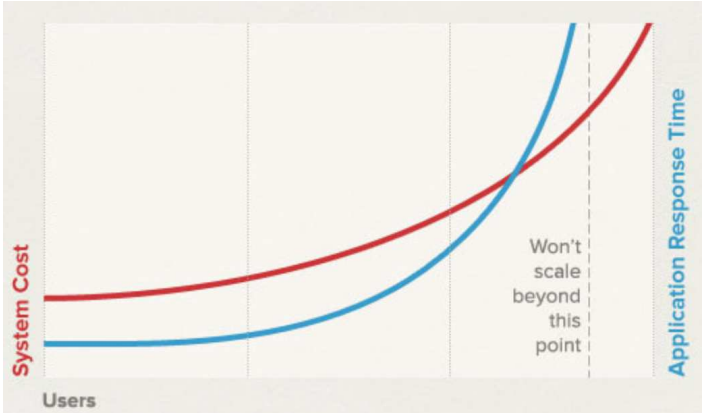
Principles of Databases

Introduction to NoSQL

David Sinclair

Scaling Databases

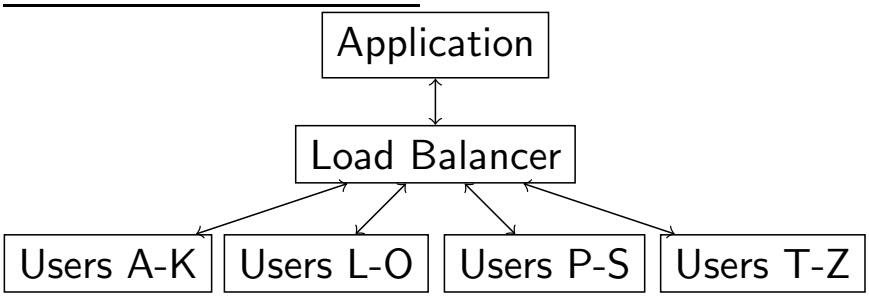
Relational database systems (RDBMS) are scaled by adding additional hardware processing power, *vertical scaling*. To support more users, or larger databases, typically requires a bigger, more powerful database server. This is also called *scaling up*. Practice has shown that that the cost of these systems increase "exponentially" with a linear increase in users, and that the system performance (response time) will approach a value that it cannot exceed.



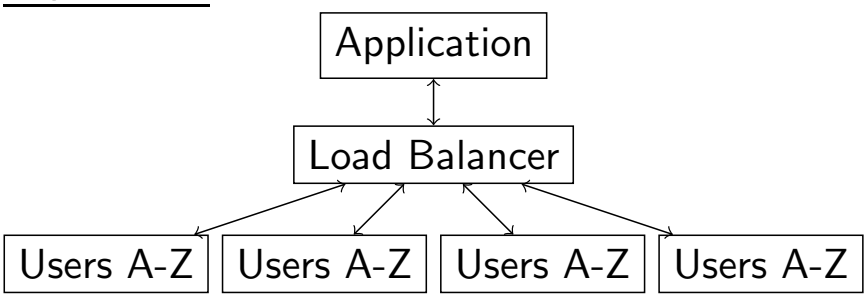
Scaling Databases (2)

Another approach is *horizontal scaling*, also called *scaling out*, where the load is spread out by either partitioning (sharding) or replication the data.

Partitioning (Sharding):

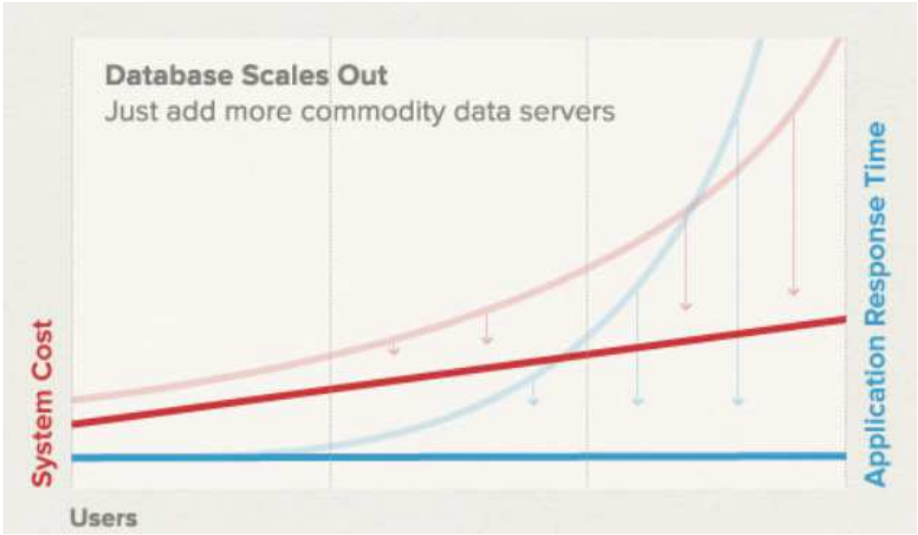


Replication:



Scaling Databases (3)

Horizontal scaling is better at handling load spikes.



Horizontal scaling has the effect of flattening both the cost and performance curves.

NoSQL

NoSQL stands for "Not only SQL" and represents a class of DBMS where:

- SQL is not the query language.
- The internal architecture, and in particular the data, is distributed (sharding and replication) and fault-tolerant.
- There is no fixed schema.
- There are no joins.
  - Joins require a fixed schema and strong consistency.
  - Joins are expensive operations in a distributed environment.

NoSQL databases are not a replacement for relational databases. In specific environment they can be used to compliment relational databases.

CAP Theorem

This is more of a conjecture rather than a theorem and it states that it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

- Consistency: all nodes see the same data at the same time.
- Availability: a guarantee that every request receives a response about whether it was successful or failed.
- Partition tolerance: the system continues to operate despite arbitrary message loss or failure of part of the system.

A distributed system can satisfy any two of these guarantees at the same time, but not all three.

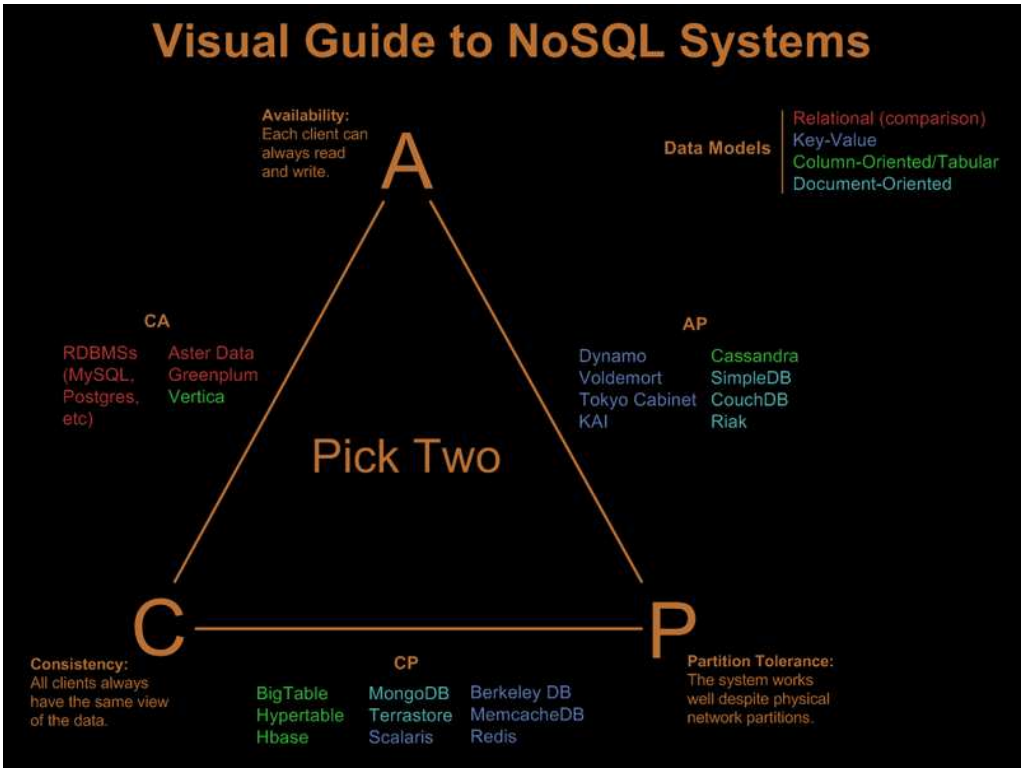
CAP Theorem (2)

In relational databases we do not have the Partition tolerance property and hence Consistency and Availability properties are achieved.

In NoSQL databases we want to have the Partition tolerance property, so we need to select to keep either the Consistency and Availability property.

- If we drop the Availability property then we have to accept waiting until data is consistent.
- If we drop the Consistency property then we have to accept getting inconsistent data sometimes.

CAP Theorem (3)



Source: Nathan Hurst's Blog, http://blog.nahurst.com/visual-guide-to-nosql-systems

BASE

Relational databases are built on the ACID properties.

Atomicity All of the operations in the transaction will complete, or none will.

Consistency Transactions are never observe or result in inconsistent data.

Isolation Transactions will behave as if they is the only operation being performed upon the database.

Durability Upon completion of a transaction, the operation cannot be reversed and is permanent.

BASE (2)

In order to achieve scalability and improved performance, NoSQL databases have to sacrifice the ACID properties. Instead they have BASE properties.

Basically Available The use replication and sharding reduces the likelihood of data becoming unavailable. If there are failures they tend to be partial. The result is a system that is always available, even though subsets of the data may become unavailable for short periods of time.

Soft state Soft state indicates that the state of the system may change over time, becoming temporarily inconsistent, even without input. However the system will eventually become consistent.

Eventually consistent NoSQL systems ensure that at some future point in time the data assumes a consistent state.

Key-Value Stores

Key-Value Stores, such as Amazon's Dynamo, are based on a hash table where there is a unique key and a pointer to a particular item of data. These mappings are usually accompanied by cache mechanisms to maximize performance.

Advanced Key-Value Stores is able to sort the keys, which enables range queries as well as an ordered processing of keys. They are very quick, high availability systems but not capable of complex queries.

Column Family Stores

Column Family Stores are NoSQL databases where is data is stored in a column oriented manner. The main concept is a *keyspace*. A keyspace is similar to a schema in the relational model. The keyspace contains all the column families (kind of similar to tables in the relational model), which contain rows, which contain columns.

- A column family consists of multiple rows.
- Each row has a key.
- Each row can contain a different number of columns to the other rows and they can have different column names, data types, etc.
- Each column is contained to its row and doesn't span all rows like in a relational database. Each column contains a name/value pair, along with a timestamp (in Unix time).

Column Family Stores (2)

The following example shows a Column Family containing 3 rows where each row has its own set of columns.

Row Key	Columns
Bob	**email**: bob@dcu.ie, 1465676582 | **gender**: male, 1465676582 | **age**: 40, 1465676582
Brittany	**email**: brittany@dcu.ie, 1465676432 | **gender**: female, 1465676432
Tori	**email**: bob@dcu.ie, 1465676152 | **country**: Ireland, 1465676152 | **hair colour**: red, 1465676311

Column Family Stores (3)

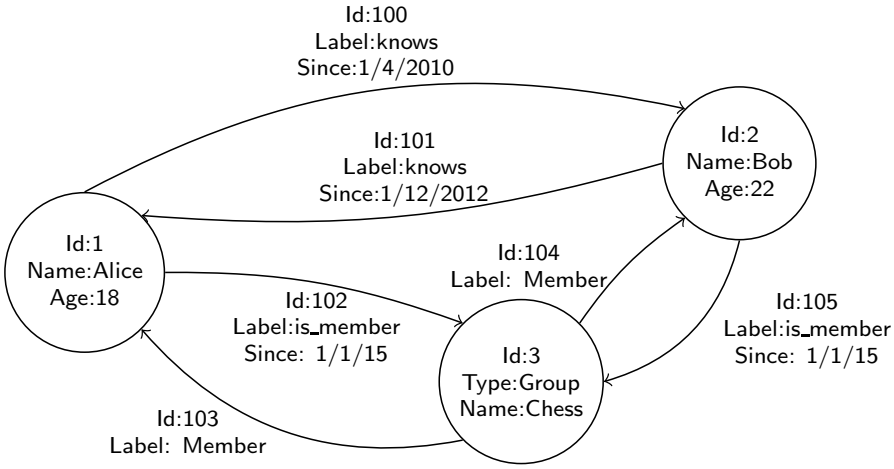
Benefits of Column Family Stores include:

- Compression: they are very efficient at data compression and/or partitioning.
- Aggregation queries: because of their structure, columnar databases perform particularly well with aggregation queries (such as SUM, COUNT, AVG, etc).
- Scalability: they are very scalable and are well suited to massively parallel processing with data spread across a large cluster of machines - typically thousands of machines.
- Fast to load and query: a billion row table could be loaded within a few seconds and queried almost immediately.

These are just some of the benefits that make Column Family Stores a popular choice for organisations dealing with big data. Examples of Column Family Stores are Google's BigTable, Apache's HBase and Cassandra databases.

Graph Databases

In *Graph Databases* entities are represented by nodes. Nodes have properties (key/value pairs) that record pertinent information about the node Edges between nodes represent a relationship between the entities the nodes represent. The edges may also have properties (key/value pairs) that record pertinent information about the relationship.



Graph Databases (2)

Graph Databases are very good at complex multi-level queries. For example, if you wanted to know "the movie about submarines with the actor who was in that movie with that other actor that played the lead in Gone With the Wind". This would require the system to:

- find the lead actor in Gone With the Wind
- find all the movies they were in
- find all the actors in all of those movies who were not the lead in Gone With the Wind
- find all of the movies they were in
- find the common films
- finally filtering that list to those with descriptions containing "submarine".

In a relational database this will require several separate searches through the movies and actors tables, doing another search on submarine movies, finding all the actors in those movies, and the comparing the (large) collected results.

Graph Databases (3)

In contrast, the graph database would simply walk from Gone With the Wind to Clark Gable, gather the links to the movies he has been in, gather the links out of those movies to other actors, and then follow the links out of those actors back to the list of movies. The resulting list of movies can then be searched for "submarine". All of this can be done via one search.

They can scale more naturally to large data sets as they do not typically need costly join operations. However, partitioning and replication in *Graph Databases* can be problematical.

Examples of *Graph Databases* are Neo4J, GraphBase and IBM's Graph.

Document Stores

Document Stores, also known as *Document-Oriented Databases*, have become very popular with the inclusion of MongoDB in the Java MEAN framework.

- *Document Stores* use semi-structured data. There are no fixed schema. Each document can have its own structure, usually represented in XML or JSON.
- Each document has a unique key, `_id`.
- In some systems, documents can be grouped into *collections*.
- A document is typically a set of Key-Value pairs.
- The Value of a Key-Value pairs can contain arrays or another document.

Document Stores (2)

Suppose we have a database of images where we store the meta-data and tags. In a relational model, this could be:

Images

id	filename	mimetype	size
1	cow.jpg	image/jpg	9123
2	bunny.png	image/png	8192

Tags

id	value
1	animal
2	cute
3	tasty

ImageTags

imageId	tagId
1	1
1	3
2	1
2	2

To find the meta-data and tags for bunny.png (`id = 2`) would require the following 2 queries:

```
SELECT * FROM Images WHERE id = 2;

SELECT value
   FROM ImageTags JOIN Tags ON (Tags.id = ImageTags.tagId)
   WHERE ImageTags.imageId = 2;
```

Document Stores (3)

In MongoDB the same dataset could be represented as:

Images

```
{
_id:  1,
filename:  'cow.jpg',
mimetype:  'image/jpg',
size:  9123,
tags:  [ 'animal', 'tasty' ]
},
{
_id:  2,
filename:  'bunny.png',
mimetype:  'image/png',
size:  8192,
tags:  [ 'animal', 'cute' ]
}
```

and the following query would gather all the data we need

```
db.Images.find( { _id:2} );
```

as the data is in one place.

Document Stores (4)

One big advantage of *Document Stores* is that you can add key-value pairs to one document in a store or collection without having to add a similar field to every other document (you do not have to change the schema). In our example we could add a pair {`compression_ratio:10`} to document with `_id:1` to record the variable compression ratio for the .jpg image. This field would make no sense for a .png image.

The big disadvantage of *Document Stores* is data integrity. Some *Document Stores* allow documents to include *document references* to address this but this comes at a performance cost.

Examples of *Document Stores* are CouchDB, MongoDB, Lotus Notes, Elastic Search and Redis.