FEASIBILITY STUDY

IMPLEMENTATION OF LEVELDB – A KEY VALUE DATABASE FOR FUNDRACING PROJECT APPLICATION

PURPOSE

Implement a key-value database, for improve the performance of a multi-client application, trying to make easier the server-side life.

1. Make the system faster in general
2. Reduce the number of requests done to server-side
3. Improve the responsiveness of the client side making write/read operation to server faster.

Note: if the number of request to the server-side decrease, it will answer to client faster, so point 3 can be partially solved by solving point 1

PROBLEMS

As levelDb implements a distributed key-value databases, first, there is to solve

CAP Theorem - (Consistency, Availability, Partition tolerance)

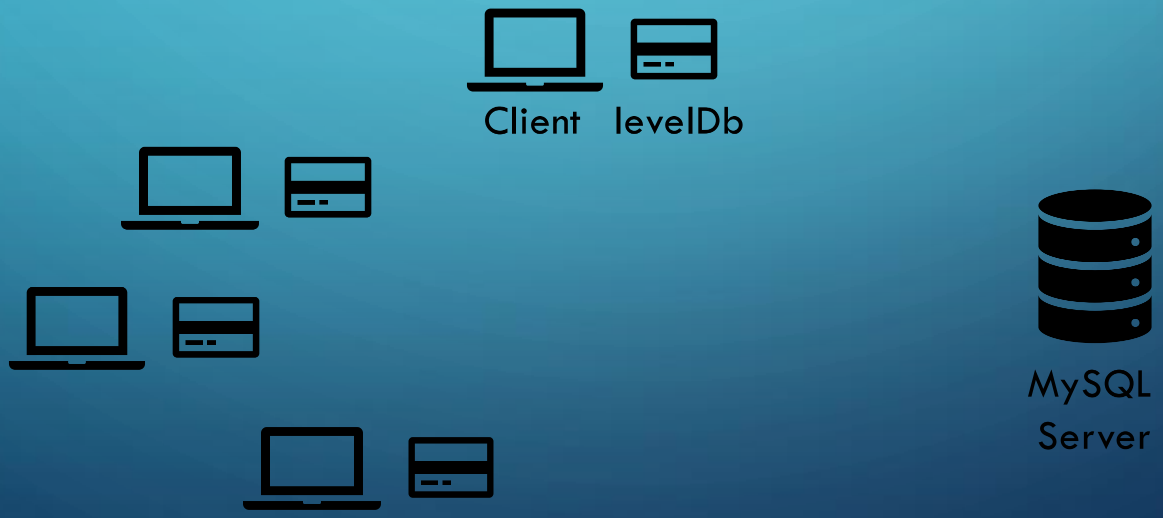
Next, due to the necessity to make cooperate levelDb and the MySQL server the system need to

1. Know what to store in the levelDb database
2. Choose appropriately the levelDb keys implementation

SOLUTION

We choose to implement levelDb key value database as a cache of the MySQL Server.

This cache is created and stored in each client node of the network. The clients can access only theirs own levelDb database and they ignore the esistence of the other ones.



CAP THEOREM – WHAT ?

In order to solve the Bewer's Theorem the system needs to provide:

- **Availabilty**

  Companies enjoy the application as they can finanance or create projects. if they could not, the application is useless. So the system must guarantee that data are always avaiable.

- **Partition Tollerance**

  Since levelDb database is implement like a personal client cache, there is no reason why, a partition of the network, must interfere with other clients work.

So the system is built not to provide a strong **consistency** of the data, wich can be implemented in a softer way.

CAP THEOREM – HOW ?

AVAILABILITY

This is garantuee using a simple cache paradigma:

- Write operation are always done in memory (MySQL Server) and they invalidate the cache.
- Read operation are always done in cache.

If we suppose that, at the beginning, the application read all the data that it needs and create the new cache. After that when it want some data from the database, it will be found it in the cache, so it will be always avaible.

If the application needs to write on the database (for example: send messages or register a new agency) it will call directly the database.

If the cache crash, the application can even restore it simply by asking data to the MySQL Server and creating a new one levelDb database.

CAP THEOREM – HOW ?

PARTITION TOLLERANCE

This is guarantee by the fact that each client operate only on his own cache, so: if a levelDb database in the network fail, the other client would never know it and it is not a problem.

SOFT CONSISTENCY

Since the user must be updated by the information that other users sent to him (messages, new project, financies ecc) sometimes the system must invalidate all the cache and rebuilt it to take the new data from the MySQL Server.

1. WHAT TO STORE ?

Since that levelDb database is accessible only for the client that created it, there is no reason why the cache must copy all the MySQL Server database.

LevelDb can store only the information that the user want to **read now** or in the future.

The application manage the tables of the MySQL Server with the JPA enties: AziendaEntity, ProgettoEntity, FinanziamentoEntity, MessaggioEntity. So is necessary to save:

- AziendaEntities – because the attribute of this entity are shown by the application to the user and permit to do the login.
- MessaggioEntities - but only the ones who are sended to the user.
- FinanziamentoEntities – no need to store because the user only write this information to the MySQL Server
- ProgettoEntities - because the attributes of this entity are shown by the application to the user. Furthermore the MySQL Server calculate even the "progress" and the "stake" values when asking for ProgettoEntity and they are shown to the user. So the cache must store even this two attributes.

2. CHOOSE THE KEY

Since that the name of the entity and their primary key are, togheter, unique in the database, we could choose the key in order to avoid collision like

NameEntity:primaryKey

So values in the key-value database will listed like

NameEntity:primaryKey:nameValue = $value

CONCLUSION

The application will not benefit a lot of the implementation of levelDb because:

- Most of the operations will be performed in any case to the MySQL server (write and read cache).

Despite this with a very cheap increasing of the application's complexity, it will be possibile to avoid a lot of *read* request to database. So it will increase a lot the scalability of the application, losing however, a strong consistency of the data that MySQL Server perform.