integrating a key-value db approach in pc booking architecture

Feasibility study

Abstract

Feasibility study about the possible integration of a key-value-based DB in a relational-DB-based PC booking application.

Dario LORENZONI

Raffele NOCERINO

Marco PETTORALI

Riccardo XEFRAJ

Index

[Introduction 2](#_Toc24110886)

[Database analysis 2](#_Toc24110887)

[Volumes involving User table 3](#_Toc24110888)

[Possible advantages of a key-value based “User” table 3](#_Toc24110889)

[Possible implementations 4](#_Toc24110890)

[Conclusions 5](#_Toc24110891)

# Introduction

Key-value databases are very efficient in terms of computational power needed for data accessing and storing when compared to traditional relational databases. Key-value databases are also useful if availability and load balancing is a requirement in the application; in fact, key-value couples are easily divisible in two or more servers, because of their simple and schema-less structure. This is the reason why complex operations such as queries and joins should not be implemented on a key-value database, and their advantages are fully exploited if only simple read/write operations are needed.

In a context of a PC booking application, it could be interesting to see if the relational database or some part of it, on which the application is built, can be implemented using a key-value approach. In the rest of the document, we will analyze each component of the database, and more detailed information will be given.

# Database analysis

The application’s database is divided in three parts: one dedicated to PCs and rooms’ information storage, another used for tracing the bookings made by users and the last one that collect all the personal information of the users.

Beginning from the first one, the PCs and rooms’ information storage implementing it with a key-value database is not a tough operation to be done, and it has low impact on the global performances of the system. This because new rooms and PCs are rarely inserted in the database after its initialization, and the effort to make so is really low. Moreover, PC and Rooms tables have to be heavily queried by the application, in order to get the exact number of available stations given a certain date and room. This led us not to choice to implement this part of the database in the key-value format.

Something similar applies to the table that recollect all the bookings made by the users. There will be many instances of the entity “Booking” in the database, but complex queries are needed in order to find the list of all the available computers in a given day. We decided to don’t implement even this table using a key-value approach.

Users’ personal information is something completely different to handle. We suppose that the database will collect many instances of the entity “User”, about one for each student in the university. This data includes the username and password of the user, so each time a student wants to log in to the system, the system will perform a read operation on the table. Students are not inserted with a high frequency in the database compared to the read operation, so we can consider our application as a read-heavy one (in relation to the “User” table), due to the fact that most of the operation that are made on the personal data of the users are read ones. Moreover, queries operations on the table “User” are not needed, because when the system checks for the current reservations made by the logged-in user in the past, it only cares about the username.

# Volumes involving User table

We suppose that each student of the university has a record in the database: we consider about 50,000 entries. Let’s consider (overestimating the values) an average of 10,000 new students enrolled each year and other 10,000 students that are leaving university. So the total amount of write operations on “User” table is 20,000 operations per year, that is nearly 55 operations per day.

Let’s suppose, instead, that each day the 10% of the students need to use the service for booking a PC. This percentage is actually underestimated, considering the high number of requests in university libraries. About 5000 people will login to the service each day, and this implies a read operation on the user table to check the user’s credentials.

To resume up, the system will perform 5000 read operations per day and only 50 write operations per day, practically the 5000/(5000+50) = 99.01% of the total operations are read ones. This means that even a little increase of performance on the read operations, due for instance to the usage of a key-value database, would lead to huge positive effects on the global performances.

# Possible advantages of a key-value based “User” table

After the analysis phase in which we found that there are no negative effects from the implementation of the “User” table using a key-value database, we decided to list all the possible advantages comparing this approach to the standard relational techniques. These are the following:

* **Lower impact on system performances:** a lot of user information are stored in the database; querying them using the traditional relational approach is certainly a way to get data from the table, but using a key-value technique is certainly faster and less expensive in terms of computational power;
* **Higher flexibility:** the fact that key-values databases are schemaless allow the system to store differentiate information about the users; for instance, some users would prefer to give their cell phone number instead of their email address to be contacted, and this is not easily possible using a relational database;
* **Simpler way to implement data sharding:** collecting a high number of information is always a critical issue. Key-value databases are simpler to handle than relational ones when doing data sharding because of their simple structure. For instance, users could be divided in multiple servers, one for each university faculty, collecting the personal information of the users attending that faculty only. This would also lead to lower costs of maintenance of the equipment, due to the lower internet traffic flowing in each server.
* **Simpler way to handle multiple replicas:** due to the fact that a key-value DB has a very simple structure, multiple copies of the database can be handled in a very simple way. For instance, the personal information of the user is a critical data and must be always available and never lost. Creating and maintaining multiple copies of these data could be really a challenge using a relational database.

In the end, it all resumes up to lower costs and higher performances of the whole system. For achieving these results, we also thought about some possible architectures to use when implementing the key-value part of our database.

# Possible implementations

For implementing the key-value database, we opted for the software “LevelDB”, by Google. We will generate keys as strings representing the hierarchical structure of the table “User”, and we will use a hash function to interpret them as indexes. This because it would be a waste of space to store long strings as keys; in fact, using integers as indexes is less impacting on the system’s performances because of their shortened size.

We also thought about a possible implementation of the data sharding using a key-value distributed database. We showed in the previous paragraphs that our application is read-heavy in relation to the key-value part of our database. This led us to move towards a client-server “Master-Slaves” architecture.

This architecture provides a differentiation of the roles between the nodes of the servers’ network. A node called “Master” is the only one in charge of listening to clients’ requests; then it redirects the read operation to one of the other machines (called “Slave”). When the user asks for a write operation, then the Master is the one that has to write directly in the correct shard on one of the Slave machines. The Master server will certainly have more data traffic to handle, but actually it only has to redirect the stream of information towards one of the slaves; data balancing is granted.

A hashing function for load balancing between the Slaves is also needed. The master knows where to put and retrieve data from because a hashing function binds each key-value couple to a certain shard.

This structure is not as simple as a traditional one-server system, but it is not difficult to implement and could return some important results in terms of efficiency.

# Conclusions

The PC booking project, which we are involved in, is compatible with the translation of the table collecting the personal information of the users to a key-value model.

This not only it is possible, but it also introduces a certain number of positive effects when compared to traditional: lower impacts on system performances, higher flexibility due to the schemaless structure of key-value databases and greater ease of horizontal portioning. The Master-Slaves architecture can also improve the global performances in case of implementation using data sharding.

Because of the important positive effects of translating the table collecting personal data of the users from the relational model to the key-value model, we decided to implement it in our project.