# Assessing NoSQL Databases for Telecom Applications

Francisco Cruz
HASlab
Universidade do Minho
Braga, Portugal
fmcruz@di.uminho.pt

Pedro Gomes
HASlab
Universidade do Minho
Braga, Portugal
pedrogomes@lsd.di.uminho.pt

Rui Oliveira
HASlab
Universidade do Minho
Braga, Portugal
rco@di.uminho.pt

José Pereira
HASlab
Universidade do Minho
Braga, Portugal
jop@di.uminho.pt

***Abstract*—The constant evolution of access technologies are turning Internet access more ubiquitous, faster, better and cheaper. In connection with the proliferation of Internet access, Cloud Computing is changing the way users look at data, moving from local applications and installations to remote services, accessible from any device. This new paradigm presents numerous opportunities that even traditional businesses like telecoms cannot ignore, in particular, enabling new and more cost effective solutions to old problems.**

**The work presented in this paper provides a detailed description of how a telecom application can be migrated to a NoSQL database. Particularly, by pointing out the necessary change of how we reason about data as well as the data structures that support it, in order to take full advantage of Cloud Computing. In addition, we also present a preliminary evaluation of different data persistency paradigms based on a fully tunable simulation platform that mimics the operation of a telecom business.**

***Index Terms*—Cloud Computing; Distributed Databases; NoSQL; Telecom;**

## I. Introduction

The constant evolution of access technologies, wired as the optical fiber, or wireless such as the WiMAX and LTE are turning Internet access more and more ubiquitous, faster, better and cheaper. The proliferation of Internet access allows users to use services directly provided by it, which results in a change of paradigm for the use of applications and how users communicate, popularizing the paradigm known as Cloud Computing (CC). In a CC environment, the majority of applications, as well as data, do not need to be installed or stored on the user's computer, as they are provided by the "cloud" through dedicated service providers, also known as Cloud Providers (CPs). The Cloud Provider is responsible, for example, for the storage, maintenance and backup of all user information, and the user just has to access the platform provided by the CP and only pay for the services she uses and when she needs them [1] – a concept known as pay-as-you-go.

The new paradigm implies the access of millions of users to the same application and partly as a result, storage of digital data has reached unprecedented levels [6]. A good example of such a applications are the social networking platforms like Facebook and MySpace, that have to deal with millions of requests everyday. But, traditional relational databases, RDBMS (Relational Database Management System), seem to be not well suited for these environments. Consequently, the major online players searched for alternatives into building extreme large scale storage systems and the result was a great set of different data stores popularly called NoSQL: Dynamo [4], PNUTS [3], BigTable [2], Cassandra [5], DataDroplets [7], among others. Such data stores provide high availability and elasticity in a distributed environment composed by a set of commodity hardware, avoiding the need to invest in very powerful and expensive servers to host the database. In addition, these data stores automatically provide replication, fail-over, load balancing and data distribution. Nevertheless, when compared to the more mature RDBMS, NoSQL databases have some fundamental limitations that should be taken into account. They provide high scalability at the expense of a more relaxed data consistency model and only provide primitive querying and searching capability. Imposing an additional complexity to the application.

This work aims at assessing how these solutions can be adapted to the business model of telecommunication operator. As with other large enterprises, telecommunications enterprises tend to rely on very centralised systems, and the field of CC offers an innovative and important opportunity in terms of new concepts and paradigms. Thus, this paper describes the process of modelling required for the transition of a relational model to a non-relational one, in the context of managing customer data, services and telephone calls.

The remainder of this paper is organized as follows. Section 2 presents the data model used for the simulation platform. Section 3 describes the process and differences between using a RDBMS and a NoSQL database. Section 5 and 6 present the workload used to benchmark the system, in order evaluate and compare the relational approach and the non-relational approach. Section 7 concludes the paper.

## II. Data Model

In its core, the system presented here is responsible for the data management and the business decisions related to customers information in relation to incoming calls. During runtime, it handles the incoming calls, fetching the caller's information and subscribed services to know if they are valid and how should they be treated (tariff cost, discounts, etc.).

In order to support this system, the telecom's data model is composed of 7 entities (Fig. 1): *Customer*, *Bucket*, *Bucket*

978-0-7695-4535-6/11 $26.00 © 2011 IEEE
DOI 10.1109/CEC.2011.48

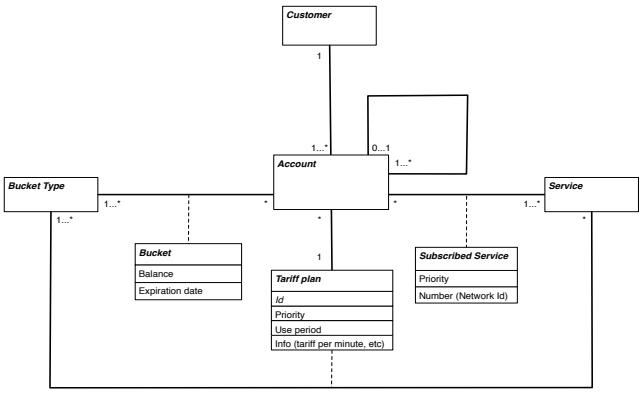



Fig. 1: Telecom's data model.

*Type*, *Subscribed Service*, *Service*, *Account* and *Tariff Plan*.

The *Customer* entity bears the personal information regarding each of the telecom's customers. In addition, this entity is related to the entity *Account* in the sense that a customer can be associated with one or more accounts.

The *Bucket* entity represents the balance associated with each account and each bucket type. The *Bucket Type* holds information related to the type of buckets existent in the system (money, SMS, etc).

As it can be inferred from its name, the *Subscribed Service* entity represents the services subscribed by each customer. In addition to the phone number, the account identifier and the service name, this entity also contains a priority field, which is used to select the most advantageous service.

With regard to the *Service* entity, it maps service names into service types. The latter's attribute, similarly to *Bucket Type*, is a finite set of possible type of services provided by the telecom, for instance, voice or text messaging.

The entity *Account* plays a central role in this model, it contains as attributes the identification of the account, the customer identification and also a tariff plan identification. In addition, it introduces the concept of hierarchy between accounts. By way of explanation, when a set of accounts have a shared balance, it is introduced the notion of a parent account. Each account can be associated at most with a single parent account.

From the assumption that different services may use different types of bucket, there is a relationship between *Services* and *Bucket Type*. Therefore, the entity *Tariff Plan* establishes this relationship by mapping service names into bucket types for each plan identification. In addition, it also provides a priority field and informations about the plan, such as the period of use or the cost per minute.

## III. Data Store

A telecom has to deal with millions of requests everyday and as with other large enterprises, they tend to rely on very centralised systems for data storage. Whenever the need for more capacity arises, the solution is always to acquire a new, more powerful and expensive server i.e. scaling vertically. However, the Cloud Computing paradigm has already proven that scaling horizontally offers the desired scalability and availability. In fact, scalability is achieved by adding new machines to the system on demand, in order to cope with the load in the system, and at the same time can provide fault-tolerance, ensuring availability. Besides, it is more cost-effective because it relies on commodity hardware and allows a granular adaptation from small to large scenarios.

For this case study, it was considered two different data stores: PostgreSQL and Cassandra. This choice is related to one of the contributions of this work: to show how a telecom's data management system can be ported to a NoSQL database, oriented towards a CC environment. Additionally, PostgreSQL represents the typical approach using relational database management systems (RDBMS) and, thus, will serve as term of comparison.

In the platform, each incoming call is composed of four main phases: 1) the first phase evaluates what services the user has subscribed verifying each one in the priority order until a valid one is found; 2) in the second phase the tariff plan is queried to assess what types of buckets can be used; 3) the third phase concerns the discovery of the account that contains such entity, in fact a client can have a service in association with a company that supports some of her costs. 4) the last phase consists of a bucket subtraction i.e. a balance update, being this the only phase that cannot be easily moved to Cassandra due to its transactional requirements.

We select the first phase of this process, *the choice of services*, as an example for the migration process presented next. This operation has the following input and output.

**Input:** The caller number and the type of incoming call.
**Output:** The available list of services and the account identification associated with them, ordered by priority.

### A. PostgreSQL

Based on the normal relational approach, the data was then analysed and the queries were built upon it. The data model showed in Figure 1 is in fact no more than a conceptual entity-relationship model and, thus was easily implemented in PostgreSQL. As a result, the schema of the database is composed of the same 7 relations, with the respective attributes.

Based in this model, the specific queries for each one of the phases was then created and implemented with JDBC. In order to achieve maximum performance, several runs of the simulation were carefully analysed resulting in some optimisations. Namely, some queries were rewritten and were also introduced secondary indexes over some attributes.

Looking at the presented example, its implementation in this data stores assumes the following form:

```
SELECT SubscribedService.account_id, SubscribedService.name
 FROM SubscribedService
  JOIN Service
   ON Service.service_name = SubscribedService.name
   AND Service.service_type = Type
 WHERE SubscribedServ.number = Id
 ORDER BY SubscribedServ.priority DESC
```

This query, which represents the relational approach to the problem, was then optimized with indexes over some of the presented fields in the *SubscribedService* and *Service*.

### B. Cassandra

Initially developed at Facebook to support the social networking application, Cassandra is, by definition, a highly available distributed data store that encompasses concepts from Dynamo's architecture and the data model from BigTable.

Moreover, Cassandra's data model implements a variant of the entity-attribute-value (EAV) model and can be thought of as a multi-dimension sorted map. The *Keyspace* is a namespace for *ColumnFamilies*, which in turn map rows to a set of columns. In fact, there is a rough correspondence between a *ColumnFamily* and a table in the relational model, but they differ on the property that within a *ColumnFamily* each row can have a completely different set of columns. As a result, there is no pre-defined schema so columns can be added dynamically. There is yet another higher data structure *SuperColumnFamilies* where each of its attribute columns (in this structure named *SuperColumns*) can have a list of ordinary columns. Within a *ColumnFamily* and a *SuperColumnFamily*, both columns and supercolumns can be ordered according to their names, using one of the following supported criteria: ASCII, UTF-8, Long, UUID or binary ordering.

The differences between the two paradigms also imply a different approach in application development. In contrast to the relational paradigm, when designing an application with a NoSQL database as back-end one should not start from defining the data model but from the features that are intended for the system. As a result, the development process of the Cassandra's interface was fundamentally different. For each of the presented phases, it was analysed both the input and the output desired, then the *ColumnFamilies* or *SuperColumnFamiles* were created, so that from the input, the desired output was easily obtained.

For the presented example, in the implementation process a *SuperColumnFamily* was created with the row key being the caller number and the name of the *SuperColumn* being the type of incoming call. Then, each *SuperColumn* has a list of columns corresponding to the list of available services and the account identification associated with them. These columns are ordered by priority. An example of the structure is depicted in Figure 2. In the migration of this phase and also in the remaining, the same method is used, where the queries based on *Joins* are translated to data structures where data is naturally indexed.

In terms of consistency, this is not a major concern in this study since customers' accounts are usually independent from each other, i.e. isolation is provided by the characteristics of the case study. The nature of the data that is in most cases rarely written also helps in this process, where such operations as the services' subscription can be implemented based on a system of delayed confirmations. With such techniques, services are activated and used even before the client is aware

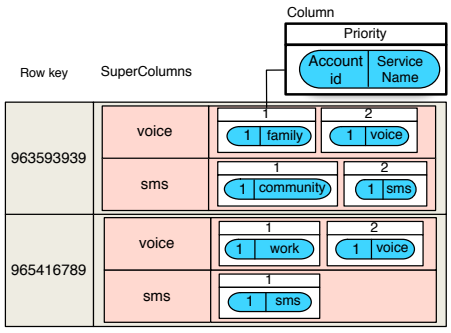



Fig. 2: *SuperColumnFamily* representing the choice of services.

of such fact, or when she removes a service, it can still be active for some moments even after the confirmation is given.

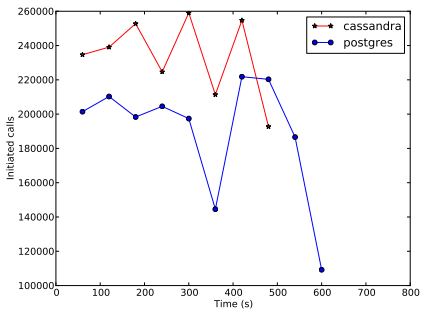
However, in the case where two or more accounts share a bucket i.e. have a shared balance, a stronger consistency criteria would be needed to prevent simultaneous concurrent updates of the balance. Even if such services only affect in average 10% of the population, they still need full transactional guarantees (ACID). There are, for this problem, two main possible solutions: (i) a hybrid approach where the bulk of data would be stored in Cassandra, except for data related to the shared buckets; (ii) using an external entity to restrict concurrent accesses to the shared bucket. The isolation of the bucket in a separated database or different code wrapper is however straightforward as this entity is only used in a restricted phase of incoming calls process.

## IV. Platform and Workloads

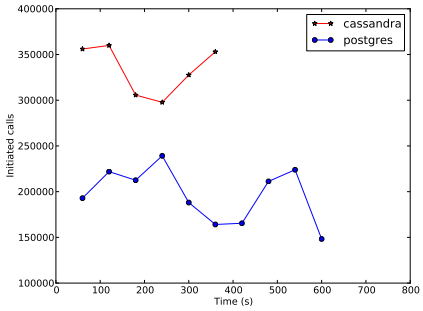
In order to test and compare the relational and non relational solutions, a simulation platform for benchmarking of the system was developed. More than just a workload execution platform, it allows for the generation of consistent data to populate the underlying database and the definition of operations to simulate different scenarios of use. Trying to mimic the real world, this platform allows the manipulation of several factors that affect the users' behaviour, such as their relationships and the associated services or their call history.

There were created different execution scenarios in order to simulate different periods of use, where the load and type of requests vary. With the implementation of several typical telecom services based in list of families or time based discounts we define their coverage over the population and its effect over the user. Many more parameters are available such as the type of incoming calls that are given to the platform, or the importance of the client's call history in the incoming calls formation.

Based on this set of parameters, there were created 2 distinct workloads: a *Day workload* representing the regular load during a day with more focus on business activities and a *Christmas workload* that simulates the Christmas holidays.




(a)




(b)

Fig. 3: Initiated calls - 7 000 000 customers

## V. Experimentation

A series of preliminary experiments were conducted to evaluate the performance of both data stores. These experiments cover the 2 defined workloads in a population of 7 million customers. Regarding the experiments setting, the simulation platform was run on a 8-core, 16GB RAM machine being the data stores limited to a single 24-core, 128GB RAM machine, with disks in RAID1.

The main goal of this experiment is to simulate a small country and for that purpose, the total number of generated calls reaches two million. During the execution time, the *Day workload* uses 160 execution clients that make 5 minute calls. The think time is 10 milliseconds and the number of simultaneous calls is 20 000, and thus, each execution client has to establish 12 500 calls. In the *Christmas workload* the think time is 0 seconds to simulate a higher load.

Figure 3 shows that in this experiment the non-relational data store achieves better results at the level of the number of initiated calls. These numbers were as expected since no transactional guarantees are offered in Cassandra and no external mechanism was implemented at this point. Even if the results are promising, there is a need for a further evaluation of the scalability of both approaches, being the Cassandra solution complemented with a transactional mechanism or a ACID compliant database.

## VI. Conclusion

With the appearance of the Cloud Computing paradigm and facing an ever-changing market, new data stores developed by Internet giants emerge. In this context, this work aims at assessing how NoSQL data stores could be adapted to the business model of a telecommunications enterprise.

In that purpose it was developed a implementation of a telecom application over two distinct paradigms and a simulation platform based on human activity that allow the creation of distinct workloads to test it. With implementations in PostgreSQL and Cassandra, a resume of the differences between the different models were shown in terms of development methods and some preliminary results. Even if at this point they are more favorable to the non relational paradigm, the platform needs to be executed in a distributed environment to test the capability of the system to scale progressively. In fact the value of this implementation would be not in its raw performance but advantages like the increased adaptability of the system and the associated cost reductions. Nonetheless, the non-relational paradigm requires an optimization of data structures, requiring data de-normalization and pre-materialization of the results not supporting also transactional operations, that is a fault that we intend to tackle in the future.

In the long term, this work intends to evaluate if the non-relational paradigm can be advantageous even in a business model like a telecom. However, it should be noted that the transition to CC is not easy, and the documentation of this extra effort is also a valuable result.

## References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53:50–58, April 2010.

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26:4:1–4:26, June 2008.

[3] 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. *Proc. VLDB Endow.*, 1:1277–1288, August 2008.

[4] 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 *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[5] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44:35–40, April 2010.

[6] D. Skillicorn. The case for datacentric grids. Technical Report ISSN-0836-0227-2001-451, Department of Computing and Information Science, Queen's University, November 2001.

[7] R. Vilaça, F. Cruz, and R. Oliveira. On the expressiveness and trade-offs of large scale tuple stores. In R. Meersman, T. Dillon, and P. Herrero, editors, *On the Move to Meaningful Internet Systems, OTM 2010*, volume 6427 of *Lecture Notes in Computer Science*, pages 727–744. Springer Berlin / Heidelberg, 2010.