Cloud Mobile Storage for Mobile Applications

The Dynamic Storage Injection Pattern

https://doi.org/10.3991/ijim.v13i03.8086

Fabrice Mourlin (✉)
UPEC University Paris, Paris, France
fabrice.mourlin@u-pec.fr

Jean-Marc Farinone
CNAM, Paris, France

Abstract—The data storage in mobile cloud computing domain is an important topic because of the large amount of data devices can provide. Different kinds of sensors retrieve information and send them to data storages. But the storage location can move even dynamically during their uses. So, these changes must be managed at runtime. In the other part, Android devices are good computers to acquire information and send them to a storage area. These devices are well-known, ergonomic and useful machines sorely used. In this paper, we present a software architecture from Android devices to Cloud storage areas to put down data when the data stores are dynamically moving. The different parts of these distributed applications are built around Android with JSON messages for REST services and moving NoSQL databases in the Google app engine. The exchanges between the cloud and the Android devices use the Google Cloud Messaging (GCM) protocol. Furthermore, even without network access, the saving is made locally in the smartphone. The synchronization with distant databases is automatically made when the network becomes again available.

Keywords—Mobile Cloud Computing; moving data storage; REST; Cloud computing service model; NoSQL cloud databases; Sync API for Android.

1 Introduction

Cloud infrastructure give hardware, data storages, APIs to manage them and so we can develop software for customers to put in it. We can use it as virtual supply and these parts are well known as IaaS (infrastructure as a service), PaaS (platform as a service), SaaS (software as a service) [1]. On the other hand, the top daily activities on smartphones and tablets are about email access, shopping on the Internet and health coaching [2]. The mobile users are now spending many times consuming digital media within mobile applications. Therefore, these applications need to save their states even if there are network disruptions. Furthermore, the mobile applications must include a software strategy to track whether a network connection is available. In case of the lack of network, a local backup can be done to keep trace of the useful business

data. Finally, the remote database can dynamically change at runtime and the mobile application must know dynamically these changes. If a database is available, applications, customers, and business users, can access it. However, any condition that involves the remote database inaccessible causes the failure of the remote persistence tier and the storage must be momentarily made locally. The required availability of an application will vary from system to system, but it must be predicted.

Our work is about the dynamically changing of data storage accessible and supplied by smartphones. For example, Leon and al. [3] change the servers and so the data storage to address the underutilization of these resources and so reducing costs. Zhangbing Zhoua et al. also treat the subject of replaceability assessment of resources in [4] and Espadas et al. propose an elastic model to use underutilized resources [5].

A network problem can involve unreachable remote databases. So, the location of the storage must be known. Another reason to change the location is the poor performance of the current database to another instance. The extreme limit occurs when no remote database is available, so a local persistence should be achieved. Therefore, the dynamically change of a database is a crucial problem.

In the context of mobile applications, the Cloud access is a crucial feature. For example, it ensures the business data access when the network is available. However, it also introduces two aspects of the communication. A Cloud point of view occurs for the selection of the database access point, and then this feature is send back to the mobile application. In addition, a mobile point of view appears about the existence of this database access or not. In the first case, it means that the work session can be saved in a remote manner. In the second case, the work session should be saved locally. Depending on the activation of the connection access, the local state will be updated to the remote database access and the local state will be erased.

So data storages are one of the main part of an application and it can be used for instance to contain the isolated applications from the OS and the low level VM when installing the whole virtual machines [6]. Our architecture is useful for example, when a user or a device need to obtain many data and store them in data storages. Such a solution helps to continue mobile users to keep a shopping session and more generally to manage a working context. It is the case when a person wants to make an inventory of a shop or a hangar. He uses his smartphone or his adapted device to read information linked to the material (QR code, barcode), and send them to a storage area. This data procurement must be made quickly. The same model must be done to obtain the knowledge of a boat shipment just arriving in a port harbor by using RFID technology, RFID tags being attached to articles. We have something similar for managing a crowd of person passing in given places by using NFC tag loaned to them in a museum for example.

A set of sensors, which spot a lot of data about cars on a highway and require to send information to a storage place need to use our kind of architecture. If the network link is damaged, these sensors must be coupled with an intern storage area. Therefore, these sensors are linked with micro programs inside micro-controllers put for example on Arduino card or microcomputers as Raspberry card. In these all cases, the local database is a kind of cache we use temporally to be flushed when the network is already available.

The treatment of the data is made on server side but, generally the problems to access or even use a remote database are not approached. In this paper, we propose an architecture were these problems (network availability, dynamic databases changes) are solved. Moreover, our architecture describes and implements a dynamically change were data are stored and describe what it must be made when the network is not available. So, we define protocols for moving databases. The specifications of data storage are made but different implementations can be dynamically chosen or injected. It is a dynamic storage injection design pattern closed to the injection dependence design pattern. Therefore, in this paper we also define an injected storage architecture.

In this paper, we use the whole stack of the services model of the cloud computing: Software as a Service (SaaS) with GCM, Platform as a service (PaaS) with our applications service appengine [7] and Infrastructure as a Service (IaaS) for the data bases in the Google cloud. In addition, of course, we developed cloud clients.

In the following of this paper, we present related works on this subject in the second section. In the third section, we introduce our general software architecture. It has two parts, a cloud part presented in the fourth section and a mobile part in the fifth section. We indicate some futures works and ideas on the subject in the sixth section and draw a conclusion.

2 Related Work

This section introduces references to existing works, which are useful to understand our approach and our contribution to the mobile software engineering.

2.1 Mobile Cloud application: service access everywhere

Cloud computing is defined as the ability to provide a set of servers and services to the large set of users. As a software point of view, mobile cloud computing allows to use services for mobile clients anywhere the mobile end users are. They use information technology as a service over the network and can consider that a data model is managed in a cloud and a view can be obtained remotely for a mobile device. Today, this kind of architecture is become popular. Several well-known applications such that Snapchat and Facebook [8], Instagram [9] and so on are a reference of such kind of applications. A user has access anywhere to any application of the suite using a mobile device.

At the design step, this choice has consequences on the software architecture. The back-end software is deployed into a cloud and visible anywhere through the network. Many mobile applications have such a structure and when a user downloads such application, it installs a client part on its mobile device. This one will be used as an ideal network client for the building of all the exchanges. As an example, the Gmail application can be download from a market [10] and then the users read their mails anywhere they are. Waze application [11] is also an example of a graphical applica-

tion whose code is downloaded from a market place and it only displays the responses of the requests.

2.2 MVC architecture evolution (Model View Cloud)

Based on a distributed architecture, the mobile device plays the role of graphical client where the data model is on a remote server. The updates come through network stimuli, which update the display. Because of the important role of the user interface thread activity, the network behavior is managed in other threads and a global asynchronous behavior is considered between the mobile client and the cloud server. For instance, during a travelling by car, a Waze's user receives data about his geographical context, which are displayed on his screen. If the traffic network is stopped, then the display is frozen until a new reception.

2.3 Remote access from mobile device

Often a mobile device plays the role of a client in a distributed application. For instance, users want a remote access to the sensors of a device or an application needs to filter the data from a set of mobile devices. However, the question is how to create a remote access to a server. In that, we consider the device as a resource set where each of them is behind a Web service. However, the Web service technology has evolved, and the REST approach is well accepted in the domain if the embedded systems.

On the server side, the management of the requests is a key feature for such a client connection. When the number of requests becomes high, the mobile service must use threads. The service launches at least on thread per requestor. By the end of a request, the memory is reused by the virtual machine to the next requests.

Several frameworks already exist for programming REST services on mobile device but the Google's framework called Restlet was the first one to provide a portable way of programming. It manages also a set of object mappings over XML, JSON and raw strings

2.4 Synchronization with the cloud

When we are looking for a cloud service provider where we can keep our files and folders, there are confusions on three terms: cloud storage, cloud backup, and cloud synchronization. While all of three serve as a remote place to stow data, the use case for each differs.

The first cloud usage stays the data storage. It is like having an external hard drive online that is accessible anywhere.

Cloud backup differs in that it is usually automated. Depending on the service, it can happen continuously or on a set schedule. When a file is created or modified, the newest version is uploaded and stored on the cloud.

Synchronization means keeping the most up-to-date version of a file or files on two or more devices. It is ideal for collaboration or people who frequently use multiple

devices. Many cloud backup and cloud storage providers have incorporated synchronization into their services. In our context, this occurs when a remote database and a local database of a mobile device need to merge their data or compute their difference.

2.5 Gap covered by our proposed architecture

Some technologies address on data treatment in the cloud and in the mobile cloud computing domains. For example, Dropbox and Google Drive offer convenient file syncing and sharing. Some papers are interested by the security (availability, reliability, fault-tolerance) of the data as in [12] or data integrity, data confidentiality, and availability in [13]. Some systems as HDFS [14] and ceph [15] are built to treat very well these particularities by using redundancy and tools to manage it. They are good systems for big data technologies. In [16], Lieyun Ding et al. consider the problem of large data management without the aspect on the access from a mobile device. In [17], B. Jiang et al. treats the synchronization data coming from mobile devices with the cloud and we address in this paper. But we didn't find papers which discuss about the dynamic moveable data bases in the cloud so the moveable data in the cloud which the main contribution of this paper. Moreover, we study, in this paper, moveable data bases coming from mobile devices so two kinds of dynamic moveable domains.

A new software architecture is proposed by Yen-Hung Kuo et al. in [18] based on the use of services. Also, the data source is exposed through a local or remote service and the client selects one of them depending on local criteria. In that case the authors consider the availability of the client and not the availability of the services. Often the data source is not reliable enough and over a long period of time, a service can reboot. The reliability must be considered in the context of mobile access where the network is a key resource. We propose in the next sections a new architecture which answers to these drawbacks.

3 Software Architecture

It is essential to identify mobile applications over a network. For instance, International Mobile Equipment Identity (IMEI) [19] is an identifier for a smartphone. An Android smartphone can also be a Wi-Fi hotspot and propose an SSID. However, these two ids cannot be used for our architecture. First IMEI is not confidential and must be avoided in most use-cases without limiting required functionalities. Second, we want to use internet network, not only Wi-Fis one. Because the identifier of a mobile device is not enough, it is necessary to build a compound identifier based on the material identifier and the software identifier. So, our choice is to use the GCM architecture.

The different entities we use in our architecture are:

- The Google Cloud Messaging API (GCM),

The regId obtained by GCM, which identifies a smartphone. Furthermore, we builta regId database which stores the regIds of the different smartphones called the regId registry

- The different remote moving databases and the Google application, which manages them and inform the smartphones every time the database is changing.

There are two parts in our architecture: a cloud part and the mobile part. In the cloud part, there are the different moving databases and an application which manages them: this application knows the current database which is used and sends this information when the database is changing and when a new smartphone wants to be connected to our architecture. This Google app is called the database manager. Furthermore, as we use regIds to contact the mobile smartphones, we built another Google application which main part is to store these regIds. This second Google cloud application is called the regIds registry.

In the mobile part, there is an Android application with receives the location of databases to put down the data or put them locally when the network shuts down for instance.

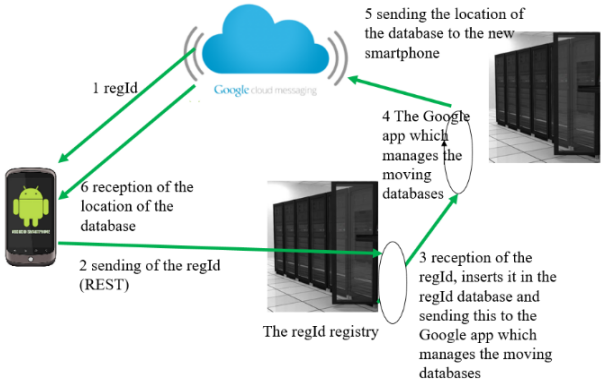

Fig. 1. The initialization step

We use the Google Cloud Messaging API (GCM) to send information from the Google cloud to Android platforms. So, a GCM project has been built in the Google cloud. When a smartphone wants to participate to our architecture, it asks for this GCM project an identifier and receives a regId. With this identifier and the GCM project, information can be sent to it.

There are two main steps in our architecture. First when a smartphone wants to participate to this distributed application: it is the initialization step. Second when the database is changing and the Google app, which manages these changing databases, must inform all the smartphones: it is the changing databases step.

The main applications, which wants to send information and data to the smartphones, are Google Cloud applications. So, the regIds of the smartphones must be known by a Google cloud application of our architecture.

We first study the initialization step. We use the fig.1 above.

- **First,** the Android app, as it participates to the GCM architecture, asks and receives a regId from the cloud (arrow labeled 1). Then it sends it to the centralized Google application called the regId registry (2). Then the Google application, which receives this new regId, informs the Google application, which manages the changing databases (3), the database manager. Therefore, the database manager can send to this smartphone the location of the database (arrows labeled 5 and 6) where the smartphone must deposit its data.
- **Next,** we study the changing databases step. We use the figure 2 below. Every time, the database changes, the database manager asks the regId registry (arrow 2), obtain all the regIds (3) and broadcast the information of the new database to every smartphone using these regIds and GCM (4). Therefore, every smartphone knows the new database (5) where it must send its data.

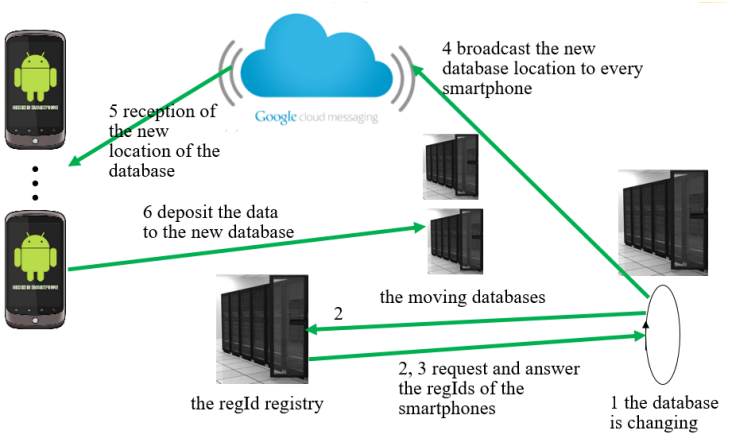

Fig. 2. The changing database step

So, we define two protocols the first one is the initialization process and the second one depicts the exchanges in a stationary mode.

4 Cloud Part

In the cloud part, our software architecture follows a layer pattern. It means that the mobile user can consider our mobile application as a layer stack where the closer layer is its graphical user interface and the farther is assigned to the network management. As shown in figure 2 a mobile phone can receive data from a service deployed in a cloud or it can send data to a web application.

Our need for mobile cloud computing is increasing because it is a way to become nomad users. In addition, anywhere we are, an Internet access provides not only the data from the cloud but also it allows the access of new HTTP services. In our soft-

ware architecture, the figure 1 highlights two main services. One is a regId receiver, which means a REST service, which collects and persists all the regIds. The second one is the database url provider. It gives a valid location of another service, which is able to persist the user data.

These two cases follow a usual architecture as multi-tier layer. Figure 3 displays the software architecture that implements the relationship of the arrow labelled 6 in Figure 2. This layer structure manages every technical role into its own layer. In our example, the rightmost layer is where the data are saved, which means a database. Because we have developed a Cloud application, the database is a NoSQL database called DataStore in the Google implementation.

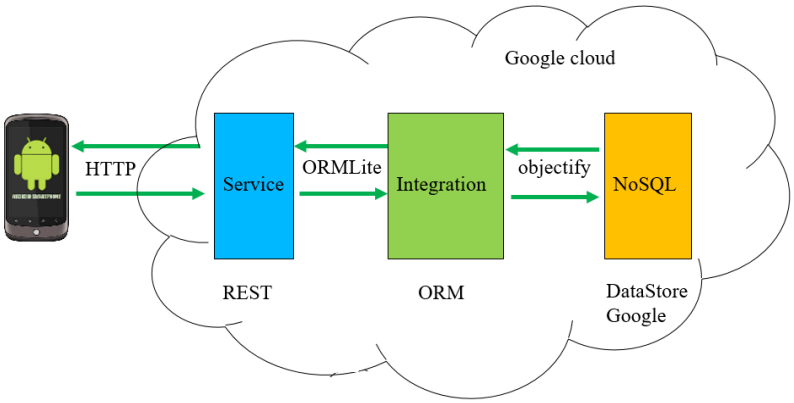

Fig. 3. The persistence layer

The next layer in the middle part plays the role of a mapping between an object world and a relational world. It provides a data access object per table of the data source. Because, some statement need the use of several data access objects, this layer called ORM layer exposes useful persistence services, which need to manage transactions over the data source.

These persistence services fill a role locally to the data manager, but they have to be called from another remote endpoint where the data are first recorded. A mobile device is such device and needs to call remotely a persistence service. We have built a REST layer with the objective is to map persistence operation on HTTP methods. These three layers equip all data manager on our network.

4.1 RegId receiver service

A registering identity is a way to identify mobile device in a mobile cloud. However, a mobile device cannot choice its identity. This depends on the network, the chosen architecture, which is used by the devices. For this reason, we have built a GCM project, which is a service to provide the identification of mobile devices. It provides a unique regId per device.

As soon as a mobile phone receives a regId, this one is saved into a remote registry where a service called regId registry records these regIds. This registry keeps all the device identifiers in case of resource location changes in our software architecture.

4.2 Database url provider

All the active devices will receive a notification about the change of the location of the databases. In our case study, the changes are given by the new uniform resource location (url) of the persistence layer and it is obtained by the database url provider. So dynamically, we can inject a right persistence layer in our software. It is a similar thing as the well-known design pattern dynamic dependency injection but in the domain of persistent layer in the mobile cloud computing. Therefore, it is a kind of new design pattern we call a dynamic storage injection design pattern.

On the server side, the database url provider service has also an update alert. It checks the availability of a persistence layer. When the connection test provides a timeout or when there is no database available, the mobile clients cannot persist their data. In the first case (no network available), the mobile is notified by an intern event. In the second case (no remote database available), the mobile clients are informed by the database url provider service. In both cases, the mobile devices save the data locally.

When several remote persistence layers are recorded into the application, the database provider can provide one of this persistence layer.

4.3 Persistence service

A first persistence application is built and deployed in the Google Cloud Platform for saving the data. For the reception, a web service exposes this application on HTTP protocol. The url database provider provides this location to any mobile client which sends a request about. However, behind this address the software architecture is defined as a classical Java EE application. This means that a mapping object relational requires first the declaration of one entity class per business table. Even if our database is not relational, a mapping is essential because, it applies a reuse approach of the persistence management, as design patterns dissociate the specification of a work and its implementation.

Next, the definition of data access object hides the use of request over objects in memory cache. Each class contains five methods, which covers all the needs over a table. CRUDS is the name attached to such a set of methods, create, read, update, delete and searchAll. Because a statement can touch more than one table, also we have created persistence services for the management of several entities in a consistent manner inside a transaction. Finally, we have created a REST service per persistence service for the usages in a SAAS of the Google cloud.

The goal of all this work is to create an access to a NoSQL Database called Google Cloud Datastore, which is built for automatic scaling, high performance, and ease of application development. It uses redundancy to minimize impact from points of failure. It uses also a distributed architecture to automatically manage scaling. Moreover,

Cloud Datastore is exposed to applications through multiple clients. It also provides a SQL-like query language. All our services have access to a precise schema that is available anywhere in the cloud.

5 Mobile Part

The mobile project management needs the use of dedicated tools not only for the built application but also for the tests and their automation. Even if the framework role stays similar from Java EE application to mobile application, we need to adapt each of them to the embedded system domain.

5.1 Network exchange

The mobile application is often a web client of a web application server. In addition, the data stream often comes from a mobile source to a service in a cloud. Our software architecture needs to take into account the data stream in the opposite sense. This means that data are imported from the cloud into a mobile application. Today, Representational State Transfer (REST) is a style of architecture based on a set of principles that describe how networked resources are defined and addressed. Roy Fielding [20] first described these principles in 2000. REST is an alternative to SOAP and JavaScript Object Notation (JSON). It is important to note that REST is a style of software architecture as opposed to a set of standards and it is particularly suitable for embedded applications. A framework called Restlet allows developers to create new REST services, which expose resource from a mobile phone. This framework is also portable, and the same structure is proposed from the development over a cloud. This allows a uniform approach for all the network exchanges.

5.2 Business mobile application

When the data exchanges occur at runtime, they play a role of business data injection into a remote workflow. Our use case is about the management of contacts with the use of a remote database. Therefore, the body of the requests contains a data structure useful for the remote service. On the service side, these data structures are considered as resources. This is a reason why REST services are more suitable than other remote access types.

Our concrete implementation of a REST Web service follows four basic design principles, use HTTP methods explicitly, be stateless, expose directory data structure-like URIs and transfer JavaScript Object Notation (JSON) objects.

HTTP GET is defined as a data-producing method that has intended to be used by a client application to retrieve a contact, to fetch contacts from a Web server or to execute a query with the expectation that the Web server will look for and respond with a set of matching contacts. The basic REST design principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods. According to this mapping.

- To create a contact on the server, use POST,
- To retrieve a contact, use GET,
- To change the state of a contact or to update it, use PUT,
- To remove a contact, use DELETE

We adopt the same naming convention for all the Web services of our distributed mobile application. Moreover, the mobile device exposes web services for the injection of a registration identifier. We apply the same mapping between the operations of the various REST services.

5.3 Local persistence layer

Our mobile client application needs to store data somewhere. We may store our data and it sometimes do it in the local embedded SQLite database. We had to decide between writing SQL queries, using a Content Provider (useful if you want to share your data with other apps), or using an ORM as explained before. We have selected an embedded ORM. However, when the data are locally saved, it becomes essential to synchronize databases when a connection is possible. This is done, using a SyncAdapter Android API that is able to update the data from the mobile device to the data store in the cloud. This requires the creation of a content provider in the mobile client. The SyncAdapter pilots the update until its termination.

A key feature of our case study is the evolution of the remote databases. Because the backup of the contacts can be done on distinct databases including the local database in the smartphone, we have adopted the same REST service interface to all data stores. This allows the mobile client to become stable even a new data store is considered. The same Web service is also useful when databases are synchronized. This happened when several contacts are saved locally to the mobile device in the case of the network is unreachable and when an access point becomes available.

6 Results and Benchmarks

Our distributed software is deployed over a network, which is the crucial resource. It is possible to track the network activity when a mobile application is making network requests. Usually data are sent to servers. These servers are virtual machine in the cloud and we have developed and deposit software to manage these data in these servers. Therefore, we use a SaaS cloud architecture.

We use the Google cloud infrastructure for that. The figures we are going to present under come from the Google console and the Google user interface to manipulate the Google cloud. The software put in the Google cloud are called Google apps.

6.1 Software architecture

In the initialization step described in the figure 1, we use two Google Cloud data stores. The first one records the different regIds. This database is unique in our archi-

tecture. The second indicates the URI of the current databases where data are stored. This second database is also unique in our architecture.

Therefore, we have a data store where regIds are recorded with their values (beginning by APA) and the date when it was recorded. We wrote a Google cloud app to manage this data store and this Google app is traced by the figure 4.

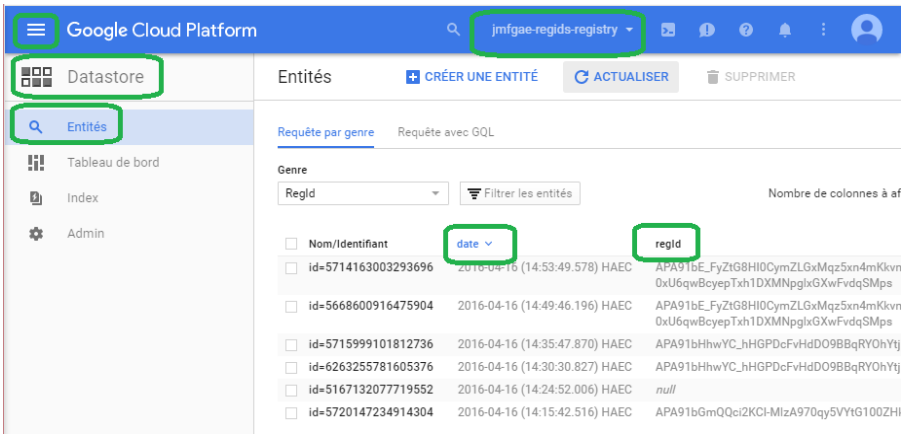

Fig. 4. The Cloud Dashboard for RegId DataStore

In our architecture, the databases where data are stored can vary. So, we must manage these moves. For that, a database for the records is located by an URI. These URIs are recorded in a data store in Google Cloud.

The second data store shown in the figure 5 records the URI of the database where the mobile phone will push their data. We put too the date when an URI is available. We wrote a Google cloud app to manage this data store.

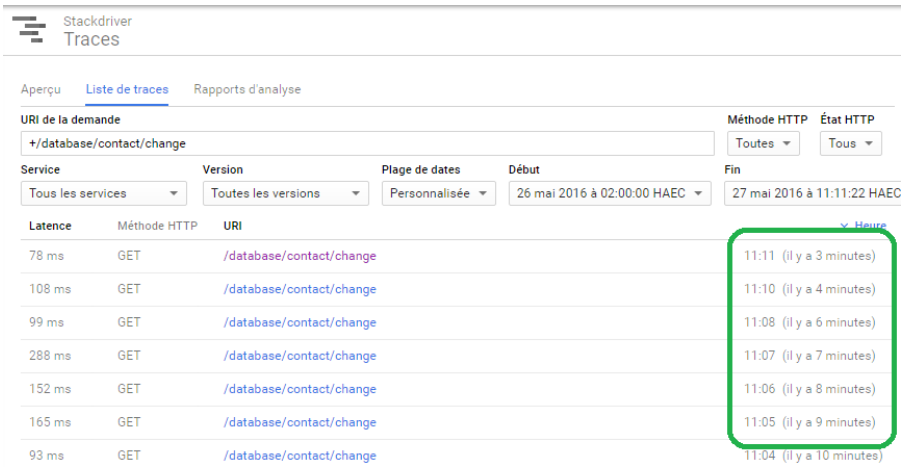

Fig. 5. The Cloud database manager

In the figure 6, we have a third database where the data coming from mobile smartphone are recorded. This database is shown in the figure 2. Even this database can change; they have all the same structure i.e. a Google Cloud data store with the kind of records as shown in the figure 6.

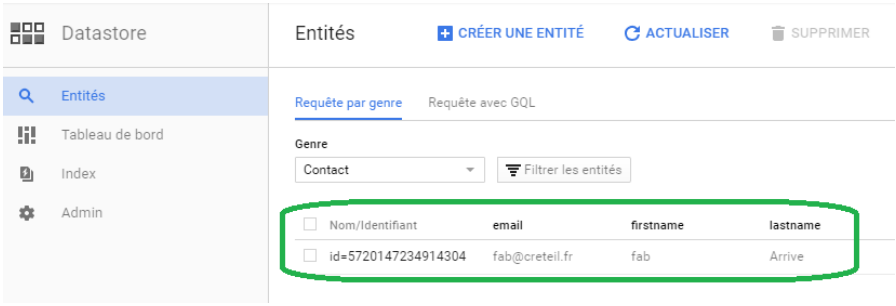

Fig. 6. A Cloud Contact Database

6.2 Benchmarks

We have used Dalvik Debug Monitor Server (DDMS) and Systrace to break down what our application is doing and for how long. On Android, every application runs in its own process, each of which runs in its own virtual machine (VM) called Android Runtime (ART). Each VM exposes a unique port that a debugger client can attach to it.

The DDMS includes a Detailed Network Analyzer (DNA) that makes it possible to track when our application is making network requests. Using this tool, we can monitor how and when our application transfers data and optimize the underlying code appropriately. We can also distinguish between different traffic types by applying a filter to network sockets before use. When we look at the behavior of the mobile application (mobile-client), it receives periodically registration identifiers and data source url from the cloud. On the other side, the mobile application sends REST requests to remote servers.

By monitoring the frequency of our data transfers, and the amount of data transferred during each connection, we identify areas of our application that are made more battery-efficient. This corresponds to the packages called contact network and regid network. We have identified the cause of transfer spikes.

6.3 Management of the transfer spikes

The main energy consumption is due to network traffic regardless the algorithm of selecting the protocol. From the Bluetooth to the WiFi, the use of communication sensors involves an energy cost. Mohammad Tawalbeh notes in [21] that the network transfers cost often more than the screen even if the luminosity is high. Also, he proposes to disable some network services such as Bluetooth service, IrDA (Infrared

Data Association) service and so on until a need appears. This allows to suppress the lookup of network base or the test of the signal range.

Good battery technology simply hasn't arrived yet in 2018, which means it's down to software and settings to configure the limited power. In the domain of hostile environment, all the resources are limited: not only the energy but also, the network capabilities, the memory, etc. Kaushik Dutta et al. [22] propose to cache a set of results concerning the most useful application. They define in a strategy of cache management which reduces the energy consumption.

There are multiple cases of moving data into and out of the cloud which can generate spikes. The first scenario is called a bursting case where the databases, applications and middle-tier servers are all based in the same location. When the cloud provides a large volume of data, it involves an overcome of activities with spikes of requests and a burst of resources. The second use case is about the replication of data and applications in the public cloud. If something goes wrong, then the whole workspace is saved and after resumption, the transfer spike occurs.

7 Conclusion and Future Works

In our paper, we present the opportunity to change dynamically storage areas. However, of course, as we use Android smartphones, the client part can move too. Therefore, our paper presents an architecture where the two parts, client and server can move.

We prove in this paper that we can save data coming from mobile devices to moving databases. The databases can change dynamically and the mobile devices know the new database to store their data. Even there is no network; data can be save by the mobile in their own local SQLite databases. In fact, as the well-known dependency injection used in object-oriented software architecture, we built a storage area injection to dynamically change the places where data can be saved. This dynamically storage area injection can be proposed as a new design pattern for software architecture using databases. May be this kind of software need to be quite generic to be easily translated from a machine to another. So, our storage area injection design pattern can be use in software built on dependence inversion.

The mobile devices are good machines to retrieve information. But a lot of other machines can make the job for example in the Internet of things domain. Raspberry PI, Arduino cards, different sensors and other technologies are good candidates to collect information. Therefore, we can suggest developing software for these connected objects. In addition, we must inform these Internet connected objects where to send their collected data to changing storage in the cloud.

Our work could be extended with the use of mobile service. This will transform our software architecture into a more elastic architecture where mobile components will income into the mobile client. These new components will play an invoker role. So, depending where the end user is, he will receive on his mobile phone a piece of code which is a service invoker of a service which is hosted in the cloud. This approach suppresses the need of an API gateway, which helps the discovery of the useful ser-

vices but introduces a new dependency on a technical API. Also, it provides the client part a more evolving behavior; when exchange format changes then the mobile component brings a request builder which take care of the new packet format. This means that the mobile part becomes easier maintainable than before.

Our architecture used Google Cloud Messaging (GCM) to inform the different smartphones. In fact, the part Firebase Cloud Messaging (FCM) [23] of the Android Firebase technology is the new version to send messages to and receive from the cloud.

Our work was implemented and tested using the local SQLite databases of Android smartphones, MySQL databases on servers and NoSQL databases in Google cloud using appengine technologies [24]. We can propose to store and share data in a peer-to-peer infrastructure between smartphones using Wi-Fi direct. Our work is accessible in a git format at https://github.com/mourlin/fmjmf.

8 References

[1] Qi Zhang, Lu Cheng and Raouf Boutaba, «Cloud computing: state-of-the-art and research challenges» at Journal of Internet Services and Applications, Volume 1, Issue 1, pp 7–18, 2010.https://doi.org/10.1007/s13174-010-0007-6

[2] Aaaron Smith, «U.S. Smartphone Use in 2015», April. 1, 2015. [Online]. Available: http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/

[3] Xavier León and LeandroNavarro, «A Stackelberg game to derive the limits of energy savings for the allocation of data center resources», Future Generation Computer Systems, vol. 29, p. 74–83, January 2013. https://doi.org/10.1016/j.future.2012.05.022

[4] Zhangbing Zhoua Walid Gaalou, Lei Shu, Samir Tata and Sami Bhiri, «Assessing the replaceability of service protocols in mediated service interactions», Future Generation Computer Systems, vol. 29, p. 287–299, January 2013. https://doi.org/10.1016/j.future.2011.08.007

[5] Javier Espadas, Arturo Molina, Guillermo Jiménez, Martín Molina, Raúl Ramírez and David Concha, «A tenant-based resource allocation model for scaling Software-as-a-Service applications over cloud computing infrastructures» Future Generation Computer Systems, vol. 29, p. 273–286, January 2013. https://doi.org/10.1016/j.future.2011.10.013

[6] Youhui Zhang, Yanhua Li and Weimin Zheng, «Automatic software deployment using user-level virtualization for cloud-computing» at Future Generation Computer Systems 29, (2013) 323–329, 2013.

[7] Yavuz Selim Yilmaz, Bahadir Ismail Aydin and Murat Demirbas, «Google cloud messaging (GCM): An evaluation», at IEEE Global Communications Conference, 2014. https://doi.org/10.1109/GLOCOM.2014.7037233

[8] Utz Sonja, Muscanell N and Khalid C, «Snapchat Elicits More Jealousy than Facebook: A Comparison of Snapchat and Facebook Use», Cyberpsychology, Behavior, and Social Networking, vol. 18, p. 7, 2015.

[9] Yuheng Hu, Lydia Manikonda and Subbarao Kambhampati, «What We Instagram: A First Analysis of Instagram Photo Content and User Types,» chez ICWSM, June 2014.

[10] «Gmail: free storage and email from Google,» 2016. [Online]. Available: https://www.google.com/gmail/about/

[11] Tobias Jeske, «Floating car data from smartphones: What google and waze know about you and how hackers can control traffic», at Proc. of the BlackHat Europe, 2013.

[12] Mai Alfawair, «A Cloud Storage Architecture for High Data Availability, Reliability, and Fault-tolerance» ICFNDS '17 Proceedings of the International Conference on Future Networks and Distributed Systems, 19, 2017
[13] Chun-Ting Huang, Lei Huang, Zhongyuan Qin, Hang Yuan «Survey on securing data storage in the cloud,» chez The 7th International Conference on Signal Processing, Image Processing and Pattern Recognition (SIP 2014), Hainan China, 2014. https://doi.org/10.1017/ATSIP.2014.6
[14] «HDFS (Hadoop Distributed File System) data architecture» Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.
[15] «ceph storage» Available: https://ceph.com/ceph-storage/
[16] Lieyun Ding, Xun Xu, «Application of Cloud Storage on BIM Life-Cycle Management» International Journal of Advanced Robotic Systems , vol. 11, 2014.
[17] Jiang, Bing & Yu Ma, Guo & Ma, Sheng & Feng Xie, Jian. (2013). Mobile Sync Client Design of Cloud Storage System. Applied Mechanics and Materials. 464. 358-364. 10.4028/www.scientific.net/AMM.464.358.https://doi.org/10.4028/www.scientific.net/AMM.464.358
[18] Yen-Hung Kuo ; Yu-Lin Jeng ; Juei-Nan Chen, «A Hybrid Cloud Storage Architecture for Service Operational High Availability» chez IEEE 37th Annual Computer Software and Applications Conference Workshops, Kyoto, Japan, 2013.
[19] I. Database, https://imeidb.gsma.com/imei/index#
[20] R. Fielding, Architectural styles and the design of network-based software architectures (Doctoral dissertation), Irvine: University of California, 2000.
[21] Mohammad Tawalbeh, Alan Eardley, Lo'ai Tawalbehb «Studying the Energy Consumption in Mobile Devices» chez 13th International Conference on Mobile Systems and Pervasive Computing, Montreal, Quebec, Canada, 2016.
[22] Kaushik Dutta, Debra Vandermeer, «Caching to Reduce Mobile App Energy Consumption» ACM Transactions on the Web (TWEB), vol. 12, 2018.
[23] https://firebase.google.com/docs/cloud-messaging/, «Firebase Cloud Messaging».
[24] Google Cloud Platform, 2016. https://cloud.google.com/appengine/

9 Authors

Dr. Fabrice Mourlin is member of the Logics, Algorithmic and Complexity Laboratory (LACL), Charles de Gaulle Avenue 60, 94010 Creteil, France. He works in the team called Systems Specification and Verification. He is expert in mobile agent application and is associate professor of Computer Science, at UPEC - Paris-Est University, Program coordinator of 2 Master programs in Computer Science.

Dr. Jean-Marc Farinone is an associate professor in Conservatoire National des Arts et Métiers in Paris (France). He is studying Cloud architecture and processing, the Internet of Objects. He works on Cloud system and mobile architecture. He has created trainings about Android, C, C++ and Java programming languages, design patterns, object-oriented designs, distributed architecture and middleware.

Article submitted 2017-12-09. Resubmitted 2018-12-28. Final acceptance 2018-12-28. Final version published as submitted by the authors.