# Understanding HBase and BigTable

### From Jimbojw.com

The hardest part about learning HBase (the open source implementation of Google's BigTable), is just wrapping your mind around the concept of what it actually is.

I find it rather unfortunate that these two great systems contain the words *table* and *base* in their names, which tend to cause confusion among RDBMS indoctrinated individuals (like myself).

This article aims to describe these distributed data storage systems from a conceptual standpoint. After reading it, you should be better able to make an educated decision regarding when you might want to use HBase vs when you'd be better off with a "traditional" database.

## It's all in the terminology

Fortunately, Google's BigTable Paper clearly explains what BigTable actually is. Here is the first sentence of the "Data Model" section:

> A Bigtable is a sparse, distributed, persistent multidimensional sorted map.

*Note: At this juncture I like to give readers the opportunity to collect any brain matter which may have left their skulls upon reading that last line.*

The BigTable paper continues, explaining that:

> The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

Along those lines, the HBaseArchitecture page of the Hadoop wiki posits that:

> HBase uses a data model very similar to that of Bigtable. Users store data rows in labelled tables. A data row has a sortable key and an arbitrary number of columns. The table is stored sparsely, so that rows in the same table can have crazily-varying columns, if the user likes.

Although all of that may seem rather cryptic, it makes sense once you break it down a word at a time. I like to discuss them in this sequence: map, persistent, distributed, sorted, multidimensional, and sparse.

Rather than trying to picture a complete system all at once, I find it easier to build up a mental framework piecemeal, to ease into it...

## map

At its core, HBase/BigTable is a map. Depending on your programming language background, you may be more familiar with the terms associative array (PHP), dictionary (Python), Hash (Ruby), or Object (JavaScript).

From the wikipedia article, a map is "an abstract data type composed of a collection of keys and a collection of values, where each key is associated with one value."

Using JavaScript Object Notation, here's an example of a simple map where all the values are just strings:

```
{
  "zzzzz" : "woot",
  "xyz" : "hello",
  "aaaab" : "world",
  "1" : "x",
  "aaaaa" : "y"
}
```

## persistent

Persistence merely means that the data you put in this special map "persists" after the program that created or accessed it is finished. This is no different in concept than any other kind of persistent storage such as a file on a filesystem. Moving along...

## distributed

HBase and BigTable are built upon distributed filesystems so that the underlying file storage can be spread out among an array of independent machines.

HBase sits atop either Hadoop's Distributed File System (HDFS) or Amazon's Simple Storage Service (S3), while a BigTable makes use of the Google File System (GFS).

Data is replicated across a number of participating nodes in an analogous manner to how data is striped across discs in a RAID system.

For the purpose of this article, we don't really care which distributed filesystem implementation is being used. The important thing to understand is that it *is* distributed, which provides a layer of protection against, say, a node within the cluster failing.

## sorted

Unlike most map implementations, in HBase/BigTable the key/value pairs are kept in strict alphabetical order. That is to say that the row for the key "aaaaa" should be right next to the row with key "aaaab" and very far from the row with key "zzzzz".

Continuing our JSON example, the sorted version looks like this:

```
{
  "1" : "x",
  "aaaaa" : "y",
  "aaaab" : "world",
  "xyz" : "hello",
  "zzzzz" : "woot"
}
```

Because these systems tend to be so huge and distributed, this sorting feature is actually very important. The spacial propinquity of rows with like keys ensures that when you must scan the table, the items of greatest interest to you are near each other.

This is important when choosing a row key convention. For example, consider a table whose keys are domain names. It makes the most sense to list them in reverse notation (so "com.jimbojw.www" rather than "www.jimbojw.com") so that rows about a subdomain will be near the parent domain row.

Continuing the domain example, the row for the domain "mail.jimbojw.com" would be right next to the row for "www.jimbojw.com" rather than say "mail.xyz.com" which would happen if the keys were regular domain notation.

It's important to note that the term "sorted" when applied to HBase/BigTable does not mean that "values" are sorted. There is no automatic indexing of anything other than the keys, just as it would be in a plain-old map implementation.

## multidimensional

Up to this point, we haven't mentioned any concept of "columns", treating the "table" instead as a regular-old hash/map in concept. This is entirely intentional. The word "column" is another loaded word like "table" and "base" which carries the emotional baggage of years of RDBMS experience.

Instead, I find it easier to think about this like a multidimensional map - a map of maps if you will. Adding one dimension to our running JSON example gives us this:

```
{
  "1" : {
    "A" : "x",
    "B" : "z"
  },
  "aaaaa" : {
    "A" : "y",
    "B" : "w"
  },
  "aaaab" : {
    "A" : "world",
    "B" : "ocean"
  },
  "xyz" : {
    "A" : "hello",
    "B" : "there"
  },
  "zzzzz" : {
    "A" : "woot",
    "B" : "1337"
  }
}
```

In the above example, you'll notice now that each key points to a map with exactly two keys: "A" and "B". From here forward, we'll refer to the top-level key/map pair as a "row". Also, in BigTable/HBase nomenclature, the "A" and "B" mappings would be called "Column Families".

A table's column families are specified when the table is created, and are difficult or impossible to modify later. It can also be expensive to add new column families, so it's a good idea to specify all the ones you'll need up front.

Fortunately, a column family may have any number of columns, denoted by a column "qualifier" or "label". Here's a subset of our JSON example again, this time with the column qualifier dimension built in:

```
{
  // ...
  "aaaaa" : {
    "A" : {
      "foo" : "y",
      "bar" : "d"
    },
    "B" : {
      "" : "w"
    }
  },
  "aaaab" : {
    "A" : {
      "foo" : "world",
      "bar" : "domination"
    },
    "B" : {
      "" : "ocean"
    }
  },
  // ...
}
```

Notice that in the two rows shown, the "A" column family has two columns: "foo" and "bar", and the "B" column family has just one column whose qualifier is the empty string ("").

When asking HBase/BigTable for data, you must provide the full column name in the form "`<family>:<qualifier>`". So for example, both rows in the above example have three columns: "A:foo", "A:bar" and "B:".

Note that although the column families are static, the columns themselves are not. Consider this expanded row:

```
{
  // ...
  "zzzzz" : {
    "A" : {
      "catch_phrase" : "woot",
    }
  }
}
```

In this case, the "zzzzz" row has exactly one column, "A:catch_phrase". Because each row may have any number of different columns, there's no built-in way to query for a list of all columns in all rows. To get that information, you'd have to do a full table scan. You *can* however query for a list of all column families since these are immutable (more-or-less).

The final dimension represented in HBase/BigTable is time. All data is versioned either using an integer timestamp (seconds since the epoch), or another integer of your choice. The client may specify the timestamp when inserting data.

Consider this updated example utilizing arbitrary integral timestamps:

```
{
  // ...
  "aaaaa" : {
    "A" : {
      "foo" : {
        15 : "y",
        4 : "m"
      },
      "bar" : {
        15 : "d",
      }
    },
    "B" : {
      "" : {
        6 : "w"
        3 : "o"
        1 : "w"
      }
    }
  },
  // ...
}
```

Each column family may have its own rules regarding how many versions of a given cell to keep (a cell is identified by its rowkey/column pair) In most cases, applications will simply ask for a given cell's data, without specifying a timestamp. In that common case, HBase/BigTable will return the most recent version (the one with the highest timestamp) since it stores these in reverse chronological order.

If an application asks for a given row at a given timestamp, HBase will return cell data where the timestamp is less than or equal to the one provided.

Using our imaginary HBase table, querying for the row/column of "aaaaa"/"A:foo" will return "y" while querying for the row/column/timestamp of "aaaaa"/"A:foo"/10 will return "m". Querying for a row/column/timestamp of "aaaaa"/"A:foo"/2 will return a null result.

## sparse

The last keyword is sparse. As already mentioned, a given row can have any number of columns in each column family, or none at all. The other type of sparseness is row-based gaps, which merely means that there may be gaps between keys.

This, of course, makes perfect sense if you've been thinking about HBase/BigTable in the map-based terms of this article rather than perceived similar concepts in RDBMS's.

## And that's about it

Well, I hope that helps you understand conceptually what the HBase data model feels like.

As always, I look forward to your thoughts, comments and suggestions.

# Comments

Sorry, comments are disabled.

## Bryan Duxbury said ...

This is a great primer. We should incorporate something like this into our official documentation/wiki.

--Bryan Duxbury 10:42, 18 May 2008 (MST)

## Jimbojw said ...

Thanks Bryan,

I'm glad you think this is valuable. The hardest part for me learning Hbase was unlearning what I already knew about relational databases.

After several conversations with friends about Hbase, it became clear to me that what really was needed was a lay-programmer's conceptual primer, presenting the concepts in the right order to minimize confusion.

--Jimbojw 07:17, 19 May 2008 (MST)

## stack said ...