Cloud Data Management: A Short Overview and Comparison of Current Approaches

Siba Mohammad
Otto-von-Guericke University Magdeburg
siba.mohammad@iti.uni-magdeburg.de

Sebastian Breß
Otto-von-Guericke University Magdeburg
sebastian.bress@st.ovgu.de

Eike Schallehn
Otto-von-Guericke University Magdeburg
eike@iti.cs.uni-magdeburg.de

ABSTRACT

To meet the storage needs of current cloud applications, new data management systems were developed. Design decisions were made by analyzing the applications workloads and technical environment. It was realized that traditional Relational Database Management Systems (RDBMSs) with their centralized architecture, strong consistency, and relational model do not fit the elasticity and scalability requirements of the cloud. Different architectures with a variety of data partitioning schemes and replica placement strategies were developed. As for the data model, the key-value pairs with its variations were adopted for cloud storage. The contribution of this paper is to provide a comprehensible overview of key-characteristics of current solutions and outline the problems they do and do not address. This paper should serve as an entry point for orientation of future research regarding new applications in cloud computing and advanced requirements for data management.

1. INTRODUCTION

Data management used within the cloud or offered as a service from the cloud is an important current research field. However, there are only few publications that provide a survey and compare the different approaches. The contribution of this paper is to provide a starting point for researchers and developers who want to work on cloud data management. Cloud computing is a new technology that provides resources as an elastic pool of services in a pay-as-you-go model [5]. Whether it is storage space, computational power, or software, customers can get it over the internet from one of the cloud service providers. Big players in the market, such as Google [8], Amazon [13], Yahoo! [9], and Hadoop [7], defined the assumptions for cloud storage systems based on analyzing the technical environment and applications workload. First, a data management system will work on a cluster of storage nodes where components failure is the normal situation rather than the exception. Thus, fault tolerance and recovery must be built in. The system must be portable across heterogeneous hardware and software platforms. It will store tera bytes of data, thus parameters of I/O operations and block sizes must be adapted to large sizes. Based on these assumptions the requirements for a cloud DBMS are: elasticity, scalability, fault tolerance and self manageability [11]. The rest of the paper is organized as follows. First, we provide an overview of the architecture and the family tree of the cloud storage systems. Then, we discuss how different systems deal with the trade-off in the Consistency, Availability, Partition tolerance (CAP) theorem. After that, we discuss different schemes used for data partitioning and replication. Then, we provide a list of the cloud data models.

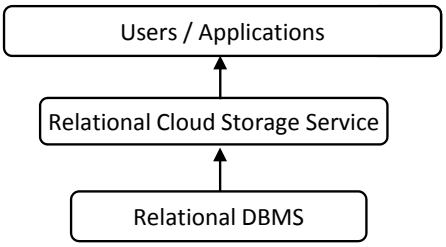

Figure 1: RDBMS as a service

2. ARCHITECTURE OVERVIEW

There are two main approaches to provide data management systems for the cloud. In the first approach, each customer gets an own instance of a Database Management System (DBMS), which runs on virtual machines of the service provider [10] as illustrated in Figure 1. The DBMS supports full ACID requirements with the disadvantage of loosing scalability. If an application requires more computing or storage resources than the maximum allocated for an instance, the customer must implement partitioning on the application level using a different database instance for each partition [1]. Another solution is on demand assignment of resources to instances. Amazon RDS is an example of relational database services, that supports MySQL, Oracle, and SQL Server.

In the second approach, data management is not provided as a conventional DBMS on a virtualized platform, but as a combination of interconnected systems and services that can be combined according to application needs. Figure 2 illustrates this architecture. The essential part is the dis-

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

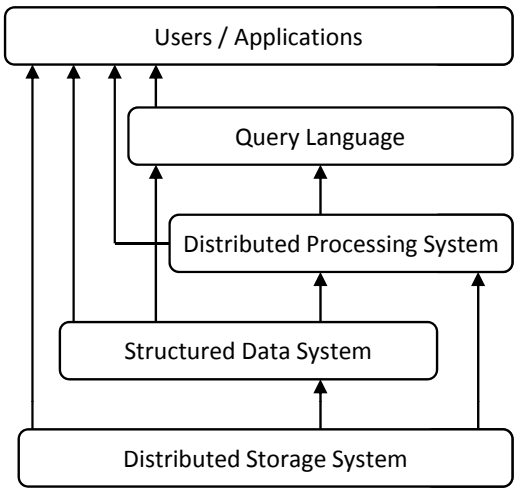

Figure 2: Cloud data management architecture

tributed storage system. It is usually internally used by the cloud services provider and not provided as a public service. It is responsible for providing availability, scalability, fault tolerance, and performance for data access. Systems in this layer are divided in three categories:

- Distributed File Systems (DFS), such as Google's File System (GFS).
- Cloud based file services, such as Amazon's Simple Storage Service (S3).
- Peer to peer file systems, such as Amazon's Dynamo.

The second layer consists of structured data systems and provides simple data models such as key-value pairs, which we discuss in Section 6. These systems support various APIs for data access, such as SOAP and HTTP. Examples of systems in this layer are Google's Bigtable, Cassandra, and SimpleDB. The third layer includes distributed processing systems, which are responsible for more complex data processing, e.g, analytical processing, mass data transformation, or DBMS-style operations like joins and aggregations. MapReduce [12] is the main processing paradigm used in this layer.

The final layer includes query languages. SQL is not supported. However, developers try to mimic SQL syntax for simplicity. Most query languages of cloud data management systems support access to one domain, key space, or table, i.e., do not support joins [4, 2]. Other functionalities, such as controlling privileges and user groups, schema creation, and meta data access are supported. Examples of query languages for cloud data are HiveQL, JAQL, and CQL. The previous components complement each other and work together to provide different sets of functionalities. One important design decision that was made for most cloud data query languages is not supporting joins or aggregations. Instead, the MapReduce framework is used to perform these operations to take advantage of parallel processing on different nodes within a cluster. Google pioneered this by providing a MapReduce framework that inspired other systems.

For more insight into connections and dependencies between these systems and components, we provide a family

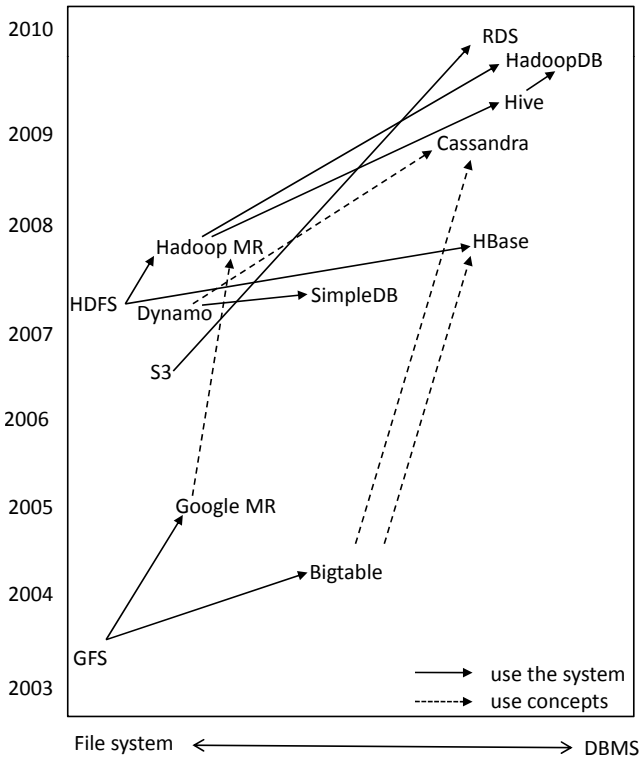

Figure 3: Family tree of cloud data management systems

tree of cloud storage systems as illustrated in Figure 3. We use a solid arrow to illustrate that a system uses another one such as Hive using HDFS. We use a dotted arrow to illustrate that a system uses some aspects of another system like the data model or the processing paradigm. An example of this is Cassandra using the data model of Bigtable. In this family tree, we cover a range of commercial systems, open source projects, and academic research as well. We start on the left side with distributed storage systems GFS and HDFS. Then, we have the structured storage systems with API support such as Bigtable. Furthermore, there are systems that support a simple QL such as SimpleDB. Next, we have structured storage systems with support of MapReduce and simple QL such as Cassandra and HBase. Finally, we have systems with sophisticated QL and MapReduce support such as Hive and HadoopDB. We compare and classify these systems based on other criteria in the coming sections. An overview is presented in Figure 4.

3. CONSISTENCY, AVAILABILITY, PARTITION TOLERANCE (CAP) THEOREM

Tightly related to key features of cloud data management systems are discussions on the CAP theorem [14]. It states that consistency, availability, and partition tolerance are systematic requirements for designing and deploying applications for distributed environments. In the cloud data management context these requirements are:

- Consistency: includes all modifications on data that must be visible to all clients once they are committed.

System / Property	Distributed Storage/File System				Structured Data Systems						Analytical Processing Systems		RDBMS as service
	Amazon S3	GFS	HDFS	Dynamo	SimpleDB	Bigtable	HBase	PNUTS	Cassandra	CouchDB	Hive	HadoopDB	RDS
Consistency Model													
ACID												X	X
BASE	X			X	X					X			
SCLA						X	X				X		
Tunable Consistency								X	X				
Partitioning Scheme													
Hash											X	X	
Range						X	X	X					
List								X			X		
Composite				X					X				
Partition Level		file	file	Key space		table	table	table	table			table	
Partition		chunk	chunk	set of items		tablet	region	tablet	set of items		bucket	chunk	
Data Models													
Key Value	X			X	X	X	X		X	X	X		
Row Oriented					X								
Wide Column						X	X		X		X		
Document Oriented										X			
Relational								X				X	X
Replication													
Rack aware		X	X				X		X		X	X	
Rack unaware	X					X			X				
Datacenter aware				X	X			X	X				
Replication Level	item	chunk	block	item	domain	DB file	DB file	tablet	record	DB	DB file	chunk	DB
Map/Reduce													
Internally											X	X	
Input for	X	X	X			X	X	X	X	X			
Interface													
Query Lang					X		X		X		X	X	X
API	X			X	X	X	X	X	X	X	X	X	X

Figure 4: Overview and classification of cloud data management systems(Gray field begins a new category of properties. Dark gray field means that the property is not applicable on a system)

At any given point in time, all clients can read the same data.

- Availability: means that all operations on data, whether read or write, must end with a response within a specified time.
- Partition tolerance: means that even in the case of components' failures, operations on the database must continue.

The CAP theorem also states that developers must make trade-off decisions between the three conflicting requirements to achieve high scalability. For example, if we want a data storage system that is both strongly consistent and partition tolerant, the system has to make sure that write operations return a success message only if data has been committed to all nodes, which is not always possible because of network or node failures. This means that its availability will be sacrificed.

In the cloud, there are basically four approaches for DBMSs in dealing with CAP:

Atomicity, Consistency, Isolation, Durability (ACID): With ACID, users have the same consistent view of data before and after transactions. A transaction is atomic, i.e., when one part fails, the whole transaction fails and the state of data is left unchanged. Once a transaction is committed, it is protected against crashes and errors. Data is locked while being modified by a transaction. When another transaction tries to access locked data, it has to wait until data is unlocked. Systems that support ACID are used by applications that require strong consistency and can tolerate its affects on the scalability of the application as already discussed in Section 2.

Basically Available, Soft-state, Eventual consistency (BASE): The system does not guarantee that all users see the same version of a data item, but guarantees that all of them get a response from the systems even if it means getting a stale version. Soft-state refers refers to the fact, that the current status of a managed object can be ambiguous, e.g. because there are several temporarily inconsistent replicas of it stored. Eventually consistent means that updates will propagate through all replicas of a data item in a distributed system, but this takes time. Eventually, all replicas are updated. BASE is used by applications that can tolerate weaker consistency to have higher availability. Examples of systems supporting BASE are SimpleDB and CouchDB.

Strongly Consistent, Loosely Available (SCLA): This approach provides stronger consistency than BASE. The scalability of systems supporting SCLA in the cloud is higher compared to those supporting ACID. It is used by systems that choose higher consistency and sacrifice availability to a small extent. Examples of systems supporting SCLA are HBase and Bigtable.

Tunable consistency: In this approach, consistency is configurable. For each read and write request, the user decides the level of consistency in balance with the level of availability. This means that the system can work in high consistency or high availability and other degrees in between. An example of a system supporting tunable consistency is Cassandra [15] where the user determines the number of replicas that the system should update/read. Another example is PNUTS [9], which provides per record time-line consistency. The user determines the version number to query at several points in the consistency time-line. In Figure 5, we classify different cloud data management systems based on the consistency model they provide.

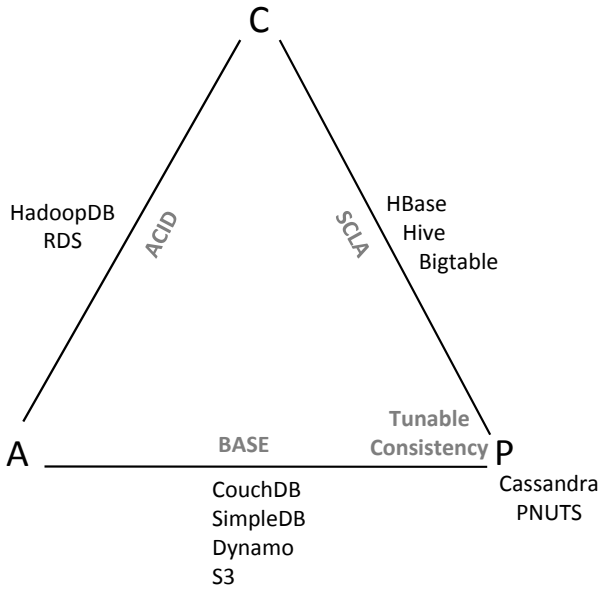

Figure 5: Classification of cloud data management systems based on CAP

4. PARTITIONING TECHNIQUES

Partitioning, also known as sharding, is used by cloud data management systems to achieve scalability. There is a variety of partitioning schemes used by different systems on different levels. Some systems partition data on the file level while others horizontally partition the key space or table. Examples of systems partitioning data on the file level are the DFSs such as GFS and HDFS which partition each file into fixed sized chunks of data. The second class of systems which partition tables or key space uses one of the following partitioning schemes [18, 6] :

List Partitioning A partition is assigned a list of discrete values. If the key of the inserted tuple has one of these values, the specified partition is selected . An example of a system using list as the partitioning scheme is Hive [20].

Range Partitioning The range of values belonging to one key is divided into intervals. Each partition is assigned one interval. A partition is selected if the key value of the inserted tuple is inside a certain range. An example of a system using range partitioning is HBase.

Hash Partitioning The output of a hash function is assigned to different partitions. The hash function is

applied on key values to determine the partition. This scheme is used when data does not lend itself to list and range partitioning. An example of a system using hash as the partitioning scheme is PNUTS.

There are some systems that use a composite partitioning scheme. An example is Dynamo, which uses a composite of hash and list schemes (consistent hashing). Some systems allow partitioning data several times using different partitioning schemes each time. An example is Hive, where each table is partitioned based on column values. Then, each partition can be hash partitioned into buckets, which are stored in HDFS.

One important design consideration to make is whether to choose an order-preserving partitioning technique or not. Order preserving partitioning has an advantage of better performance when it comes to range queries. Examples of systems using order preserving partitioning techniques are Bigtable and Cassandra. Since most partitioning methods depend on random position assignment of storage nodes, the need for load balancing to avoid non uniform distribution of data and workloads is raised. Dynamo [13] focuses on achieving a uniform distribution of keys among nodes assuming that the distribution of data access is not very skewed, whereas Cassandra [17] provides a load balancer that analyzes load information to lighten the burden on heavily loaded nodes.

5. REPLICATION TECHNIQUES

Replication is used by data management systems in the cloud to achieve high availability. Replication means storing replicas of data on more than one storage node and probably more than one data center. The replica placement strategy affects the efficiency of the system [15]. In the following we describe the replication strategies used by cloud systems:

Rack Aware Strategy: Also known as the Old Network Topology Strategy. It places replicas in more than one data center on different racks within each data center.

Data Center Aware Strategy: Also known as the New Network Topology Strategy. In this strategy, clients specify in their applications how replicas are placed across different data centers.

Rack Unaware Strategy: Also known as the Simple Strategy. It places replicas within one data center using a method that does not configure replica placement on certain racks.

Replication improves system robustness against node failures. When a node fails, the system can transparently read data from other replicas. Another gain of replication is increasing read performance using a load balancer that directs requests to a data center close to the user. Replication has a disadvantage when it comes to updating data. The system has to update all replicas. This leads to very important design considerations that impact availability and consistency of data. The first one is to decide whether to make replicas available during updates or wait until data is consistent across all of them. Most systems in the cloud choose availability over consistency. The second design consideration is to decide when to perform replica conflicts resolution, i.e., during writes or reads. If conflict resolution is done during write operations, writes could be rejected if the system can not reach all replicas or a specified number of them within a specific time. Example of that is the WRITE ALL operation in Cassandra, where the write fails if the system could not reach all replicas of data. However, some systems in the cloud choose to be always writeable and push conflict resolution to read operations. An example of that is Dynamo which is used by many Amazon services like the shopping cart service where customer updates should not be rejected.

6. DATA MODEL

Just as different requirements compared to conventional DBMS-based applications led to the previously described different architectures and implementation details, they also led to different data models typically being used in cloud data management. The main data models used by cloud systems are:

Key-value pairs It is the most common data model for cloud storage. It has three subcategories:

- Row oriented: Data is organized as containers of rows that represent objects with different attributes. Access control lists are applied on the object (row) or container (set of rows) level [19]. An example is SimpleDB.
- Document Oriented: Data is organized as a collection of self described JSON documents. Document is the primarily unit of data which is identified by a unique ID. Documents are the unit for access control [16]. Example of a cloud data management system with document oriented data model is CouchDB.
- Wide column: In this model, attributes are grouped together to form a column family. Column family information can be used for query optimization. Some systems perform access control and both disk and memory accounting at the column family level [8, 17, 3]. An example of that is Bigtable. Systems of wide column data model should not be mistaken with column oriented DB systems. The former deals with data as column families on the conceptual level only. The latter is more on the physical level and stores data by column rather than by row.

Relational Model (RM) The most common data model for traditional DBMS is less often used in the cloud. Nevertheless, Amazon's RDS supports this data model and PNUTS a simplified version of it.

7. SUMMARY AND CONCLUSION

The cloud with its elasticity and pay-as-you-go model is an attractive choice for outsourcing data management applications. Cloud service providers, such as Amazon and Microsoft, provide relational DBMSs instances on virtual machines. However, the cloud technical environment, workloads, and elasticity requirements lead to the development of new breed of storage systems. These systems range from highly scalable and available distributed storage systems with simple interfaces for data access to fully equipped DBMSs that support sophisticated interfaces, data models, and query languages.

Cloud data management systems faced with the CAP theorem trade-off provide different levels of consistency ranging from eventual consistency to strict consistency. Some systems allow users to determine the level of consistency for each data input/output request by determining the number of replicas to work with or the version number of the data item. List, range, hash, and composite partitioning schemes are used to partition data to achieve scalability. With partitioning comes the need for load balancing with two basic methods: uniform distribution of data and workloads, and analyzing load information. With data partitioned and distributed over many nodes, taking into consideration the possibility of node and network failures, comes the need for replication to achieve availability. Replication is done on the partition level using rack aware, data center aware, and rack unaware placement strategies. Cloud data management systems support relational data model, and key-value pairs data model. The key-value pairs is widely used with different variations: document oriented, wide column, and row oriented.

As outlined throughout this paper, several typical properties of traditional DBMS, such as advanced query languages, transaction processing, and complex data models, are mostly not supported by cloud data management systems. Some of them, because they are simply not required for current cloud applications. Others, because their implementation would lead to losing some of the important advantages, e.g., scalability and availability, of current cloud data management approaches. On the one hand, providing features of conventional DBMS appears to lead toward worthwhile research directions and is currently addressed in ongoing research. On the other hand, advanced requirements, which are completely different may arise from future cloud applications, e.g., interactive entertainment, on-line role playing games, and virtual realities, have interesting characteristics of continuous, collaborative, and interactive access patterns, sometimes under real-time constraints. In a similar way, new ways of human-computer interaction in real-world environments addressed, for instance, in ubiquitous computing and augmented reality are often very data-intensive and sometimes require expensive processing, which could be supported by cloud paradigms. Nevertheless, neither traditional DBMS nor cloud data management can currently sufficiently support those applications.

8. ACKNOWLEDGMENTS

We would like to thank the Syrian ministry of higher education for partially funding this research.

9. REFERENCES

[1] Amazon RDS FAQ. http://aws.amazon.com/rds/faqs/. [online; accessed 10-March-2012].

[2] Cassandra Query Language Documentation. http://caqel.deadcafe.org/cql-doc. [online; accessed 25-July-2011].

[3] HBase: Bigtable-like structured storage for Hadoop HDFS. http://wiki.apache.org/hadoop/hbase. [online; accessed 01-March-2012].

[4] Jaql Overview. http://www.almaden.ibm.com/cs/projects/jaql/. [online; accessed 31-August-2011].

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, EECS Department, University of California, Berkeley, 2009.

[6] S. S. Avi Silberschatz, Henry F. Korth. *Database System Concepts Fifth Edition*. McGraw-Hill, 2010.

[7] D. Borthakur. *The Hadoop Distributed File System: Architecture and Design*. The Apache Software Foundation, 2007.

[8] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *In Proceedingss of the 7th Conference on Usenix Symposium on Operating Systems Design and Implementation - Volume 7*, pages 205–218, 2006.

[9] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc VLDB Endow.*, 2008.

[10] C. Curino, E. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational Cloud: A Database Service for the Cloud. In *5th Biennial Conference on Innovative Data Systems Research*, 2011.

[11] S. Das, S. Agarwal, D. Agrawal, and A. E. Abbadi. ElasTraS: An Elastic, Scalable, and Self Managing Transactional Database for the Cloud. Technical report, 2010.

[12] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 2008.

[13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 2007.

[14] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. *SIGOPS Oper. Syst. Rev.*, 1997.

[15] E. Hewitt. *Cassandra The Definitive Guide*. O Reilly Media, Inc, 2010.

[16] J. L. J. Chris Anderson and N. Slater. *CouchDB The Definitive Guide*. OReilly Media, Inc, 2010.

[17] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 2010.

[18] S. B. e. a. Lance Ashdown, Cathy Baird. *Oracle9i Database Concepts*. Oracle Corporation., 2002.

[19] R. H. Prabhakar Chaganti. *Amazon SimpleDB Developer Guide Scale your application's database on the cloud using Amazon SimpleDB*. Packt Publishing Ltd, 2010.

[20] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, 2009.