High-Performance Fault-Tolerant Data Caching and Synchronization Architecture for Smart-Home Mobile Application

Chapter · January 2016

DOI: 10.1007/978-981-10-1409-3_15

Ramesh Guntha
Amrita Vishwa Vidyapeetham
12 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Aryadevi Remanidevi Devidas
Amrita Vishwa Vidyapeetham
18 PUBLICATIONS **45** CITATIONS
SEE PROFILE

Maneesha Vinodini Ramesh
Amrita Vishwa Vidyapeetham, Amritapuri, Ko…
117 PUBLICATIONS **431** CITATIONS
SEE PROFILE

High-Performance Fault-Tolerant Data Caching and Synchronization Architecture for Smart-Home Mobile Application

Ramesh Guntha, Aryadevi Remanidevi Devidas, Maneesha Vinodini Ramesh

Amrita Center for Wireless Networks and Applications, Amrita School of Engineering, Amritapuri Campus, Amrita Vishwa Vidyapeetham University
Kollam, 690525, Kerala, India
{rameshg, aryadevird, maneesha }@am.amrita.edu;

Abstract. Mobile devices are becoming the preferred choice for internet access as they are getting increasingly powerful and affordable. But because of lack of ubiquitous high bandwidth wireless internet, many mobile applications suffer from performance and reliability issues while accessing data from the servers. While many of the applications use caching mechanism to store data locally on mobile device to improve data access performance, they are not thoroughly focusing on related areas such as two-way data synchronization, fault tolerance and recovery, offline mode support and real-time update support. And hence even many reputed applications show inconsistent and out-of-date data; especially during network and battery outages. In this paper we propose usage of Replication process, Pending data process, targeted data update broadcasts to solve the above issues in the software architecture of Smart-Home project.

1 Introduction

Mobile devices are becoming increasingly powerful in terms of processing power, storage and memory, and they are becoming increasingly affordable to common man. But the availability of ubiquitous high bandwidth wireless network is still a dream, especially in the developing countries. Apart from bandwidth, many mobile devices also suffer from poor battery backup.

Because of bandwidth issues many mobile applications suffer from performance and reliability issues while accessing data from servers. Many mobile applications often use local data caching on mobile device to avoid going to server for every data request, but they fail to thoroughly take care of related functionalities such as synchronization, replication, fault tolerance and recovery, offline-mode etc. Due to this, even many reputed applications like Twitter, Instagram, Evernote, Dropbox etc., display inconsistent or out-dated data to users under certain fault conditions, some of them do not allow offline mode activity, some of them hang under fault conditions [1], [2], [3].

The Smart-Home architecture presented in this paper shows how these issues can be addressed. The relevant data is stored on the mobile and the Replication and Pending data processes which run on the mobile device perform the data synchronization with the server. The Replication process performs the server to mobile data synchronization. User's data updates on mobile are synchronized to server by the Pending data process. These data updates can be done in offline mode as well. The targeted data update broadcast mechanisms ensure that user sees the data updates happening to relevant entities on server in real-time. They take care of complex inter-dependent tables as well. Conflicting data updates to server are resolved based on the server time-stamp. Only the data related the particular user is considered for synchronization. The sync mechanisms are triggered on periodic-basis, based on events such as network recovery, application start, server data updates, and permission changes, thus are able to recover from network and battery outages quickly.

The rest of the paper is organized in following sections. Smart-Home application description, deep-dive into the proposed architecture, analysis of related work and conclusions.

2 Smart-Home Application

Lot of power wastage occurs due to carelessness of users. Bigger organizations can save thousands of dollars per month if they properly manage the power consumption. Smart-Home application is being developed to take care of this need.

2.1 Functionality

Smart-Home application helps in controlling the electric equipment in home/office building(s) it has both mobile and web client interfaces. The users can configure various communities, buildings, floors, rooms, common areas which they control. Under each of these locations they can add the electric equipments to be monitored and controlled. Users get the real-time status updates and the power consumption statistics for each entity they have access to. Users can control the equipment from both mobile and web applications. The updates done in offline mode on mobile would be propagated once the network becomes available. Users can also set pre-configured rules to control how much power can be consumed at an entity level.

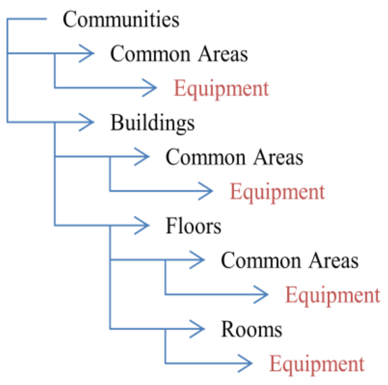

Fig. 1. Entity relations in Smart-Home application

2.2 Entity Relations

Fig. 1 describes the entity hierarchy in Smart-Home application. Community is the highest. Communities may have some common areas like streets, grounds etc. A community contains a set of buildings. Buildings may have some common areas like lobby, back-yard etc. A building contains set of floors. Floors may have some common areas like corridors, lobbies etc. A floor contains set of Rooms. Rooms do not have common areas. Finally the electrical equipment is placed under rooms and various common areas.

2.3 Roles and Permissions

Smart-Home application has two types of roles, Admin and Viewer. These roles are assigned to a particular user at a given entity level and the same privileges apply for all the child entities. Admin can control entities and view power consumption data and status. Admin also can create child entities under the entity. Viewer can only view data and status for the given entity and its children. If a user is admin for a Building, then he/she is the admin for Floors, Rooms and Equipment in that building. A given entity can have more than one user as Admin and/or Viewer, similarly, a given user can be Admin and/or Viewer for more than one entity.

2.4 Data Estimations

In a given community, such as a university campus, there are 10s of buildings, each building would have 3-4 floors, each floor would 10-20 rooms, each room would have around 10 electric equipment such as lights, fans, air conditioning equipment etc. In addition to this there could be several electric equipment like lights, water purifiers, cameras etc in common areas such as grounds, streets, lobbies, corridors etc. So for a given community we are looking at around 10,000 to 20,000 equipment. This could amount to huge data if status and power consumption details for each equipment as it's turned on/off is stored and broadcasted in real-time.

3 Smart-Home Architecture

In this section we elaborate on overall system architecture, mobile client database, replication process, pending data process, real-time data updates to mobile clients.

3.1 System Architecture

The system provides both mobile and web interfaces, supported by scalable, high performance, event-driven, and robust server architecture (Fig. 2). The data is exchanged between client and server in the light-weight and universal JSON format. On the server side high performance NGINX used as web server to handle all the media. The Application server is powered by high performance, highly flexible, extensible, robust, and heavily adopted NodeJS server. Express package is used as command router inside NodeJS, and Sequelize as the ORM (Object Relational Mapping) package to interact with MySQL database. The application logic is written in reusable modular fashion. The logic consists of periodic event code to run any scheduled tasks and broadcasting code to push the relevant data updates to clients based on user's permissions, thereby reducing bandwidth and resource consumption. The DSS module runs the various aggregation algorithms on the power consumption data and also executes various configured rules and triggers various alerts and commands to automatically control the electronic equipment.

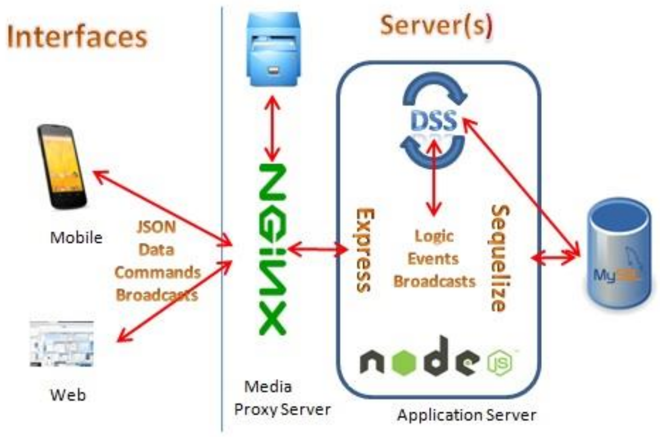

Fig. 2. System architecture of Smart-Home application

3.2 Mobile Database Design Considerations

To provide high performance user experience irrespective of network bandwidth and to support offline mode, all the relevant data for the user is cached on the mobile database. The mobile database is closely mirrored to server database so that it can support all the functionality by accessing the local database itself. In addition to application tables mobile database also has a set of Replication tables for synchronizing data from server to mobile and Pending tables for synchronizing data from mobile to server.

The "modifiedDate" Field. Every row of every table of Smart-Home application contains the last updated timestamp for the row, stored in the modifiedDate field. It plays a critical role in the two-way data synchronization.

3.3 Replication Process for Server to Mobile Synchronization

Replication process depends on Replication tables for synchronizing data from server to the mobile database. Only the relevant data to the user is replicated.

E.R Diagram of Replication Tables. The ReplicationTable table shown in the E.R diagram (Fig. 3) contains list of tables to be replicated on the mobile. The table names are listed in the replicationTableName field. The replicationParentTableId is a self referencing foreign key pointing to the parent replication table record. This relation is used to order the replication requests to server, parent first and then child, and so on. The lastDataDate field stores the "modifiedDate" of the record till which the data from server for this table is replicated on mobile. The lastReplicationDate stores the last successful replication run for this table.

The ReplicationRun table stores the information about all the Replication processes runs. The startDate field stores the date-time at the start of a run, similarly endDate stores the end date-time. The triggeredBy stores the event which triggered that particular run. The replicationStatusId stores the status of replication process. It can be either "Success", "In Progress" or "Failed".

The ReplicationRunDetail is the child table of ReplicationRun table through the replicationTableId foreign key. It stores the replication run information at the table level. The lastDataDate stores the latest modifedDate of a given table replicated from the server. The requestDate stores when the process requested data from server. The responseDate stores the response time from server. The responseStatusId can be either "Success" or "Failed". The responseNumRows indicate the total number of rows promised by server in this request. The responseActualRows indicate the actual number of rows downloaded from server. The replicationStatusId stores the status of replication process for this table. It can be either "Success", "In Progress" or "Failed".

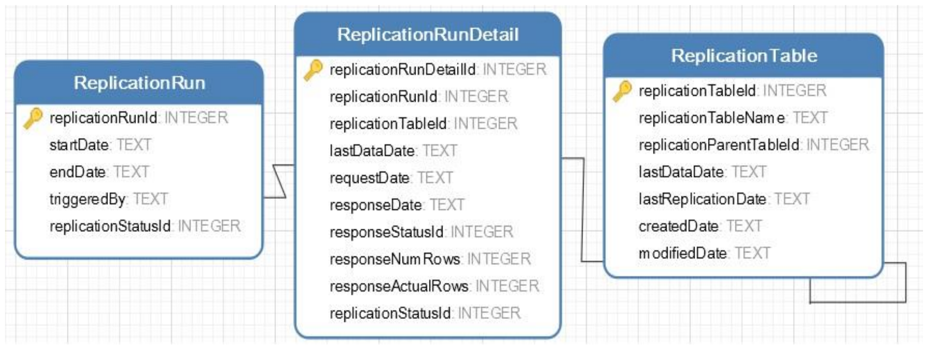
ReplicationRun
replicationRunId: INTEGER
startDate: TEXT
endDate: TEXT
triggeredBy: TEXT
replicationStatusId: INTEGER
ReplicationRunDetail
replicationRunDetailId: INTEGER
replicationRunId: INTEGER
replicationTableId: INTEGER
lastDataDate: TEXT
requestDate: TEXT
responseDate: TEXT
responseStatusId: INTEGER
responseNumRows: INTEGER
responseActualRows: INTEGER
replicationStatusId: INTEGER
ReplicationTable
replicationTableId: INTEGER
replicationTableName: TEXT
replicationParentTableId: INTEGER
lastDataDate: TEXT
lastReplicationDate: TEXT
createdDate: TEXT
modifiedDate: TEXT

Fig. 3. E.R diagram of replication tables

Replication Algorithm. Fig. 4 shows the replication algorithm. Upon starting the replication process, it puts an entry in the Replication table, and then it finds all tables to replicate from the ReplicationTable's replicationTableName field. Then for each of the tables it makes a server call to fetch the rows which are updated later to the lastDataDate. It records the details of each table replication in the ReplicationRunDetail table. Once all the tables are replicated, it would update the ReplicationRun table.

Incremental Replication of Relevant Data. The replication process is incremental, as it fetches the rows from the server for the corresponding table, only if there are any rows which have modifiedDate later to the lastDataDate in ReplicationTable. This lastDataDate is not affected by time differences between the server and mobile, as it's the latest modifiedDate of server records from the previous replication run. In other words, lastDataDate is based on the server time.

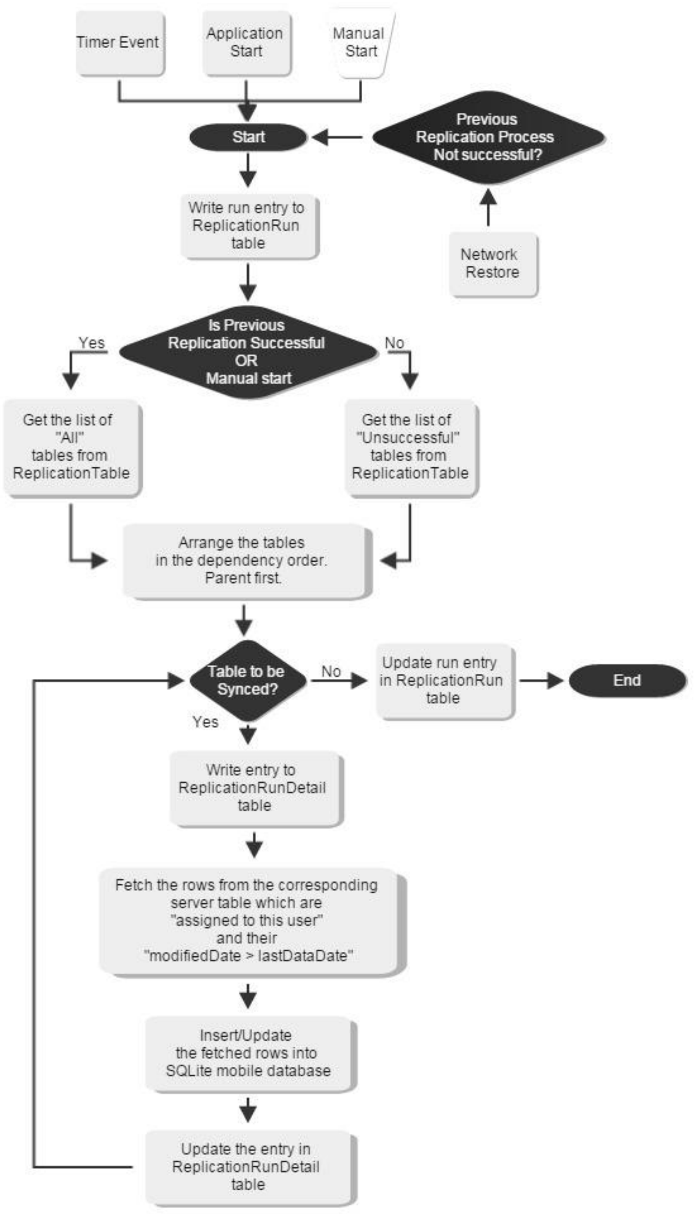
Timer Event
Application Start
Manual Start
Start
Previous Replication Process Not successful?
Network Restore
Write run entry to ReplicationRun table
Is Previous Replication Successful OR Manual start
Yes
No
Get the list of "All" tables from ReplicationTable
Get the list of "Unsuccessful" tables from ReplicationTable
Arrange the tables in the dependency order. Parent first.
Table to be Synced?
No
Update run entry in ReplicationRun table
End
Yes
Write entry to ReplicationRunDetail table
Fetch the rows from the corresponding server table which are "assigned to this user" and their "modifiedDate > lastDataDate"
Insert/Update the fetched rows into SQLite mobile database
Update the entry in ReplicationRunDetail table

Fig. 4. Algorithm of replication process

The replication process only fetches the entities for which the user is either Admin or Viewer. When the user permission is added or revoked from a given entity, the data comes in through real-time update, and then the real-time update process adds or removes the entity from local database. In case of permission removal, the children of the entity are removed first and then the parent.

Replication of Inter-dependent Entities. Almost any database has parent-child relations between the tables enforced by foreign key constraints. So if a child record is inserted without its parent record being present, it would result in an error. So the replication process must take care of this order. The replicationParentTableId field points to the parent table record in ReplicationTable table, using this field the process replicates parent tables first and then replicates the child tables. This ensures that replication is done without violating foreign key constraints.

Replication Triggers, Fault Tolerance, and Recovery. Replication process can be triggered by periodic timer event, application start, user or network restore event. This ensures immediate recovery from network failures and battery power failures. ReplicationRun table stores whether the previous run failed or successful, so upon network recovery, the process checks this table to see if previous run got stuck, if so, it resumes the replication process. The ReplicationTable stores the table level replication status, so that it can pick-up at which table it's stuck and resume the replication process from then onwards. This ensures no wastage of network and computing resources.

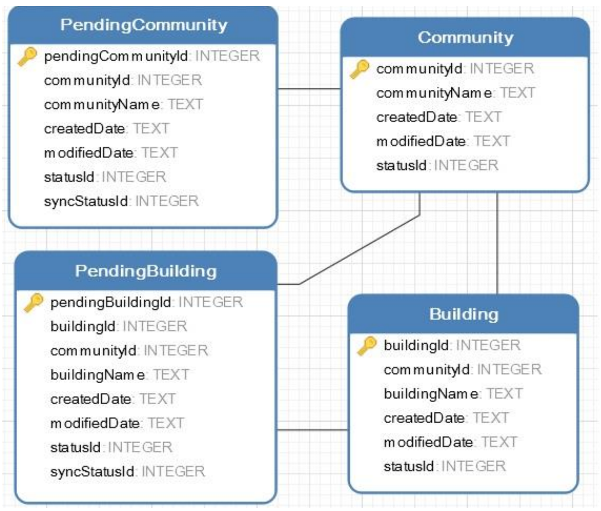

Fig. 5. E.R diagram of pending tables

3.4 Pending Table Process for Mobile to Server Synchronization

The Pending table process propagates updates made by the user on mobile to server. When the user makes an update to any entity, the update is written to the corresponding pending table. Once the update is propagated to server successfully, it is applied to the main table on the mobile. User is allowed to make any number of updates at once, but only the latest update is sent to server and all the others are marked as Overridden.

E.R Diagram of Pending Tables. The E.R diagram in Figure 5 shows pending tables for Community and Building entities. The pending table closely mirrors the corresponding main table. The Pending table has a new primary key instead of the main table's primary key. This is to allow storage of historical updates in Pending table. The syncStatusId is used to store the status of synchronization for the update. It can have any of "Pending", "In Progress", "Retry", "Success", "Failed" or "Overridden".

Lifecycle of "syncStatusId" field.

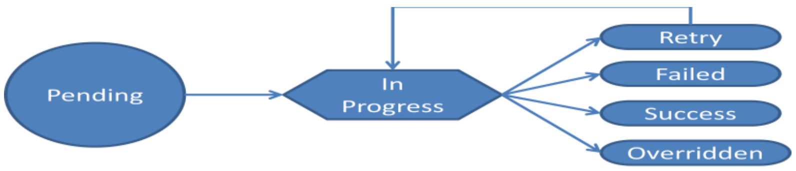

Fig. 6. Lifecycle diagram of "syncStatusId" field

When the user makes an update, the syncStatusId has the value "Pending" in the PendingTable. Once the process picks-up the change for server synchronization it is marked as "In Progress". If the network is down or server is not reachable, then the status is changed to "Retry", so that it can get picked up by later Pending process run. If the update is failed at Server for any other reason such as "Unique constraint violation" etc, it's marked as "Failed". If the server's record has a later 'modifiedDate' compared to the mobile record being synchronized, it will reject the update with an "Overridden" status. If the update goes through successfully, the syncStatusId is updated with "Success" and the changes are copied to the main table. The Pending table records with "Overridden" and "Failed" status are not retried.

Conflict Resolution. The data on mobile could become out-of-date due to network issues preventing the replication process or real-time update process. If the user updates such stale data it would be rejected by server as Overridden as the server data has later "modifiedDate" compared to that of the mobile update. Similarly, the updates made by the user might get delayed to reach the server due to network or battery outages, or some other user might have updated before this user. In this case also the server would the update as "Overridden". This ensures that conflicts are resolved with policy of rejecting stale data updates.

Synchronization of Inter-dependent Entities. As elaborated in section "Replication of Inter-dependent Entities" above, the replicationParentTableId field from ReplicationTable gives the order of synchronizing the updates to server, following which the Pending table process would avoid foreign key constraint violations.

Pending Table Process Triggers, Fault Tolerance, and Recovery. The pending table process is triggered on periodic basis, on application start, on user update and on network recovery. This ensures that updates are sent to server at earliest possible time to prevent stale data updates as much as possible. This also ensures that pending process resumes immediately after a network or power restoration, resulting in quick recovery from faults. Since pending data is copied to main table only on successful synchronization, network or battery faults, which might stop ongoing pending process in middle, would not affect application functionality, so that's how fault tolerance is achieved.

3.3 Real-Time Mobile Data Updates

Apart from the Pending data process, the mobile data gets updated by real-time targeted broadcast updates from the server. As soon as the user logs into the mobile application, it establishes a persistent stateful socket connection with server, using which the server can push fresh relevant updates that occur on the server database. Since it’s an event based, push mechanism, it involves very less network resources and results in real-time data synchronization from server to mobile. When the user updates, inserts or deletes any data on the server either through web application or mobile application, the server derives all the relevant users to be notified using the access permissions, and broadcasts these updates to their mobile apps for synchronization.

4 Related Work

Research to improve mobile application performance by caching data on mobile has been going on for couple of decades [7]. But still as noted by [1], [2], [3] many reputed mobile applications are unable to display consistent and up-to-date data in case of network or battery outages. References [1], [2], [3] take a novel approach in their research to implement a database abstraction layer both on mobile and cloud server for performing data synchronization, replication which is resilient to faults. Since their approach abstracts the database completely, there is no visibility into how the data is stored, hence this approach does not give much freedom to developers and business analysts to access and analyze data from the server. It's reasonable to assume that any enterprise application's team would want to access server database for performance turning, reporting, and analyzing the data for multiple other functionalities. Also [1], [2], [3] do not outline how inter-dependent complex table structures are specified and how to enforce critical database integrity constraints like unique key, not null, or foreign key. Our architecture shows developers how to achieve the synchronization, reliability, fault tolerance and performance while still keeping the complete access to their databases and hence has much better chance of getting adopted. Reference [4] approaches the data-caching problem to store only the relevant data on the mobile in-terms of user own data and user's relationship data, which the user most likely to access on frequent basis, thereby ensuring high hit-ratio. Reference [5] provides a complete mobile-middleware-server architecture for data caching, replication and network fault recovery, but their approach is not scalable as it centralizes the replication and synchronization process on middle-ware server instead of decentralizing on to the mobile system. Due to this updates might take a lot of time to reach mobile and server under high load conditions. Reference [6] surveys various mobile data solutions and categorizes them into the segments of extended client-server systems, data caching systems and adaptive systems. Reference [7] discourages replication anywhere-anytime-anyway approach as it is not scalable because it requires eager and strict mechanisms of caching and transactions. This paper achieves the anywhere-anytime-anyway replication with lazy mechanism with much simpler architecture, which can be easily implemented. Reference [8] suggests a pre-write and pre-read mechanism and mobile host and mobile support station machines for data synchronization which is similar to our Pending data process, but our pending data process outlines mechanisms to synchronize the dependent tables and triggering mechanisms with much simpler conflict resolution mechanism which helps users to understand what went wrong when the updates get overridden or rejected.

5 Conclusion

The in-depth explanation of Smart-Home architecture in this paper demonstrates a way to implement mobile database caching so that it also takes care of two-way data synchronization, conflict resolution, dependent table synchronization, and broadcasting of real-time targeted updates. It shows that architecture is distributed and takes full advantage of storage and processing power of the mobile devices to deliver high performance user experience. It also demonstrates that architecture is fault tolerant and recovers as soon as network or power becomes available.

6 Acknowledgement

This work is funded by Amrita Vishwa Vidyapeetham University as part of “Smart Home” project.

References

1. Y. Go, N. Agrawal, A. Aranya, and C. Ungureanu. Reliable, Consistent, and Efficient Data Sync for Mobile Apps. In FAST, 2015.
2. D. Perkins, N. Agrawal, A. Aranya, C. Yu, Y. Go, H. Madhyastha, and C. Ungureanu. Simba: Tunable End-to-End Data Consistency for Mobile Apps. In Proceedings of the European Conference on Computer Systems (EuroSys ’15),Bordeaux, France, April 2015.
3. N. Agrawal , A. Aranya , C. Ungureanu, Mobile data sync in a blink, Proceedings of the 5th USENIX conference on Hot Topics in Storage and File Systems, p.3-3, June 27-28, 2013, San Jose, CA
4. Q Xing, Y Li, J Wang, Y Han, A User-Relationship-Based Cache Replacement Strategy for Mobile Social Network - Frontier of Computer Science and Technology, 2015 - ieeexplore.ieee.org
5. Ricardo A, Nuno L, Ana A, Lino F and Paulo N: Creating and optimizing client-server applications on mobile devices. In IE 2013.
6. Jing, J., Helal, A., Elmagarmid, A., “Client-server computing in mobile environments”, ACM Computing Surveys, vol. 31, n° 2, 1999.
7. Gray J., Helland P., O’Neil P., Shasha D., “Dangers of replication and a solution” , ACM Int. Conf. On Management of Data (SIGMOD), 1996.
8. Madria, S. K., & Bhargava, B. (2001). A transaction model to improve data availability in mobile computing. Distributed and Parallel Databases, 10(2), 127-160.