A Review of Data Synchronization and Consistency Frameworks for Mobile Cloud Applications

Yunus Parvej Faniband[1], Iskandar Ishak[2], Fatimah Sidi[3], Marzanah A. Jabar[4]
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
43400 UPM, Serdang, Selangor Darul Ehsan, Malaysia

***Abstract*—Mobile devices are rapidly becoming the predominant means of accessing the Internet due to advances in wireless communication techniques. The development of Mobile applications (“apps”) for various platforms is on the rise due to growth in the number of connected devices. Numerous apps rely on cloud infrastructure for data storage and sharing. Apart from advances in wireless communication and device technology, there is a lot of research on special data management techniques that addressed the limitations of mobile wireless computing to make the data appear seamless for accessing and retrieval. This paper is an effort to survey the frameworks that support data consistency and synchronization for mobile devices. These frameworks offer a solution for the unreliable connection problem with customized synchronization and replication processes and hence helps in synchronizing with multiple clients. The frameworks are compared for the parameters of consistency and data models (table, objects or both) support along with techniques of synchronization protocol and conflict resolution. The review paper has produced interesting results from the selected studies in areas such as data consistency, handling offline data, data replication, synchronization strategy. The paper is focused on client-centric data consistency and the offline data synchronization feature of various frameworks.**

Keywords*—Mobile cloud computing; data consistency; mobile back-end as a service; distributed systems; mobile apps

I. Introduction

The model of mobile cloud computing utilizes the services of cloud computing. The mobile cloud environment consists of portable computing devices, mobile Web and location-based services, supported by wireless communication infrastructure, to provide mobile devices online access to large storage space and unlimited computing power [1]. A wireless network with mobile clients is fundamentally a distributed system but suffer from the primary challenges such as limited computational capacity and storage of mobile devices, intermittent loss of connectivity and battery power restrictions. The transmission bandwidth of the mobile device is likely to be lesser than the transmission bandwidth of the mobile support stations (MSSs) and this leads to the phenomenon of communication asymmetry. The effective management of data in systems with the mobile client is affected by these limitations. The environment of frequent disconnections and limited bandwidth, impact the data and transaction management as well as the data consistency guarantees.

To provide the illusion of uninterrupted data access, the data management must hide the constraints of mobile wireless computing. The technique of replicating data locally on the mobile device enables the user to carry offline data without the need to always be connected to the data server. The ability to disconnect with the network, do local changes, and then reintegrating (synchronize) these changes back into the system makes the mobile gadget an essential extension to modern distributed databases and collaborative tools [1].

Data synchronization is an empowering process that eliminates the critical requirement of having steady connectivity and permits users to run data-centric mobile applications while being offline. Hence data synchronization allows users to carry out operations of additions, deletions, and updates on the offline data, while disconnected from the network.

Generally, the mobile applications (which we generally refer to as ‘Apps’) are developed according to the different application programming interfaces (API) abstractions supported by the underlying mobile middleware. The middleware may provide a simple file-based API (possibly extended with replication-specific methods). It may also support complex abstraction such as objects, tuples, relational entities or an object which may contain pointers to other interdependent objects. Middleware with database replication primarily provides query oriented CRUD APIs(Create, Read, Update and Delete) to application developers for typical operations on data with declaratively defined by SQL queries for the update, creation/insertion, and deletion of records.

This section covers the introduction to replication, data-centric and client-centric consistency models. Section II and III classify and describe various consistency and synchronization frameworks in mobile cloud computing. In Section IV we discuss research findings and recommendations followed by related work (Section V) and conclusion with future work (Section VI). Table I shows the list of acronyms used in the paper.

A. Mobile Computing Environment and Limitations

Generally, a mobile cloud computing environment has two unique sets of entities namely Fixed Hosts (FH) and Mobile Hosts (MH) [1]. FHs are machines (Works stations and Servers) with efficient computation power and reliable storage of data and run large databases. FHs that are connected through the fixed network. MHs with limited processing and storage power(cellular phone, palmtops, laptops, notebooks) are not continually communicating with the fixed network. They may be disconnected for various reasons such as due to the battery power saving measures and also due to disconnections during frequent relocation between different cells. Additional dedicated fixed hosts called mobile support stations (MSSs) acts

TABLE I. LIST OF ACRONYMS.

Symbol	Description
C	Causal Consistency
CMP	Consistency scheme with PRACTI property [2]
CRDT	Conflict-Free Replicated Data Types
HU	Hoarding Unit
E	Eventual Consistency
FSC	Fork-sequential consistency
MRC	Monotonic read consistency
MWC	Monotonic write consistency
PRACTI	PR - Partial Replication, AC - Arbitrary Consistency, TI -Topology Independence
RAWC	Read after writes consistency
RYWC	Read your writes consistency
S	Strong Consistency
SC	Sequential Consistency
T , O	Table , Objects
WFRC	Write Follows Read Consistency

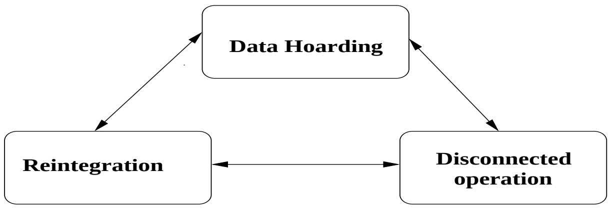

Fig. 1. States of Disconnected Operation.

as the channel between the FH and MH through wireless LAN (local area network) connections, cells or connections to the network with standard modems.

When the network connectivity becomes unavailable or unacceptable, the MH enters the disconnected state. Disconnected operation (see Fig. 1) is a three-stage changeover between the following states [3]:

1) Data hoarding: This is the process of preloading or prefetching the data in anticipation of a foreseeable disconnection. Before going to offline mode (disconnection), the data structures necessary for operation during disconnection are either replicated (catched) or moved (partitioned) at the MH.
2) Disconnected operation: When the MH is offline (disconnected from the network), data might be changed, added or even removed at either the MH or the FH.
3) Synchronization or Reintegration: When the connection is reestablished, each operation executed at the MH should be synchronized (reintegrated) with appropriate updates executed at other sites in order to attain seamless consistency.

For a given distributed system, the complexity of operations in each of the above three states is determined by the interdependence of data operated on. The issues pertaining to three states is summarized in Table II.

B. Replication

Replication is a basic strategy to support fault resilience, high data availability and quick response for universally available services. Replication process creates many instances of the identical object in different machines, over a distributed or local network ([3], [4], [5], [6], [7], [8]). The copies of these multiple objects (replicas) are persistently maintained over time in order to allow the workload to be divided over the possible number of replicas. The replication strategy involved in a different distributed system depends on the requirements of data freshness tolerance. The use cases in some applications need only read operations, while others high ratio of writes(updates) compared to read. Banking systems require that data is always consistent over time and some social networks may tolerate stale data.

TABLE II. ISSUES IN DISCONNECTED OPERATION.(SOURCE [1])

State	Problem	Resolution
Hoarding	Unit of caching/hoarding	System dependent (e.g. a file or a database fragment)
	Which items to cache (hoard)?	Application dependent , based on purpose of the system Defined distinctly by the user Generate from the knowledge of past operations
	When to execute hoarding ?	Based on regular intervals Before disconnection
	Call for locally unavailable data	Add requests to queue for future service Raise an exception/error
Disconnection	What to log?	Timestamps Data Values Operations
	When to optimize the log?	Before synchronization Incrementally
	How to optimize the log?	System dependent
	How to synchronize?	Re-execute an operational log
Reintegration or Synchronization	How to resolve conflicts?	Automatic resolution Use application-semantics Provide utility to aid the user

C. Consistency Models

The literature [9] [10] describes data-centric and client-centric consistency as the two principle viewpoints on consistency. The data-centric consistency manages the internal state details by guaranteeing that all the replicas are same and ensures system maintains consistency for updates. Data-centric consistency is important to system developers. Client-centric consistency deals with only observing data updates as a black box and hence application developers focus on client-centric guarantees. Ordering and Staleness are the two criteria for measuring guarantees of both data-centric and client-centric consistency. Staleness is measured in the unit of time (t-perceivability) or versions (k-staleness), calculated based on how much a given copy is falling behind [11] [12].

D. Data-Centric Models (Server-Side Consistency Models)

1) Strong consistency - A system adopting a strong consistency model is in a consistent state all times. The strong consistency is a single-copy consistency model that is not suitable for mobile applications dealing with cloud data due to the availability and performance issues as mandated by CAP theorem [13].
2) Sequential consistency - This is a slightly weaker form of strong consistency with the condition that, same order of execution is maintained for all the sequentially related requests. Subsequently, the clients observe the same order and sequence of updates.

3) Causal consistency - In a system adopting the Causal Consistency (CC), the same sequence of execution is maintained on all replicas, for all the causally related requests. The non-related requests are followed in random order.
4) Grouping operations - This model deals with handling the cases of, series of reading and write operations. The Grouping Operation model allows raising the level of granularity to span multiple reads and writes, into an atomically executed unit.

E. Client-Centric Models (Client-Side Consistency Models)

1) Weak consistency - A weak consistency model does not guarantee that subsequent accesses will return the updated value. The term 'inconsistency window' [10] attribute to the time between the update and the instant when it is guaranteed that any observer will always see the refreshed value.
2) Eventual consistency - Eventual consistency is considered as another model of Weak consistency with an added guarantee that when no new updates are made to an object, eventually all replicas will see the last. Eventual consistency provides the following four main ordering guarantees [14]:
    a) Monotonic Read Consistency (MRC) - In this model after reading a version 'n' of an object, the same client will never access a version less than 'n' on a subsequent read.
    b) Read Your Writes Consistency (RYWC) - In this model, after writing version 'n', the same client will never again read an older version less than 'n'. This is a unique case of the causal consistency model [9] [10].
    c) Monotonic Write Consistency (MWC) - In this model, all writes by the same client guaranteed to be serialized in the order of time of update It guarantees that a write operation is always ended prior to any subsequent write operation on the same data item [9] [10]
    d) Write Follows Read Consistency (WFRC) - It guarantees that an update succeeding a read of version 'x' will never be carried out on replicas that are prior to version 'x' [9].

The studies [15] [16] conclude that it is mandatory to guarantee all four client-centric models (MRC, RYWC, MWC, and WFRC) for the system to achieve client-centric consistency.

II. Classification of Consistency, Synchronization and Replication Systems in Mobile Computing

This section classifies the current efforts into different types such as systems for weakly connected clients, sync services and systems supporting geo-replication. The studies are also classified based on three PRACTI properties [2].

A. Systems for Weakly-Connected Clients

Many previous attempts have dealt with data replication and management in systems where mobile clients intermittently connected either to servers or to peers ([3], [4],[17], [2], [18], [19]). Systems like Coda [3] and Ficus [19] address the issues in handling disconnected operations and replicate files providing high availability at the cost of consistency. Bayou [4] is a distributed relational database system that provides eventual data consistency, under offline mode. These systems differ on their procedures to handle conflicts. For instance, Bayou performs application-level conflict resolution, while Coda and Ficus allow system level resolution of conflicts. Some Systems (like Simba [20]) are aimed to provide more control for mobile applications to select suitable consistency abstractions for data synchronization services.

There are several studies which explicitly focus on the efficiency of data management systems for weakly connected clients ([21], [22], [23], [24]). In compliance to different requirements of apps, Odyssey [23] system give OS support for applications to modify the fidelity of their data to accommodate resource changes, such as wireless network bandwidth fluctuations and battery conditions. Cedar [24] increase the query processing capability by identifying the commonality between client and server query results and hence provides productive mobile database access. In LBFS [21] (low-bandwidth network file system), the content-based chunking technique prevents redundant transfer of files and also detect inter-file similarities.

B. Geo-Replication

There are several studies which focus on the tradeoff between consistency, availability, and performance for geo-distributed services. These system handle data replications within and across and data centers. Some systems primarily aimed at providing low-latency causal consistency at scale (e.g. , COPS [25] and Eiger [26]) and others (e.g., Red Blue consistency [27], Walter [8], Transaction Chains [28], and) focus to reduce the latency involved in supporting other forms of stronger-than-eventual consistency, including serializability under limited conditions. Arbitrary consistency selection systems (e.g., Pileus [29] and SPANStore [30]) attempt to provide more control for applications to choose suitable consistency across data centers, to meet SLAs or to minimize operating costs.

C. PRACTI Paradigm

In a distributed system, an optimal replication system should support all the three PRACTI [2] properties. 1) PR-system (Partial Replication) allow any node to store a subset of data and metadata. 2) AC-system (Arbitrary Consistency) provide flexibility of selection of consistency semantics (different types of configurable consistency guarantees like both strong and weak consistency) for applications. 3) The TI-systems (Topology Independence) permit all nodes to send updates to all other nodes (TI).

Applying PRACTI taxonomy to the current studies, the existing replication systems fall into the following four protocol groups. Each system compliant to most two of the PRACTI paradigm properties.

1) Server replication: Some systems use the log-based peer-to-peer update exchange protocol for server-side replication. This protocol follows full replication mechanism and allow all nodes to store complete data from any volume and also all nodes collect

all updates. This protocol helps to achieve topology independence (TI) in some systems (e.g., Bayou [4] and Replicated Dictionary [31]), where any node to send updates to any other node. Some Systems like in TACT [32] and Lazy Replication [33]) use this protocol to provide more control to select suitable consistency guarantees for data synchronization (AC). Since the protocol does not support efficient network usage due to full replication, these systems may be not suitable for devices with limited resources.

2) Some systems with client-server architecture (e.g., Coda [3] and Sprite [34]) and hierarchical caching systems (e.g., hierarchical AFS [35]) implement a protocol to selectively replicate/cache arbitrary subsets of content (PR). Apart from supporting a group of consistency policy by the system, a supplementary extension of consistency guarantees are provided by changing the basic architecture (AC). In order to support consistency, the partial replication protocols need intercommunication between a child and its parent and also serialize control messages at the central server node [36]. Due to these communication complexities, the performance, availability and data sharing features may be paralyzed in such systems.
3) In the Distributed hash table (DHT)-based storage systems (e.g., PAST [37], CFS [38] and BH [39]), the scalability is achieved by load balancing the server across various nodes, on a per-object or per-block basis. For high availability, the data is also replicated to multiple nodes and such architecture becomes challenging for providing the consistency guarantees.
4) Object replication systems (e.g., WinFS [40], Ficus [41] and Pangaea [42]) permit nodes for selective replication/caching of arbitrary subsets of data (PR) and communication with every other peer (TI).These protocols lack consistency guarantees since they do not mandate ordering constraints on updates across multiple objects.

D. Synchronization Service Frameworks

The existing services mainly offer sync services into three categories: (1) File-only, (2) Table-only and combination of (3) Table and Object. Izzy [43] and Mobius [44] sync services provide a platform for structured data like tables only, to expedite the development and deployment of data-centric mobile apps. Mobius guarantee that all clients observe write operations in the same order, maintaining the flexibility of local client views to diverge (fork-sequential consistency [45]).Dropbox sync service provides dedicated API for tables and do not store files and tables together. Many Mobile apps are developed using the file sync services of Google Drive [46], iCloud [47], Dropbox[48] [49] and Box Sync[50]. StackSync [51] is an open-source Personal Cloud framework that provides scalable file synchronization and sharing. QuickSync [52] is a system that focuses on improving the synchronization performance of cloud storage, in wireless networks depending on network conditions.

Sapphire [53] is a cloud-enabled distributed programming platform for mobile and cloud applications. Sapphire makes a smooth application execution using the techniques of code-offloading, caching, and fault-tolerance. Sapphire lacks in data management services but provides smooth application execution. The work on Pebbles[54] revealed that apps massively depend on structured data (table) to manage unstructured objects (files). Simba [20] extended the table interface of Izzy [43] to provide a unified abstraction for both table and object, the benefits of which are explored the context of local systems in these studies [[55], [56]].

CouchDB [57] is a schema-free “document store” supporting eventual consistency and provide “document” sync with coordination from its client TouchDB [58].

SwiftCloud [59] and Cloud types [60] provide cloud-enabled programming interface to facilitate the mobile apps for storing local replicas of data on the devices and subsequently sync with the cloud servers. The programmer needs to handle synchronization in SwiftCloud and Cloud types, while Simba permits automatic synchronization.

Mobile operating systems provide some kinds of data storage abstractions to developers. Apple expanded its iCloud [47] service with CloudKit [61], a new means for applications to store and access data stored in iCloud [47]. There are some open source mobile back-end-as-a-service offering, such as Parse Server platform [62] and StackSync [51].

Many commercial services provide back-end cloud storage services to link mobile and web applications to the cloud, such as IBM Bluemix Mobile Cloud Service [63] [64]. Services of Firebase [65] and Kinvey [66], also aid app developers to connect their apps to cloud backend.

III. Discussion on Literature of Consistency and Synchronization Frameworks

The existing literature from the database community and distributed systems community focus on consistency models, their implementations and their measurement. This paper focuses on the reference implementations helping the mobile clients for end-to-end data consistency and data synchronization service utilizing the cloud resources. The literature has case studies investigating the difficulties related to consistent replication across mobile devices with intermittent network connectivity and bandwidth constraints. Some studies in the literature address the frameworks designed to handle the current constraints in Mobile app development.

Coda [3], was one of the initial client-server architecture systems, to emphasize the difficulties in addressing the offline operations. BlueFS [67] is another system that focuses on energy efficiency in resource-constrained mobile devices. Bayou [4] is based on client-server architecture and supports a disconnected system and provides a programming interface to application-specific conflict detection and resolution to handle optimistic updates (eventual consistency). Odyssey [23] support application-aware adaptation based on type-specific operations. The Rover [68] toolkit is a client-server, mobile applications development platform that relocatable dynamic object (RDO) and queued remote procedure call (QRPC) for data communication.

Simba [20] provides end-to-end data consistency framework with a data abstraction for a combination of tabular and object data models. Additionally, the applications written to this abstraction are allowed to select from a set of distributed consistency schemes and sync data with the cloud. Simba Server implementation of data Storage use OpenStack Swift [69] for object data and Cassandra [6] to store tabular data. Simba configure OpenStack Swift and Cassandra to utilize three-way replication, in order to achieve high availability. Also, the framework mentioned in [70] support using Cassandra as a backend datastore. Our work [71] proposes to extend Simba with support for large data objects.

Mobius [44] is designed as a cloud-enabled data replication and messaging platform for the mobile applications. It provides table consistency and uses PNUTS [7] as the back-end store.

A middleware framework for a mobile network that performs reliable and real-time data synchronization is proposed by Xue [72]. Izzy [43] and Mobius [44] frameworks provide a platform for structured data like tables only, to expedite the development and deployment of data-centric mobile apps. Simba [20] is built upon the sync framework of Izzy and supports cloud-based data synchronization service, which reduces development complexity of mobile apps.

Cimbiosys [17] is a peer-to-peer system platform (clients share updates directly with each other) that enables various apps to manage cloud-based data on personal computers and mobile devices. Perspective [73] is another platform like Cimbiosys that use filters for selective replication of data on mobile devices. PRACTI [2] is a unique replication system that supports all the three ideal PRACTI properties of partial replication, arbitrary consistency and topology-independence.

Currently, researchers are proposing new principles to deal with weak consistency. Strong correctness guarantees can be achieved without the use of costly global synchronization when all operations in a program are purely monotonic. Built on this monotonic principle, some data structures like sets and sequences can be correctly replicated without the need of synchronization.

The Conflict-Free Replicated Data Types (CRDTs) ([74] [75] [76]) are asynchronous data types that do not need synchronization for updates. They comply Strong Eventual Consistency Model [75] and can be utilized to build other data models, required by applications. Asynchronous quality of CRDTs makes it more qualified for replication in eventual consistency environments.

More recently researchers utilize these special data types (CRDTs) to build the frameworks using Key-Value stores. Riak [77] distributed database is used as a back-end store by systems like SwiftCloud [59] to implement a Key-CRDT. In order to support strong eventual consistency, the SwiftCloud middleware, convert a Key-Value store in a Key-CRDT store, into a data-model that utilize properties of CRDT. The system allows clients to execute updates concurrently without synchronization. By executing automatic conflict resolution specified in CRDTs, the systems guarantees the clients with zero conflict for simultaneous updates. Walter [8] and Gemini [27] are other systems that use CRDT for providing eventual consistency. Indigo [78] enhance SwiftCloud, wherein an application specifies the invariants, or consistency rules, that the system must maintain.

Consistency As Logical Monotonicity (CALM) is another technique used in built consistency frameworks. According to the CALM theorem, logically monotonic programs are guaranteed to be eventually consistent without the requirement of any coordination protocols (distributed locks, two-phase commit, paxos, etc.). Hence CALM approach ensures eventual consistency by necessitating a monotonic logic [79]. In logic languages (e.g. Bloom[80]) CALM analysis helps to analyze whether the code flow is sufficient towards consistency without the integrating co-ordination protocols [79].

The study claimed [81] that the use of revision diagrams along with special abstract Cloud Types is a useful technique for eventually consistent distributed programs. Revisions diagrams are semi lattices designed for the context of multiple versions and eventual consistency and work same as the version control systems. In this approach, the distributed state is stored using special cloud abstract data types. These Cloud types expose interface with well defines update and query operations [60]. Cloud types provide eventually consistent storage and hide the complex backend implementation details of network and coordination protocols. They offer the functionality to perform the optimized fork and join implementations and storing of updates in the form of logs [60]. The prototype implementation of this technique is available in TouchDevelop language and as a library in C# [82]. While the CRDTs help to carry out only commutative operations, the cloud types support non-commutative operations still accomplishing eventual consistency.

Open Data Kit (ODK) 2.0 [83] support to build Android-based application-specific information modules for offline operations. StoArranger [84] is another system framework that aid the programmers to manage cloud data storage on mobile devices by addressing issues of rearranging, and coordinating cloud storage communications. BlueMountain [85] is a modern mobile data management platform supporting solutions for file and database management, which allow to achieve wider deployability and help app developers to spend more efforts on app logic. Unidrive [86] is a client-side middleware system which can integrate multi-cloud capabilities to mobile apps. CacheKeeper [87] allows caching of browser data on mobile devices using system-wide, kernel level caching support for mobile applications.

Parse [62] is a back-end as a service platform that uses MongoDB as the back-end datastore. Parse platform allow the developer to create loosely or strongly typed objects and easily save, update, query, and delete these in a backend data store.

Mobile apps are developed using the file sync services of Google Drive [46], iCloud [47], Dropbox [48] [49] and Box Sync[50]. StackSync [51] is an open-source Personal Cloud framework that provides scalable file synchronization and sharing. QuickSync [52] is a system that optimizes cloud storage synchronization performance in wireless networks based on network conditions. IBM Bluemix Mobile Cloud Service [63] [64] provides back-end cloud storage services to link mobile and web applications to the cloud. Other commercial platforms such as Kinvey [66] and Firebase [65], help app developers to connect the apps to cloud backend.

TABLE III. COMPARISON OF THREE REFERENCE IMPLEMENTATIONS.

Reference Design	Strength	Weaknesses
Simba [20]	- Allow apps for the programmatic delay tolerant data transfer - Uses a single persistent TCP connection to the cloud data , resulting in bandwidth saving	Since multiple apps access the same instance of client, certain poorly written apps may adversely affect other Simba apps
Mobius [44]	- All in one solution with a combination of messaging and data platform - Linear scalability for number of applications, users and size of data	Can be improved in the area of cross-app synchronization, optimization strategies and caching
SwiftCloud [59]	- Allow execution of transactions in the client side as well as at the data centers - Efficient use of caching methods, executing both reads and updates at the client	- Lack support for combined weak and strong consistency, and for object composition - DC implementation is not modular

IV. RESEARCH FINDINGS, DISCUSSION AND RECOMMENDATIONS

This section discusses the selected case studies, based on the criteria to understand the different technologies used for building the frameworks. SwiftCloud uses the CRDT with the Riak key store. Mobius uses PNUTS distributed database and supports P2P communication model. Simba supports configuring different consistency levels using Cassandra and OpenStack Swift object storage. TouchDevelop library utilizes the Cloudtypes using the Revision diagrams. BloomL library covers the BloomL language supported framework. Due to space constraints, we are only covering the three frameworks Swiftcloud, Simba and Mobius. These reference solutions are aimed at providing data replication, synchronization, and offline services to ease the development complexity of mobile apps. The solutions use the client side caching technique to offer offline services. The solutions are backed by the cloud storage to store the data. Table III summarizes the strength and weaknesses of the studied three reference implementations. Table IV (See Appendix) summarizes the consistency and data models (table, object or both) support in the various reference implementations. Table IV also lists PRACTI property supported by each framework along with mechanism of synchronization protocol and conflict resolution.

A. Synchronization Services

Some of the solutions provide the sync services for structured data like table only (Mobius and SwiftCloud). Simba supports both tabular and objects data models. Synchronization operation execution required to be handled by the programmers in case of Mobius and Swiftcloud. In contrast, Simba supports automatic synchronization process in the background.

B. Consistency Support

In order to satisfy the diverse consistency needs the frameworks should support different types of data and independently define their consistency. Mobius provides per-record sequential [88] and fork-sequential [45] consistency through the exclusive type of read operations. Simba provides three consistency semantics, resembling strong, causal and eventual consistency. The extent of consistency specification permit may be per-row or per-request, per-table.

Caching Policy and Offline support: The strategies of caching (replication) data at the client side enable higher availability and improve latency. Caching policies need to take care of the consistency semantic (ordering, updates and fetching of fresh updates). Solutions provide options to access data from client-side storage or remotely. Cache policies can be determined by the server-side back-end. The server-generated policies can be context-aware, globally configurable and dynamic. These policies are created based on run-time usage or access patterns of all users collected from each application. Efficient write caching capabilities group possible numbers of write operations in a one network message to reduce bandwidth. A Prototype of Mobius clients uses the trained decision tree model (policy selector) to determine whether to fetch locally or remotely. Mobius uses cost-sensitive decision tree classifiers to write batch updates. In SwiftCloud the clients can access the causally- consistent view of the stable version of data (cached at multiple servers). In Mobius, MUD tables are partitioned across mobile nodes and one or more server backends. Data access during offline is from the local tables. The write updates are stored locally and forwarded to the backend on reconciliation of client. During offline, reads are delivered from the local scout in SwiftCloud. Scout cache handles the write updates. On network availability, finally, they will be committed at its DC.

C. Limitations of Reference Frameworks

Even though incredible researches have been done in providing end-to-end data consistency solutions, many challenges still remain. This section points out some of the challenges that are needed to be addressed in various reference frameworks.
For app developers, currently, Mobius [44] provide higher level APIs (blocking or asynchronous) abstracted around the basic MUD APIs. The researchers propose the opportunity to support richer interfaces with the declarative query language. Mobius can be improved in the area of cross-app synchronization, optimization strategies and caching. Mobius can be improved in cache operations such as dynamic caching strategies, clearance policies and push-based cache maintenance. For bandwidth consumption and improving access, the outstanding updates stored locally should be compressed. There should be a smooth deterioration of response quality during disconnected operation. For the scalability improvements, the authors of Mobius [44] propose to improve partitioning schemes by adapting their earlier efforts on automatic and fine-grained partitioning ([89], [90]).

Simba's [20] sync protocol does not support streaming APIs to handle big size objects (e.g. Media file like Videos). Simba proposes to handle atomic multi-row transactions as prospective enhancement and currently support only atomic transactions on individual rows.

SwiftCloud [59] can be enhanced with a better caching mechanism and support for transaction migration. Also, better data encapsulation across software stack through API level, to address efficient data access.

V. Related Work

The work of [91] conducts an analysis of concepts of mobile client-server computing and mobile data access with a detailed review of early research prototypes (Bayou [4], Odyssey [23] and Rover [68]) for mobile data management. Our work extends this work by analyzing the consistency support for the latest frameworks. The survey of contributions on data dissemination and support for data consistency techniques for mobiles devices is discussed here [92]. The paper [93] compares and analyze the several contributions to models for mobile transaction. A survey of literature work on synchronization between the mobile device and server-side databases can be found here [94]. A survey of academic work on mobile/cloud computing can be found here [95]. The paper [96] conducts a comprehensive review of the data replication techniques in the cloud environments. Recent review article deals with the comparison of different categories of data synchronization algorithms based on scalability, consistency, accuracy parameters in ubiquitous network [97].

VI. Conclusion and Future work

In this paper, we presented a review of data consistency and synchronization frameworks in Mobile Cloud Computing for Mobile Apps. We considered the latest studies done from 2010 to 2017, and the advantages and disadvantages of three reference implementations in the literature have been presented. Then, the approaches to handle consistency support, sync services, conflict handling and offline operations in reference solutions have been discussed. Furthermore, out of the review, several findings and potential future works have been identified. We believe that this is an important research area, that will attract more contributions from the research community.

The Conflict free replicated data type, logically monotonic programs (CALM approach) and Revisions diagrams as semi lattices are some of the techniques used in these frameworks. Frameworks make use of the backend stores implemented using these technologies to support the data consistency features.

Out of the three frameworks explored, *Simba* is a superior framework ensuring three types of consistency guarantees (strong, causal and eventual consistency) for both table and objects data models. Simba reduces programmer's efforts as it supports automatic synchronization process in the background. Simba lacks multi-row transactions and streaming APIs to access to large objects.

Swiftcloud uses a client-assisted failover solution with CRDT store to support both mergeable and strongly consistent transactions. Programmers need to manage the synchronization process. It utilizes properties of CRDTs to support automatic conflict resolution. SwiftCloud needs to improve in providing efficient data access through APIs.

Mobius provides table consistency and uses PNUTS as the back-end store to support cloud-enabled data replication and messaging platform. Mobius provides per-record sequential and fork-sequential consistency through the exclusive type of read operations. Programmers need to manage the synchronization process. Mobius uses cost-sensitive decision tree classifiers to write the batch update. Mobius needs improvements in the area of caching and optimization strategies with richer client interfaces. It has to be noted that the literature review is limited by sources and keywords, terminologies used in the search, and the search date. Hence it is possible to include more relevant papers while replicating this study in the future. Our final outputs of this research are limited to the current availability of frameworks that address the data consistency, synchronization , and other features. While the current study did not deal with the full details of measurements of numerical deviation, order deviation and staleness (latency) of each framework, we intend to conduct detailed research with simulations on the comparison of these performance parameters for each platform.

Acknowledgment

Authors acknowledge Universiti Putra Malaysia and Faculty of Computer Science and Information Technology, Malaysia, for all the support. This research is funded by Ministry of Education of Malaysia under the Fundamental Research Grant Scheme (FRGS) (FRGS/1/2015/ICT04/UPM/02/07).

References

[1] E. Pitoura and G. Samaras, *Data management for mobile computing*. Springer Science & Business Media, 2012, vol. 10.

[2] N. M. Belaramani, M. Dahlin, L. Gao, A. Nayate, A. Venkataramani, P. Yalagandula, and J. Zheng, "Practi replication." in *NSDI*, vol. 6, 2006, pp. 5–5.

[3] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 1, pp. 3–25, 1992.

[4] 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*, vol. 29. ACM, 1995, pp. 172–182.

[5] 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," in *ACM SIGOPS operating systems review*, vol. 41. ACM, 2007, pp. 205–220.

[6] 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*. ACM, 2009, pp. 5–5.

[7] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts: Yahoo!'s hosted data serving platform," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1277–1288, 2008.

[8] Y. Sovran, R. Power, M. K. Aguilera, and J. Li, "Transactional storage for geo-replicated systems," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 385–400.

[9] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[10] W. Vogels, "Eventually consistent," *Communications of the ACM*, vol. 52, no. 1, pp. 40–44, 2009.

[11] 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*. IEEE, 2012, pp. 38–46.

[12] 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*. ACM, 2011, p. 1.

[13] A. Bradic, "The cap theorem : Brewer ' s conjecture and the feasibility of consistent , available , partition-tolerant web services," no. August, 2010.

[14] 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*. IEEE, 1994, pp. 140–149.

[15] J. Brzezinski, C. Sobaniec, and D. Wawrzyniak, “From session causality to causal consistency.” in *PDP*, 2004, pp. 152–158.

[16] ——, “Session guarantees to achieve pram consistency of replicated shared objects,” in *International Conference on Parallel Processing and Applied Mathematics*. Springer, 2003, pp. 1–8.

[17] V. Ramasubramanian, T. L. Rodeheffer, D. B. Terry, M. Walraed-Sullivan, T. Wobber, C. C. Marshall, and A. Vahdat, “Cimbiosys: A platform for content-based partial replication,” in *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, 2009, pp. 261–276.

[18] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, “Ivy: A read/write peer-to-peer file system,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 31–44, 2002.

[19] R. G. Guy, J. S. Heidemann, and T. W. Page Jr, “The ficus replicated file system,” *ACM SIGOPS Operating Systems Review*, vol. 26, no. 2, p. 26, 1992.

[20] D. Perkins, N. Agrawal, A. Aranya, C. Yu, Y. Go, H. V. Madhyastha, and C. Ungureanu, “Simba: Tunable end-to-end data consistency for mobile apps,” in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 7. [Online]. Available: https://github.com/SimbaService/Simba

[21] A. Muthitacharoen, B. Chen, and D. Mazieres, “A low-bandwidth network file system,” in *ACM SIGOPS Operating Systems Review*, vol. 35. ACM, 2001, pp. 174–187.

[22] N. Tolia, J. Harkes, M. Kozuch, and M. Satyanarayanan, “Integrating portable and distributed storage.” in *FAST*, vol. 4, 2004, pp. 227–238.

[23] 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*, vol. 31. ACM, 1997, pp. 276–287.

[24] N. Tolia, M. Satyanarayanan, and A. Wolbach, “Improving mobile database access over wide-area networks without degrading consistency,” in *Proceedings of the 5th international conference on Mobile systems, applications and services*. ACM, 2007, pp. 71–84.

[25] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, “Don’t settle for eventual: scalable causal consistency for wide-area storage with cops,” in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 401–416.

[26] ——, “Stronger semantics for low-latency geo-replicated storage.” in *NSDI*, vol. 13, 2013, pp. 313–328.

[27] C. Li, D. Porto, A. Clement, J. Gehrke, N. M. Preguiça, and R. Rodrigues, “Making geo-replicated systems fast as possible, consistent when necessary.” in *OSDI*, vol. 12, 2012, pp. 265–278.

[28] Y. Zhang, R. Power, S. Zhou, Y. Sovran, M. K. Aguilera, and J. Li, “Transaction chains: achieving serializability with low latency in geo-distributed storage systems,” in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 276–291.

[29] 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*. ACM, 2013, pp. 309–324.

[30] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, “Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services,” in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 292–308.

[31] G. T. Wuu and A. J. Bernstein, “Efficient solutions to the replicated log and dictionary problems,” in *Proceedings of the third annual ACM symposium on Principles of distributed computing*. ACM, 1984, pp. 233–242.

[32] H. Yu and A. Vahdat, “Design and evaluation of a conit-based continuous consistency model for replicated services,” *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 3, pp. 239–282, 2002.

[33] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat, “Providing high availability using lazy replication,” *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 4, pp. 360–391, 1992.

[34] M. Nelson, B. Welch, and J. Ousterhout, *Caching in the Sprite network file system*. ACM, 1987, vol. 21, no. 5.

[35] D. Muntz and P. Honeyman, “Multi-level caching in distributed file systems,” Center for Information Technology Integration, Tech. Rep., 1991.

[36] S. Chandra, M. Dahlin, B. Richards, R. Y. Wang, T. E. Anderson, and J. R. Larus, “Experience with a language for writing coherence protocols,” in *Proceedings of the Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (DSL), 1997*. Usenix Association, 1997, pp. 5–5.

[37] A. Rowstron and P. Druschel, “Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility,” in *ACM SIGOPS Operating Systems Review*, vol. 35. ACM, 2001, pp. 188–201.

[38] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Wide-area cooperative storage with cfs,” in *ACM SIGOPS Operating Systems Review*, vol. 35. ACM, 2001, pp. 202–215.

[39] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay, “Design considerations for distributed caching on the internet,” in *Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference on*. IEEE, 1999, pp. 273–284.

[40] D. Malkhi and D. Terry, “Concise version vectors in winfs,” in *International Symposium on Distributed Computing*. Springer, 2005, pp. 339–353.

[41] R. G. Guy, J. S. Heidemann, W.-K. Mak, T. W. Page Jr, G. J. Popek, D. Rothmeier *et al.*, “Implementation of the ficus replicated file system.” in *USENIX Summer*, 1990, pp. 63–72.

[42] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam, “Taming aggressive replication in the pangaea wide-area file system,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 15–30, 2002.

[43] S. Hao, N. Agrawal, A. Aranya, and C. Ungureanu, “Building a delay-tolerant cloud for mobile data,” in *2013 IEEE 14th International Conference on Mobile Data Management*, vol. 1. IEEE, 2013, pp. 293–300.

[44] 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*. ACM, 2012, pp. 141–154.

[45] A. Oprea and M. K. Reiter, “On consistency of encrypted files,” in *International Symposium on Distributed Computing*. Springer, 2006, pp. 254–268.

[46] G. Drive, “Google drive,” 2016, https://developers.google.com/drive/.

[47] A. Inc, “icloud for developers,” 2016.

[48] Dropbox, “Build your app on the dropbox platform,” 2016, https://www.dropbox.com/developers.

[49] I. Drago, M. Mellia, M. M Munafo, A. Sperotto, R. Sadre, and A. Pras, “Inside dropbox: understanding personal cloud storage services,” in *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM, 2012, pp. 481–494.

[50] B. Inc, “Box sync app,” 2016, ”http://box.com”.

[51] P. Garcia-Lopez, M. Sanchez-Artigas, C. Cotes, G. Guerrero, A. Moreno, and S. Toda, “Stacksync: architecturing the personal cloud to be in sync.”

[52] Y. Cui, Z. Lai, X. Wang, and N. Dai, “Quicksync: Improving synchronization efficiency for mobile cloud storage services,” *IEEE Transactions on Mobile Computing*, vol. 16, no. 12, pp. 3513–3526, 2017.

[53] I. Zhang, A. Szekeres, D. Van Aken, I. Ackerman, S. D. Gribble, A. Krishnamurthy, and H. M. Levy, “Customizable and extensible deployment for mobile/cloud applications,” in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 97–112.

[54] R. Spahn, J. Bell, M. Lee, S. Bhamidipati, R. Geambasu, and G. Kaiser, “Pebbles: Fine-grained data management abstractions for modern operating systems,” in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 113–129.

[55] P. Shetty, R. P. Spillane, R. Malpani, B. Andrews, J. Seyster, and

E. Zadok, "Building workload-independent storage with vt-trees." in *FAST*, 2013, pp. 17–30.

[56] K. Ren and G. Gibson, "Tablefs: Embedding a nosql database inside the local file system," in *APMRC, 2012 Digest*. IEEE, 2012, pp. 1–6.

[57] "Apache couchdb," 2018, http://couchdb.apache.org.

[58] "Touchdb," 2018, http://tinyurl.com/touchdb.

[59] N. Preguiça, M. Zawirski, A. Bieniusa, S. Duarte, V. Balegas, C. Baquero, and M. Shapiro, "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)*. IEEE, 2014, pp. 30–33.

[60] S. Burckhardt, M. Fähndrich, D. Leijen, and B. P. Wood, "Cloud types for eventual consistency," in *European Conference on Object-Oriented Programming*. Springer, 2012, pp. 283–307.

[61] A. Shraer, A. Aybes, B. Davis, C. Chrysafis, D. Browning, E. Krugler, E. Stone, H. Chandler, J. Farkas, J. Quinn *et al.*, "Cloudkit: structured storage for mobile applications," *Proceedings of the VLDB Endowment*, vol. 11, no. 5, pp. 540–552, 2018.

[62] P. Platform, "Parse platform," 2016, https://parseplatform.github.io/.

[63] A. Gheith, R. Rajamony, P. Bohrer, K. Agarwal, M. Kistler, B. W. Eagle, C. Hambridge, J. Carter, and T. Kaplinger, "Ibm bluemix mobile cloud services," *IBM Journal of Research and Development*, vol. 60, no. 2-3, pp. 7–1, 2016.

[64] A. Popov, A. Proletarsky, S. Belov, and A. Sorokin, "Fast prototyping of the internet of things solutions with ibm bluemix," in *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017.

[65] Firebase, "Firebase," 2017, https://firebase.google.com/.

[66] Kinvey, "Kinvey baas," 2016, https://www.kinvey.com/.

[67] E. B. Nightingale and J. Flinn, "Energy-efficiency and storage flexibility in the blue file system." in *OSDI*, vol. 4, 2004, pp. 363–378.

[68] 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*, vol. 29. ACM, 1995, pp. 156–171.

[69] O. Swift, "Openstack swift object storage service," 2018, http.//swift.openstack.org.

[70] 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.

[71] 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*. ACM, 2017, pp. 383–387.

[72] Y. Xue, "The research on data synchronization of distributed real-time mobile network," in *Computer Science and Software Engineering, 2008 International Conference on*, vol. 3. IEEE, 2008, pp. 1104–1107.

[73] B. Salmon, S. W. Schlosser, L. F. Cranor, and G. R. Ganger, "Perspective: Semantic data management for the home." in *FAST*, vol. 9, 2009, pp. 167–182.

[74] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A comprehensive study of convergent and commutative replicated data types," Ph.D. dissertation, Inria–Centre Paris-Rocquencourt; INRIA, 2011.

[75] ——, "Conflict-free replicated data types," in *Symposium on Self-Stabilizing Systems*. Springer, 2011, pp. 386–400.

[76] S. Burckhardt, A. Gotsman, H. Yang, and M. Zawirski, "Replicated data types: specification, verification, optimality," in *ACM SIGPLAN Notices*, vol. 49. ACM, 2014, pp. 271–284.

[77] R. Klophaus, "Riak core: Building distributed applications without shared state," in *ACM SIGPLAN Commercial Users of Functional Programming*. ACM, 2010, p. 14.

[78] 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*. ACM, 2015, p. 6.

[79] P. Alvaro, N. Conway, J. M. Hellerstein, and W. R. Marczak, "Consistency analysis in bloom: a calm and collected approach." in *CIDR*. Citeseer, 2011, pp. 249–260.

[80] 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*. ACM, 2012, p. 1.

[81] S. Burckhardt, D. Leijen, M. Fähndrich, and M. Sagiv, "Eventually consistent transactions," in *European Symposium on Programming*. Springer, 2012, pp. 67–86.

[82] S. Burckhardt, "Bringing touchdevelop to the cloud," 2013, https://www.microsoft.com/en-us/research/blog/bringing-touchdevelop-to-the-cloud/.

[83] W. Brunette, S. Sudar, M. Sundt, C. Larson, J. Beorse, and R. Anderson, "Open data kit 2.0: A services-based application framework for disconnected data management," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 440–452.

[84] Y. Bai and Y. Zhang, "Stoarranger: Enabling efficient usage of cloud storage services on mobile devices," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 1476–1487.

[85] S. Chandrashekhara, T. Ki, K. Jeon, K. Dantu, and S. Y. Ko, "Bluemountain: An architecture for customized data management on mobile systems," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. ACM, 2017, pp. 396–408.

[86] H. Tang, F. Liu, G. Shen, Y. Jin, and C. Guo, "Unidrive: Synergize multiple consumer cloud storage services," in *Proceedings of the 16th Annual Middleware Conference*. ACM, 2015, pp. 137–148.

[87] Y. Zhang, C. Tan, and L. Qun, "Cachekeeper: a system-wide web caching service for smartphones," in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. ACM, 2013, pp. 265–274.

[88] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE transactions on computers*, vol. 100, no. 9, pp. 690–691, 1979.

[89] C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: a workload-driven approach to database replication and partitioning," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 48–57, 2010.

[90] A. L. Tatarowicz, C. Curino, E. P. Jones, and S. Madden, "Lookup tables: Fine-grained partitioning for distributed databases," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 102–113.

[91] J. Jing, A. S. Helal, and A. Elmagarmid, "Client-server computing in mobile environments," *ACM computing surveys (CSUR)*, vol. 31, no. 2, pp. 117–157, 1999.

[92] D. Barbará, "Mobile computing and databases-a survey," *IEEE transactions on Knowledge and Data Engineering*, vol. 11, no. 1, pp. 108–117, 1999.

[93] P. Serrano-Alvarado, C. Roncancio, and M. Adiba, "A survey of mobile transactions," *Distributed and Parallel databases*, vol. 16, no. 2, pp. 193–230, 2004.

[94] A. A. Imam, S. Basri, and R. Ahmad, "Data synchronization between mobile devices and server-side databases: a systematic literature review," *Journal of Theoretical and Applied Information Technology*, vol. 81, no. 2, p. 364, 2015.

[95] T. Soyata, H. Ba, W. Heinzelman, M. Kwon, and J. Shi, "Accelerating mobile-cloud computing: A survey," in *Cloud Technology: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2015, pp. 1933–1955.

[96] B. A. Milani and N. J. Navimipour, "A comprehensive review of the data replication techniques in the cloud environments: Major trends and future directions," *Journal of Network and Computer Applications*, vol. 64, pp. 229–238, 2016.

[97] L. B. Bhajantri and V. V. Ayyannavar, "Cognitive agent based data synchronization in ubiquitous networks: A survey," *International Journal of Advanced Pervasive and Ubiquitous Computing (IJAPUC)*, vol. 10, no. 2, pp. 1–17, 2018.

Appendix

TABLE IV: Summary of reference designs

Framework	CMP	HU	Sync Protocol	Conflict Resolution
Coda [3] ∗	E ,PR	O	Use callbacks based on RPC	System level conflict resolution
Bayou [4]	E , TI	T, O	Two way Log exchange protocol	Application level resolution
Rover [68]	- , AC	O	↕ Two way using relocatable dynamic objects (RDOs) and queued remote procedure call (QRPC)	Object consistency is provided by application-level locking or by using application-specific algorithms
Perspective [73] ↔	E , PR, TI	O	↕ Modified update log to limit the exchanges to only the needed information,	Use two phase conflict resolver(Pre-resolver and a full resolver).
Cimbiosys [17] ↔	EF, PR,TI	O	↕ Use Eventual filter consistency and technique of Eventual knowledge singularity ,a compact synchronization-specific state for making economical use of bandwidth and system resource	Any device whose filter selects both conflicting versions may detect the conflict and either resolve it automatically or store both versions pending manual resolution
PRACTI replication [2] ↔	S, C, PR, AC , TI	O	Two types of communication 1.causally ordered Streams of Invalidations and 2. unordered Body messages. The protocol for sending streams of invalidations use log exchange protocol	Interface for detecting and resolving write- write conflicts according to application-specific semantics
SwiftCloud [59]	E , C, RB , PR	CRDTs	CRDTs do not require consensus on replica reconciliation	CRDTs can be updated without synchronization
Indigo [78]	EC , C, S, RB , PR , AC	CRDTs	CRDTs do not require consensus on replica reconciliation	CRDTs can be updated without synchronization
Izzy [43]	E , PR	T	Provide preferences to applications for setting on when and how to transfer data	Supports per-row versions for synchronization and conflict resolution
Simba [20] ∗	S, C , E , PR , AC	T + O	Versioning scheme which uses compact version numbers instead of full version vectors.3 consistency models also rely on this versioning scheme.Objects are stored and synced as a collection of fixed-size chunks for efficient network transfer.	Conflicts are stored in a separate conflict table until explicitly resolved by the user.
Sapphire [53]	S, E , PR	O	Cross Sapphire Objects (SOs) management and cross-SO transactions are NOT supported.Doc not clear about conflict resolution and handling	One of the deployment manager (DM) for the provide support for Dynamic allocation of load-balanced M-S replicas w/ eventual consistency. The extension OptimisticTransactions provide transactions with optimistic concurrency control, abort on conflict.
Mobius [44]	FSC , PR , AC	T	use a publish subscribe mechanism (Yahoo Message Broker (YMB))for many features such as asynchronous replication.	Supply the client with three kind of information to resolve conflict and can discard its local changes or issue a new update.
Key-Vaue Store with BloomL [80]	E	Built-in lattices	State of two replicas synchronized by exchanging their 'kv-store' maps.	The 'lmap' merge function automatically resolves conflicting updates made to the same key.
TouchDevelop [82] ∗ [60]	E	O (Cloud Types)	Special cloud types data is automatically shared between all devices, and is automatically persisted both on local storage and in cloud storage.	Automatic conflict resolution and no special code needed to handle merging
Middleware for client-centric consistency [70]	MRC , RYWC , E , PR	O	Protocol reads data from the storage system and adds a copy of that datum to its local cache, if the cache does not already contain that datum in that exact version (identified by its vector clock).	The application must take care of the data violation-handling task by reloading data
Open Data Kit 2.0 [83] *	E	T	Single database row is the base unit and use a small granularity of change-tracking to enable smaller data transmission.	User must resolve the conflict by either taking the server's change, the local change, or mix of the local change and the server change
StoArranger [84]	E , PR	T	Delay and batch cloud backup requests from apps to minimize the impact of transmission promotion/tail energy	Conflict detection is based on the folder meta data with folder sync APIs.
Unidrive [86]	S, E, PR	O	Metadata of content is stored in cloud and synced to all clients using a quorum based distributed locking protocol and chunking technique	It support multiple Consumer cloud storage in which synchronization logic is purely implemented at client devices and all communication is conveyed through file upload and download operations
QuickSync [52] ∗	E	O	Sync efficiency is improved by using three key components, the Network-aware Chunker, the Redundancy Eliminator , and the Batched Syncer	Do not address consistency or conflict handling but improves Sync efficiency.
Parse Server [62] ∗	S, E, PR	T	The Parse Server SDKs provides a local datastore which can be used to store and retrieve the PFObjects by 'pinning' process	Server side conflicts are handled using generic 'beforeSave' function in the cloud. Applications are responsible to handle conflicts.
StackSync [51]	S, E, PR	O	Sync protocol is based on RPCs or method calls by ObjectMQ middleware	A copy of the conflicted document is created and user needs to decide about this.

Continued on next page

TABLE IV – continued from previous page

Framework	CMP	HU	Sync Protocol	Conflict Resolution
Dropbox [48]	RAW	O	The basic object in the system is a chunk of data with size of up to 4MB. Files larger than that are split into several chunks. Reduces the amount of exchanged data by using delta encoding when transmitting .	File meta data has a unique revision identifier used to detect changes and avoid conflicts
iCloud with CloudKit [47]	S, E	O	CloudKit APIs support to fetch for only the changes since the last time it updated.	iCloud server returns the error code with objects that contains the different versions of the conflicting record and user can perform whatever resolution logic is needed to resolve the conflict
Amazon Dynamo [5]	E, PR	O	Implements an anti-entropy (replica synchronization) protocol and Merkle trees for faster and to minimal the amount of data transfer.	It makes use of object versioning and application-assisted conflict resolution
Bluemix Mobile Cloud Service [63]	E, PR	O	Applications use Cloudant Sync (an Apache CouchDB^{TM} replication-protocol-compatible) to store, index and query local JSON data on a device	It is the application's responsibility to detect the conflicts and resolve them based on conflict details
Firebase [65]	E, PR	O	Proprietary sync protocol and lack of documentation	Timestamp based conflict resolution
Kinvey [66]	S, E, PR, AC	O	Delta Sync allows only sync of new and updated entities only.	The libraries and backend implement a default mechanism of "client wins", which implies that the data in the backend reflects the last client that performed a write. Custom conflict management policies can be implemented with Business Logic.

Note: See Table-I for the list of symbols used.
*: open-source