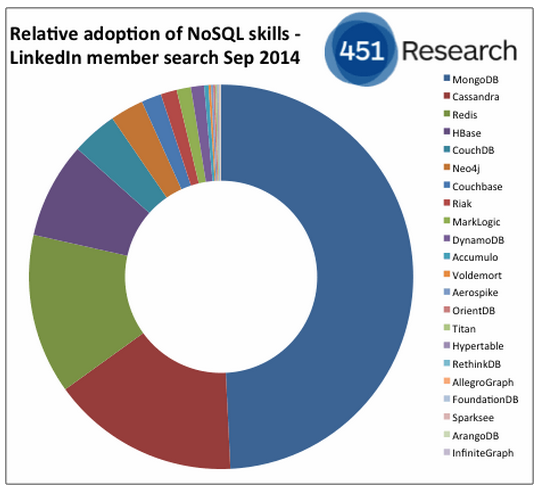
Facebook's NoSQL friend

**An introduction to HBase, the ‘Hadoop database’**

May 28, 2015 Ghislain Mazars

[](http://jaxenter.com/wp-content/uploads/2015/02/Screen-Shot-2015-02-09-at-15.11.17.png)

[Facebook](http://www.shutterstock.com/downloading_tips.mhtml?code=&id=176941343&size=medium&image_format=jpg&method=download&super_url=http%3A%2F%2Fdownload1.shutterstock.com%2Fgatekeeper%2FW3siZSI6MTQyMzUyMzk2OCwiYyI6Il9waG90b19zZXNzaW9uX2lkIiwiZGMiOiJpZGxfMTc2OTQxMzQzIiwicCI6InYxfDI1MTc2NjA3fDE3Njk0MTM0MyIsImsiOiJwaG90by8xNzY5NDEzNDMvbWVkaXVtLmpwZyIsIm0iOiIxIiwiZCI6InNodXR0ZXJzdG9jay1tZWRpYSJ9LCIzWHcvMFJDM0EyQzVMZitkWTFtMlBuY2F0clUiXQ%2Fshutterstock_176941343.jpg&racksite_id=ny&chosen_subscription=1&license=standard&src=0UJBzMJNWUwzZx-8vTprUg-1-1) image via Shutterstock NoSQL skill adoption

Although Facebook famously ditched Cassandra to use HBase for its messenger service, the NoSQL database remains largely overlooked. Ubeeko CEO Ghislain Mazars takes a look under the hood of HBase features.

In the myriad of [NoSQL](http://jaxenter.com/tag/nosql) databases today available on the market, HBase is far from having a comparable mindshare to market leader [MongoDB](http://jaxenter.com/tag/mongodb). Easy to learn, MongoDB is the NoSQL darling of most application developers. The document-oriented database interfaces well with lightweight data exchanges format, typically JSON, and has become the natural NoSQL database choice for many web and mobile apps.

Where MongoDB (and more generally JSON databases) reaches its limits is for highly scalable applications requiring complex data analysis (the oft denominated “data-intensive” applications). That segment is the sweet spot of column-oriented databases such as HBase. But even in that particular category, HBase has lately oft been overlooked in favour of Cassandra. Quite a surprising turn of event actually, as Facebook, the “creator” of Cassandra, ditched its own creation in 2011 and selected HBase as the database for its Messages application. We will come back to the technical differences between the two databases, but the main reason for Cassandra’s remarkable comeback is to be found elsewhere.

**Cassandra, the comeback kid of NoSQL databases**

With Cassandra, we find a pattern common to most major NoSQL databases, i.e. the presence of a dedicated corporate sponsor. Just as MongoDB (with MongoDB Inc, formerly called 10gen) and Couchbase (with Couchbase Inc.), the technical and market development of Cassandra is spearheaded by Datastax Inc. From a continued effort on documentation (Planet Cassandra) to the stewardship of the user community with meetups and summits, Datastax has been doing a remarkable job in waving high the Cassandra flag. These efforts have paid off, and Cassandra now holds the pole position among wide-column databases.

It is worth noting however that in the process, Cassandra has lost a lot of its open-source nature. 80% of the committers on the Apache project are from Datastax and the management features beloved by enterprise customers are proprietary and part of DSE (“DataStax Enterprise”). Going one step further, the integration with Apache Spark, the new whizz-kid of Big Data, is currently only available as part of DSE…

**HBase, a community-driven open-source project**

Unlike Cassandra, HBase very much remains a community-driven open-source project. No less than 12 companies are represented in the Apache project committee and the three Hadoop distributors, Cloudera, Hortonworks and MapR, share the responsibility of marketing the database and supporting its corporate users.

As a result, HBase sometimes lacks the marketing firepower of one company betting its life on the product. If it had been the case, no doubt that HBase would be in a 1.x release by now: while Hadoop made a big jump from 0.2x to 1.0 in 2011, HBase continued to move steadily in the 0.9x range! And the three companies including the database in their portfolio show a tendency to privilege other (more proprietary) offerings of theirs and thus provide a restrictive image of HBase.

In this context, it is a quite an achievement that HBase occupies such an enviable place among NoSQL databases. It owes this position to its open-source community, strong installed base within web properties (Facebook, Yahoo, Groupon, eBay, Pinterest) and distinctive Hadoop connection. So in spite or maybe thanks to its unusual model, HBase could still very much win… As Cassandra has shown in the last 2/3 years, things can move fast in this market. But we will come back to that later on, for now, let us take a more technical look at HBase.

**Under the hood**

**Hadoop implementation of Google’s BigTable**

HBase is an open-source implementation of Big Table as described in the 2005 paper from Google (http://research.google.com/archive/bigtable.html). Initially developed to store crawling data, BigTable remains the distributed database technology underpinning some of Google’s most famous services, Google Docs & Gmail. Of course, as should be expected from a creation of Google, the wide-column database is super scalable and works on commodity servers. It also features extremely high read performance, ensuring for example that a Gmail user instantaneously retrieves all its latest emails.

Just like BigTable, HBase is designed to handle massive amounts of data and is optimized for read-intensive applications. The database is implemented on top of the Hadoop Distributed File System (HDFS) and takes advantage of its linear scalability and fault tolerance. But the integration with Hadoop does not stop at using HDFS as the storage layer: HBase shares the same developer community as Hadoop and offers native integration with Hadoop MapReduce. HBase can serve as both the source or the destination of MapReduce jobs. The benefit here is clear: there is no need for any data movement between batch MapReduce ETL jobs and the host operational and analytics database.

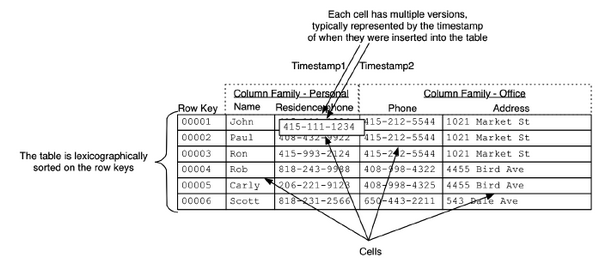
**HBase schema design**

HBase offers advanced features to map business problems to the data model, which makes it way more sophisticated than a plain key-value store such as Redis. Data in HBase is placed in tables, and the tables themselves are composed of rows, each of which has a rowkey.

The rowkey is the main entry point to the data: it can be seen as the equivalent of the primary key for a traditional RDBMS database. An interesting capability of HBase is that its rowkeys are byte arrays, so pretty much anything can serve as the rowkey. As an example, compound rowkeys can be created to mix different criteria into one single key, and optimize data access speed.

In pure key-value mode, a query on the rowkey will give back all the content of the row (or to take a columnar view, all of its columns). But the query can also be much more precise, and specifically address:

* A family of columns
* A specific column, and as a result a cell which is the intersection of a row and a column
* Or even a specific version of a cell, based on a timestamp

[](http://jaxenter.com/wp-content/uploads/2015/02/Screen-Shot-2015-02-09-at-15.27.19.png)

Source: Introduction to HBase Schema Design (Amandeep Khurana)

Combined, these different features greatly improve the base key-value model. With one constraint, the rowkey cannot be changed, and should thus be carefully selected at design stage to optimize row-key access or scan on a range of rowkeys. But beyond that, HBase offers a lot of flexibility: new columns can be added on the fly, all the rows do not need to contain the same columns (which makes it easy to add new attributes to an existing entity) and nested entities provide a way to define relationships within what otherwise remains a very flat model.

**HBase features**

**Sorted rowkeys**

Manipulation of HBase data is based on three primary methods: Get, Put, and Scan. For all of them, access to data is done by row and more specifically according to the rowkey. Hence the importance of selecting an appropriate rowkey to ensure efficient access to data. Usually, the focus will be on ensuring smooth retrieval of data: HBase is designed for applications requiring fast read performance, and the rowkey typically closely aligns with the application’s access patterns.

As scans are done over a range of rows, HBase lexicographically orders rows according to their rowkeys. Using these “sorted rowkeys”, a scan can be defined simply from its start and stop rowkeys. This is extremely powerful to get all relevant data in one single database call: if we are only interested in the most recent entries for an application, we can concatenate a timestamp with the main entity id to easily build an optimized request. Another classical example relates to the use of geohashed compound rowkeys to immediately get a list of all the nearby places for a request on a geographic point of interest.

**Control on data sharding**

In selecting the rowkey, it is important to keep in mind that the rowkey strongly influences the data sharding. Unlike traditional RDBMS databases, HBase provides the application developer with control on the physical distribution of data across the cluster. Column families also have an influence (all column members for a family share the same prefix), but the primary criteria is the rowkey to ensure data is evenly distributed across the Hadoop cluster (data is sorted in ascending order by rowkey, column families and finally column key). As rowkeys determine the sort order of a table’s row, each region in the table ends up being responsible for the physical storage of a part of the row key space.

Such an ability to perform physical-level tuning is a bit unusual in the database world nowadays, but immensely powerful if the application has a well-defined access pattern. In such case, the application developer will be able to guide how the data is spread across the cluster and avoid any hotspotting by skillfully selecting the rowkey. And, at the end of the day, disk access speed matters from an application usability perspective, so it is really good to have some control on it!

**Strong consistency**

In its overall design, HBase tends to favour consistency over availability. It even supports ACID-level semantics on a per-row basis. This of course has an impact on write performance, which will tend to be slower than comparable eventually consistent databases. But again, typical use cases for HBase are focused on a high read performance.

Overall, the trade-off plays in favour of the application developer, who will have the guarantee that the datastore always (vs eventually…) delivers the right value of the data. In effect, the choice of delivering strong consistency frees the application developer from having to implement cumbersome mechanics at the application level to mimic such guarantee. And it is always best when the application developer can focus on the business logic and user experience vs the plumbing…

**What’s next for HBase?**

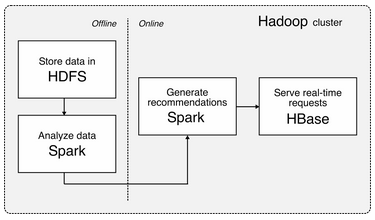
In a first section, we had a look at HBase position in the wider NoSQL ecosystem, and vis-à-vis its most direct competitor, Cassandra. In our second and third sections, we reviewed the key technical characteristics of HBase, and highlighted some key features of HBase that make it stand out from other NoSQL databases. In this final section, we will discuss recent initiatives building out on these capabilities and the chances of HBase becoming a mainstream operational database in a Hadoop-dominated environment.

**Support for SQL with Apache Phoenix**

Until recently, HBase did not offer any kind of SQL-like interaction language. That limitation is now over with Apache Phoenix, an open-source initiative for ad hoc querying of HBase.

Phoenix is an SQL skin for HBase, and provides a bridge between HBase and a relational model and approach to manipulate data. In practice, Phoenix compiles SQL queries to native HBase calls using another recent novelty of HBase, coprocessors. Unlike standard Hadoop SQL tools such as Hive, Phoenix can both read and write data, making it a more generic and complete HBase access tool.

**Further integration with Hadoop and Spark**

Over time, Hadoop has evolved from being mainly a HDFS + MapReduce batch environment to a complete data platform. An essential part of that transformation has been the advent of YARN, which provides a shared orchestration and resource management service for the different Hadoop components. With the delivery of project Slider end of 2014, HBase cluster resource utilisation can now be “controlled” from YARN, making it easier to run data processing jobs and HBase on the same Hadoop cluster.

With a different spin, the ongoing integration work behind HBase and Spark also contributes to the unification of database operations and analytic jobs on Hadoop. Just as for MapReduce, Spark can now utilize HBase as both a data source and a target. With nearly 2/3 of users loading data into Spark via HDFS, HBase is the natural database to host low-latency, interactive applications from within a Hadoop cluster. Advanced analytics provided by Spark can be fed backdirectly into HBase, delivering a closed-loop system, fully integrated with the Hadoop platform.

**Final thoughts**

With Hadoop moving from exploratory analytics to operational intelligence, HBase is set to further benefit from its position as the “Hadoop database”.  The imperative of limiting data movements will play strongly in its favour as enterprises start building complete data pipelines on top of their Hadoop “data lake”.

In parallel, HBase is a strong contender for emerging use cases such as the management of IoT-related time series data. Incidentally, the recent launch by Microsoft of a HBase as a Service offering on Azure should be read in that context.

For these reasons, there is no doubt that HBase will continue to grow steadily over the next few years. Still the opportunity is here for more, and for HBase to have a much bigger impact on the enterprise market. MapR has in this perspective recently made a promising move by incorporating its HBase-derived MapR-DB in its free community edition. For their part, Hortonworks and Cloudera have been active on the essential integrations with Slider and Spark. Now is the time for the HBase community and vendors to move to the next stage, and drive a rich enterprise roadmap for the “Hadoop database” – to make HBase sexy and attractive for mainstream enterprise customers!

Be the first to share this article with your network!