


Dan Sullivan



# NOSQL

FOR MERE MORTALS®

An abstract painting with a complex, layered composition. It features bold, dark, diagonal strokes in black and dark brown that crisscross the frame. These strokes are set against a background of vibrant, textured brushstrokes in shades of blue, purple, white, and yellow. The overall effect is one of dynamic energy and visual complexity.

## Software-Independent Approach!

If you find yourself working around the constraints of relational databases, then a NoSQL database might be a better option. This book will help you identify and implement the best NoSQL database for your application.

# **NoSQL for Mere Mortals<sup>®</sup>**

❖ **Tip** You would, of course, need to update your code to handle both ways of representing customer names or convert all instances of one form into the other.

Part III, “Document Databases,” returns to the concept of schemaless databases and discusses the related concept of a polymorphic database, which is something of a middle ground between fixed schemas found in relational databases and schemaless models used in key-value databases.

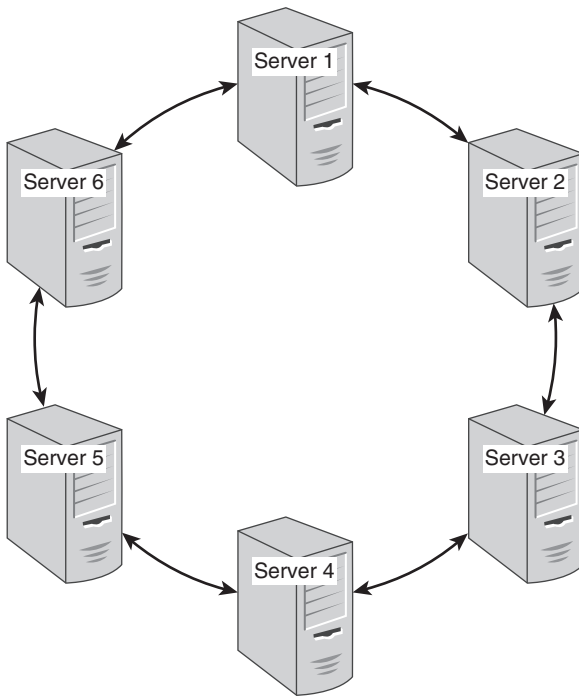
## Key-Value Architecture Terms

The architecture of a key-value database is a set of characteristics about the servers, networking components, and related software that allows multiple servers to coordinate their work. Three terms frequently appear when discussing key-value architectures:

- Clusters
- Rings
- Replication

### Cluster

*Clusters* are sets of connected computers that coordinate their operations (see Figure 4.7). Clusters may be loosely or tightly coupled. Loosely coupled clusters consist of fairly independent servers that complete many functions on their own with minimal coordination with other servers in the cluster. Tightly coupled clusters tend to have high levels of communication between servers. This is needed to support more coordinated operations, or calculations, on the cluster. Key-value clusters tend to be loosely coupled.



**Figure 4.7** *A ring architecture of key-value databases links adjacent nodes in the cluster.*

Servers, also known as nodes, in a loosely coupled cluster share information about the range of data the server is responsible for and routinely send messages to each other to indicate they are still functioning. The latter message exchange is used to detect failed nodes. When a node fails, the other nodes in the cluster can respond by taking over the work of that node.

Some clusters have a master node. The master node in Redis, for example, is responsible for accepting read and write operations and copying, or replicating, copies of data to slave nodes that respond to read requests. If a master node fails, the remaining nodes in the cluster will elect a new master node. If a slave node fails, the other nodes in the cluster can continue to respond to read requests.

Masterless clusters, such as used by Riak, have nodes that all carry out operations to support read and write operations. If one of those nodes fails, other nodes will take on the read and write responsibilities of the failed node.

Because the failed node was also responsible for writes, the nodes that take over for the failed node must have copies of the failed node's data. Ensuring there are multiple copies of data on different nodes is the responsibility of the replication subsystem. This is described in the section "Replication," later in this chapter.

Each node in a masterless cluster is responsible for managing some set of partitions. One way to organize partitions is in a ring structure.

## Ring

A ring is a logical structure for organizing partitions. A *ring* is a circular pattern in which each server or instance of key-value database software running on a server is linked to two adjacent servers or instances. Each server or instance is responsible for managing a range of data based on a partition key.

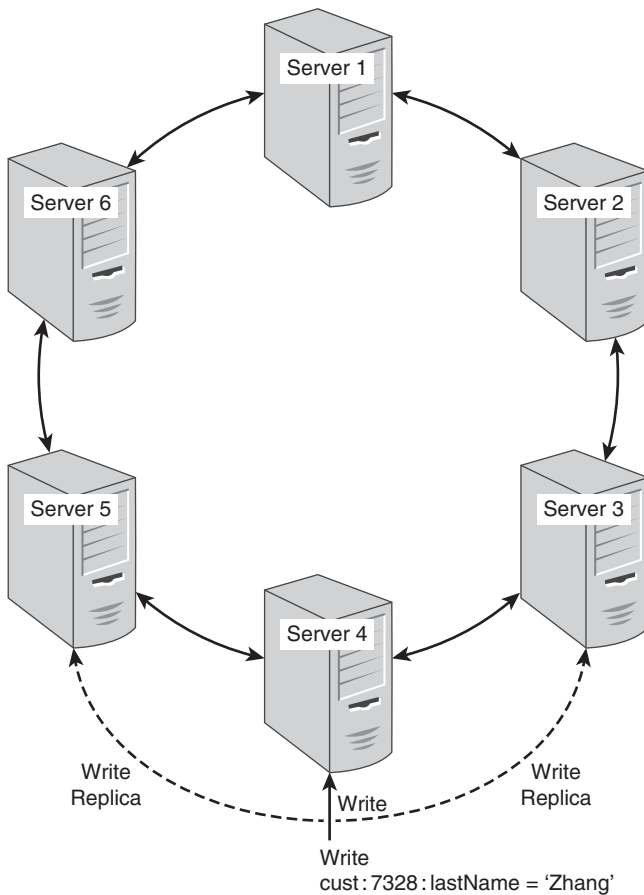
Consider a simple hashlike function that maps a partition key from a string; for example, 'cust:8983:firstName' to a number between 0 and 95. Now assume that you have an eight-node cluster and the servers are labeled Server 1, Server 2, Server 3, and so on. With eight servers and 96 possible hashlike values, you could map the partitions to servers, as shown in Table 4.1.

**Table 4.1** *Server to Partition Mapping*

| Server Name | Partition Range |
|-------------|-----------------|
| Server 1    | 0–11            |
| Server 2    | 12–23           |
| Server 3    | 24–35           |
| Server 4    | 36–47           |
| Server 5    | 48–59           |
| Server 6    | 60–71           |
| Server 7    | 72–83           |
| Server 8    | 84–95           |

In this model, Server 2 is linked to Server 1 and Server 3; Server 3 is linked to Server 2 and Server 4; and so on. Server 1 is linked to Server 8 and Server 2. Refer to Figure 4.7 to see a graphical depiction of a ring architecture.

A ring architecture helps to simplify some otherwise potentially complex operations. For example, whenever a piece of data is written to a server, it is also written to the two servers linked to the original server. This enables high availability of a key-value database. For example, if Server 4 fails, both Server 3 and Server 5 could respond to read requests for the data on Server 4. Servers 3 and 5 could also accept write operations destined for Server 4. When Server 4 is back online, Servers 3 and 5 can update Server 4 with the writes that occurred while it was down (see Figure 4.8).



**Figure 4.8** One way to replicate data is to write copies of data to adjacent nodes in the cluster ring.

## Replication

Replication is the process of saving multiple copies of data in your cluster. This provides for high availability as described previously.

One parameter you will want to consider is the number of replicas to maintain. The more replicas you have, the less likely you will lose data; however, you might have lower performance with a large

number of replicas. If your data is easily regenerated and reloaded into your key-value database, you might want to use a small number of replicas. If you have little tolerance for losing data, a higher replica number is recommended.

Some NoSQL databases enable you to specify how many replicas must be written before a write operation is considered complete from the perspective of the application sending the write request. For example, you may configure your database to store three replicas. You may also specify that as soon as two of the replicas are successfully written, a successful write return value can be sent to the application making the write request. The third replica will still be written, but it will be done while the application continues to do other work.

You should take replicas into consideration with reads as well. Because key-value databases do not typically enforce two-phase commits, it is possible that replicas have different versions of data. All the versions will eventually be consistent, but sometimes they may be out of sync for short periods.

To minimize the risk of reading old, out-of-date data, you can specify the number of nodes that must respond with the same answer to a read request before a response is returned to the calling application. If you are keeping three replicas of data, you may want to have at least two responses from replicas before issuing a response to the calling program.

The higher the number required, the more likely you are to send the latest response. This can add to the latency of the read because you might have to wait longer for the third server to respond.

Up to this point, most of the terms described have dealt with logical modeling and the organization of servers and related processes. Now it is time to address algorithms implemented in and processes that run within the key-value database software to implement higher-level functions.