Sharding and Master-Slave Replication of NoSQL Databases: Comparison of MongoDB and Redis

Anikó Vágner[a] and Mustafa Al-Zaidi[b]
Department of Information Technology, Faculty of Informatics, University of Debrecen, Hungary

Keywords: NoSQL Databases, Cluster, Replication, Sharding, Distribution Models.

Abstract: One of the reasons that NoSQL databases were born is that they can be used in clusters, namely, computers work together, share data and from the client side, the clustered computers look as if there is only one computer. In this paper, the distribution models of two NoSQL databases are introduced. We chose the databases from the database ranking website (noa, 2023a), exactly the two first NoSQL databases: MongoDB and Redis. However they belong to two different NoSQL database categories, they use similar key-value pairs which is the main basis of the clustering. Additionally, the distribution models do not depend on the categories of the databases, both database management systems know sharding and master-slave replication, and can use these two distribution models together. These two database management systems do not know peer-to-peer replication. Our goal was to get to know whether there are similarities between the structures of clustered computers of each database management system. If we consider the theory, the answer should be yes, they are similar to each other: for the sharding 2 computers are enough, similarly for the replication 2 computers are also enough, and if both of the techniques are used, 4 computers should be enough.

1 INTRODUCTION

In the early 21st century new types of database management systems were born, which got a new name: NoSQL databases. They were born because developers needed easier development, faster solutions, and distributed databases compare to relational database management systems. Considering the distributed database concept, reliability can be mentioned as the main advance of the NoSQL databases, since in clustered environment, if a computer fails the whole system can work and can be available. (Sadalage and Fowler, 2013)

The NoSQL database topic distinguishes more categories, like key-value, column-family, document, graph, and search engines. The list of categories is long, and each category is not exactly clean. For example, MongoDB is a document database but can also be considered a search engine. Additionally, MongoDB can also be regarded as a key-value database since it stores JSON documents as values under keys. Moreover, Redis is a key-value database, but it can store JSON also, so in an extended sense it can be a document database. Despite this fact, Redis does not offer many tools which manage JSON documents well. (noa, 2023a) (noa, 2023b) (noa, 2023c) (Banker et al., 2016) (Sadalage and Fowler, 2013)

Many NoSQL databases offer more kinds of distribution models, like master-slave replication or peer-to-peer replication and sharding. They can call it some other names, but the concepts finally are the same. The replication and the sharding can be used together in a cluster. The sharding splits the whole database into many parts, whereas the replication copies each piece of the database to many computers. (Sadalage and Fowler, 2013)

MongoDB as a document database and Redis as a key-value database offer master-slave replication and sharding (noa, 2023b) (noa, 2023c) (Banker et al., 2016) (Carlson and Sanfilippo, 2013). In this paper, we introduce their exact architecture, examine how they work, and finally, we compare them.

2 DISTRIBUTION MODELS

In this section, we consider the distribution models of NoSQL databases in general.

[a] https://orcid.org/0000-0001-8843-2403
[b] https://orcid.org/0009-0003-2561-4500

VÃ¡gner, A. and Al-Zaidi, M.
Sharding and Master-Slave Replication of NoSQL Databases: Comparison of MongoDB and Redis.
DOI: 10.5220/0012142700003541
In *Proceedings of the 12th International Conference on Data Science, Technology and Applications (DATA 2023)*, pages 576-582
ISBN: 978-989-758-664-4; ISSN: 2184-285X

2.1 Master-Slave Replication

In the master-slave replication solution the master contains all data of the database, it gets every update for the data. A slave copies every data from the master. The master can have many slaves, additionally, the slaves can also have slaves.

This solution is good when the application needs a lot of reading and only a little writing. The slaves can support the reads, as there can be more slaves, and the reading workload can go to the slaves. In a very heavy reading environment, the master can send the data to 2-3 slaves, which can have also slaves. The task of the first-level slaves is only to give the data to the second-level slaves which satisfy the reading tasks.

If a slave fails, it will not influence the work of the whole system. If the master fails, a slave can easily take over the task of the master. (Sadalage and Fowler, 2013)

2.2 Peer-to-Peer Replication

In this case, every node has the same data, namely the whole database, and every node can accept writes and reads. If one node has a write the changed data should be sent to other nodes to update there also. Redis and MongoDB do not support this solution. (noa, 2023b) (noa, 2023c) (Banker et al., 2016) (Carlson and Sanfilippo, 2013) (Sadalage and Fowler, 2013)

2.3 Sharding

Sharding means that the database has pieces. One piece is a shard. In the three NoSQL models (key-value, column-family, and document), every piece has a key, and under the key, there is a kind of ”value”. In the document database, the “value” is a document, in the column-family database, the “value” is a column-family, and in the key-value database, the “value” is a value. So a shard is a key-”value” pair, where the key helps to grab a shard.

In the sharding distribution model, every shard is stored only on one node. There are more opportunities for how the shards are assigned to the nodes:

- the application (the developer) decides it,
- an additional software decides it, or
- the NoSQL database itself decides it.

From these three solutions, we will consider only the last one, where the NoSQL database decides where the shared should be stored because the advantage of the NoSQL databases and the distribution model is that the cluster can be easily expanded even with more nodes at the same time, and the cluster should give the solution on which node will be the new place of each shard and how they will be moved there.

With the sharding, we can realize the horizontal scaling, namely, we can add more nodes to the cluster, so we can increase the capacity of the whole system.

2.4 Master-Slave Replication and Sharding

In this case, there are more masters, and we can consider that these more masters realize the sharding. Every master has their own slaves, who get copies of the data of the master. We can say that the slaves do not need to know that there is a sharding also in the system. (Sadalage and Fowler, 2013)

2.5 Peer-to-Peer Replication and Sharding

In this case, every shard is copied more times. It is a parameter called replication factor that shows each shard how many times is copied to the different nodes. The replication factor is 3 in most cases, the literature and the documentation take it as a simple fact, and everybody uses this number, even though the database management systems support other numbers as replication factors also. (noa, 2023b) (noa, 2023c) (Banker et al., 2016) (Carlson and Sanfilippo, 2013) (Sadalage and Fowler, 2013)

If the replication factor is one it means that there is no replication. If the replication factor is two, it means there is one copy of every piece of data, but they say that it is not enough, if we lose one of them, the other copy is not enough. With replication factor 3, there are 2 copies of each shard, if we lose one, the two copies help our application, namely, the system can find the other two copies of the data. Finally, if the replication factor is 4 or more, it is very hard to maintain the database, it needs a lot of waiting when the data is inserted, updated, or deleted. So in the NoSQL world, it is decided that the replication factor is 3 in most cases.

If the replication factor is 3, we need at least 4 computers to realize this distribution model. Every shard is stored 3 times on three different nodes. So each node does not contain every shard.

3 LITERATURE

Many research papers compare NoSQL databases, but in most cases not in detail. They introduce the four main categories and give information about distribution models, as they know the replication and the sharding. (Corbellini et al., 2017) (Gajendran, 2012) (Hecht and Jablonski, 2011) (Indrawan-Santiago, 2012) (Jing Han et al., 2011) (Lourenço et al., 2015) (Tudorica and Bucur, 2011)

Gajendran (Gajendran, 2012) says that MongoDB uses multi-version concurrency control (MVCC), which means that multiple versions are stored for each data, but only one is the latest. They state that MongoDB uses automatic sharding with automatic failover and load balancing. Sharding is based on a collection and not on the database. They state that MongoDB uses asynchronous replication.

Cattell (Cattell, 2011), Diogo (Diogo et al., 2019), Gessert (Gessert et al., 2017), Hecht (Hecht and Jablonski, 2011), and Moniruzzaman (Moniruzzaman and Hossain, 2013) compared MongoDB and Redis with some other database management systems. They considered the sharding and the replication of the DBMSs with some other aspects like storage and query processing.

Davoudian (Davoudian et al., 2018) gives a very comprehensive research paper about NoSQL databases. They introduce the evolution of NoSQL databases and speak about the four main categories and distribution models. They introduce the key-oriented and traversal-oriented sharding strategies. MongoDB and Redis use the key-oriented sharding strategy, where the unit of the shard is the key-”value” pair. Using the traversal-oriented strategy NoSQL databases partition less connective nodes and group highly related nodes. This solution is used in for example graph databases.

4 MongoDB

MongoDB (noa, 2023b), (Banker et al., 2016) stores BSON documents. It organizes documents into collections. A collection stores similar documents, namely, it acts as a database table of relational database management systems. However, MongoDB does not offer a predefined structure for a collection like the relational DBMSs, only a kind of optional validation that is similar to the XML validation. So a MongoDB collection stores many documents, which can be considered to be similar to each other, but their structure does not need to be the same.

The collections are stored in a MongoDB database. A MongoDB server can store more MongoDB databases.

4.1 Master-Slave Replication of MongoDB

The master node is called primary by MongoDB (noa, 2023b), (Banker et al., 2016) and it receives the write operations. The slaves are called secondaries, which copy the operations from the primary node based on a kind of log stream called oplog.

MongoDB thinks in replica sets for which the documentation recommends using three nodes: a primary node and two secondary nodes. In the initial replica set, we have to add at least three nodes with some configuration parameters and statements, and the nodes of the replica set vote which one is the primary (master). Despite the recommended 3 nodes, MongoDB can build the replica set only with two nodes also. In a replica set, there must be one and only one primary node.

If the primary (master) fails, one of the secondaries (slaves) can be a new primary. The nodes of the replica set vote for the new master from among the secondary nodes.

Next to the primary and secondary nodes, there can be some arbiters in the replica set. The arbiter takes part only in the voting if it is needed. An arbiter node does not store any data. A ready replica set can be expanded with an arbiter.

Of course, secondaries can be also added to the replica set. Altogether a replica set can have a maximum of 50 members from which a maximum of 7 arbiters can be.

The primary can transmit the data and the operations to the secondaries in two ways: initial synchronization when a new slave (secondary) connects to the cluster the primary node sends the full data set to the slave, and after the full synchronization, the primary node sends the ongoing changes to the new (and all) secondary nodes. For the ongoing changes, namely, for the new write operations, MongoDB uses the oplog, which is a kind of log of the write statements. MongoDB supports asynchronous replication, but with the options of the write statements, we can reach that the write becomes synchronous.

The client can connect to the replica set itself, so it should know all nodes. It means that the client does not know with which node it communicates. So the read and write statements go to the replica set itself, and the database decides on which node the statement is performed.

4.2 Sharding of MongoDB

In MongoDB, the unit of the sharding is a document. MongoDB (noa, 2023b), (Banker et al., 2016) uses shard keys to distribute the documents of a collection. The shard key is a field or more fields of the document. The shard key is not the same as the document key (which is the document id in MongoDB) in most cases.

To shard the data, the collection has to have an index in which the first part should be the shard key. This means that the sharding settings belong to a collection. If a collection is sharded, MongoDB does not provide any method to unshard the collection.

MongoDB offers two solutions to span the shard keys:

- ranges: based on the keys it creates non-overlapping ranges and based on the value of the shard key(s) it distributes the documents among the nodes.
- hashed shard key: using a hash function on the shard key, MongoDB creates a hashed shard key. Similar to the range solution, in this case, the ranges will also be created but based on the hashed shard keys. The hashed shard key supports more even distribution of the sharded data.

If a new shard is added to the sharded cluster, MongoDB can migrate the shards to the new node if we ask it. MongoDB automatically migrates a shard if the system is unbalanced.

In the database, there can be sharded and non-sharded collections. Non-sharded collections are stored on the primary shard of the database. Each database should have its own primary shard.

To build sharding with MongoDB, we need at least two nodes for the shards, and additionally at least a config server node and at least a router (mongos) (in most cases in another node). The router (mongos) connects to the config server, and the administrator adds the shard nodes (replica sets) to the cluster. The only way that the clients can communicate with the sharded database is by the mongos (router).

4.3 Combination of Master-Slave Replication and Sharding in MongoDB

To build the combination of master-slave replication and sharding in MongoDB, we need at least two replica sets, each of which with at least two nodes (a master and a slave), a replica set for the config server with at least two nodes (a master and a slave), at least a router (mongos), additional a client node, with which we can connect to the cluster. In the replica sets, there can be more than one slave, in a replica set the nodes (master and slaves) will vote on who is the master. So a minimal number of nodes to build this cluster is 7 plus the client.

4.4 An Environment of a Cluster of MongoDB

To try out the distribution models of MongoDB we installed MongoDB 5.0 on Ubuntu Linux 20.10 server. As a client, we connected to the MongoDB cluster from Python 3.10 with PyMongo 4.0.1.

5 Redis

Redis (noa, 2023c) (Carlson and Sanfilippo, 2013) is a key-value database, it offers five data structures: string, set, list, hash, and sorted set (zset). Every data structure has a key, which can be used as a key of the sharding.

5.1 Master-Slave Replication in Redis

The realization is very simple, the slave gets a config entry, that shows it should copy the data of the master which can be found in a given node. The master recognizes the slaves. So in a minimal configuration, Redis needs only two nodes to build this cluster.

The synchronization between the master and slave is based on an initial replication in which the whole database is copied to the slave and a replication stream in which the new statements are sent to the slave. When the slave loses the master, an initial replication can happen again.

A master can have more slaves. Moreover, the slaves can have also slaves. In this case, we can build a tree with three (or more) levels. In the first level of the tree, the root is the master, which sends data to the second-level slaves. The task of second-level slaves can only transmit the data to the third-level slaves. Finally, the third-level slaves can be read by the clients.

Redis allows the clients to read every node in this cluster. The master can also be read, but the documentation suggests reading the slaves. The clients can write the master only. If we try to write the slave it will cause an error. (noa, 2023c) (Carlson and Sanfilippo, 2013)

5.2 Sharding in Redis

For the sharding, at least three nodes are needed, Redis is not able to build a sharding with only two nodes. Redis does the sharding automatically based on a hash function of the keys. If we want to store two keys on the same node, we can use the key hashtag, where the hash function of the sharding will use only the hashtag part of the key in order that the system put the key to a node.

A new node can be added to the sharded cluster and the cluster can be resharded.

If we want to connect with a client to the cluster we have to name all nodes where the cluster run. (noa, 2023c)

5.3 Combination of Sharding and Master-Slave Repication in Redis

For the combination of sharding and master-slave replication in Redis, at least six nodes are needed. Redis needs three nodes to realize the sharding, and every node has at least one copy (replication). When we set up the cluster, we give the nodes and the number how many replicas we want. So, with six nodes and with cluster-replicas=1, we will have 3 masters and each of which has its own slave. With 9 nodes and cluster-replicas=2, we will have 3 masters and each of which has 2 slaves.

A new master can be added to the cluster in the same way as in the sharded cluster. Similarly, a new replica (slave) can also be added to the cluster. When a new slave is added to the cluster, the master can be specified, otherwise, a random master will be assigned to the slave.

If we want to connect to the cluster with a client we have to name all nodes where the cluster run.

5.4 An Environment of a Cluster of Redis

To try out the distribution models of Redis we installed Redis 5.0 on Ubuntu Linux 20.10 server. Considering the client, we connected to the Redis from Python 3.10 with RedisPy 4.1.4, but it did not work with the Redis cluster, so finally, we used Java 17 with Lettuce 6.1.6 to connect to the cluster.

6 COMPARISON

In this section, we compare the distribution solutions of MongoDB and Redis, considering the master-slave replication, the sharding, and the combination of the sharding and replication.

6.1 Master-Slave Replication of MongoDB and Redis

The minimum number of nodes to build the master-slave replication is 2 in both cases. No additional nodes are needed to use the cluster in both cases. The client should know all nodes of the replica set considering MongoDB, meanwhile, in the case of Redis, the client can connect to the master or a slave. The reason for this is that MongoDB itself chooses the master from the nodes of the replica set, meanwhile, in the case of Redis the database administrator can set the slave and the master nodes.

Redis supports multilevel master-slave replication, namely, the slaves can be masters of other slaves, whereas the MongoDB replica set does not offer this solution. The default synchronization type is asynchronous in both cases, and both databases can realize the synchronous write with parameters.

6.2 Sharding in MongoDB and Redis

Redis needs at least 3 nodes to build sharding where all nodes contain shards of the database. Meanwhile, to build sharding in MongoDB, 2 nodes would be enough to store the shards of the database, but MongoDB needs additional nodes, namely a config node, which stores metadata about the shard nodes only, a mongos which is a router, connects to the config node and accept the requests from the clients.

MongoDB can have more mongos services, to which the clients can connect, and with the help of the config node, the shard nodes give back or update the data to answer the client requests. Redis has another solution for the client connection, a client knows all shard nodes. The client driver chooses a node randomly, the node calculates on which node the asked shard is, and reroutes the driver to this node. It does not need additional nodes, the client driver and the shard nodes solve everything.

Both MongoDB and Redis can place shards based on using a hash function on the keys. But additionally, MongoDB can use also range sharding, where ranges of the sharding key are assigned to the nodes. In the case of both databases, we can realize with settings that two different shards will be stored on the same node, namely, in Redis, the key hashtag can be used for the keys of the shards, in MongoDB, we can manipulate with the composite shard keys.

Both databases support dynamic horizontal scale, namely, new nodes can be easily added to the sharded cluster. Using the sharding based on a hash function, the resharding is automatically performed by the system.

In MongoDB, the sharding is assigned to the collections, which means that each collection can have different sharding settings. In contrast, Redis can use an automatic sharding solution on the whole database, so the database administrator has not so much opportunity to influence the sharing in a way.

6.3 Combination of Sharding and Master-Slave Replication in MongoDB and Redis

To build the combination of sharding and master-slave replication in Redis at least 6 nodes are needed, 3 for the sharding, and every shard should have its own replication. Considering the same problem of MongoDB, it needs at least 7 nodes, namely mongos (router), config servers (with replication 2 nodes), for the sharding 2 nodes, and each shard node has its own replication.

With this cluster type, both databases support the dynamic horizontal scale, namely new nodes can be added to the cluster even when the system work.

In this environment, it is not easy for Redis to support the multi-level master-slave replication, since the built system makes the decisions about who is the masters and the slaves in the cluster and not the database administrator (as in the case of the master-slave replication). However, each node can have additional slaves, since the slave is configured to be a slave. MongoDB does still not support multilevel master-slave replication.

The rest point of views are similar to the sharding or the master-slave replication. The client knows all the nodes of the cluster in the case of Redis, meanwhile, in the case of MongoDB, the client connects to one of the mongos (routers). Both databases can use a hash function to shard the keys, but MongoDB has a range shard solution, too. The default synchronization is asynchronous in both cases, and in both cases, the synchronous write can be achieved. And finally, Redis can add shard settings for the whole database only, meanwhile, MongoDB can granulate the sharing to collections.

7 CONCLUSION

In this paper, the distribution models of NoSQL database management systems were introduced, highlighting the master-slave replication, the sharding, and their combinations. The real solutions were introduced on two NoSQL databases: MongoDB and Redis. They are not in the same NoSQL categories, but both of them realize the sharding and master-slave distribution models and their combinations. And they do not realize the peer-to-peer replication.

Finally, we compared the distribution models of the two NoSQL databases. We found that there is no big difference between the two realizations, but of course, they are not the same. The main difference is that MongoDB needs additional nodes to realize the sharding, namely, it uses config servers and the mongos as a router, but it is enough of two nodes to store the shards. At the same time, the Redis needs 3 nodes to realize sharding. Both of them can use a hash function to shard the data, but MongoDB has another solution to this question. Both of them have a solution to store two different shards on the same node. Both of them support horizontal scalability with sharding. By default, both of them use asynchronous replication, but both of them have solutions to realize synchronous replication.

Altogether we found that it is easy to build the clusters with both NoSQL databases. We can state that the distribution models of MongoDB are more complex than the distribution model of Redis. In this way, it is easier to build a Redis cluster as a MongoDB cluster. All in all, the application to be developed will show which database management system should be chosen.

REFERENCES

(2023a). *DB-Engines Ranking*. https://db-engines.com/en/ranking.

(2023b). *MongoDB documetation*. https://www.mongodb.com/docs/manual/tutorial/getting-started/.

(2023c). *Redis Documentation*. https://redis.io.

Banker, K., Bakkum, P., Verch, S., Garrett, D., and Hawkins, T. (2016). *MongoDB in action*. Manning, Shelter Island, NY, second edition.

Carlson, J. L. and Sanfilippo, S. (2013). *Redis in action*. Manning, Shelter Island, NY.

Cattell, R. (2011). Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, 39.

Corbellini, A., Mateos, C., Zunino, A., Godoy, D., and Schiaffino, S. (2017). Persisting big-data: The NoSQL landscape. *Information Systems*, 63.

Davoudian, A., Chen, L., and Liu, M. (2018). A Survey on NoSQL Stores. *ACM Computing Surveys*, 51.

Diogo, M., Cabral, B., and Bernardino, J. (2019). Consistency Models of NoSQL Databases. *Future Internet*, 11.

Gajendran, S. (2012). A survey on nosql databases.

Gessert, F., Wingerath, W., Friedrich, S., and Ritter, N. (2017). NoSQL database systems: a survey and decision guidance. *Computer Science - Research and Development*, 32.

Hecht, R. and Jablonski, S. (2011). NoSQL evaluation: A use case oriented survey. In *2011 International Conference on Cloud and Service Computing*. IEEE.

Indrawan-Santiago, M. (2012). Database Research: Are We at a Crossroad? Reflection on NoSQL. In *2012 15th International Conference on Network-Based Information Systems*. IEEE.

Jing Han, Haihong E, Guan Le, and Jian Du (2011). Survey on NoSQL database. In *2011 6th International Conference on Pervasive Computing and Applications*. IEEE.

Lourenço, J. R., Cabral, B., Carreiro, P., Vieira, M., and Bernardino, J. (2015). Choosing the right NoSQL database for the job: a quality attribute evaluation. *Journal of Big Data*, 2.

Moniruzzaman, A. B. M. and Hossain, S. A. (2013). NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *arXiv:1307.0191 [cs]*.

Sadalage, P. J. and Fowler, M. (2013). *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Addison-Wesley, Upper Saddle River, NJ.

Tudorica, B. G. and Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. In *2011 RoEduNet International Conference 10th Edition: Networking in Education and Research*. IEEE.