HBASE DESIGN CHOICES

Ignacio Amaya and Hugo Santana

Main considerations

We have considered two different designs to create the row key:

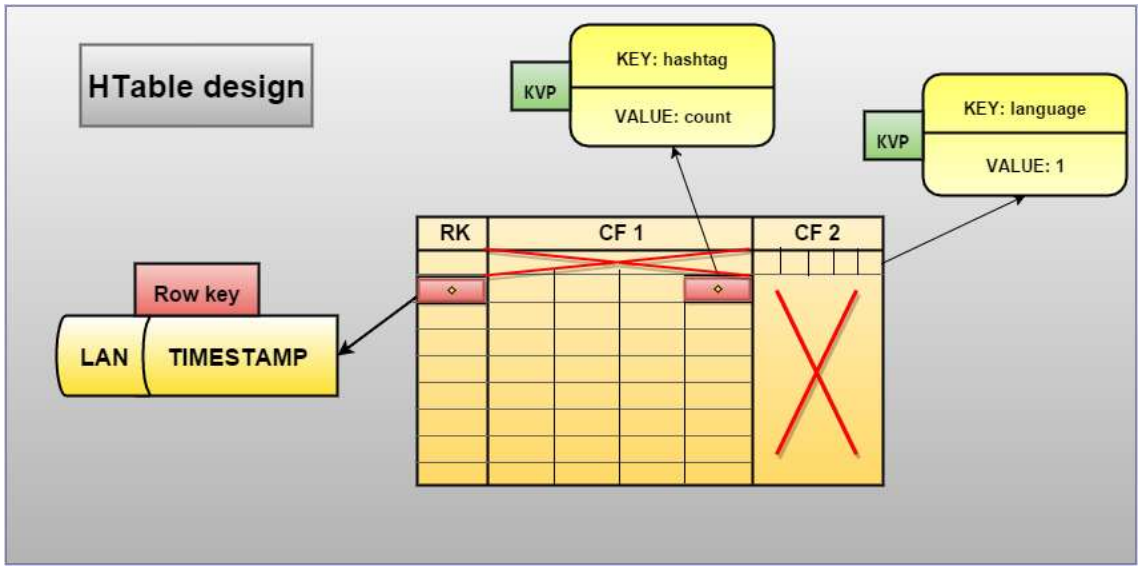
- Language + timestamp
- Timestamp + language

Both combinations are unique because given a timestamp and a language it is not possible to have the same hashtag several times. This is important because we don't want to overwrite values in the table when inserting. One of the main problems of creating the row key using the language + timestamp is that, although we have our first two queries optimized, the third query is not going to have a good performance.

We are assuming that our table is going to store a huge amount of data (terabyte order). Therefore we want all the queries to run fast when the timestamp's range is large. That's why we decided to add another column family to store the languages that are in the table, so the third query would perform as many scans as number of languages in our table (never more than 34, which is the maximum number of languages). Although with this approach the third query would be slower, the first two would be much faster because no filter operation will be needed. Another problem of the key starting with the timestamp is the overhead of inserting always at the end of the table, which could cause overhead in the node in charge of that part of the table. This could be bad is there is a big volume of inserts.

The second column family stores all the languages in the row with lowest value (any row could be chosen for that). The cells contain the languages as keys and a 1 as a key (we are not using that value for anything, so other could be chosen).

We can see in the next page a sketch of our design with the advantages and disadvantages of our design listed.



Pros

- Takes advantage of the multidimensional properties of HBase. If files contain more than 3 hashtags per line our design would still work.
- Filter operations have been avoided in all the three queries. Only scan operations are executed, which are more efficient.
- The overhead of writing always in the same node is solved.

Cons

- In the case of a range of time not very long and a large number of languages, doing a lot of scans could be more time consuming than executing filtering operations. However, we expect this table to be huge, so usually the ranges won't be so small.