Cloud Data Management for Online Games: Potentials and Open Issues

Ziqiang Diao, Eike Schallehn

Department of Technical and Business Information Systems
Otto-von-Guericke University Magdeburg, Germany
diao, eike@iti.cs.uni-magdeburg.de

Abstract: Popular massively multiplayer online role-playing games (MMORPG) typically have millions of players and are played throughout the world. Worldwide revenues for MMORPGs have increased to billions of dollars each year. Unfortunately, the complex architecture of MMORPGs make them hard to maintain, resulting in considerable costs and development risks. For normal operation, MMORPGs have to access huge amounts of diverse data. With increasing numbers of players, managing growing volumes of data in a relational database becomes a big challenge, which cannot be overcome by simply adding new servers. Cloud storage systems are emerging solutions, focusing on providing scalability and high performance for Cloud applications, social media, etc. However, Cloud storage systems are in general not designed for processing transactions or providing high levels of consistency. In this paper, we analyze the existing architecture of MMORPGs, classify data, highlight the design requirements, identify the major research challenges, propose a Cloud-based model for MMORPGs, and present a series of data management solutions.

1 Introduction

A massively multiplayer online game (MMOG) can generally support hundreds or thousands of players from all over the world in parallel. In a virtual game world, players can choose a new identity, establish a new social network, compete or cooperate with other players, or even realize dreams that cannot be completed in real-life. Because of these attractiveness, in 2011 US Americans overall spend 26 million hours per day and 2.6 billion dollars in total for playing MMOGs[New13a]. The game industry is developing rapidly, and the global MMOG market volume has grown to $13 billion in 2012[New13b].

Different from some types of MMOGs such as first-person shooters and real-time strategy games, MMORPGs keep the virtual game environment running even in the case of no players. In addition to the account information, the state data of objects and characters must be recorded on the server side in real-time, so that players can quit and continue the game at any time. All of the player behavior in the game should be monitored and backed up in order to maintain the order of the virtual world. Furthermore, MMORPGs usually have more concurrent players than any other MMOGs, (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 [ZKD08]. However, existing RDBMS cannot

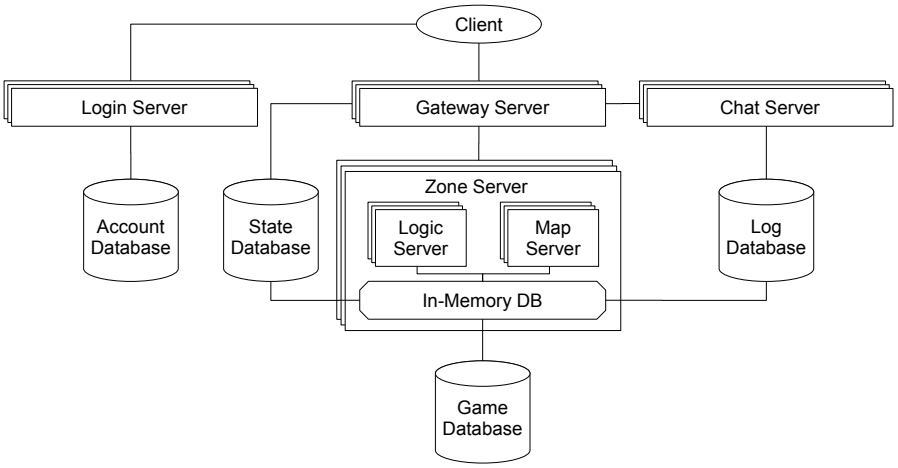

Figure 1: A simplified architecture of MMORPGs

fully satisfy all these requirements simultaneously [WKG+07, Cat10]. Therefore, with the increasing data volume, the storage system becomes a bottleneck, and solving scalability and availability issues become a major cost factor and development risk.

Cloud storage systems, which have the ability to support highly concurrent data accesses and huge storage, may become a solution. Unfortunately, in contrast with the conventional DBMS, cloud systems are generally designed for Web applications that have different access characteristics and require lower or different consistency levels. Therefore, we need to analyze MMORPGs in more detail, to assess the usability of Cloud storage systems for MMORPGs, open issues, and possible solutions.

In this paper, we focus on MMORPGs, and propose to customize a Cloud-based data management for them. The rest of the paper is structured as follows. In Section 2, we analyze the existing architecture and data management of MMORPGs. In Section 3, we classify data of MMORPGs into four groups and discuss their requirements. In Section 4, we propose a Cloud-based data management model for MMORPG, and, finally, conclude and summarize this paper in Section 5.

2 A short Analysis of MMORPG's current Data Management

In order to support player interaction and ensure system security [ZKD08] the MMORPG typically employs a Client-Server model or a Browser-Server model. However, MMORPGs usually have a similar architecture on the server side. Figure 1 shows a simplified architecture of MMORPGs [WKG+07, Ale03].

A player sends a request to the login server, which is responsible for determining the validity of the request. The login server cooperates with an account database that stores user account information. If the validation is passed, the login server encrypts the user ID,

generates a token, and then returns it to the player. The player uses this token to communicate with the gateway server, which is an information transit station. The gateway server is generally composed by multiple servers, the main functions of which are to maintain the connection state of a player, monitor the player's requests and transmit it to a zone server or a chat server. MMORPGs need to satisfy the connection requests from global players, so these servers are usually deployed in parallel in several data centers around the world. Note that only one zone server provides services in one session. The zone server is responsible for maintaining a global virtual game world for players, executing the game rules, 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, and accesses state data of characters from a state database. When a player starts an MMORPG, all character state data of this player must be read from the state database at one time, and then cached in an in-memory database. During the game, the zone server manages data in memory in real-time, and writes them to the disk periodically. When a player quits the game, the character state data of this player is persisted to the state database and deleted from the active game data in memory. The log database in this architecture records all player operations and sometimes also the chat history. The data is collected for purposes of data mining and intrusion detection.

Currently, MMOPRGs apply distributed RDBS for data persistence, which can commit complex transactions and are proved to be stable. As an example, consider MySQL Cluster[Ora13] 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 is able to help clients to access correct shards to satisfy a query or commit a transaction. For the purpose of guaranteeing 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.

3 Data Management Requirements of MMORPGs

From a system's point of view, the essence of a game is data processing, storage, and transmission among databases, servers, and players. According to data management requirements, for our following considerations we classify data 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

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

Table 1: Data classification and requirements analysis

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 [LLL04]. 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 [Vog08] 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 [Vog08] is sufficient for these two classes of data.

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

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 costs of a system initially laid out for too many users, the data management needs to be extremely scalable [GDG08]. Furthermore, log data will be appended continually and retained in the database statically for a long time [WKG+07]. 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 [IHK04].

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 a game database 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 DMS 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.

4 The case for Cloud-based Data Management for MMORPGs

Currently, MMORPGs apply RDBMS for data persistence. However, RDBMS cannot fulfill some data management requirements well, or there must be some provisos[Ale03, Cat10]. 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 [FMC+11, Cat10]. With their simplified data model, Cloud storage systems are proven to have a great potential for scalability [FMC+11].

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.

4.1 Proposal of an Architecture for Cloud-based MMORPGs

For these reasons, we propose a set of Cloud services to improve the existing game architecture. The main idea is that subsets of data having strong requirements to system scalability or flexible data model are persisted in a Cloud-based distributed file system or in a Cloud-based structured data system; other data sets are provided Cloud services that are similar with existing solutions. Figure 2 shows the new Cloud-based architecture for MMORPGs.

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. Furthermore, the scale of account data is not large. Accordingly, we suggest storing account data in RDBMS as a service. We do not suggest managing the account data in a public Cloud for the data security reasons. Before the public Cloud providers bring trustworthy data security solutions, we recommend game providers to build a private Cloud, which employs Cloud architecture and is maintained and protected by game providers themselves.

Log data has a large scale. Once the log data has been written to disk, they do 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,

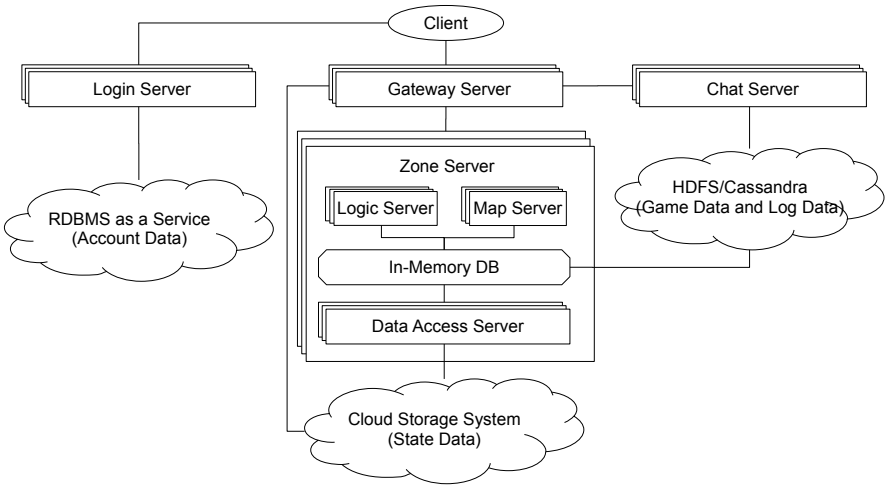

Figure 2: A Cloud based architecture of MMORPGs

management requirements of log data are not difficult to meet. Therefore, there are some existing Cloud storage systems that can be used directly, such as Cassandra[Apa13] or the Hadoop Distributed File System (HDFS) [SKRC10]. Both of them can provide high throughput to access data across a large number of servers and have the feature of high fault tolerance. Cooperating with Hadoop MapReduce, both of them can process and analyze large data set in parallel.

Game data processing does not pose a challenge to the existing solutions. Unless they are stored on the client side, we can manage them in a traditional distributed file system or in HDFS because these data exist typically in the form of documents.

As discussed above, managing the state data in real-time is a big challenge for a disk resident RDBMS. Unfortunately, Cloud storage systems with their shared nothing architecture are not intended to provide real-time support. Although some Cloud storage systems manage data in the memory of each node, 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 continue applying an in-memory database in each zone server.

Note that state data must be persisted to the disk. In this process, a flexible data model, simplified data processing, and a high performance for concurrent data access are required, all of which are weaknesses of RDBMS. State data is backed up to disk in frequent intervals, typically every 5 to 10 minutes. Consequently, the database in MMORPGs is write-intensive. Cassandra is designed for Web applications that perform more equal shares of write and read operations, such as a social networking website. Cassandra ensures that it is always writable. Nevertheless, Cassandra can only partially meet the demand of the

state data. It is a plan for our future research, to customize a new Cloud storage system for MMORPGs based on Cassandra. In the following, we discuss some features of Cassandra.

4.2 Using Cassandra for MMORPGs

Cassandra has a decentralized (peer-to-peer) structure [Lak10], 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 different with RDBMS (e.g., MySQL Cluster) and some other Cloud storage systems (e.g., Google Bigtable [CDG+06]), 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. That is also the reason that we can manage state and log data in Cassandra. 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 problems that cannot be well solved by Cassandra directly.

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[GeBS11]. Note that Cassandra records timestamps in each column, and uses it as a version identification. Therefore, a possible solution is that we record the timestamps from a global server (in this paper it is the access server) in both server side and columns in Cloud storage system. When we read state data from the Cloud, the timestamps 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 access servers in parallel. For these two reasons, accessing these timestamps in the server side is not a challenge.

Customized functionality: We need to develop some new functions based on features of

MMORPGs. There are two examples: the task of the Cloud storage system is data backup. Each value that has been written in the Cloud 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 workload of write. A more efficient data partitioning method based on semantics (e.g. map zones) or a mix of both has to be proposed.

Network traffic: In shared-nothing systems, nodes communicate with each other via messaging. As a result, a large amount of messages and data are transmitted over the network, which may lead to potential bandwidth bottlenecks. This risk is particularly high for MMORPGs that need to backup the state data snapshot to the disk regularly. Here are two possible proposals: If no or only some unimportant attributes (e.g., the position of a character) of a state data are modified, the current round backup of this state data can be skipped [ZK11, ZKD08]; only the timestamp, the row key, and the modified values are sent as a message to the Cloud storage system.

5 Conclusion

In this paper, we proposed a Cloud-based data management for MMORPGs, which has a high performance and scalability. We outlined the main criteria for this proposal, by classifying and describing the typical requirements of MMORPGs. The proposed model adopts a shared-nothing architecture to ensure scalability, and customizes a set of data management solutions, so that the new system can operate efficiently. We classified data into four groups, and found out that only the state data and the log data should be recorded in the Cloud storage system and managed differently. For example, the model promises a strong consistency guarantee to the state data, but only an eventual consistency guarantee to the log data. To further improve the system's performance, we also proposed some solutions to reduce the number and size of messages. In our future work, we will focus on the cooperation of multiple DMSs in one MMORPG and the customization of a new Cloud storage system for MMORPGs.

References

[Ale03] Thor Alexander. *Massively Multiplayer Game Development*. Thomson Learning, 2003.

[Apa13] Apache. Cassandra, January 2013. `http://cassandra.apache.org/`.

[Cat10] Rick Cattell. Scalable SQL and NoSQL Data Stores. *ACM Special Interest Group on Management of Data (SIGMOD)*, 39(4):12–27, 2010.

[CDG+06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of 7th Symposium on Operating System Design and Implementation(OSDI)*, pages 205–218, 2006.

[FMC+11] Craig Franke, Samuel Morin, Artem Chebotko, John Abraham, and Pearl Brazier. Distributed Semantic Web Data Management in HBase and MySQL Cluster. In *IEEE International Conference on Cloud Computing, CLOUD 2011*, pages 105–112, 2011.

[GDG08] Nitin Gupta, Alan Demers, and Johannes Gehrke. SEMMO : A Scalable Engine for Massively Multiplayer Online Games [Demonstration Paper]. In *ACM SIGMOD Conference 2008*, pages 1234–1238, 2008.

[GeBS11] Francis Gropengieß er, Stephan Baumann, and Kai-Uwe Sattler. Cloudy Transactions Cooperative XML Authoring on Amazon S3. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, pages 307–326, 2011.

[IHK04] Takuji Iimura, Hiroaki Hazeyama, and Youki Kadobayashi. 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, 2004.

[Lak10] Avinash Lakshman. Cassandra - A Decentralized Structured Storage System. *Operating Systems Review*, 44(2):35–40, 2010.

[LLL04] Frederick W.B. Li, Lewis W.F. Li, and Rynson W.H. Lau. Supporting continuous consistency in multiplayer online games. In *12. ACM Multimedia 2004*, pages 388–391, 2004.

[New13a] Newzoo. 2011 MMO Trend Report, January 2013. `http://www.newzoo.com/trend-reports/mmo-trend-report/`.

[New13b] Newzoo. 2012 MMO Games Market Report, January 2013. `http://www.newzoo.com/infographics/the-global-mmo-market-sizing-and-seizing-opportunities/`.

[Ora13] Oracle. MySQL Cluster Overview, January 2013. `http://dev.mysql.com/doc/refman/5.5/en/mysql-cluster-overview.html`.

[SKRC10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[Vog08] Werner Vogels. Eventually consistent. *ACM Queue*, 6(6):14–19, 2008.

[WKG+07] Walker White, Christoph Koch, Nitin Gupta, Johannes Gehrke, and Alan Demers. Database research opportunities in Computer Games. *ACM Special Interest Group on Management of Data (SIGMOD)*, 36(3):7–13, 2007.

[ZK11] Kaiwen Zhang and Bettina Kemme. Transaction Models for Massively Multiplayer Online Games. In *30th IEEE Symposium on Reliable Distributed Systems (SRDS 2011)*, pages 31–40, 2011.

[ZKD08] Kaiwen Zhang, Bettina Kemme, and Alexandre Denault. 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, 2008.