International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 11, Issue 10, October 2022 ISSN 2319 – 4847

Handling End-to-end data consistency in a cloud-based sync framework for data object access and management

Aslam B Nandyal[1], Mohammed Rafi[2]

[1]DOS, Computer Science & Engg, University BDT College of Engineering, Davanagere, Karnataka, India

[2]DOS, Computer Science & Engg, University BDT College of Engineering, Davanagere, Karnataka, India

Abstract

The majority of mobile enterprise applications are designed to work with preexisting backend enterprise systems, and certain usage scenarios call for the storage of large files on mobile devices. In a wireless mobile environment, issues with latency, bandwidth, speed, mistakes, and service disruptions make it challenging for Mobile Application Developers to upload and retrieve huge files to and from a mobile app. Some situations need that these files be utilized offline, updated by a single user, and shared with all other users. Universally available services rely on replication technique to provide fault tolerance, high data availability, and rapid response times. The term "consistency" relates to the concept that the state of the data is determined in some manner by the varied clients that use a storage system. This could be the most current status of the data or the collection of modifications that led to that state. Two factors contribute to the difficulty of maintaining consistency: To begin with, storage systems need to store a large number of copies of their data in case of failure or performance issues. Second, storage operations can be performed on a large number of data items or objects. This paper proposes an improved Mobile Sync framework (known as LOSMob) to improve end-to-end data consistency in mobile cloud environments, with three distinct consistency plans (Strong, Causal, and Eventual) and synchronization of both table and object data. The framework is evaluated based on the end-to-end latency (in seconds) for a particular size object under three different consistency schemes (Strong, Causal, and Eventual) in a predetermined setting.

Keywords: Mobile cloud computing, Mobile backend as a Service, Large files, Distributed systems.

1. INTRODUCTION

Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service (PaaS) are the most common cloud computing service paradigms[1]. Mobile Backend as a Service is a new type of service that facilitates the integration of cloud-based apps with mobile platforms (MBaaS). The Sync framework model can be summarized by the following five characteristics [2]–[4]:

- Enable non-blocking, responsive (high availability), and reliable mobile applications when the internet is unavailable.
- Apps that need to handle interdependent data locally and across numerous devices with cloud storage should be supported by the cloud for multi-user, shared data mobile apps.
- Allow developers to customize the synchronization mechanism and how data conflicts are handled by providing a synchronization model with customizable consistency guarantees.
- Provide high-level APIs that are aware of synchronization and can be used by programs to perform on demand or background synchronization operations.
- Mobile clients require efficient use of power and bandwidth, with sync processes that can be set to run at regular intervals, bidirectional protocol for exchanging logs.

The key to providing fault-tolerant, highly available, and quickly responsive universally available services is replication. To replicate something means to create multiple identical copies of it on different computers, either in a local network or across the distributed network[5] , [6]. To make sure the workload is spread out among the most possible replicas, copies of these multiple objects (replicas) are persistently kept over time. The replication mechanism employed by a given distributed system is subject to change based on the system's individual requirements for ensuring data consistency over time. There are certain programs that only need to read data, while there are others that require a large amount of writing (updates). Unlike banking systems, which require data to be consistent over time, some social networks may be willing to accept stale information.

"Consistency" means that different clients accessing a storage system determine the state of data, such as the most current status or the collection of alterations that led to that state. It’s challenging to provide consistency for two reasons: First, storage systems must keep many replicas of fault tolerance and performance data. Second storage operations can be performed on a large number of data items or objects.

This paper is constructed as follows: Section 2 explain two principle viewpoints on consistency, the data-centric and client-centric consistency. Section 3 explains the bench-marking and evaluating techniques for data consistency solutions, followed by brief description of Distributed databases which are important for the design of Sync frameworks in Section 4. Section 5 provides the details of consistency measurement method with architecture and Sequence Diagram of different client apps involved during measuring the consistency schemes in proposed LOSMob framework. Related work regarding the data synchronization and consistency support for different framework in literature is discussed in Section 6. Section 7 discussed the implementation details to support the different consistency schemes n proposed LOSMob Sync framework. The effectiveness of the proposed Mobile Sync framework is discussed in Sections 8 and 9, and a conclusion is provided in Section 10.

2. CONSISTENCY MODELS IN SYNC FRAMEWORKS

The literature primarily describes two basic perspectives on consistency: data-centric consistency and client-centric consistency. Both of these perspectives are presented below. [7] [8].

a. **Data- centric models or Server-side consistency models**

The data-centric consistency management system is responsible for managing the specifics of the system's internal state. It does this by ensuring that all system replicas are exactly the same and that the system maintains its consistency throughout the update process. When it comes to system development, data-centric consistency is critical. Data-centric or Server-side consistency models include the following:

i. **Strong consistency** - One of the most important characteristics of the strong consistency model is that it maintains a consistent state all of the time. As specified by the CAP theorem, a single-copy consistency model such as strong consistency is not suited for mobile apps working with cloud data because of concerns with availability and performance [9].
ii. **Causal consistency (CC)** - When implemented, Causal Consistency (CC) ensures that all requests with a causal relationship execute in the same order across all copies. Requests that aren't related to the ones that came before them are handled at random.
iii. **Sequential consistency (SC)** - The requirement of sequential consistency (SC) is that all sequentially linked requests are processed in the same order. This is a less stringent type of strong consistency. In this way, all clients will view the same updates at the same time.
iv. **Grouping operations** - This concept is for dealing with scenarios when a sequence of read and write operations are required. To increase the granularity of numerous reads and writes into one atomically executed unit, the Grouping Operation model can be used.

b. **Client- centric models or Client-side consistency models**

To put it another way, application developers put a premium on client-centric reliability because client-centric consistency is preoccupied with black-box monitoring of data updates. Client-centric or client-side consistency models include the following:

i. **Weak consistency -** Unfortunately, this approach does not ensure that subsequent queries will always return the correct, up-to-date information. 'Inconsistency window' refers to the interval between an update and the instant at which all observers will always view the modified value [8].
ii. **Eventual consistency (EC)** - When no new changes are made to an object, eventual consistency (EC) is considered to be a variant of Weak consistency with the additional promise that, ultimately, all replicas will receive the last modification made to an object. The following four main ordering guarantees are provided by eventual consistency [10]:

1. In Monotonic read consistency (**MRC**), when a client reads an object’s version number ’n’ and then reads it again, the client will never access a lower version of the object.
2. In Read your writes consistency (**RYWC**), After writing version 'n', the same client will never read an older version less than 'n'. This is a one-of-a-kind of causal consistency model [7] [8].
3. In Monotonic write consistency (**MWC**), Under the monotonic write consistency (MWC) model, all changes from the same client are recorded in the sequence in which they were received. It ensures that any write operation on the same data item model is always terminated before any subsequent write operation[8].
4. The Write Follows Read Consistency (WFRC) model guarantees that an update that occurs after reading version "x" will never be applied to replicas that were created before version "x."[7].

For the system to attain client-centric consistency, the studies [11] [12] a conclusion can be drawn from the fact that all four client-centered models MRC, RYWC, MWC, and WFRC) must be guaranteed.

3. CONSISTENCY BENCHMARKING AND 3D DESIGN FRAMEWORK

The benchmarking techniques to estimate staleness or to compute violations of (MRC, RYWC etc.) is found in the literature [13] [14]. Application frameworks use latency as a benchmarking factor to measure the influence of high-level abstraction for effective mobile application data synchronization. [15] [16]. Three deviation metrics have been identified in the literature [7] as being appropriate for mobile contexts when it comes to consistency models. 3D Design Framework [17] describes this categorization [18]. The three axes of the 3D Design Framework are formed by three different metric parameters:

a. The difference in the number of updates applied to replicas is referred to as **Numerical Deviation**.
b. **Order deviation**: The variation in the sequence in which operations are carried out between replicas of a given object.
c. The time it takes for replicas to see an update is referred as **Staleness**.

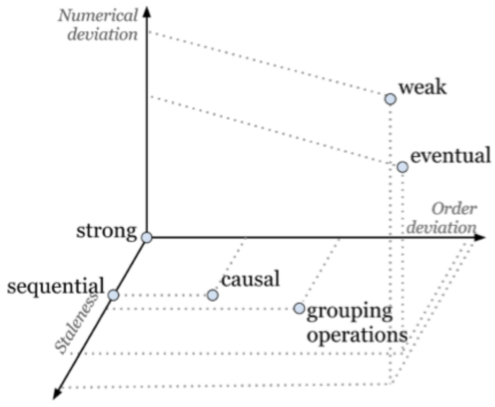

Fig 1. 3D Design framework classification

The actual variances may vary based on the system’s tolerance for irregularity. Figure 1 shows the classification of the 3D Design Framework as a result of applying the metrics to distinct consistency models (Data centric and Client-centric models)[17]. Based on the allowed quantity of discrepancies, a suitable consistency model can be constructed. Concrete values are not included on the axes because they are dependent on the framework. The system’s nature also influences the allowable level of inconsistency.

The 3D design framework serves as a foundation for developing, testing, and evaluating data consistency solutions in mobile wireless contexts. Because the three dimensions are separate, combining each attribute can result in a wide range of consistency maintenance systems. In this study, the suggested LOSMob framework is an application framework that is utilized as a pre-built backend to quickly and affordably prototype data-centric mobile applications that manage huge files. Therefore, the purpose of this research is to examine the effect of abstraction levels on the effective synchronization of large amounts of data in mobile applications. [15] [16].

4. DISTRIBUTED DATABASES

Backend as a Service, also known as BaaS, is a type of service that makes it easy for application developers to store data in the cloud. The supported interface can be queried by developers to access their data. The architecture of the BaaS Provider controls how data is stored, replicated, and partitioned internally. Scalability, availability, consistency, and adaptability is all affected by these metrics. The replication and synchronization algorithms of frameworks for cloud data management for various mobile, online, and desktop platforms are also influenced by the data store's design. Because frameworks are constructed on top of data stores, it is critical to have a thorough understanding of data stores in distributed systems.

Dynamo [5] is a high-availability key-value storage system that trades off consistency for more availability in the event of a network breakdown. Using object versioning and application assisted conflict resolution, it provides a "always-on" user experience. Nodes that adhere to ring infrastructure are part of the data distribution mechanism. Consistent hashing is used for dataset partitioning to reduce the number of ownership shifts [19]. The object is allocated a coordinator node and then replicated at N of its successors for data replication. Asynchronous propagation of updates to all replicas leads to eventual consistency. The use of vector clocks for object versioning, concurrency management, and application-level conflict resolution is facilitated by vector clocks.

Dynamo storage uses partial replication and offers eventual consistency. Appendix provides a summary of the sync reference frameworks' consistency and data structure (table, object, or both) support: Table II.

Another open source multi-master, key-value cloud datastore is Cassandra [20]. Cassandra distributes data over several nodes for redundancy and resilience against failure. The location of the replicas is determined by the replication strategy. Replication factor or replication level refers to the total number of cluster replicas. Each row in the cluster has exactly one copy if the replication factor is one. Each row in the cluster has exactly one copy if the replication factor is one. A replication factor of two indicates that each row has two copies, each of which is located on a separate node in the network. No primary or master replica exists; all of the replicas are equally valuable. As a general rule, it is best to keep the replication factor at a level that does not exceed the number of cluster nodes. However, before adding the appropriate number of nodes, the replication factor can be adjusted. Cassandra clusters are fixed in size (e.g., eight nodes) and use three-way replication (replication level3), whereas clients use the QUORUM consistency level. Because transactions aren't supported in Apache Cassandra, it's necessary to use specialized read and write procedures to get at the objects. Operations can be run under one of three different Consistency Levels: One (Quorum), One (All), and One (Quorum). Cassandra uses the same replication method as Dynamo.

As a result of the Eventual Consistency concept[10] , Cassandra provides a weak notion of consistency, which states that all replicas of a data element will have the same value in the end if no further changes are communicated to that data element. In addition, it provides a significant level of uniformity on every read, but the decision must be made system-wide. Amazon S3 and OpenStack Swift [21], both object-based storage frameworks, are popular choices for large-scale storage because of their nearly limitless scalability and low cost.

The Simba [2] server implementation stores tabular data in Cassandra [20] and object data in Open Stack Swift (Open- Stack Swift Object Storage Service, 2018). For high availability, Simba sets up Cassandra and Swift to employ threeway replication. Additionally, the methodology described in [16] allows for the use of Cassandra as the backend datastore. A recent sync framework NetMob[22] also stores table and object data in Cassandra and the Open Stack Swift object store. In addition, the suggested framework in this research (LOSMob) also makes use of Cassandra for table data storage and Open Stack Swift for object data storage. Three-way or three-level Optimistic replication is used in the proposed framework (LOSMob) to ensure high availability [23].

Data can be stored in a key-value format in a NoSQL open-source database called Basho Riak [24]. Map/Reduce operations [25], HTTP, JSON, and REST queries are all supported by this fault-tolerant, highly scalable database. Data is automatically copied to several nodes using the Riak ring, which consists of identical nodes for replication. The data (both existent and newly created) is automatically distributed among the nodes. There are no bottlenecks or single points of failure because every node in the cluster is identical. Each object has a version vector linked with it so that conflicts can be addressed. It's possible to configure Riak to employ the "lastwriter- wins" consistency policy, or else return both versions for the application to resolve the conflicts. For semantics in which the last writer prevails, the most recent update (as ordered by the wall clock) takes precedence over all other updates (Lloyd et al., 2011).

Additionally, Basho Riak ensures convergent causal consistency with transactions in addition to the last-writer-wins policy. An implementation of a Key-CRDT using SwiftCloud [26] uses the Basho Riak distributed database. CRDTs are asynchronous data types for which updates do not require synchronization. ([27] [28] [29]). It is possible to utilize them to construct alternative data models required by applications because CRDTs are compliant with the Strong Eventual Consistency Model [29]. For eventual consistency contexts, the asynchronous nature of CRDTs is ideal.

Key-Value stores such as Yahoo! PNUTS [30] allow users to choose between eventual and strong consistency for each read. In order to keep write latency low while providing geographic replication, it uses a publish-subscribe mechanism called Yahoo Message Broker (YMB). This allows for asynchronous replication and low write latency. Fully replicated PNUTS systems are created by separating the system into regions, with each zone replicating a copy of every table. Using a per-record timeline consistency paradigm, PNUTS ensures conflict-free access to data. Per-record timeline consistency ensures that all records are updated in the same sequence, regardless of where they are stored. PNUTS utilizes a single master for all writes, with read-only mirror replicas also present. As a result of the operations being performed in a sequential order, no conflicting replicas have been created.

For each record, PNUTS supports a per-record timeline consistency model and provides a choice of eventual or strong consistency. In order to offer cloud-enabled data replication and messaging mechanism, Mobius[3] uses PNUTS as the back-end storage. Table Consistency refers to the MUD (Messaging Unified with Data) table, which represents the programming abstraction of a logical data table across devices and clouds [3].

MongoDB [31] is a NoSQL document database that enables horizontal scaling of massive amounts of data. Initially, all data is stored on a single server, which is then replicated. When the application develops in size and the amount of traffic increases, MongoDB changes to a shared cluster. This improves the system’s failure tolerance and availability. Replica Sets is the name of the replication strategy employed by MongoDB. MongoDB is a database that combines strongly consistent writing with eventual consistency for readers [31]. A single master node is responsible for all writing operations. According to the CAP Theorem, despite the fact that this gives high consistency, it decreases availability (Gilbert Lynch, 2002). Clients can use slave nodes to perform read operations. Read operations are eventually consistent because updates must propagate from master to slave. Monotonic Writes and Write-follow-reads, on the other hand, can both be offered.

MongoDB blends strongly consistent writing with eventually consistent reads to provide a highly reliable database ([31]. There is also support for Monotonic Writes and Writefollow- reads (WFR). An abstraction of the MongoDB database is provided by the Parse Server [32] Query mechanism to read, write, and update database entries. The application data in Kinvey [33] is stored in a MongoDB instance.

The key goals of Apache CouchDB [34] are to create a ”distributed database system with built-in bidirectional replication, providing fault tolerance, high scalability, and offline access,” according to the CouchDB documentation. Clustering is one of CouchDB’s scalability capabilities, and it allows to divide the data into separate clusters. CouchDB’s clustering feature is similar to MongoDB’s sharding feature. CouchDB Loung is the name of the clustering program developed for CouchDB. CouchDB is a database that uses eventual consistency. As a result of this policy, all nodes in the system are master nodes, which means that all nodes are capable of completing write operations. In CouchDB, replication is done incrementally. This means that all nodes receive updates on a regular basis. There isn’t a single point of failure in CouchDB because it contains numerous master nodes.

5. PROCEDURE OF CONSISTENCY MEASUREMENT

Using the 3D design framework outlined in Section 3 [17] [18], data consistency challenges in mobile wireless contexts can be designed, studied, and evaluated. There are a large variety of systems that can be constructed by combining the properties of the three dimensions. Using the LOSMob framework suggested in this research, it is possible to quickly and cheaply prototype data-centric mobile applications that can manage data.

LOSMob data stores provide less robust consistency qualities, such as eventual consistency. Additionally, as illustrated in Table 1, LOSMob supports Strong and Causal Consistency schemes. It is possible for a LOSMob framework client app to notice values that are stale, meaning that they are not from the most recent write. This design characteristic is due to the fact that a distributed system with partition tolerance may only guarantee one of two properties: data consistency or availability, as defined by the CAP theorem [9]. In order to evaluate the impact of high-level abstraction on efficient sync of mobile application data, Latency is a benchmarking factor for application frameworks (Bermbach, Kuhlenkamp, Derre, Klems, Tai, 2013). It is feasible to give inconsistent attributes such as read-your-writing, monotonic reading, or session settings. However this can alter the collection of conceivable scenarios and the code that is developed to manage and test various consistency models [14].

In an application developer's perspective, LOSMob maintains a set of replicated key-value tables. There is no limit to the number of tables that can be used to store application specific data. A single data object or a collection of data objects can be stored in a table. Each data object is made up of a unique key, which is a string value, and an opaque value, which is a byte sequence. In addition to each table having a unique global name, each server has its own setup and replication policies, which most programs can't see.

a. **Strong:** A Get(key) returns the value already stored by any client in an earlier Put(key).
b. **Eventual:** When a Get(key) is called, it returns the value written by any Put(key), which is to say, any version of the object associated with the key that was specified. If no other Puts are made, the latest version will eventually be returned, but there are no guarantees as to how long this might take.
c. **Monotonic:** A Get(key) returns the same or a later version of a previous Get(key) in this session; if the session does not already have the Gets, the Get may return the value of any Put (key).
d. **Read-my-writes**: In this scenario Get(key) is implemented in an eventual consistent manner. Thus, Get(key) returns the value written by the previous Put(key) in this session, or an earlier version if no Puts were executed in this session, in an eventual consistent manner.
e. **Causal:** Depending on what happened before the key was put, Get(key) will return either the value from the most recent Put(key) or a later version of that value.. If any of the following three conditions are met, the causal precedence relation '¡' is defined such that op1¡op2 is true: (1) op1 occurs earlier in the same session than op2, and (2) In this case, op1 is Put(key), and op2 is Get(key), which always returns the same version as op1 and (3) there exists a relationship for some value of op3 such that op1¡op3 and op3¡op2.

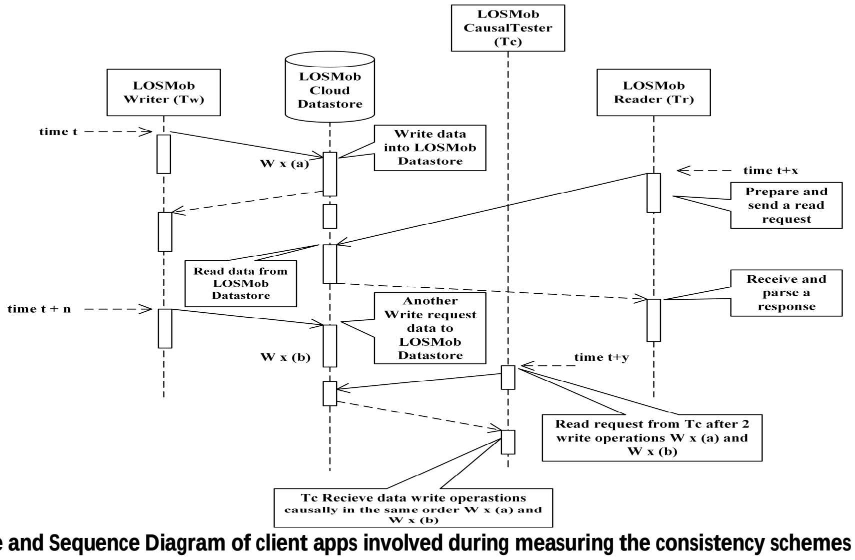
LOSMob CausalTester (Tc)
LOSMob Writer (Tw)
LOSMob Cloud Datastore
LOSMob Reader (Tr)
time t
Write data into LOSMob Datastore
W x (a)
time t+x
Prepare and send a read request
Read data from LOSMob Datastore
Receive and parse a response
time t + n
Another Write request data to LOSMob Datastore
W x (b)
time t+y
Read request from Tc after 2 write operations W x (a) and W x (b)
Tc Recieve data write operastions causally in the same order W x (a) and W x (b)

Fig 2. Architecture and Sequence Diagram of client apps involved during measuring the consistency schemes in LOSMob.

The architecture of the benchmark test applications used in this research to measure consistency parameters and investigate the likelihood of a client observing stale data in cloud data storage is illustrated in Figure 3. [14] [35]. Read and write latencies, as well as the latency experienced by LOSMob client apps, and the latency experienced when syncing data with CloudLM, are all quantified by these four cloud-deployed roles: (1) Cloud data store CloudLM, and three mobile computation clients, (2) Reader (Tr), (3) Writer (Tw), and (4) CausalTester (Tc). The contents of the LOSMob data store are read by a LOSMob Custom Reader (Tr), which records the read operation's time stamp. During the assessment experiment, the reading procedure occurs for a specific number of times (for example four times) for every specific period (for example every second).

The Writer (Tw) uses the LOSMob data APIs to repeatedly write specific amounts of test data (100 MiB) to the LOSMob datastore. Because the written value corresponds to the current time, it is easier to determine which write was observed in a read operation. In the evaluation experiment, a writing operation occurs for every specific period (for example every four or five seconds). It is possible to utilize a single thread to implement the Writer role, one or more threads to implement the Reader role in a single configuration, or a single thread to implement both the Reader and Writer roles. Both of these configurations are conceivable..

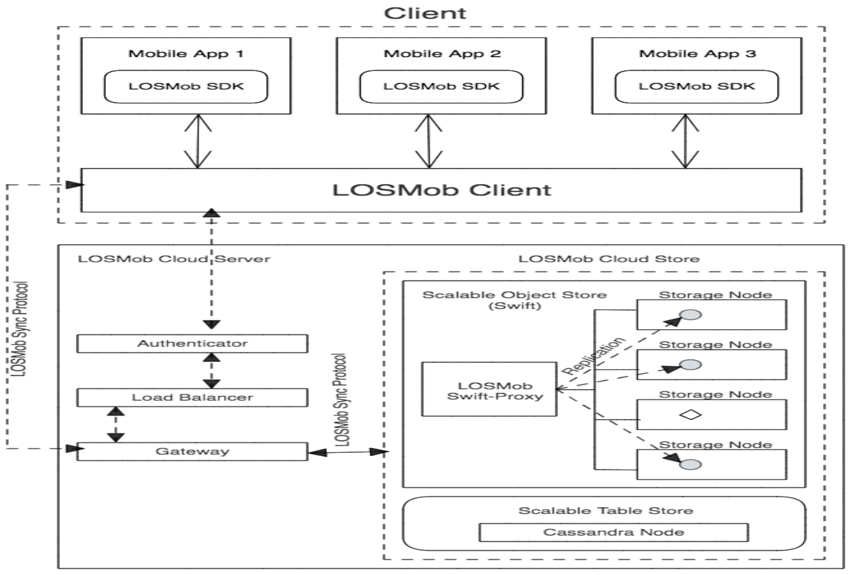

Fig 3. Experimental Setup for measuring the consistency schemes in LOSMob.

A similar process occurs when the CausalTester (Tc) client reads the contents of the LOSMob datastore, which were written in a specific order. For example two writers clients Wx(a) and Wx(b) write to the datastore at time x(a) and x(b) as shown in Figure 3. The CausalTester (Tc) notifies the writer client of the time stamp at which the read takes place. The client Tc must write to a row with the same row-key as Tw before Tw can perform a write operation.

This evaluation procedure uses a series of measurements to calculate average values. Specifically, one measurement of the experiment in this research is configured to running the writing process and reading process for 4 minutes, by doing 50 write operations and an appropriate number of read operations (250- 500).

Latency will be calculated in the post-processing data analysis step by determining how long it has been since the last writing operation based on the write and read time stamps (Tr and Tc). Finally, the values are expressed as the latency of a single measurement run or, alternatively, as an average of several measurement runs. The evaluation consisted of repeating measurements ten times and then aggregating the results.

The evaluation experiment uses the following metrics to measure the results of this study:

1. Write latency, or the time it takes for an app to perceive an update to Write (Tw).
2. The term "Sync" refers to the delay in synchronization updates that occurs between a Writer (Tw) and a Reader (Tr).
3. The time it takes for the app to read newly updated data from Reader (Tr) is known as the Read latency.

6. RELATED WORK

Mobile client-server computing and mobile data access are examined in [36] together with a thorough analysis of the first working models from the field of study (Bayou [37], Odyssey [38], and Rover [39])) for mobile data management. Extending this work, the detailed analysis of data synchronization and consistency support for the latest framework, such as handling offline data, data consistency, data replication, and synchronizing strategy, is presented in [4].

This survey talks about how data is shared and how mobile devices can get consistent data. [40]. Several contributions to mobile transaction models are compared and analyzed in the paper [41]. The researchers carried out a comprehensive literature review on synchronization between mobile devices and server databases in [42]. An in-depth analysis of cloud data replication methods is provided in the article [6]. In a review article, different types of data synchronization algorithms are compared based on their ability to scale, their consistency, and their accuracy in a ubiquitous network [43].

In the paper [44], the five most prominent NoSQL databases—Redis, MongoDB, Cassandra, Neo4j, and OrientDB—are compared and evaluated based on their consistency implementation.

Per-record sequential [45] and fork-sequential [46] consistency are provided by Mobius [3] through the use of exclusive read operations. Consistency can be specified on a row-by-row, request-by-request, or table-by-table basis.

With the help of cloud-enabled programming interfaces, mobile apps may save data copies on users' devices and synchronize them with cloud servers, such as SwiftCloud [26] and Cloud types [47].

Pileus [35] supports the three most commonly used consistency schemes. These three consistency schemes are also supported by Simba[2] , NetMob [22] as well as the framework proposed in this article (LOSMob [48]).

The early proposal to support large files and consistency in a Sync framework is provided by [49]. The frameworks NetMob [22] and LOSMob [48] deal with dedicated support for large file and consistency.

An alternative method for building consistency frameworks is known as Consistency As Logical Monotonicity (CALM). In the absence of coordination protocols, the CALM theorem ensures that all logically monotonic programmes are eventually consistent (distributed locks, two-phase commit, paxos, etc.). As a result, the CALM approach guarantees eventual consistency by necessitating the use of monotonic logic [50]. The CALM analysis of logic languages such as Bloom [51] can be used to tell if the code flow alone is enough to guarantee consistency, or if the integrated coordination protocols are also required. [50].

Developers can take advantage of data storage abstractions provided by mobile operating systems. CloudKit is a new Apple feature that allows programmes to store and retrieve data from the iCloud service. [52]. There are open source choices for mobile back-end-as-a-service, such as the Parse Server framework[32] and StackSync [53].

Commercial services like IBM Bluemix Mobile Cloud Service [54] exist to help connect mobile and web apps to cloud storage and consistency support services [55]. App developers can also benefit from the other commercial services provided by Firebase [56] and Kinvey [33]in order to connect their apps to cloud backends.

The current paper focus on end-to-end data consistency for large file object by LOSMob framework in mobile cloud environment. Previous article based on LOSMob [48] was focused on large file handling support feature of this framework, specifically on mobile data management method based on object segmentation and object chunking in order to improve large file object access and synchronization in a mobile cloud environment.

7. METHOD OF HANDLING CONSISTENCY IN LOSMOB

On the client side, LOSMob uses SQLite for tables and LevelDB for objects. On the mobile side, LOSMob uses LevelDB to manage large files. In order to support large working loads, the LevelDB makes use of the concepts Sorted String Table (LSM) and Sorted String Table (SST).

A data model provided by LOSMob is known as LOSMob Table (TableLM for short), and it supports tables as well as objects. RowLM is the name given to a single TableLM row. When working with multiple ClientNM, TableLM, and LOSMob applications, the LOSMob Cloud Server (also referred to as CloudLM) is in charge of keeping track of the data. With three distinct consistency plans and TableLM synchronization, CloudLM enables a tunable storage consistency system.

To accommodate the wide variety of sync needs among applications, LOSMob allows developers to set their own per-table sync rules using sync methods (register - writeSync- Subscribe etc). Each RowLM can, therefore, hold the data for the table and objects that it is linked to, with the configurable distributed consistency for the whole table. The consistency schemes offered by LOSMob are listed in Table 1.

Client - Operation	**Causal consistency**	**Strong consistency**	**Eventual consistency**
Permission for Offline write	Yes	No	Yes
Permission Offline read	Yes	Yes	Yes
Require Conflict resolution	Yes	No	No

Table 1. Consistency schemes in LOSMob.

Cassandra is used to store table data and Open Stack Swift is used to store object data on the LOSMob data storage server. LOSMob configures Cassandra and OpenStack Swift for three-way replication (replication level 3) to achieve high availability.

For object storage, LOSMob relies on OpenStack Swift's built-in support for Eventual consistency. A new object is created and then deleted after the updated RowLM is committed by StoreLM in order to ensure Strong consistency. In order to ensure exclusive write access for updates, StoreLM attaches a read/write lock to each TableLM. This allows for multiple threads to read and write simultaneously.

8. EXPERIMENTAL SETUP

As shown in the experimental setup diagram (Fig.3)The replication factor or replication level refers to the total number of replicas across the cluster. Openstack Swift environment with replication level of three, as depicted in the setup diagram.

9. CONSISTENCY MEASUREMENT IN LOSMOB

Two Xiaomi RedMi 5 Plus handsets with 6GB of RAM and Android 10 were used to verify the LOSMob consistency measuring setup. LOSMob client applications and the CloudLM data sync latency will be measured using a 100 MiB file as the write payload. The notification time of one second was used in the evaluation method to assess the eventual and causal consistency. It is essential that all updates are completed prior to the end of this
time period.

Both the Eventual and the Strong consistency models are supported by Cassandra. Cassandra is set up to use three-way replication for high availability. To further guarantee great consistency when reading, Casandra is set up with ReadConsistency=ONE. This setting indicates the instant response from the consistent copy. To provide strong consistency while updating the commit log and memtable, Casandra will be configured with WriteConsistency=ALL, instructing the cluster to write to all replica nodes in the cluster for that partition.

StoreLM generates a new object and then deletes the old one when the modified RowLM has been properly committed, ensuring strong consistency despite OpenStack Swift's default support for eventual consistency. In order to ensure exclusive write access for updates, StoreLM attaches a read/write lock to each TableLM. This allows for multiple threads to read and write simultaneously.

Using three different mobile test clients, Reader (Tr), Writer (Tw), and CausalTester (Tc), LOSMob's consistency parameters are measured. It is the responsibility of Tc to ensure that it writes to a row with the same row-key as Tw before Tw's write operation can be carried out.

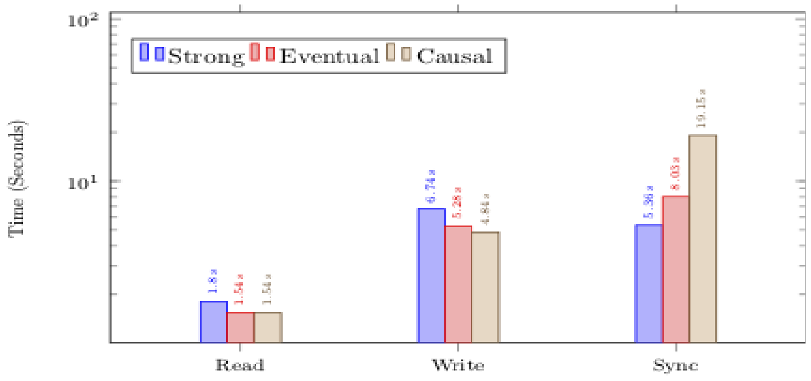

Fig 4. End-to-end latency of 100 MiB object for different consistency schemes in LOSMob.

Figures 4 and 5 show the WiFi latency and data transfer associated. Figure 4 shows the following metrics:

1) Write latency, or the time it takes for an app to perceive an update to Write (Tw).
2) The term "Sync" refers to the delay in updating that occurs between a Writer (Tw) and a Reader (Tr).
3) The time it takes for the app to read newly updated data from Reader (Tr) is known as the Read latency.

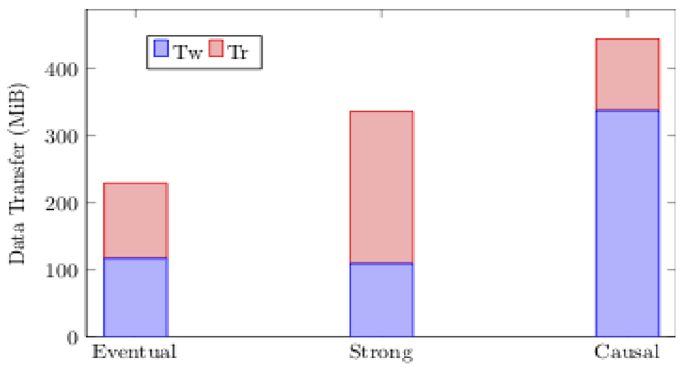

Fig 5. Data transfer for 100 MiB object in different consistency schemes of LOSMob.

Figure 5 illustrates the total amount of data that is exchanged between the Writer (Tw) and Reader (Tr) clients for each consistency method. According to the rule of the last write, there is less data transmission since only the most recent version is required for eventual consistency (see Figure 5). Strong consistency necessitates immediate propagation of all updates, resulting in a higher data transfer, but with the lowest possible sync latency. In the event of a conflict, Causal has a longer synchronization latency than Eventual because it necessitates more round-trip times to resolve the conflict. Because the Writer's (Tw) first attempt at syncing fails, the Writer (Tw) must read the conflicting data from the Causal Tester (Tc) before attempting to update again, resulting in a larger-than-usual data transfer for conflict resolution with Causal. As long as there are no conflicts, synchronization latency and data transfer for Causal and Eventual are the same (not depicted).

The above graph indicates that LOSMob supports three consistency schemes: Strong, Causal, and Eventual (See in TableI), and the results demonstrated the desired behavior and data transfer in each consistency scheme, as described below.

Strong consistency has a Write latency of 6.74 seconds. Additionally, this technique demands all updates to propagate promptly, which results in a greater data transmission rate (more than 300 MiB). Also this scheme has the shortest sync latency (5.36 seconds) because data is synced immediately.

A single sync process (8.03 seconds) results in a smaller data transfer (approximately 210 MiB) in the case of Eventual Consistency, which requires reading only the most recent version.

Since reads are always performed in the same location, there is little variation in read latency between consistency schemes. Even when using Strong, the local copy is always up to date, and reading it doesn't need any interaction with the server at all.

There is a difference in Sync latency between Causal and Eventual in the event of a conflict because Causal requires more RTTs to resolve the conflict than Eventual. Because the first sync attempt by Writer (Tw) fails, Writer (Tw) is forced to read Causal Tester's (Tc) conflicting data and retry its update. This results in an increase in data transfer. Without conflicts, the latency of synchronization and data transfer for Causal and Eventual are comparable (not depicted).

10. CONCLUSION

This research examined the perceived latency of reads, writes, and data synchronization by LOSMob client applications. Data handling and object synchronization in LOSMob are efficient enough to satisfy strong, causal, and eventual consistency guarantees.

In order to evaluate the perceived latency of reads, writes, and data synchronisation with the Mobile client Apps of LOSMob, the LOSMob framework was tested with a write payload of 100 MiB files. The overall amount of data exchanged by Writer (Tw) and Reader (Tr) clients was reduced for the Eventual consistency scheme compared to the Last Writer wins rule. In the absence of conflicts, the synchronization latency and data transmission for Causal and Eventual consistency systems were identical. LOSMob's handling of the consistency scenario verifies that efficient data reduction and bandwidth reduction techniques are utilized during data transport. Hence LOSMob effectively support the three types of consistency guarantees (strong, casual and eventual consistency) with efficient data handling.

REFERENCES

[1] P. Mell, T. Grance, and others, "The NIST definition of cloud computing," 2011.

[2] D. Perkins *et al.*, "Simba: Tunable end-to-end data consistency for mobile apps," in *Proceedings of the Tenth European Conference on Computer Systems*, 2015, p. 7. doi: 10.1.1/jpb001.
[3] B.-G. Chun, C. Curino, R. Sears, A. Shraer, S. Madden, and R. Ramakrishnan, "Mobius: unified messaging and data serving for mobile apps," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, 2012, pp. 141–154.
[4] Y. P. Faniband, I. Ishak, F. Sidi, and M. A. Jabar, "A Review of Data Synchronization and Consistency Frameworks for Mobile Cloud Applications," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 12, pp. 601–611, 2018.
[5] G. DeCandia *et al.*, "Dynamo: amazon's highly available key-value store," in *ACM SIGOPS operating systems review*, 2007, vol. 41, pp. 205–220.
[6] B. A. Milani and N. J. Navimipour, "A comprehensive review of the data replication techniques in the cloud environments: Major trends and future directions," *J. Netw. Comput. Appl.*, vol. 64, pp. 229–238, 2016.
[7] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.
[8] W. Vogels, "Eventually consistent," *Commun. ACM*, vol. 52, no. 1, pp. 40–44, 2009.
[9] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, vol. 33, no. 2, p. 51, 2002, doi: 10.1145/564585.564601.
[10] D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer, and B. B. Welch, "Session guarantees for weakly consistent replicated data," in Parallel *and Distributed Information Systems, 1994., Proceedings of the Third International Conference on*, 1994, pp. 140–149.
[11] J. Brzezinski, C. Sobaniec, and D. Wawrzyniak, "From Session Causality to Causal Consistency.," in *PDP*, 2004, pp. 152–158.
[12] J. Brzezinski, C. Sobaniec, and D. Wawrzyniak, "Session guarantees to achieve PRAM consistency of replicated shared objects," in *International Conference on Parallel Processing and Applied Mathematics*, 2003, pp. 1–8.
[13] D. Bermbach and S. Tai, "Eventual consistency: How soon is eventual? An evaluation of Amazon S3's consistency behavior," in *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing*, 2011, p. 1.
[14] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu, "Data Consistency Properties and the Trade-offs in Commercial Cloud Storage: the Consumers' Perspective.," *Cidr*, pp. 134–143, 2011.
[15] M. Klems, D. Bermbach, and R. Weinert, "A runtime quality measurement framework for cloud database service systems," in *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*, 2012, pp. 38–46.
[16] D. Bermbach, J. Kuhlenkamp, B. Derre, M. Klems, and S. Tai, "A Middleware Guaranteeing Client-Centric Consistency on Top of Eventually Consistent Datastores.," in *IC2E*, 2013, pp. 114–123.
[17] J. Cao, Y. Zhang, G. Cao, and L. Xie, "Data consistency for cooperative caching in mobile environments," *Computer*, 2007.
[18] Y. Huang, J. Cao, B. Jin, X. Tao, J. Lu, and Y. Feng, "Flexible cache consistency maintenance over wireless ad hoc networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 8, pp. 1150–1161, 2010.
[19] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 1997, pp. 654–663.
[20] A. Lakshman and P. Malik, "Cassandra: structured storage system on a p2p network," in *Proceedings of the 28th ACM symposium on Principles of distributed computing*, 2009, pp. 5–5.
[21] "OpenStack Swift Object Storage Service." 2018.
[22] Y. P. Faniband, I. Ishak, F. Sidi, and M. A. Jabar, "NetMob: A Mobile Application Development Framework with Enhanced Large Objects Access for Mobile Cloud Storage Service," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 7, 2019, doi: 10.14569/IJACSA.2019.0100784.
[23] S. R. Alapati, "Cassandra Data Modeling, and the Reading and Writing of Data," in *Expert Apache Cassandra Administration*, Springer, 2018, pp. 99–148.
[24] R. Klophaus, "Riak core: Building distributed applications without shared state," in *ACM SIGPLAN Commercial Users of Functional Programming*, 2010, p. 14.
[25] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[26] N. Preguiça *et al.*, "SwiftCloud: Fault-tolerant geo-replication integrated all the way to the client machine," in *2014 IEEE 33rd International Symposium on Reliable Distributed Systems Workshops (SRDSW)*, 2014, pp. 30–33.
[27] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A comprehensive study of convergent and commutative replicated data types," PhD Thesis, Inria–Centre Paris-Rocquencourt; INRIA, 2011.
[28] S. Burckhardt, A. Gotsman, H. Yang, and M. Zawirski, "Replicated data types: specification, verification, optimality," in *ACM SIGPLAN Notices*, 2014, vol. 49, no. 1, pp. 271–284.
[29] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Symposium on Self-Stabilizing Systems*, 2011, pp. 386–400.
[30] B. F. Cooper *et al.*, "PNUTS: Yahoo!'s hosted data serving platform," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1277–1288, 2008.
[32] P. Platform, "Parse Platform." 2016.
[33] Kinvey, "Kinvey BaaS." 2016.
[34] "Apache CouchDB." 2018.

[35] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh, “Consistency-based service level agreements for cloud storage,” in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013, pp. 309–324.
[36] J. Jing, A. S. Helal, and A. Elmagarmid, “Client-server computing in mobile environments,” *ACM Comput. Surv. CSUR*, vol. 31, no. 2, pp. 117–157, 1999.
[37] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser, “Managing update conflicts in Bayou, a weakly connected replicated storage system,” in *ACM SIGOPS Operating Systems Review*, 1995, vol. 29, no. 5, pp. 172–182.
[38] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, “Agile application-aware adaptation for mobility,” in *ACM SIGOPS Operating Systems Review*, 1997, vol. 31, no. 5, pp. 276–287.
[39] A. D. Joseph, A. F. de Lespinasse, J. A. Tauber, D. K. Gifford, and M. F. Kaashoek, “Rover: A toolkit for mobile information access,” in *ACM SIGOPS Operating Systems Review*, 1995, vol. 29, no. 5, pp. 156–171.
[40] D. Barbará, “Mobile computing and databases-a survey,” *IEEE Trans. Knowl. Data Eng.*, vol. 11, no. 1, pp. 108–117, 1999.
[41] P. Serrano-Alvarado, C. Roncancio, and M. Adiba, “A survey of mobile transactions,” *Distrib. Parallel Databases*, vol. 16, no. 2, pp. 193–230, 2004.
[42] A. A. Imam, S. Basri, and R. Ahmad, “Data Synchronization Between Mobile Devices and Server-side Databases: a Systematic Literature Review,” *J. Theor. Appl. Inf. Technol.*, vol. 81, no. 2, p. 364, 2015.
[43] L. B. Bhajantri and V. V. Ayyannavar, “Cognitive Agent Based Data Synchronization in Ubiquitous Networks: A Survey,” *Int. J. Adv. Pervasive Ubiquitous Comput. IJAPUC*, vol. 10, no. 2, pp. 1–17, 2018.
[44] M. Diogo, B. Cabral, and J. Bernardino, “Consistency Models of NoSQL Databases,” *Future Internet*, vol. 11, no. 2, p. 43, Feb. 2019, doi: 10.3390/fi11020043.
[45] L. Lamport, “How to make a multiprocessor computer that correctly executes multiprocess programs,” *IEEE Trans. Comput.*, vol. 100, no. 9, pp. 690–691, 1979.
[46] A. Oprea and M. K. Reiter, “On consistency of encrypted files,” in *International Symposium on Distributed Computing*, 2006, pp. 254–268.
[47] S. Burckhardt, M. Fähndrich, D. Leijen, and B. P. Wood, “Cloud types for eventual consistency,” in *European Conference on Object-Oriented Programming*, 2012, pp. 283–307.
[48] A. B. Nandyal, M. Rafi, M. Siddappa, and B. B. Sathish, “Improving Data Services of Mobile Cloud Storage with Support for Large Data Objects Using OpenStack Swift,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 6, 2021, doi: 10.14569/IJACSA.2021.01206101.
[49] Y. P. Faniband, I. Ishak, F. Sidi, and M. A. Jabar, “Enhancing Mobile Backend As a Service Framework to Support Synchronization of Large Object,” in *Proceedings of the 2017 International Conference on Information Technology*, New York, NY, USA, 2017, pp. 383–387. doi: 10.1145/3176653.3176719.
[50] P. Alvaro, N. Conway, J. M. Hellerstein, and W. R. Marczak, “Consistency Analysis in Bloom: a CALM and Collected Approach.,” in *CIDR*, 2011, pp. 249–260.
[51] N. Conway, W. R. Marczak, P. Alvaro, J. M. Hellerstein, and D. Maier, “Logic and lattices for distributed programming,” in *Proceedings of the Third ACM Symposium on Cloud Computing*, 2012, p. 1.
[52] A. Shraer *et al.*, “Cloudkit: structured storage for mobile applications,” *Proc. VLDB Endow.*, vol. 11, no. 5, pp. 540–552, 2018.
[53] P. Garcia-Lopez, M. Sanchez-Artigas, C. Cotes, G. Guerrero, A. Moreno, and S. Toda, “StackSync: architecturing the personal cloud to Be in sync.”
[54] A. Gheith *et al.*, “IBM Bluemix Mobile Cloud Services,” *IBM J. Res. Dev.*, vol. 60, no. 2–3, pp. 7–1, 2016.
[55] A. Popov, A. Proletarsky, S. Belov, and A. Sorokin, “Fast Prototyping of the Internet of Things solutions with IBM Bluemix,” 2017.
[56] Firebase, “Firebase.” 2017.
[57] N. M. Belaramani *et al.*, “PRACTI Replication.,” in *NSDI*, 2006, vol. 6, pp. 5–5.

APPENDIX

Table 2. Summary of Reference Design.

Reference Framework	Mechanism of Sync Protocol	Handling of Conflict Resolution	PRACTI [57]property	Hoarding unit
Coda [59] ∗	Make use of RPC-based callbacks.	conflict resolution is done at the system level	PR, E	O
Bayou [38]	Bidirectional Protocol for Exchanging Logs.	Resolution at the Application Level	E , TI	T, O
Rover [40]	A two-way street built on RDOs (relocatable dynamic objects) and RPC (remote procedure call) queues (QRPC)	Application-level locking or application-specific algorithms ensure object consistency.	- , AC	O
Perspective [60] ↔	A few minor adjustments were made to the log of updates to ensure that only pertinent information was shared.	Utilize a Two-Step Approach to Resolving Conflict (Pre-resolver and a full resolver).	E , PR, TI	O

Cimbiosys [61] ↔	Bandwidth and system resources are conserved with the help of the Eventual filter consistency and the Eventual knowledge singularity technique.	It is possible for any device with a filter that chooses both versions to detect the conflict and either automatically resolves it or stores both versions until the issue is resolved manually.	E, PR,TI	O
PRACTI replication [58] ↔	There are two methods of interaction in this world. 1. Invalidation Streams in a causal order, and 2. Body messages that are not in any particular order. To send streams of invalidations, the log exchange protocol is used.	An API to identify and resolve write-write conflicts with the help of domain-specific logic	S, C, PR, AC , TI	O
SwiftCloud [26]	Replica reconciliation is not a matter for consensus in CRDTs.	It is possible to update CRDTs asynchronously.	E , C, RB , PR	CRDTs
Indigo [62]	Replica reconciliation is not a matter for consensus in CRDTs.	It is possible to update CRDTs asynchronously.	EC , C, S, RB , PR , AC	CRDTs
Izzy [63]	Apps need options for configuring when and how data is transferred, and you must provide them with those.	Offers row-by-row versioning for use in synchronisation and conflict resolution	E , PR	T
Simba [2] ∗	To accommodate Strong and Causal consistency schemes, compact version numbers are used instead of the full version vector. Objects in Simba are stored and synchronised as a collection of chunks of fixed size, which allows for efficient network transfer.	In order for a conflict to be removed from its own table, users must take affirmative action to resolve it.	S, C , E , PR , AC	T + O
Sapphire [64]	Incompatibilities include the inability to manage or conduct business across different Sapphire Objects (CSOs). The document is unclear on how to handle and resolve conflicts.	Dynamic allocation of load-balanced M-S replicas with eventual consistency is a feature that should be supported by the deployment manager (DM). Optimistic Transactions is a plugin that adds conflict-aborting transaction termination in the event of a concurrency disagreement.	S, E , PR	O
Mobius [3]	support features like asynchronous replication by means of a publish-subscribe mechanism (Yahoo Message Broker, YMB).	Provide the client with three sorts of information so it can choose whether to keep its local modifications or start over with a new update if a conflict arises.	FSC , PR , AC	T
NetMob [65]	To accommodate Strong and Causal consistency schemes, compact version numbers are used rather than a full version vector. NetMob utilises chunking techniques and object storage for streamlined network transfer. and synced as a collection of fixed-size chunks.	Until the user resolves a conflict, it will remain in a separate conflict table.	S, C , E , PR , AC	T + O
Key-Vaue Store with BloomL [66]	Two copies about something are brought into sync by exchanging "kv- store" map data.	When multiple changes are made to the same key at once, the 'lmap' merge function will automatically resolve the conflicts.	E	Built-in lattices
TouchDevelop [67] ∗ [48]	Data stored in particular clouds is automatically persisted in multiple locations, including local storage, the cloud, and all devices.	Merging conflicts are resolved automatically, and no specialised code is required.	E	O (Cloud Types)
Middleware for client-centric consistency [68]	When a protocol retrieves data from storage, it uses the vector clock technique to add a copy of the retrieved data to the local cache if the cache did not already contain the retrieved data in the exact version that was retrieved.	The application is responsible for reloading data in the event of a data violation.	MR , RYW , E , PR	O
Open Data Kit 2.0 [69] *	A single database row serves as the fundamental unit, and fine-grained change tracking is employed to facilitate more compact data transfer.	If there is a conflict, the user can choose to implement either the server's update or the local update, or some combination of the two.	E	T
StoArranger [70]	Apps should be instructed to postpone their cloud backup requests until transmission promotion/tail energy impacts are reduced, and backups should be stored in batches.	Folder sync APIs utilise the meta data of the folders to detect conflicts.	E , PR	T
Unidrive [71]	The cloud is used to store metadata about content and synchronises it to all clients in real time using a quorum-based distributed locking protocol and chunking method.	It allows for multiple users to store their data in the cloud, with synchronization logic being handled entirely by the client devices and all communication being handled via file transfers.	S, E, PR	O
QuickSync [72] ∗	The Network-aware Chunker, the Redundancy Eliminator, and the Batched Syncer are the three main building blocks of a sync system that significantly increase its efficiency.	Avoids fixing inconsistencies or resolving conflicts, but speeds up Sync.	E	O
Parse Server [32] ∗	For 'pinned' PFObjects, the Parse Server SDKs provide a local datastore for storing and retrieving them.	Cloud services use a generic beforeSave function to resolve conflicts on the server side. Software programmes must resolve any conflicts that arise.	S, E, PR	T

StackSync [54]	The Sync protocol is built on top of Remote Procedure Calls and Method Calls provided by the ObjectMQ middleware.	The user is prompted to decide whether or not to create a copy of the document that is in conflict.	S, E, PR	O
Dropbox [73]	The smallest data object that can be stored in Dropbox is 4MB in size. If a file is larger than 4 MB, it will be split into multiple smaller files. When sending information over a network, delta encoding is used to reduce the amount of information sent and received.	All file metadata includes a revision identifier, which is checked for updates and used to prevent conflicts.	RAW	O
iCloud with CloudKit [74]	The CloudKit APIs permit users to retrieve only the modified data since the last check-in.	To resolve a conflict, a user receives an error code from the iCloud server along with objects containing each version of the record in question.	S, E	O
Amazon Dynamo [5]	Includes a protocol for replica synchronisation (anti-entropy) and Merkle trees, both of which speed up and reduce the volume of data transfers.	Tools like object versioning and ap-assisted conflict resolution are used.	E, PR	O
Bluemix Mobile Cloud Service [55]	Cloudant Sync (compatible with the Apache CouchDBTM replication protocol) is used by apps to index and query locally stored JSON data.	The application must identify the conflicts and find a solution based on the specifics of each case.	E, PR	O
Firebase [57]	Undocumented proprietary syncing protocol	Dispute resolution using time stamps	E, PR	O
Kinvey [33]	Delta Sync is a method of synchronisation that allows for the transfer of only modified or newly created entities.	The libraries and backend used by Kinvey all follow the "client wins" policy by default, which means that the data received from the backend is always the result of the last client to perform a write. We also accommodate user-defined conflict resolution policies.	S, E, PR, AC	O
Note: T: tables, O: objects, CS: consistency scheme, S: strong, C: causal, E:eventual, RAW: Read After Write, MRC:Monotonic Read RYWC:Read your writes consistency, PR: Partial Replication, AC: Arbitrary Consistency, TI - Topology Independence, *: open-source				

AUTHORS

	Mr. Aslam B Nandyal, Currently pursuing Ph.D fron Visvesvaraya Technological University, Belgavi. He has previously worked as an assistant professor in St. Francis institute of Technology, Borivali west Mumbai. His current research domain is cloud computing. His areas of interest in network security, open source cloud platforms, Mobile backend as a service MBaaS.
	Dr. Mohammed Rafi is a professor in UBDTCE Davanagere, He has delivered many technical talks and published papers in various international journals in his career and area of interest are machine learning, Blockchain, computer vision,Database Management, Cyber Security . His is also an expert member of various national bodies of India. Expert Member AICTE, New Delhi, Co- Author for Text Book on Operating System, Chairman of Faculty Feedback committee at UBDTCE, Project Guide for Winner of I prize Avishkar project exhibition 2020 at VTU,