the game; in Figure 8(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 [17]

cluster in the new architecture: sending avatar'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 avatar's state data after a player quits the game. (The avatar's state data will then be 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 constraints, which is much faster than the 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 specify the sum of consistency level of write and read 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 [17]. The data access server is responsible for data exchange among players, game servers, and the Cassandra cluster. It maintains a timestamp table, which records the last time (timestamp) of when an avatar's state data was backed up to Cassandra (see Figure 8(a)). This timestamp is used here as version identification, which is also stored with the state data in Cassandra in parallel. Figure 8(b) shows that when a player starts a game, the read request will be sent with a relative timestamp from the data access server to Cassandra. Through comparing the timestamp, Cassandra can accurately obtain the latest record. Our proposal can significantly reduce the amount of data propagation inside the Cassandra cluster. That is because state data of an avatar needs 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 the game engine do not need to compare the timestamp and thereby may get outdated values.

Support of causal consistency. Inconsistent data caused by data redundancy cannot be detected and fixed by Cassandra itself. The 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 avatars' state data (see Figure 2(b)), 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, the data access server is 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 *Anti-Entropy*, will then ensures that the log data are eventually consistent and ordered.

5. Conclusion

Within the CloudCraft project we investigate 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. In this article, we described our test environment and summarized key findings. 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 overview of relevant data sets and derived suitable consistency levels and how these could be achieved by extending existing Cloud data management solutions and mixing them with existing technology (in-memory transactional DBMS). 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 persistence 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] Hewitt E. *Cassandra: The Definitive Guide*. Sebastopol: OReilly Media, 2010. 332p.

[5] 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.

[6] Li F. W. B., Li L. 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.

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

[8] 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.

[9] 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.

[10] Zhang K., Kemme B., Denault A. Persistence in Massively Multiplayer Online Games. In: Claypool M. (ed.) *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games [NETGAMES 2008], 21-22 October, Worcester, Massachusetts, USA* NY:ACM, 2008. 53-58.

[11] Zhang K., Kemme B. Transaction Models for Massively Multiplayer Online Games. In: Jimnez-Peris R.,Madrid P. D. (eds.) *Proceedings of the 2011 IEEE 30th International Symposium on Reliable Distributed Systems [SRDS 2011], 4-7 October, Madrid, Spain*. Piscataway:IEEE, 2011. 31-40.

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

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

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

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

[16] 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.

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

[18] Kienzle J., Verbrugge C., Kemme B., Denault A., Hawker M. Mammoth A Massively Multiplayer Game Research Framework. In: Whitehead J., Young R. M. (eds.) *Proceedings of the 4th International Conference on Foundations of Digital Games [FDG 2009], 26-30 April, Orlando, FL, USA*. NY: ACM. 2009. 308-315.

[19] Fischer T., Daum M., Irmert F., Neumann C., Lenz, R. Exploitation of event-semantics for distributed publish/subscribe systems in massively multiuser virtual environments. In: Desai B. C., Bernardino J. (eds.) *Proceedings of the Fourteenth International Database Engineering and Applications Symposium on [IDEAS 2010], 16-18 August, New York, New York, USA*. NY: ACM Press, 2010. 90–97.

[20] Iosup A., Lascateu A., Japui N. CAMEO: Enabling social networks for Massively Multiplayer Online Games through Continuous Analytics and cloud computing. In Cheok A., Huang J., Ishibashi Y. (eds.)*Proceedings of the 9th Annual Workshop on Network and Systems Support for Games [NetGames 2010], 16-17 November, Taipei, Taiwan*. Piscataway: IEEE, 2010. 1–6.

[21] Burns B. *Darkstar: The Java Game Server*. Sebastopol: O'Reilly Media, 2007. 77p.

[22] Abadi D. J. Data Management in the Cloud Limitations and Opportunities. *IEEE Data Engineering Bulletin*, 2009, 32(1), 3-12.

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

[24] Cao T., Vaz Salles M., Sowell B., Yue Y., Demers A., Gehrke J., White W. Fast checkpoint recovery algorithms for frequently consistent applications. In: Sellis T. K., Miller R. J., Kementsietsidis A., Velegrakis Y. (eds.) *Proceedings of the 2011 international conference on Management of data [SIGMOD 2011], 12-16 June, New York, New York, USA*. NY: ACM, 2011. 265-276.

[25] Vaz Salles M., Cao T., Sowell B., Demers A., Gehrke J., Koch C., White W. An evaluation of checkpoint recovery for massively multiplayer online games. *Proceedings of the VLDB Endowment*, 2009, 2(1), 1258-1269.

[26] Cao T., Sowell B., Salles M. V., Demers A., Gehrke J. BRRL : A Recovery Library for Main-Memory Applications in the Cloud Categories and Subject Descriptors. In: Sellis T. K., Miller R. J., Kementsietsidis A., Velegrakis Y. (eds.) *Proceedings of the 2011 international conference on Management of data [SIGMOD 2011], 12-16 June, New York, New York, USA*. NY: ACM, 2011. 1233–1235.