Noname manuscript No.
(will be inserted by the editor)

A Brief Survey on Replica Consistency in Cloud Environments

Robson A. Campêlo · Marco A. Casanova · Dorgival Guedes · Alberto H. F. Laender

Received: date / Accepted: date

Abstract Cloud computing is a general term that involves delivering hosted services over the Internet. With the accelerated growth of 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. This survey reviews 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.

Keywords Replica consistency · Cloud environments · Storage systems · Consistency models

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".

Robson A. Campêlo · Dorgival Guedes · Alberto H. F. Laender
Department of Computer Science, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brazil
E-mail: {robson.campelo, dorgival, laender}@dcc.ufmg.br

Marco A.Casanova
Department of Informatics, Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brazil
E-mail: casanova@inf.puc-rio.br

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 that paradigm, users acquire computing and storage resources in a pricing model which is known as *pay-as-you-go* [4]. According to that 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 home utilities model.

Depending on the kind of resource offered to the users, cloud services tend to be grouped in the following three basic models: *Software as a Service* (SaaS) [26], *Platform as a Service* (PaaS) [9], and *Infrastructure as a Service* (IaaS) [12]. As an extension of this classification, when the service refers to a database, the model is known as *Database as a Service* (DBaaS) [23], which is the focus of this survey. Such a model provides transparent mechanisms to create, store, access and update databases. Moreover, the database service provider takes full responsibility for the database administration, thus guaranteeing backup, reorganization and version updates.

The use of DBaaS solutions enables service providers to replicate and customize their data over multiple servers, which can be physically separated, even placed in different data centers [56]. By doing so, they can meet growing demands 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 [17, 37].

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. Enforcing synchronous replication to ensure strong consistency in such an environment incurs in significant performance overheads, due to the increased network latency between data centers [34], and to the fact that network partitions may lead to service unavailability [15]. As a consequence, specific models have been proposed to offer weaker or relaxed consistency guarantees [54].

Several cloud storage services choose to ensure availability and performance even in the presence of network partitions rather than to offer a stronger consistency model. NoSQL-based data storage environments provide consistency properties in eventual mode [54], which means that all changes to a replicated piece of data eventually reach all its replicas. However, using this type of consistency increases the probability of reading obsolete 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 run-time in order to improve performance or reduce costs, while maintaining the percentage of obsolete reads at low levels [19,28,52].

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 open challenges in cloud computing [27]. In this survey, we focus on state-of-the-art methods for consistency in cloud environments. Considering the different solutions, we categorize those methods into three distinct categories: (1) static consistency methods, (2) dynamic consistency methods and (3) consistency monitoring methods. Other surveys on distinct issues related to replica consistency have been recently published [3,53]. We refer the reader to them for further considerations on this topic.

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 Section 4, we first propose a taxonomy to categorize the most prominent consistency methods found in the literature and then present an overview of the main approaches adopted to implement them. In Section 5, we provide a sum-up discussion emphasizing the main aspects of the surveyed methods. Finally, in Section 6, 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 replica consistency in cloud environments. Initially, we highlight the cloud storage infrastructure requirements and describe the ACID properties. Then, we introduce the CAP theorem and discuss its associated trade-offs.

2.1 Cloud data storage requirements

A trustworthy and appropriate data storage infrastructure is a key aspect to provide an adequate cloud data storage infrastructure, so that all resources can be efficiently used and shared to reduce consistency issues. Next we list some crucial requirements that must be considered by a shared infrastructure model [15,38,48].

Automation. The data storage must be automated to be able to quickly execute infrastructure changes required to maintain replica consistency with no 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 a Service Level Agreement (SLA).
Fault tolerance. The data storage must be able to recover in case of failure, *e.g.*, 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.
Partition Tolerance. The data storage must be tolerant to network partitions, *i.e.*, the system must continue to operate despite them.
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, thus 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).

2.2 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 [36], which describes several approaches to achieve them. However, in the DBaaS model, due to the architectural aspects previously described, assuring the ACID properties is a nontrivial task.

Despite the difficulty to ensure the ACID properties in a cloud data storage, strategies have been proposed in an attempt to manage them in web application transactions. For instance, atomicity might be guaranteed by implementing the two-phase commit (2PC) protocol [35], whereas isolation can be obtained by a multiversion concurrency control or by a global timestamp, and durability 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 [55]. However, replication represents an important obstacle to guarantee consistency. [1]. Thus, maintaining a replicated database in a mutually consistent state implies that, in all of its replicas, each of their data items must have identical values. [47]. Therefore, strategies for data update and propagation must be implemented to ensure that, if a copy is updated, all others must also be updated [51].

2.3 The CAP theorem

The CAP theorem was introduced by Brewer as a conjecture [15] and subsequently proved (in a restricted form) by Gilbert and Lynch [32]. Since then it has become an important concept in cloud systems [16]. 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 [33]. 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.

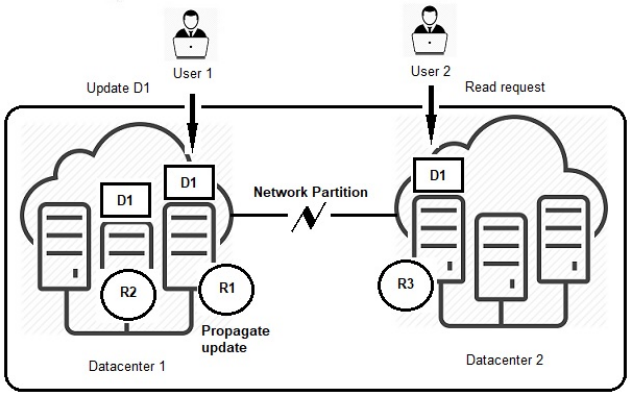

Fig. 1 Consistency vs. Availability in Replicated Systems.

The trade-offs caused by the CAP theorem led to the proliferation of not-ACID systems 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 not-ACID systems, known as BASE [30], offer distinct 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 [51], thus defining strategies that support consistency within a distributed data storage system. However, trade-offs due to the CAP theorem require choosing from a range of models to address different consistency levels, which may vary from a relaxed model to a strict one [10]. In this context, there are two distinct perspectives to be considered in a distributed data storage system with respect to consistency [51]: the *data-centric perspective* and the *client-centric perspective*.

From the data-centric perspective, the operations from all processes that want to access the data must be synchronized and ordered to guarantee correct results. From the client-centric perspective, it is often the case that shared updates are rare and mostly access private data. In such cases, the major concern is to make

their own operations consistent, which may be simpler to achieve.

3.1 Data-centric consistency models

In this perspective, the consistency models seek to ensure that the data access process follows certain rules that guarantee that the storage system works correctly. The period between the update and the moment when this guarantee is reached is called *inconsistency window*. These rules are based on the definition of the results that are expected after read and write operations, even considering that those 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. The consistency models that fall in this category are [51]: *weak consistency, PRAM consistency, causal consistency, sequential consistency* and *strict consistency*.

Weak Consistency. As its name indicates, weak consistency is the model that offers the lowest possible ordering guarantee, since it allows data to be written across multiple nodes and always returns the version that the system finds first. This means that there is no guarantee that the system will eventually become consistent, which is the case of the stricter consistency models discussed next (PRAM, causal, sequential and strict) [51, 54].

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 the other processes in the same order that they were issued, 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 [41, 51].

Causal Consistency. Causal consistency is a model in which a sequential ordering is always maintained only between requests that have a *causal dependency*. 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) request B reads a value that has been written by a request A. Moreover, this dependency is transitive, so if A and B have a causal dependency, and B and C also have a causal dependency, then there is also a causal dependency between A and C [51, 54]. 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 causal model cannot be achieved due to trade-offs of the CAP theorem [44].

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 in all replicas and also that those related to the same process are executed in the order that they are received by the storage system [51].

Strict Consistency. Strict consistency is a 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 of in which replica the operation occurred. To achieve that, an absolute global time order must be maintained [51].

3.2 Client-centric consistency models

In this perspective, a distributed data store is characterized by a relative absence of simultaneous updates. 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. The consistency models that fall in this category are [51]: *eventual consistency, monotonic reads consistency, monotonic writes consistency, read-your-writes consistency* and *writes-follow-reads consistency*.

Eventual Consistency. The eventual consistency model states that all updates will propagate through the system and all replicas will gradually become consistent, after all updates have stopped for some time [51, 54]. Although this model does not provide concrete consistency guarantees, it is advocated as a solution for many practical situations [6, 7, 20, 54] and has been implemented by several distributed storage systems [17, 24, 31, 39].

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 a scenario where data visibility is not guaranteed to be instantaneous, at least the versions of a data item will become visible in

chronological order [51, 54].

Monotonic Write Consistency. This 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 [51,54]. For instance, if the initial write operation w_1 is delayed, it is not allowed for a subsequent write w_2 to overwrite that data item before w_1 completes.

Read-Your-Writes Consistency. This consistency model is closely related to the monotonic read one. 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 [51, 54]. 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.

Writes-Follow-Reads Consistency. This consistency model 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 r on d by the same process, then it is guaranteed that w will only be executed on the same or more recent value of d previously read. [51].

4 Replica consistency methods

In this section, we present an overview of state-of-the-art methods for replica consistency in cloud environments. This overview includes those methods that we considered to be among the most representative in the literature. Based on the similarities of their core ideas, we classified these methods into three distinct categories: (1) static consistency methods, (2) dynamic consistency methods and (3) consistency monitoring methods. In the following sections, we first describe the generic characteristics of each category and then present an overview of its specific methods.

4.1 Static consistency methods

This category includes those methods that provide consistency guarantees in cloud storage systems, but which 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 options. Representative methods in this category are of two types: *Event Sequencing-based Consistency* and *Clock-based Strict Consistency*. They are described next.

4.1.1 Event sequencing-based consistency

This type of consistency method aims at hiding 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 [7].

The most representative system that implements this type of method is PNUTS [21], 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 localities. 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.

4.1.2 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 [22, 25]. 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. According to Bravo et al. [14], this type of technique is implemented by the data storage systems themselves and might impact the consistency guarantees that they provide.

Spanner [22] and Clock-SI [25] 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 storage. In this strategy, read-only operations read from a consistent snapshot and other op-

erations 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. Another example of a system that implements *snapshot isolation* is Vela [49]. It uses this technique to provide fault tolerance, performance and scalability, without sacrificing consistency.

4.2 Dynamic consistency methods

The methods included in this category implement mechanisms that provide dynamic characteristics such as self-adaptive or flexible consistency guarantees, which allow the selection or specification of the desired consistency level at any given point. These methods are of two types, *Automated and Self-adaptive Consistency* and *Flexible Consistency*, and are described next.

4.2.1 Automated and self-adaptive consistency

This type of consistency method aims at dynamically enforcing 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[1] rate. Once the estimation model is computed, it is possible to identify the key parameter that affects the stale reads and then to 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 [19], which is a consistency-based system that aims at gradual and dynamically tuning 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 the other replicas. The most representative system that implements this approach is VFC[3] (*Versatile Framework for Consistency in Cloud Computing*) [28], 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 a 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 [52] 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.

4.2.2 Flexible consistency

This type of consistency method is characterized by its flexibility to adapt to predefined consistency models. In this context, there are four main approaches that implement this type of consistency: (1) combining both weak and strict consistency depending on the operation, (2) providing configuration options for actors in a multi-cluster, (3) adopting the notion of consistency regions and (4) performing eventually or strongly consistent reads as needed [8, 11, 18, 50].

The first approach aims at strengthening eventual consistency, thus allowing the applications to specify consistency rules, or *invariants*, that must be maintained by the system. Once those 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 [8] 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 aims at supporting updates with a choice of linearizable and eventual consistency. GEO [11] is an open-source geo-distributed actor[2] system that implements this approach. In GEO, an actor may be declared as *volatile* or *persistent*. In the first

[1] Read operations that return old instances of a data object that has already been updated to a newer value [43].

[2] Service applications may use actors to provide a programming model to simplify synchronization, fault-tolerance and scalability. It represents a useful abstraction for the middle tier of scalable service applications that run on a virtualized cloud infrastructure in a datacenter [11].

case, the latest version resides in memory and may be lost when servers fail, where in the second case the latest version resides in the storage layer. In case a user requests linearizability, GEO guarantees true linearizability by applying to the persistent storage in real time in between call and return.

The third 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 be concurrently delivered is set by the system administrators. Scalable Service Oriented Replication (SSOR) [18] 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.

Finally, the fourth approach aims at providing 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 [50] 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 [24]. 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.

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 are described next.

4.3.1 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 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) [13]. 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*) [46]. 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. Such scheme is based on a technique called *Provable Data Possession* [5], which is used to audit and validate the integrity and consistency of data stored on remote servers.

4.3.2 Consistency auditing

This kind of method is based on an architecture that consists 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) [42] 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 causal 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 [42].

5 Discussion

As proposed in Section 4, replica 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. They capture 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 which are well-understood concepts in distributed systems [29, 40, 45]. 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 at providing mechanisms that adjust the degree of consistency without any 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 predefined 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 remotely stored data in a potentially misbehaving service and need to rely on the CSP to guarantee their 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.

Table 1 presents a brief summary of the surveyed methods, whereas Table 2 summarizes the storage requirements supported by the systems that we have addressed 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.

As previously mentioned in Section 2.3, the existing trade-off determined by the CAP theorem implies that applications must sacrifice consistency to be able to satisfy other application requirements. Thus, Table 2 shows Availability and Partition Tolerance as storage requirements, which are related to the CAP theorem. In regard to the remaining requirements in Table 2, they are not directly related to the CAP theorem, but have some impact on the consistency methods (See Sect. 2.1).

In this context, Table 2 shows that PNUTS [21], Spanner [22], Clock-SI [25], DynamoDB [50] and Pileus [52] guarantee at the same time Availability and Partition Tolerance, thus providing a weaker type of consistency. Thus, these systems address the CAP's trade-off by sacrificing consistency.

PNUTS, Spanner and Clock-SI implement static consistency methods, so that the consistency criterion is not flexible, which means that the consistency criteria these systems adopt are not flexible. On the other hand, DynamoDB and Pileus implement dynamic consistency methods, thereby tuning the consistency level is not a problem under certain scenarios. For the remaining systems, consistency is not impacted by the CAP's trade-off. Vela [49], GEO [11], Harmony [19] and VFC [28] do not guarantee partition tolerance, which means that availability and consistency might be achieved. Indigo [8] does not guarantee availability, thereby achieving consistency and partition tolerance. Finally, SSOR [18] does not guarantee availability and partition tolerance, which means that its dynamic method concerns only in providing consistency.

Table 2 also shows that availability and scalability are the most addressed storage requirements supported by the surveyed systems, whereas elasticity is the least one. Although elasticity is not exploited by most of the

Table 1 Summary of the Surveyed Replica Consistency Methods

Category	Method	Brief Description
Static Consistency	Event Sequencing-based Consistency [21]	Establishes that all replicas of a given record apply all updates to a record in the same order.
	Clock-based Strict Consistency [22, 25, 49]	Uses clock-based mechanisms to control timestamps to enforce strict consistency.
Dynamic Consistency	Automated and Self-Adaptive Consistency [19, 28, 52]	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 [8, 11, 18, 50]	Allows applications to specify consistency rules, or *invariants*, that must be maintained by the system. Supports updates with a choice between linearizable and eventual consistency. 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 [13, 46]	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 [42]	Implements a Local and Global Auditing structure to allow a group of clients to detect the occurrence of consistency violations.

surveyed systems, this is a desirable and important feature for large scale applications, since it provides 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 the 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. By contrast, an elastic system can scale from 10 servers to 20 servers (or vice-versa) on-demand. Thus, we argue that it would be important to identify the challenges for supporting an elastic and consistent cloud data store.

6 Conclusions

In this survey we have reviewed several methods proposed in the literature to ensure replica consistency in distributed cloud data storage systems. 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 of 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 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 [13, 28, 42, 46], we highlight that some consistency methods consider very specific contexts [8, 18, 19, 21, 22, 25, 50, 52]. Moreover, very few of the proposed solutions [18, 19, 50] 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 many open issues to be exploited by other approaches.

Funding

This research was funded by the authors' individual grants from CAPES, CNPq, FAPEMIG and FAPERJ.

Availability of data and materials

This research was based on an extensive bibliography review and did not involved the use of any public or private dataset.

Table 2 Storage Requirements Supported by the Replica Consistency Methods

Category	Representative Systems	Storage Requirements [15, 38, 48]								
		Automation	Availability	Elasticity	Fault Tolerance	Low Latency	Partition Tolerance	Performance	Reliability	Scalability
Static Consistency Methods	PNUTS [21]		✓		✓		✓		✓	✓
	Spanner [22]		✓		✓	✓	✓			✓
	Clock-SI [25]		✓			✓	✓	✓		✓
	Vela [49]		✓		✓	✓		✓		✓
Dynamic Consistency Methods	Indigo [8]				✓	✓	✓			
	GEO [11]		✓		✓	✓		✓		✓
	SSOR [18]			✓	✓			✓	✓	✓
	Harmony [19]	✓	✓			✓		✓		
	VFC [28]	✓	✓			✓		✓	✓	✓
	DynamoDB [50]		✓	✓		✓	✓	✓	✓	✓
	Pileus [52]	✓	✓		✓		✓	✓	✓	✓
Consistency Monitoring Methods	VICOS [13]	not applied								
	CaaS [42]	not applied								
	DMR-PDP [46]	not applied								

Authors' contributions

The authors equally contributed to the elaboration of this survey.

Competing interests

The authors declare that they have no competing interests.

References

1. D. J. Abadi. Data Management in the Cloud: Limitations 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 *Proceedings of the International Conference on Database Systems for Advanced Applications*, pages 2–15, Hong Kong, China, 2011.
3. D. Agrawal, A. El Abbadi, and K. Salem. A taxonomy of partitioned replicated cloud-based database systems. *IEEE Data Engineering Bulletin.*, 38(1):4–9, 2015.
4. 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.
5. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable Data Possession at Untrusted Stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 598–609, Alexandria, VA, USA, 2007.
6. P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Probabilistically Bounded Staleness for Practical Partial Quorums. *Proceedings of the VLDB Endowment*, 5(8):776–787, 2012.
7. P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Quantifying Eventual Consistency with PBS. *Communications of the ACM*, 57(8):93–102, 2014.
8. V. Balegas, S. Duarte, C. Ferreira, R. Rodrigues, N. Preguiça, M. Najafzadeh, and M. Shapiro. Putting Consistency Back into Eventual Consistency. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 6:1–6:16, Bordeaux, France, 2015.
9. D. Beimborn, T. Miletzki, and D. I. S. Wenzel. Platform as a Service (PaaS). *Wirtschaftsinformatik*, 53(6):371–375, 2011.
10. D. Bermbach and J. Kuhlenkamp. Consistency in Distributed Storage Systems: An Overview of Models, Metrics and Measurement Approaches. In *Proceedings. of the First International Conference on Networked Systems*, pages 175–189. Marrakech, Morocco, 2013.
11. P. A. Bernstein, S. Burckhardt, S. Bykov, N. Crooks, J. M. Faleiro, G. Kliot, A. Kumbhare, M. R. Rahman, V. Shah, A. Szekeres, and J. Thelin. Geo-Distribution of Actor-Based Services. *Proceedings of the ACM on Programming Languages*, 1:107:1–107:26, 2017.
12. 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.
13. 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.
14. M. Bravo, N. Diegues, J. Zeng, P. Romano, and L. Rodrigues. On the use of clocks to enforce consistency in the cloud. *IEEE Data Engineering Bulletin*, 38(1):18–31, 2015.
15. E. A. Brewer. Towards Robust Distributed Systems. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*, pages 7–14, Portland, Oregon, USA, 2000.

16. Eric Brewer. Pushing the CAP: Strategies for Consistency and Availability. *IEEE Computer*, 45(2):23–29, 2012.
17. 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.
18. 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.
19. H. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Perez. Harmony: Towards Automated Self-adaptive Consistency in Cloud Storage. In *Proceedings of the 2012 IEEE International Conference on Cluster Computing*, pages 293–301, Beijing, China, 2012.
20. H. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Pérez. Consistency management in cloud storage systems. In M. Gaber S. Sakr, editor, *Large Scale and Big Data: Processing and Management*. CRC Press, 2014.
21. 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. *Proceedings of the VLDB Endowment*, 1(2):1277–1288, 2008.
22. J. C. Corbett, J. Dean, and M. Epstein et al. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems*, 31(3):8:1–8:22, 2013.
23. 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 *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research*, pages 235–240, Asilomar, CA, USA, 2011.
24. 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 Operating Systems Review*, 41(6):205–220, 2007.
25. J. Du, S. Elnikety, and W. Zwaenepoel. Clock-SI: Snapshot Isolation for Partitioned Data Stores Using Loosely Synchronized Clocks. In *Proceedings of the IEEE 32nd International Symposium on Reliable Distributed Systems*, pages 173–184, Braga, Portugal, 2013.
26. A. Dubey and D. Wagle. Delivering software as a service. *The McKinsey Quarterly*, 6:1–7, 2007.
27. M. M. Elbushra and J. Lindström. Eventual Consistent Databases: State of the art. *Open Journal of Databases*, 1(1):26–41, 2014.
28. S. Esteves, J. Silva, and L. Veiga. Quality-of-service for Consistency of Data Geo-replication in Cloud Computing. In *Proceedings of the 18th European Conference on Parallel Processing*, pages 285–297, Rhodes Island, Greece, 2012.
29. C. Fidge. Logical Time in Distributed Computing Systems. *IEEE Computer*, 24(8):28–33, 1991.
30. A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based Scalable Network Services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pages 78–91, New York, NY, USA, 1997.
31. S. Ghemawat, H. Gobioff, and S. Leung. The Google File System. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003.
32. S. Gilbert and N. Lynch. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services. *ACM SIGACT News*, 33(2):51–59, 2002.
33. S. Gilbert and N. Lynch. Perspectives on the CAP theorem. *IEEE Computer*, 45(2):30–36, 2012.
34. S. Goel and R. Buyya. Data replication strategies in wide-area distributed systems. In R.G. Qiu, editor, *Enterprise Service Computing: From Concept to Deployment*, pages 211–241. IGI Global, 2007.
35. Jim Gray. Notes on Data Base Operating Systems. In *Operating Systems, An Advanced Course*, pages 393–481, London, UK, UK, 1978. Springer-Verlag.
36. T. Haerder and A. Reuter. Principles of Transaction-oriented Database Recovery. *ACM Computing Surveys*, 15(4):287–317, 1983.
37. S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu. Maestro: Replica-Aware Map Scheduling for MapReduce. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 435–442, Ottawa, Canada, 2012.
38. J. Ju, J. Wu, J. Fu, Z. Lin, and J. Zhang. A Survey on Cloud Storage. *Journal of Computers*, 6(8):1764–1771, 2011.
39. A. Lakshman and P. Malik. Cassandra: A Decentralized Structured Storage System. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.
40. L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978.
41. R. J. Lipton and J. S. Sandberg. *PRAM: A Scalable Shared Memory*. Tecnical Report TR-180-88, Department of Computer Science, Princeton University, 1988.
42. 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.
43. H. Lu, K. Veeraraghavan, P. Ajoux, J. Hunt, Y. J. Song, W. Tobagus, S. Kumar, and W. Lloyd. Existential Consistency: Measuring and Understanding Consistency at Facebook. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 295–310, 2015.
44. 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.
45. F. Mattern. Virtual Time and Global States of Distributed Systems. *Parallel and Distributed Algorithms*, 1(23):215–226, 1989.
46. R. Mukundan, S. Madria, and M. Linderman. Replicated Data Integrity Verification in Cloud. *IEEE Data Engineering Bulletin*, 35(4):55–64, 2012.
47. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.
48. B. P. Rimal, E. Choi, and I. Lumb. A Taxonomy and Survey of Cloud Computing Systems. In *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, pages 44–51, Seoul, Korea, 2009.
49. T. Salomie and G. Alonso. Scaling off-the-shelf databases with vela: An approach based on virtualization and replication. *IEEE Data Engineering Bulletin*, 38(1):58–72, 2015.
50. S. Sivasubramanian. Amazon DynamoDB: A Seamlessly Scalable Non-relational Database Service. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 729–730, Scottsdale, Arizona, USA, 2012.
51. A. S. Tanenbaum and M. V. Steen. *Distributed Systems: Principles and Paradigms*. Prentice-Hall, 2007.
52. D. B. Terry, V. Prabhakaran, and R. Kotla et al. Consistency-based Service Level Agreements for Cloud

Storage. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 309–324, New York, NY, USA, 2013.
53. P. Viotti and M. Vukolić. Consistency in Non-transactional Distributed Storage Systems. *ACM Computing Surveys*, 49(1):19, 2016.
54. W. Vogels. Eventually Consistent. *Communications of the ACM*, 52(1):40–44, 2009.
55. Z. Wei, G. Pierre, and C. Chi-Hung. Scalable Transactions for Web Applications in the Cloud. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, pages 442–453, Delft, The Netherlands, 2009.
56. P. Xiong, Y. Chi, and S. Zhu et al. Intelligent Management of Virtualized Resources for Database Systems in Cloud Environment. In *Proceedings of the 27th International Conference on Data Engineering*, pages 87–98, Washington, DC, USA, 2011.