There has been a surge in use of mobile devices across the globe for using different services. Mobile devices access data on the server, which at times need responsiveness. Each of the users can have these applications on different devices, such as mobiles, laptop, tablet. Thus, to ensure this responsiveness, applications maintain local replicas of the data, which can be accessed in case of slow or unavailable server connection. These local replicas can be used to read or update the data as and when needed. To sync or resolve the consistency between these local replicas and the data on the server, thereby providing eventual consistency is very tough. Typical challenges of sharing the data in the local replicas and handling eventual consistency, comes with it own set of challenges of:
Representation - Different app platforms will have to maintain different web services for the data representation.
Consistency - Changing different local replicas, leads to conflicts while resolving them after a while, at the server.
Change sets - Thus, to reliably resolve conflict, the clients are required to maintain logs of different versions of the data at local replicas, so these logs can be used to resolve conflicts at the server.

The paper proposes uses of Cloud data types, which are programmed at the server, to achieve eventual consistency, thereby abstracting details of servers, networks, caches, and protocols. Different data types such as simple ones: CString (cloud string) and CInt (cloud integers) and structured ones: (Array, Entities, Sets) are defined at the cloud level. Different aspects of the data modeling such as the declarations, queries, updates are all directly integrated at the programming language level, thus, there is only one data format across platforms and one program controlling the data. This program can read / update the data directly, without need of copying or blocking. Revision diagrams, similar to the version control systems, are used to guarantee the eventual consistency. The main revisions of the data are stored in the cloud, whereas the mobile devices store the local revisions. Thus multiple versions can be replayed and can be merged into other revision, on conflicts by traversing the revision diagrams of the data.

Fork-join automata provides the implementation of the abstract update and query operations, and fork join methods. Each of the local replicas are forked from the main server, and after a period of time, are joined in the main server. This automata is defined by a tuple consisting of a set of query operations, set of update operations, an initial state (Server), set of states, functions for splitting the states, and merging states. Different fork-join automata can have different definitions of revision diagrams. Different functionalities such as flush-pull, flush-push, yield-pop, yield-push, yield-pull are used to achieve consistency and synchronization between data on different local replicas and the main server. yield which are non blocking operations propagate changes from a local replica to other replicas, and from other replicas to current replica. flush which are blocking operations, are used to guarantee synchronization between

replicas. sync and create are a few different rules for the servers. new, delete and operation expressions are a few different rules for the local replicas.

Strong Points:
1). Cloud types enable ease of the data sharing in the cloud for the client applications, as it requires the client to just write efficient code, without writing data structures at different client platforms, thereby minimizing the development time.
2). Conflict resolution is automatic, due to preconceived notions of the conflicts that will arise, thereby eliminating the need to write extra code for resolution.
3). Due to use of fork automation, Cloud types provide optimized fork and join implementations, and also less space consuming representation of logs. Also, any schema and extensions can be supported in future.

Weak Points:
1). The paper provides little information for an example of defining a definition of revision diagram, as we can specify different revision diagrams.
2). Paper provides no information about the barrier function.
3). Paper fails to provide more detail about the difference between the performance with CRDT(Conflict-Free Replica Data Types) and Bayou's weakly consistent replication.

Question:
What could be the performance impact for using such data types for eventual consistency?