A Brief Survey on Cloud Database Consistency

Robson A. Campêlo* Dorgival Guedes Alberto H. F. Laender

Department of Computer Science
Universidade Federal de Minas Gerais
Belo Horizonte, MG, Brazil
{robson.campelo, dorgival, laender}@dcc.ufmg.br

ABSTRACT

Cloud computing is a general term that involves delivering hosted services over the Internet. With the accelerated growth in the volume of data used by applications, several organizations have moved their data into cloud servers to provide scalable, reliable and highly available services. A particularly challenging issue that arises in the context of cloud storage systems with geographically-distributed data replication is how to reach a consistent state for all replicas. In this survey, we review major aspects related to consistency issues in cloud data storage systems, categorizing recently proposed methods into three categories: (1) static consistency methods, (2) dynamic consistency methods, and (3) consistency monitoring methods.

1. INTRODUCTION

Cloud computing is a general term that includes the idea of delivering hosted services over the Internet. The term "cloud" is an abstraction of this new model that arose from a common representation of a network, since the particular location of a service is not relevant, which means that services and data providers are seen as existing "in the network cloud".

In recent years, cloud computing has emerged as a paradigm that attracts the interest of organizations and users due to its potential for cost savings, unlimited scalability and elasticity in data management. In this paradigm, users acquire computing and storage resources in a pricing model which is known as *pay-as-you-go* [3]. According to this model, IT resources are offered in an unlimited way and the payment is made according to the actual resources used for a certain period, similarly to the traditional energy pricing model.

Depending on the kind of resource offered to the users, cloud services tend be grouped in the following three basic models: *Software as a Service* (SaaS) [22], *Platform as a Service* (PaaS) [7] and *Infrastructure as a Service* (IaaS) [9]. As an extension of this classification, when the service refers to a database, the model is known as *Database as a Service* (DBaaS) [19], which is the focus of this survey. This model provides transparent mechanisms to create, store, access and update databases. Moreover, the database service provider ensures full responsibility for the database administration, guaranteeing backup, reorganization and migration to new system versions.

The use of DBaaS cloud servers enables service providers to replicate and customize their data over multiple servers, which can be physically separated and placed in different data centers. By doing so, they can meet a growing demand by directing users to the nearest, or most recently accessed server. In that way, replication allows them to achieve features such as fast access,improved performance and higher availability. Thus, replication has become an essential feature of this storage model and is extensively exploited in cloud environments [14, 33].

A particularly challenging issue that arises in the context of cloud storage systems with geographically-distributed data replication is how to reach a consistent state in all replicas. In the cloud environment, computer failures and network partitions cannot be completely avoided. Enforcing synchronous replication to ensure strong consistency in such an environment incurs in a significant performance overhead, due to the problem of high network latency between data centers [30] and the fact that network partitions may lead to service unavailability [11]. As a consequence, specific models have been proposed to ensure weaker or relaxed consistency guarantees.

Several cloud storage services choose to ensure availability and performance even in the presence of network partitions rather than to use a stronger consistency model. NoSQL-based data storage environments provide consistency properties in eventual mode [48]. However, using this type of consistency increases the probability of reading obsolete

*Corresponding Author

data, since the replicas being accessed may not have received the most recent writes. This led to the development of adaptive consistency solutions, which were introduced to allow adjusting the level of consistency at runtime in order to improve performance or reduce costs, while maintaining the percentage of obsolete reads at low levels [16, 24, 47].

A consistency model in distributed environments seeks to guarantee the consistency of an update operation, as well as the access to an updated object. Obtaining the correct balance between consistency and availability is one of the challenges still existing in cloud computing [23]. In this survey, we focus on state-of-the-art methods for data consistency in cloud environments. Considering the different solutions, we categorize these recently proposed methods into three distinct categories: (1) static consistency methods, (2) dynamic consistency methods and (3) consistency monitoring methods.

The remainder of this survey is organized as follows. In Section 2, we present general concepts related to cloud database management. In Section 3, we approach the main consistency models adopted by existing distributed storage systems. In Sections 4 and 5, we present, respectively, a taxonomy proposed to categorize the most prominent consistency methods found in the literature and an overview of the main approaches adopted to implement them. In Section 6, we provide a sum-up discussion emphasizing the main aspects of the surveyed methods. Finally, in Section 7, we conclude the survey by summarizing its major issues.

2. CLOUD DATABASE MANAGEMENT

In this section, we present general concepts related to cloud database management in order to provide a better understanding of the key issues that affect data consistency. Initially, we approach the architectural aspects of the DBaaS model. After that, we highlight the cloud storage infrastructure requirements and describe the ACID properties. Finally, we introduce the CAP theorem and discuss its associated trade-offs.

2.1 Architectural Aspects of the DBaaS Model

In the DBaaS model, the architecture is often distributed and data can be stored in possibly hundreds of machines, where the resources are shared among the clients [50]. As previously mentioned, one of the techniques used to obtain specific features such as fast access, improved performance and high availability is to replicate data in geographically distinct locations.

For example, in a scenario in which a failure occurs in a replica, possibly the system will continue to operate by switching requests between the replicas. Another benefit is scalability, which allows the system to handle an increase in the number of requests as well as in the volume of stored data [46]. Once considering the possibility of data replication in geographically distant areas, it is necessary to define how updates will be spread among the replicas, since the manner how this operation is executed directly affects data consistency.

Data partitioning represents another distribution method in the DBaaS model. It is performed by breaking a table into two or more fragments or partitions, which can be stored in different locations [43]. This method differs from replication in the sense that instead of copying the data it splits them according to certain criteria. In cloud environments, partitioning may occur dynamically and in real time.

2.2 Cloud Data Storage Requirements

A trustworthy and appropriate data storage infrastructure is a key aspect to achieve an adequate cloud database management, so that all resources can be efficiently utilized and shared to reduce consistency issues. Therefore, the following are some needed and crucial requirements that must be considered in a shared infrastructure model [34, 44].

Automation. The data storage must be automated to be able to quickly execute infrastructure changes necessary to meet the previous requirements without human intervention.

Availability. The data storage must ensure that data continues to be available at a required level of performance in situations ranging from normal to adverse.

Elasticity. Not only must the data storage be able to scale with increasing load, but it must also be able to adjust to reductions in load by releasing cloud resources, while guaranteeing compliance with any Service Level Agreement (SLA).

Fault Tolerance. The data storage must be able to recover in case of failure by providing a backup instance of the application that will be ready to take over without disruption.

Low Latency. The data storage must handle latency issues by measuring and testing the network latency before committing an application.

Performance. The data storage must provide an infrastructure that supports fast and robust data access, update and recovery.

Reliability. The data storage must ensure that the data can be recovered in case a disaster occurs.

Scalability. The data storage needs to quickly scale to meet workload demands, providing horizontal and vertical scalability. Horizontal scalability refers to the ability to increase capacity by adding more machines or setting up a new cluster or a new distributed environment. Vertical scalability, on the other hand, refers to the increase of capacity by adding more resources to a machine (*e.g.*, more memory or an additional CPU).

2.3 The ACID Properties

Conventional Database Management Systems must conform to four transaction properties: *Atomicity*, *Consistency*, *Isolation* and *Durability*. Known as the ACID properties, they are well discussed in the literature [32], existing several approaches to achieve them. However, in the DBaaS model, due to architectural aspects previously described, assuring the ACID properties is a nontrivial task.

Despite the difficulty for assuring the ACID properties in a cloud data storage, strategies have been proposed in the attempt to manage them in web application transactions. For instance, atomicity might be guaranteed by implementing the two-phase commit (2PC) protocol [31]. Isolation can be obtained by a multi-version concurrency control or by a global timestamp, while durability can be achieved by applying queuing strategies such as FIFO (*First In, First Out*) to concurrent write transactions, so that old updates do not override the latest ones [49].

However, replication represents an important obstacle to guarantee consistency [1]. Thus, mantaining a replicated database in a mutually consistent state implies that in all of its replicas each of their data items must have identical values [42]. Therefore, strategies for data update and propagation must be implemented to ensure that, if a copy is updated, all the other ones must be updated too [46].

2.4 The CAP Theorem

The CAP theorem was introduced by Brewer as a conjecture [11] and subsequently proved (in a restricted form) by Gilbert and Lynch [28]. Since then it has become an important concept in cloud systems [12]. It establishes that, when considering the desirable properties of *Consistency*, *Availability*, and *Partition tolerance* in distributed systems, at most two of them can be achieved simultaneously.

It is evident that the CAP theorem introduces conflicts and imposes several challenges to distributed systems and service providers. Among the conflicts derived from it, considering that network partitions are inevitable in a geographically distributed scenario, we highlight the trade-off between Consistency and Availability [29]. To illustrate this situation, in Figure 1 we observe that User 2 performs a read request for data item D1 in replica R3 (Datacenter 2), after User 1 has updated data item D1 in replica R1 (Datacenter 1) in the presence of a network partition that isolates the two data centers. Assuming that the update made by User 1 has not been propagated, there are two possible scenarios: the replicas may be available and User 2 will read obsolete data, thereby violating consistency, or User 2 must wait until the network partition is fixed and the update has been propagated to replica R3, thus violating availability.

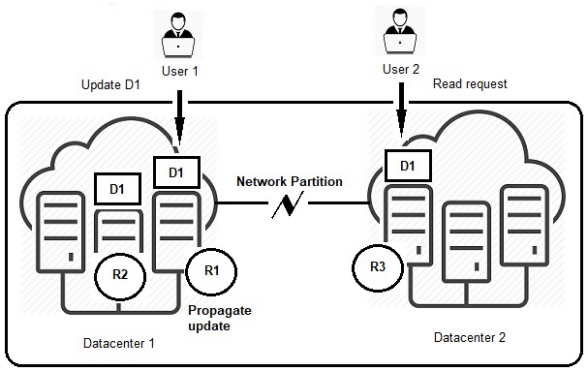

Figure 1: Consistency versus Availability in Replicated Systems.

The realization of the trade-offs caused by the CAP theorem led to the proliferation of not-ACID models for building cloud-based applications, *i.e.,* systems that are *Basically Available*, rely on the maintenance of a *Soft state* that can be rebuilt in case of failures and are only *Eventually consistent* to be able to survive network partitions. Such a model became know as BASE [26] and may offer different consistency models, which are discussed next.

3. CONSISTENCY MODELS

A consistency model may be defined as a contract between a data storage system and the data processes that access it [46]. It defines strategies for supporting consistency within a distributed data storage system. However, trade-offs due to the CAP theorem require the choice from a range of models to address different consistency levels, which may vary from a relaxed model to a strict one [8]. In this context, there are two distinct perspectives to be considered in a distributed data storage system with respect to consistency [46]: the *data provider* and the *clients*.

From the perspective of the data provider, the operations from all processes that want to access the data must be synchronized and ordered to guarantee correct results. From the perspective of the clients, it is often the case that shared updates are rare and they mostly access their private data. In that case, their major concern is that their own operations are consistent, what may be simpler to achieve. These two consistency perspectives are called *data-centric* and *client-centric* [46]. Next, we discuss the consistency models related to each perspective.

3.1 Data-centric Consistency Models

In this perspective, the consistency models seek to ensure that the data access process follows certain rules so that the storage system can work correctly. These rules are based on the definition of the results that are expected after read and write operations, even considering that these operations are concurrently performed. However, the absence of a global clock makes the identification of the last write operation a difficult task, which requires some restrictions on the data values that can be returned by a read operation, thus leading to a range of consistency models [46].

The consistency models that fall in this category are: *weak consistency, PRAM consistency, causal consistency, sequential consistency and strict consistency.*

3.1.1 Weak Consistency

The weak consistency model offers the lowest possible ordering guarantee. As this guarantee is very weak, we can say that it does not really exist, since an implementation may or may not have a protocol to synchronize replicas. It provides consistency to a group of transactions instead of to individual reads and writes, which is done by the stricter consistency models that follow next (PRAM, causal and sequential) [46, 48].

3.1.2 PRAM Consistency

PRAM (Pipelined Random Access Memory) consistency, also known as FIFO consistency, is a model in which write operations from a single process are seen by other processes in the same order that they were issued by its original process, whereas writes from different processes may be seen in a different order by different processes. In other words, there is no guarantee on the order in which the writes are seen by different processes, although writes from a single source must keep their order as if they were in a pipeline [37, 46].

3.1.3 Causal Consistency

Causal consistency is a model in which a sequential ordering is always maintained only between requests that have a causal relationship. Thus, concurrent requests do not share this relationship. In this case, all requests must be serialized in the same order on all replicas [46]. Thus, in a scenario of an always-available storage system in which requests have causal dependencies, a consistency level stricter than that provided by the casual model cannot be achieved due to trade-offs of the CAP theorem [39].

More specifically, two requests have a causal dependency if at least one of the following two conditions is achieved: (1) both requests are executed on a single thread and the execution of one precedes the other in time; (2) if a request B reads a value that has been written by a request A. Moreover, the relationship is transitive, so if A and B have a causal relation, and B and C also have a causal relation, then A and C share the same relationship [46, 48].

3.1.4 Sequential Consistency

Sequential consistency is a stricter model that differs from the previous one in the sense that it extends to all requests the need of the replicas to agree on the ordering of non-causally related requests. It requires that all operations be serialized in the same order on all replicas and that those related to the same process are executed in the order that they are received by the storage system [46]

3.1.5 Strict Consistency

Strict consistency is the model that provides the strongest consistency level. It states that if a write operation is performed on a data item, the result needs to be instantaneously visible to all processes, regardless in which replica the operation has occurred. To achieve this, an absolute global time order must be maintained [46].

3.2 Client-centric Consistency Models

In this perspective, the distributed data store is characterized by an relative absence of simultaneous updates. Also, when a simultaneous update occurs, the solution of any concurrency problem is simple to achieve. The emphasis is then to maintain a consistent view of data items for an individual client process that is currently operating on the data store [46]. The models in this category are: *eventual consistency, monotonic reads consistency, monotonic writes consistency, read-your-writes consistency* and *writes-follow-reads consistency.*

3.2.1 Eventual Consistency

The eventual consistency model states that all updates will propagate through the system and all replicas will gradually become consistent in the case of the absence of updates for a long time [46, 48]. Although this model does not provide concrete consistency guarantees, there are several distributed storage systems that implement it [14, 20, 27, 35].

3.2.2 Monotonic Read Consistency

The monotonic read consistency model guarantees that if a process reads a version of a data item d at time t, it will never see an older version of d at a later time. In an scenario where data visibility is not guaranteed to be instantaneous, at least the versions of a data item will become visible in chronological order [46, 48].

3.2.3 Monotonic Write Consistency

The monotonic write consistency model guarantees that a data store must serialize two writes w_1 and w_2 in the same order that they were sent by the same client [46, 48]. For instance, if the initial write operation w_1 is lost, it is not allowed to the subsequent write w_2 to overwrite that data item.

3.2.4 Read-Your-Writes Consistency

Read-your-writes consistency is closely related to the monotonic reads model. It guarantees that once a write operation is performed on a data item d, its effect will be seen by any successive read operation performed on d by the same process [46, 48]. This means that if a client has written a version v of a data item d, it will always be able to read a version at least as new as v.

3.2.5 Writes-Follow-Reads Consistency

Writes-follow-reads consistency guarantees that if a write operation w is requested by a process on a data item d, but there has been a previous read operation on d by the same process, then it is guaranteed that w will only be executed on the same or on a more recent value of d previously read [46].

3.3 Hierarchical View of the Consistency Models

In order to achieve a better understanding of the relationships among the consistency models, they can be hierarchically organized according to their degree of strictness from a lower consistency level to a higher one (see Figure 2).

This hierarchical organization also considers some model combinations, such as the PRAM consistency model that may be seen as a combination of the monotonic read (MR), monotonic write (MW) and read-your-writes (RYW) models [5]. In addition, according to Brzeziński et al. [13], there is also a relationship between causal consistency, which is a data-centric model, and the client-centric perspective, thus meaning that causal consistency is similar to write-follows-reads (WFR).

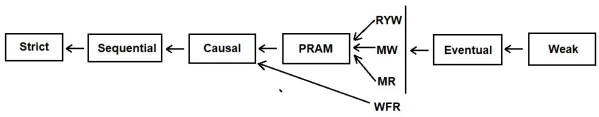

Figure 2: Hierarchical View of the Consistency Models.

4. A TAXONOMY FOR CONSISTENCY METHODS

In this section, we propose a taxonomy for categorizing the various consistency methods addressed in this survey. The proposed taxonomy categorizes these methods as follows: *static consistency methods, dynamic consistency methods, and consistency monitoring methods.*

Our criteria for proposing this taxonomy are based on the similarities of the core ideas behind the methods, which led us to consider these three categories as the most representative to categorize them. In what follows, we describe the main characteristics of the methods that belong to each category.

4.1 Static Consistency Methods

This category includes those methods that provide consistency guarantees in cloud storage systems, but are not flexible enough to support self-adaptivity according to the applications' consistency requirements, *i.e.*, such methods do not offer a diversity of consistency model options. Representative methods in this category are of two types: *Event Sequencing-based Consistency* and *Clock-based Strict Consistency*, which have been implemented in systems like PNUTS [17], and Spanner [18] and Clock-SI [21], respectively.

4.2 Dynamic Consistency Methods

The methods in this category implement mechanisms that provide dynamic characteristics such as self-adaptive or flexible consistency guarantees, which allows the selection or specification of the desired consistency level. They are of two types: *Automated and Self-Adaptive Consistency,* and *Flex-*

ible Consistency. The first type has been implemented in systems like Harmony [16], VFC^3 [24] and Pileus [47], whereas the second one supports systems like Indigo [6], SSOR [15] and Amazon DynamoDB [45].

4.3 Consistency Monitoring Methods

Alternatively, instead of directly handling data consistency issues, some methods focus on providing mechanisms that allow data owners to detect the occurrence of consistency violations in the cloud storage. This means that clients might audit their own data and make decisions based on how the Cloud Service Provider (CSP) stores and manages their data copies according to the consistency level that has been agreed upon in the service level contract. The methods in this category are of two types, *Consistency Verification* and *Consistency Auditing*, and have been respectively implemented in VICOS [10], and in DMR-PDP [41] and CaaS [38].

5. OVERVIEW OF CLOUD DATABASE CONSISTENCY METHODS

In this section, we present an overview of the main approaches adopted by the consistency methods addressed in this survey. These methods are among the most representative in the literature, although our survey is not exhaustive. The presentation follows the taxonomy introduced in Section 4.

5.1 Static Consistency Methods

As mentioned earlier, the methods in this category provide fixed guarantees, not being able, for example, to adjust their behavior to take advantage of situations where a stronger consistency might be temporarily available. As we shall see next, they can be divided into two types: those based on the sequence of events and those based on clock timestamps.

5.1.1 Event Sequencing-based Consistency

This type of consistency method aims to hide the replication complexity by providing a consistency model that lies between the two extremes of general serializability and eventual consistency. The key aspect behind this type of method is the observation that transaction serializability is costly and often unnecessary in web applications [17].

The most representative system that implements this type of method is PNUTS [17], which is a massively parallel and geographically distributed DBMS developed by Yahoo for web applications. PNUTS developers observed that web applications typically manipulate one record at a time, whereas different records may be located in different geographic locality. Hence, an event sequencing-based consistency method establishes that all replicas of a given record receive all updates applied to that record in the same order. This strategy is implemented by designating one of the replicas as the master for each record, so that this master receives all writes sent to that record by the other replicas. If a record has the majority of its writes sent to a particular replica, this replica becomes the master for that record.

5.1.2 Clock-based Strict Consistency

Systems that implement this type of consistency method are characterized by the use of clock-based mechanisms to control timestamps to enforce strict consistency [18, 21]. They offer the guarantee that arbitrary objects in the data store are accessed atomically and isolated from concurrent accesses. The approach behind this consistency method is based on the ability of a system to provide a timestamp log to track the order in which operations occur.

Spanner [18] and Clock-SI [21] are representative systems that implement this type of consistency method. The former is a relational-like distributed data store system developed by Google. It assigns globally-meaningful commit timestamps that reflect the serialization order of the transactions, which may be distributed. Spanner also enforces that if a transaction T_2 begins after the commit of a transaction T_1, then the commit timestamp of T_2 must be greater than the commit timestamp of T_1.

On the other hand, Clock-SI implements the so called *snapshot isolation*, which is a consistency criterion for partitioned data storages. In this strategy, read-only operations read from a consistent snapshot and other operations perform a commit if no objects written by these transactions were concurrently written. The local physical clock is used by each transaction to identify its read timestamp.

5.2 Dynamic Consistency Methods

Some methods may be able to adjust guarantees based on the observation of access patterns and system structural aspects. We discuss them next.

5.2.1 Automated and Self-adaptive Consistency

This type of consistency method aims to dynamically enforce multiple consistency degrees over distinct data objects. Three main approaches have been adopted to implement such methods.

The first approach is based on probabilistic computations that provide an estimation of the stale reads rate. Once the estimation model is computed, it is possible to identify the key parameter that affects the stale reads and scale up/down the number of replicas involved in read operations in order to maintain a low tolerable fraction of stale reads. This approach is mainly implemented by Harmony [16], which is a consistency-based system that aims to gradual and dynamically tune the consistency level at runtime according to the applications' consistency requirements.

The second approach allows the evaluation and enforcement of divergence bounds over data objects (table/row/column), so that the consistency levels can be automatically adjusted based on statistical information. The evaluation takes place on a divergence vector every time an update request is received, although it is necessary to identify the affected data objects. If any limit is exceeded, all updates since the last replication are placed in a FIFO-like queue to be propagated and executed on other replicas. The most representative system that implements this approach is VFC^3 (*Versatile Framework for Consistency in Cloud Computing*) [24], which provides three-dimensional consistency Quality-of-Service vectors, where users can specify consistency parameters according to the QoS level desired, thus letting the system automatically adjust the consistency degree.

Finally, the third approach is characterized by dynamically selecting to which server (or even set of servers) each read of a record must be directed, so that the best service is delivered given the current configuration and system conditions. Hence, this approach is adaptable to distinct configurations of replicas and users, as well as to changing conditions such as variations on the network performance or server load. Another important aspect of this approach is the fact that it allows application developers to provide a Service Level Agreement (SLA) that specifies the applications' consistency/latency desires in a declarative manner. Pileus [47] is a key-value storage system that implements this approach. It provides a diversity of consistency guarantee options for globally distributed and replicated data environments. It also allows several systems or even clients of a system to achieve different consistency degrees, even while sharing the same data.

5.2.2 *Flexible Consistency*

This type of consistency method is characterized by its flexibility to adapt to predefined consistency models. In this context, there are three main approaches that implement this type of consistency: combining both weak and strong consistency depending on the operation, adopting the notion of consistency regions and performing eventually or strongly consistent reads as needed [6, 15, 45].

The first approach aims to strengthen eventual consistency allowing the applications to specify consistency rules, or *invariants*, that must be maintained by the system. Once these invariants are defined, it is possible to identify the operations that are potentially unsafe under concurrent execution, thus allowing one to select either a violation-avoidance or an invariant-repair technique. Indigo [6] is a middleware system that implements this approach. It supports an alternative consistency method built on top of a geo-replicated key-value data store.

The second approach is based on a formalism proposed to support the applications' consistency requirements. The trade-off between consistency and scalability requirements is handled by introducing the notion of consistency regions and service-delivery-oriented consistency policies. A consistency region is defined as a logical unit that represents the application state-level requirements for consistency and scalability. This concept is used to define consistency boundaries that separate each group of services that need to be ordered. Hence, services that need to be kept consistent must be associated to a certain region. The definition of what region a service belongs to and which services that can concurrently be delivered is set by the system administrators.

Scalable Service Oriented Replication (SSOR) [15] is a middleware that implements this approach. It covers three distinct types of consistency region: (1) Conflict Region (CR), which is a region composed by services that have conflicting requirements for consistency regardless of the session; (2) Sessional Conflict Region (SCR), which is a region that includes services of a particular session with conflicting consistency requirements; and (3) Non-Conflict Region (NCR), which is a region that does not impose any consistency constraints or requirements.

The third approach aims to provide elasticity and flexibility in order to handle unpredictable workloads, but allowing a system to be tuned for capacity. Due to this characteristic, it allows applications to perform eventually or strongly consistent reads as needed. Amazon DynamoDB [45] is a highly reliable and cost-effective NoSQL database service that implements this approach. It was built based on the experience with its predecessor Dynamo [20]. DynamoDB adopts eventual consistency as its default model, which does not guarantee that an eventually

consistent read will always reflect the result of a recently completed write. On the other hand, when adopting a stronger consistency model, it returns a result that reflects all writes that have received a successful response prior to that read.

5.3 Consistency Monitoring Methods

The methods in this category allow the applications to evaluate the level of consistency that has been actually obtained during execution, so that they can make their own decisions on how to handle any problems when they arise.

5.3.1 Consistency Verification

These methods rely on two approaches to consistency verification. The first consists of a protocol that enables a group of mutually trusting clients to detect consistency violations on a cloud storage, whereas the second provides a verification scheme that allows data owners to ensure whether the Cloud Service Provider (CSP) complies with the SLA for storing data in multiple replicas.

This protocol-based approach is adopted by VICOS (Verification of Integrity and Consistency for Cloud Object Storage) [10]. VICOS supports the optimal consistency concept of *fork-linearizability*, which captures the strongest achievable notion of consistency in multi-client models. The method may guarantee this notion by registering the causal evolution of the user's views into their interaction with the server. In case the server creates only a single discrepancy between the views of two clients, it is ensured that these clients will never observe operations of each other afterwards. That is, if these users communicate later and the server ever lies to them, the violation will be immediately discovered. Thus, users can verify a large number of past transactions by performing a single check.

The verification approach is implemented by DMR-PDP (*Dynamic Multi-Replica-Provable Data Possession*) [41]. The context addressed by this scheme is that whenever data owners ask the CSP to replicate data at different servers, they are charged for this. Hence, data owners need to be strongly persuaded that the CSP stores all data copies that are agreed upon in the service level contract, as well as that all remotely stored copies correctly execute the updates requested by the users. This approach deals with such problems by preventing the CSP from cheating the data storage; for instance, by maintaining fewer copies than paid for. The scheme is based on a technique called *Provable Data Possession* [4], used to audit and validate the integrity and consistency of data stored on remote servers.

5.3.2 Consistency Auditing

This kind of method is based on an architecture that conists of a large data cloud maintained by a CSP and multiple small audit clouds composed of a group of users that cooperate on a specific job (*e.g.*, revising a document or writing a program). The required level of consistency that should be provided by the data cloud is stipulated by an SLA involving the audit cloud and the data cloud. Once the SLA exists, the audit cloud can verify whether the data cloud violates it, thus quantifying, in monetary terms or otherwise, the severity of the violation.

Consistency as a Service (CaaS) [38] implements this method. It relies on a two-level auditing structure, namely: *local auditing* and *global auditing*. Local auditing allows each user to independently perform local tracing operations, focusing on monotonic read and read-your-write consistencies. Global auditing, on the other hand, requires that an auditor is periodically elected from the audit cloud to perform global tracing operations, focusing on casual consistency. This type of method is supported by constructing a directed graph of operations, called the *precedence graph*. If the constructed graph is a directed acyclic graph (DAG), the required level of consistency is preserved [38].

6. DISCUSSION

As proposed in Section 4, consistency methods can be grouped in three categories: static consistency methods, dynamic consistency methods and consistency monitoring methods.

Static Consistency methods are mostly based on versioning of events. This captures the idea of event ordering by means of control strategies such as a sequence number that represents a data object version or clock-based mechanisms. We note that this is a well-understood concept in distributed systems [25, 36, 40]. The idea of an event happening before another represents a causal relationship and the total ordering of events among the replicas has been shown quite useful for solving synchronization issues related to data consistency. Thus, consistency methods in this category extend this concept on specific scenarios.

Dynamic consistency methods, in turn, generally aim to provide mechanisms that adjust the degree of consistency without human intervention. This is an important feature for applications that have temporal characteristics, as well as for real-time workload cloud storage systems. Specifically, flexible consistency methods are suitable to address applications' consistency requirements that need to adapt to pre-

Table 1: Storage Requirements Supported by the Consistency Methods

Category	Representative Systems	Storage Requirements [34, 44]							
		Automation	Availability	Elasticity	Fault Tolerance	Low-Latency	Performance	Reliability	Scalability
Static Consistency Methods	PNUTS [17]		✓		✓			✓	✓
	Spanner [18]		✓		✓	✓			✓
	Clock-SI [21]		✓			✓	✓		✓
Dynamic Consistency Methods	Indigo [6]				✓	✓			
	SSOR [15]			✓	✓		✓	✓	✓
	Harmony [16]	✓	✓	✓		✓	✓		
	VFC [24]	✓	✓			✓	✓	✓	✓
	DynamoDB [45]		✓	✓		✓	✓	✓	✓
	Pileus [47]	✓	✓		✓		✓	✓	✓
Consistency Monitoring Methods	VICOS [10]	not applied							
	CaaS [38]	not applied							
	DMR-PDP [41]	not applied							

defined consistency models.

On the other hand, consistency monitoring methods do not provide specific guarantees, but focus on detecting the occurrence of consistency violations in the cloud data storage. Despite that, these methods offer significant contributions that are suitable for scenarios where multiple clients cooperate on data remotely stored in a potentially misbehaving service, and need to rely on the Cloud Service Provider to guarantee their confidentiality and correctness. Furthermore, those clients need to verify if the requested data updates were correctly executed on all remotely stored copies, while maintaining the required consistency level.

Finally, Table 1 summarizes the storage requirements supported by the systems that we have surveyed in order to stress what are the main consistency trade-offs considered by them. Note that we only address those systems that implement a specific consistency method, since consistency monitoring methods only focus on detecting consistency violations. Table 1 also shows that availability and scalability are the most addressed storage requirements supported by the surveyed system, whereas elasticity is the least one.

As previously mentioned, existing trade-offs determined by the CAP theorem imply that applications must sacrifice consistency under certain scenarios to be able to satisfy other application requirements. This suggests that most of the consistency methods are also involved in providing other features such as high availability and scalability, even reducing the degree of consistency to a level that may be deemed acceptable.

Although elasticity is not exploited by most of the surveyed systems, this is a desirable and important feature for large scale applications. It is characterized by the ability to deal with load increases by adding more resources during high demand and releasing them when load decreases. Elasticity differs from scalability in sense that the latter is a static property of the system whereas the former is a dynamic feature that allows the system's scale to be increased [2]. For instance, a scalable system might scale to hundreds or even to thousands of nodes. In contrast, an elastic system can scale from 10 servers to 20 servers (or vice-versa) on-demand. Therefore, we argue that it would be important to identify the challenges for supporting an elastic and consistent cloud data store.

7. CONCLUSIONS

In this survey, we have reviewed several methods proposed in the literature to ensure consistency in distributed cloud data storage systems (Table 2). Ensuring consistency in replicated databases is an important research topic that offers many challenges in the sense that such systems must provide a consistent state in all replicas, despite the occurrence of concurrent transactions. In other words, it must be provided a suite of strategies for data update and propagation in order to guarantee that if one copy

Table 2: Summary of the Surveyed Methods

Category	Method	Brief Description
Static Consistency	Event Sequencing-based Consistency [17]	Establishes that all replicas of a given record apply all updates to a record in the same order.
	Clock-based Strict Consistency [18, 21]	Uses clock-based mechanisms to control timestamps to enforce strict consistency.
Dynamic Consistency	Automated and Self-Adaptive Consistency [16, 24, 47]	Provides a gradually and dynamically tunable consistency level at runtime according to the applications' consistency requirements. Enforces increasing degrees of consistency for different types of data, based on their semantics.
	Flexible Consistency Guarantees [6, 15, 45]	Allows applications to specify consistency rules, or *invariants*, that must be maintained by the system. Supports the applications' consistency requirements and flexibly adapt to predefined consistency models. Allows applications to perform eventually or strongly consistent reads as needed.
Consistency Monitoring	Consistency Verification [10, 41]	Enables a group of mutually trusting clients to detect data-integrity and consistency violations. Allows the data owner to ensure that the Cloud Service Provider stores all data copies that are agreed upon in the service level contract.
	Consistency Auditing [38]	Implements a Local and Global Auditing structure to allow a group of clients to detect the occurrence of consistency violations.

is updated, all the other ones must be updated as well. The proposed taxonomy presented here provides researchers and developers with a framework to better understand the current main ideas and challenges in this area.

As presented in the surveyed works, there have been significant efforts in the area. However, although some challenges have been addressed, there are still many open issues. For instance, despite existing solutions that aim at supporting general-purpose applications [10, 24, 38, 41], we highlight that some consistency methods consider very specific contexts [6, 15, 16, 17, 18, 21, 45, 47]. Moreover, very few of the proposed solutions [15, 16, 45] address the elasticity property when dealing with consistency requirements in large scale cloud data storage systems, which allows us to say that there are still open issues to be exploited by others.

8. ACKNOWLEDGEMENTS

This research is funded by projects InWeb (grant MCT/CNPq 573871/2008-6), MASWeb (grant FAPEMIG/PRONEX APQ-01400-14) and EUBR ATMOSPHERE, and by the authors' individual grants from CAPES, CNPq and FAPEMIG. The authors would also like to thank José Palazzo Moreira de Oliveira for his comments on a draft of this work.

9. REFERENCES

[1] D. J. Abadi. Data Management in the Cloud: Limitation and Opportunities. *IEEE Data Engineering Bulletin*, 32:3–12, 2009.

[2] D. Agrawal, A. E. Abbadi, S. Das, and A. J. Elmore. Database Scalability, Elasticity, and Autonomy in the Cloud. In *Proc. of the International Conference on Database Systems for Advanced Applications*, pages 2–15, Hong Kong, China, 2011.

[3] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais, and I. Ahmad. Cloud computing pricing models: A survey. *International Journal of Grid and Distributed Computing*, 6(5):93–106, 2013.

[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and

D. Song. Provable Data Possession at Untrusted Stores. In *Proc. of the 14th ACM Conference on Computer and Communications Security*, pages 598–609, Alexandria, VA, USA, 2007.

[5] P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Highly available transactions: Virtues and limitations. *Proc. of the VLDB Endowment*, 7(3):181–192, 2013.

[6] V. Balegas, S. Duarte, C. Ferreira, R. Rodrigues, N. Preguiça, M. Najafzadeh, and M. Shapiro. Putting Consistency Back into Eventual Consistency. In *Proc. of the Tenth European Conference on Computer Systems*, pages 6:1–6:16, Bordeaux, France, 2015.

[7] D. Beimborn, T. Miletzki, and D. I. S. Wenzel. Platform as a Service (PaaS). *Wirtschaftsinformatik*, 53(6):371–375, 2011.

[8] D. Bermbach and J. Kuhlenkamp. Consistency in Distributed Storage Systems: An Overview of Models, Metrics and Measurement Approaches. In *Proc. of the First International Conference on Networked Systems*, pages 175–189. Marrakech, Morocco, 2013.

[9] S. Bhardwaj, L. Jain, and S. Jain. Cloud computing: A study of infrastructure as a service (IAAS). *International Journal of Engineering and Information Technology*, 2(1):60–63, 2010.

[10] M. Brandenburger, C. Cachin, and N. Knezevic. Don't Trust the Cloud, Verify: Integrity and Consistency for Cloud Object Stores. *ACM Transactions on Privacy and Security*, 20:16:1–16:11, 2015.

[11] E. A. Brewer. Towards robust distributed systems. In *Proc. of the 19th Annual ACM Symposium on Principles of Distributed Computing*, pages 7–14, Portland, Oregon, USA, 2000.

[12] Eric Brewer. Cap twelve years later: How the "rules" have changed. *Computer*, 45:23–29, 2012.

[13] J. Brzezinski, C. Sobaniec, and D. Wawrzyniak. From Session Causality to Causal Consistency. In *Proc. of the 12th Conference on Parallel, Distributed and Network-Based Processing*, pages 152–158, Coruña, Spain, 2004.

[14] 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. *ACM Transactions on Computer Systems*, 26(2):4:1–4:14, 2008.

[15] T. Chen, R. Bahsoon, and A. H. Tawil. Scalable service-oriented replication with flexible consistency guarantee in the cloud. *Information Sciences*, 264:349–370, 2014.

[16] H. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Perez. Harmony: Towards Automated Self-adaptive Consistency in Cloud Storage. In *Proc. of the 2012 IEEE International Conference on Cluster Computing*, pages 293–301, Beijing, China, 2012.

[17] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: Yahoo!'s Hosted Data Serving Platform. *Proc. of the VLDB Endowment*, 1(2):1277–1288, 2008.

[18] J. C. Corbett, J. Dean, and M. Epstein et al. Spanner: Google's Globally Distributed Database. *ACM Trans. Comput. Syst.*, 31(3):8:1–8:22, 2013.

[19] C. Curino, E. Jones, R. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational Cloud: A Database-as-a-Service for the Cloud. In *Proc. of the 5th Biennial Conference on Innovative Data Systems Research*, pages 235–240, Asilomar, CA, USA, 2011.

[20] 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. *ACM SIGOPS Oper. Syst. Rev.*, 41(6):205–220, 2007.

[21] J. Du, S. Elnikety, and W. Zwaenepoel. Clock-SI: Snapshot Isolation for Partitioned Data Stores Using Loosely Synchronized Clocks. In *Proc. of the IEEE 32nd International Symposium on Reliable Distributed Systems*, pages 173–184, Braga, Portugal, 2013.

[22] A. Dubey and D. Wagle. Delivering software as a service. *The McKinsey Quarterly*, 6:1–7, 2007.

[23] M. M. Elbushra and J. Lindström. Eventual Consistent Databases: State of the art. *Open Journal of Databases*, 1(1):26–41, 2014.

[24] S. Esteves, J. Silva, and L. Veiga. Quality-of-service for consistency of data geo-replication in cloud computing. In *Proc. of the 18th European Conference on Parallel Processing*, pages 285–297, Rhodes Island,

Greece, 2012.

[25] C. Fidge. Logical time in distributed computing systems. *Computer*, 24(8):28–33, 1991.

[26] Ar. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *Proc. of the 16th ACM Symposium on Operating Systems Principles*, SOSP '97, pages 78–91, New York, NY, USA, 1997. ACM.

[27] S. Ghemawat, H. Gobioff, and S. Leung. The Google File System. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003.

[28] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[29] S. Gilbert and N. Lynch. Perspectives on the CAP theorem. *Computer*, 45(2):30–36, 2012.

[30] S. Goel and R. Buyya. Data replication strategies in wide-area distributed systems. In *Enterprise service computing: from concept to deployment*, pages 211–241. 2007.

[31] Jim Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, pages 393–481, London, UK, UK, 1978. Springer-Verlag.

[32] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, 15(4):287–317, December 1983.

[33] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu. Maestro: Replica-Aware Map Scheduling for MapReduce. In *Proc. of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 435–442, Ottawa, Canada, 2012.

[34] J. Ju, J. Wu, J. Fu, Z. Lin, and J. Zhang. A Survey on Cloud Storage. *Journal of Computers*, 6(8):1764–1771, 2011.

[35] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

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

[37] R. J. Lipton and J. S. Sandberg. *PRAM: A Scalable Shared Memory*. Tecnical Report TR-180-88, Department of Computer Science, Princeton University, 1988.

[38] Q. Liu, G. Wang, and J. Wu. Consistency as a Service: Auditing Cloud Consistency. *IEEE Transactions on Network and Service Management*, 11(1):25–35, 2014.

[39] P. Mahajan, L. Alvisi, and M. Dahlin. *Consistency, Availability, and Convergence*. Tecnical Report UTCS TR-11-22, Department of Computer Science, The University of Texas at Austin, 2011.

[40] F. Mattern. Virtual Time and Global States of Distributed Systems. *Parallel and Distributed Algorithms*, 1(23):215–226, 1989.

[41] R. Mukundan, S. Madria, and M. Linderman. Replicated data integrity verification in cloud. *IEEE Data Eng. Bull.*, 35(4):55–64, 2012.

[42] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Springer Science & Business Media, 2011.

[43] S. K. Rahimi and F. S. Haug. *Distributed Database Management Systems: A Practical Approach*. Wiley-IEEE Computer Society, USA, 1st edition, 2010.

[44] B. P. Rimal, E. Choi, and I. Lumb. A Taxonomy and Survey of Cloud Computing Systems. In *Proc. of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, pages 44–51, Seoul, Korea, 2009.

[45] S. Sivasubramanian. Amazon DynamoDB: A Seamlessly Scalable Non-relational Database Service. In *Proc. of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 729–730, Scottsdale, Arizona, USA, 2012.

[46] A. S. Tanenbaum and M. V. Steen. *Distributed Systems: Principles and Paradigms*. Prentice-Hall, 2007.

[47] D. B. Terry, V. Prabhakaran, and R. Kotla et al. Consistency-based Service Level Agreements for Cloud Storage. In *Proc. of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 309–324, New York, NY, USA, 2013.

[48] W. Vogels. Eventually consistent. *Comm. of ACM*, 52(1):40–44, 2009.

[49] Z. Wei, G. Pierre, and C. Chi-Hung. Scalable Transactions for Web Applications in the Cloud. In *Proc. of the 15th International Euro-Par Conference on Parallel Processing*, pages 442–453, Delft, The Netherlands, 2009.

[50] P. Xiong, Y. Chi, and S. Zhu et al. Intelligent Management of Virtualized Resources for Database Systems in Cloud Environment. In *Proc. of the IEEE 27th International Conference on Data Engineering*, pages 87–98, Washington, DC, USA, 2011.