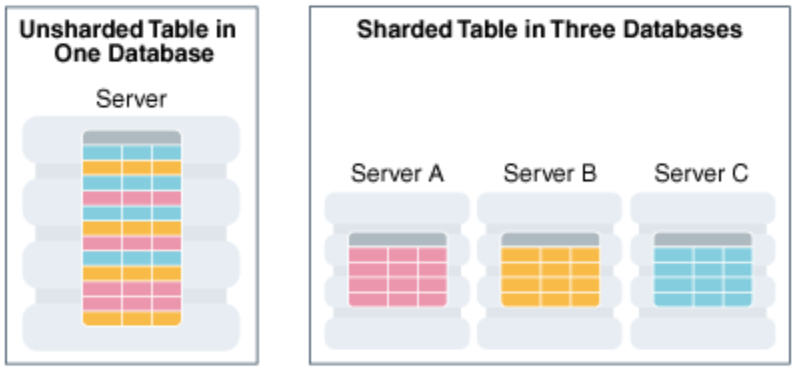
devopedia.org

Database Sharding

lokesh.rawat, arvindpdmn, tintin

20-26 minutes


Unsharded Table in One Database
Server
Sharded Table in Three Databases
Server A
Server B
Server C

An SQL database table split into three shards. Source: Oracle Docs 2020b, fig. 49-1.

In the age of Big Data, popular social media platforms and IoT sensor data, datasets are huge. If such a dataset is stored in a single database, queries become slow. Overall system performance suffers. This is when database sharding becomes useful.

A single logical dataset is **split into multiple databases** that are then **distributed across multiple machines**. When a query is made, only one or a few machines may get involved in processing the query. Sharding enables effective scaling and management of large datasets. There are many ways to split a dataset into shards.

Sharding is possible with both SQL and NoSQL databases. Some databases have out-of-the-box support for sharding. For others, tools and middleware are available to assist in sharding.

Database replication, partitioning and clustering are concepts related to sharding.

Discussion

- When do I need to shard my database?

  When to do database sharding. Source: Nasser 2020.

  SQL query performance can be improved by simply **indexing** tables. But on large tables (millions of rows) searching even the indices can be slow.

  One approach is to do **partitioning**. Large tables are split into smaller tables. Each partition has it's own index. Suppose tables are partitioned by rows, query is processed on a smaller index. All partitions reside on the same server.

  A complementary approach is **vertical scaling**, which is a hardware upgrade: faster processor, more RAM, faster disk, etc. However, this is not scalable or cost effective in the long run.

  Even with partitioning and vertical scaling, the server can get overwhelmed with too many requests. One solution is **replication**. The master database is replicated in other servers called slaves. Read query is processed by one of the slaves. This speeds up database reads but not writes, which happen only on the master server. Reads can also be improved with **caching**.

  When all these have failed, use sharding as the last resort. Sharding enables **horizontal scaling**, which involves adding more servers that share the load and process requests in parallel.

- How is sharding different from partitioning?

  Some use the terms partitioning and sharding interchangeably. However, there are clear differences.

All partitions of a table reside on the same server whereas sharding involves multiple servers. Therefore, sharding implies a distributed architecture whereas partitioning does not.

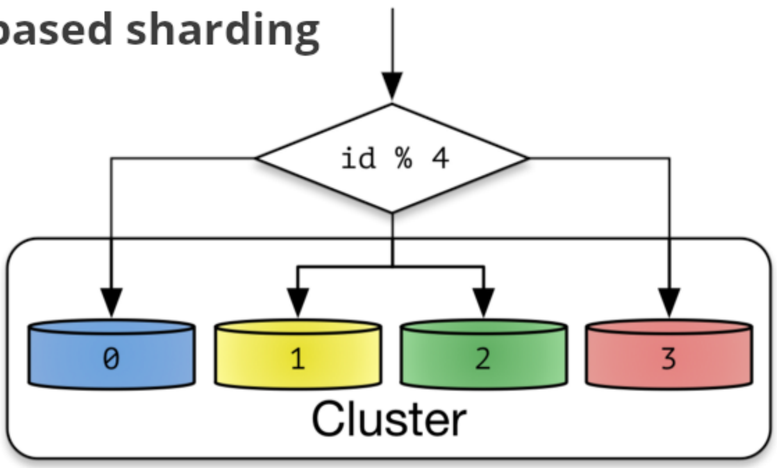
Partitions can be horizontal (split by rows) or vertical (by columns). Shards are usually only horizontal. In other words, all shards share the same schema but contain different records of the original table. For this reason, sharding is sometimes called *horizontal partitioning*.

Due to its distributed nature, sharding is more complex than partitioning. Many databases including older ones offer built-in support for partitioning. Sharding is supported mainly in modern databases. Where databases lack support for sharding, and no relevant middleware is used, the logic of which shard to hit resides in application code. With partitions, the logic resides at the database level.

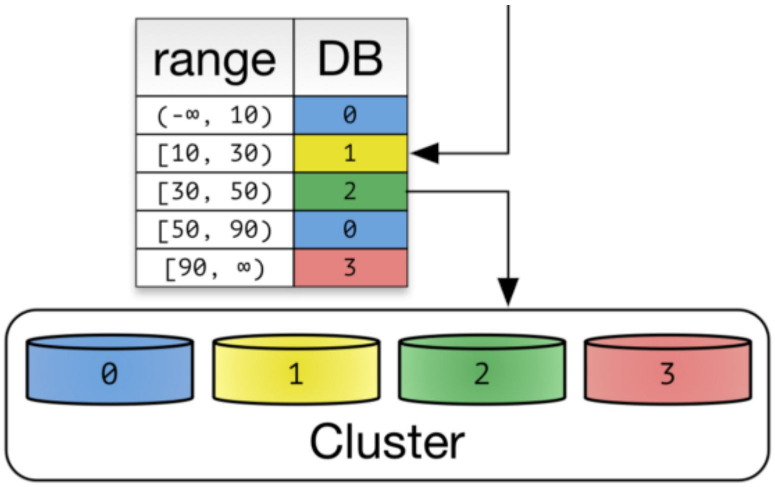
Sharding adopts a **shared-nothing architecture**. A shard is not aware of other shards. This is often not the case with database clustering or replication.

- What are some strategies for database sharding?

(a) Hash-based sharding



(b) Range-based sharding



Illustrating hash-based and range-based sharding. Source: Adapted from Kim 2014.

A sharding strategy is necessary to determine which record goes into which shard:

- **Key-Based**: One of the columns, called *shard key*, is put through a hash function. The result determines the shard for that row. Also called hash-based sharding or algorithmic sharding.
- **Range-Based**: For example, given a product-price database, prices in range 0-49 go into shard 1, 50-99 into shard 2, and so on. Price column is the shard key. If the store sells lot more low-value products, this will result in unbalanced shards and hotspots.
- **Dictionary-Based**: A lookup table is used. This is a flexible approach since there's no predetermined hash function or ranges. An example is to shard by customer's region such as UK, US, or EU.
- **Hierarchical**: A combination of row and column is used as the shard key. Previously mentioned sharding strategies can be used on the key.
- **Entity Groups**: To facilitate queries across multiple tables, this strategy places related tables within the same shard. This brings

stronger consistency. For example, all data pertaining to a user reside in the same shard.

- How should I select a suitable shard key?
  Selecting a shard key is an important decision. Once selected, it's hard to change this in future. Consider how the choice will affect the current schema, queries and query performance. The choice should support current and future business use cases. Identify main parts of the data and clustering patterns.

  For efficiency, the data type of the shard key is ideally an integer.

  Sharding must balance two constraints: minimize cross-partition queries and distribute the load evenly by sharding at the right granularity.

  Shard key must be on a column (SQL) or field (NoSQL) that meet the following:

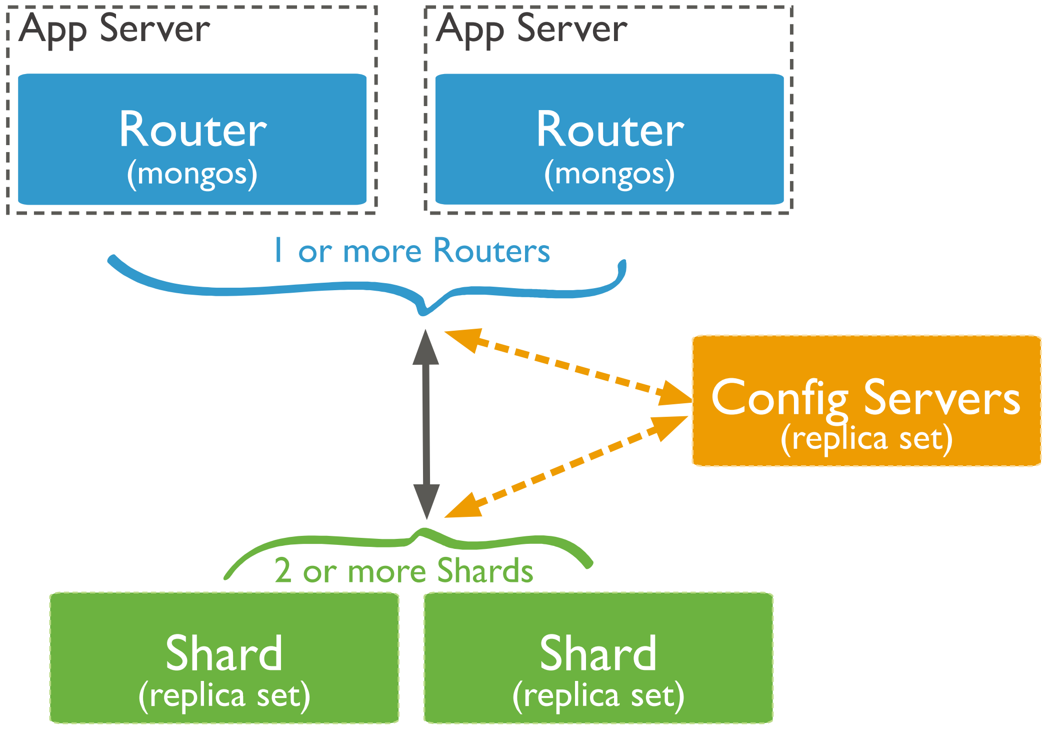
- **Stability**: It almost never changes. Otherwise, we could incur expensive movement of data from one shard to another.
- **High Cardinality**: It has many more unique values relative to the number of shards.
- **Low Frequency**: Every value occurs with a low frequency.
- **Non-Monotonic**: The value doesn't increase or decrease monotonically.
- Could you share some examples of shard keys?
  In an eCommerce application, Inventory, Sales and Customers could be functional areas with shard keys `productID`, `orderID` and `customerID` respectively. Alternatively, `customerID` could be the shard key for Sales as well. Typically `customerID` is unique and of high cardinality. Hence, this could be the shard key rather

than `country`.

In an IoT application, if all devices are storing data at similar intervals then `deviceID` is a suitable key. If some devices store lot more data than others, this would create hotspots and therefore `deviceID` may not suitable. For an application with a few status codes, using status code as shard key is not a good idea.

Writing across shards is to be avoided. As an example, an eCommerce order may insert buyer address and seller address. Though both are part of the same logical table, they could reside on different shards. Another example is when a transaction inserts an order entry and an inventory entry. These two could be on different shards. It's preferable to locate all data pertaining to a transaction on a single shard.

- What's a typical database sharding architecture?



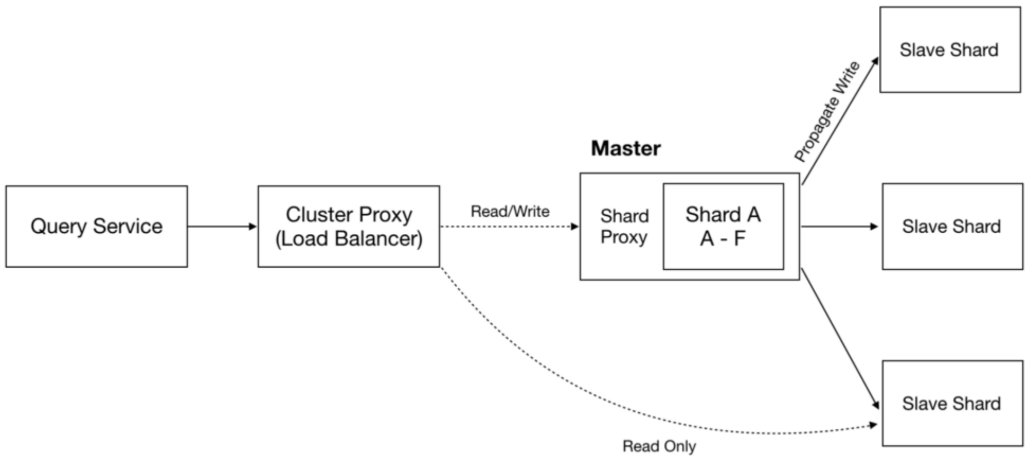
A MongoDB sharded cluster. Source: MongoDB Docs 2020a.

There's no requirement that a server must host a single shard. Multiple shards can be hosted on the same server. When load increases, they can be moved to separate servers. Original database is split by a shard key. Data that have the same key is called a **logical shard**. One or more logical shards when hosted on the same server is called **physical shard**.

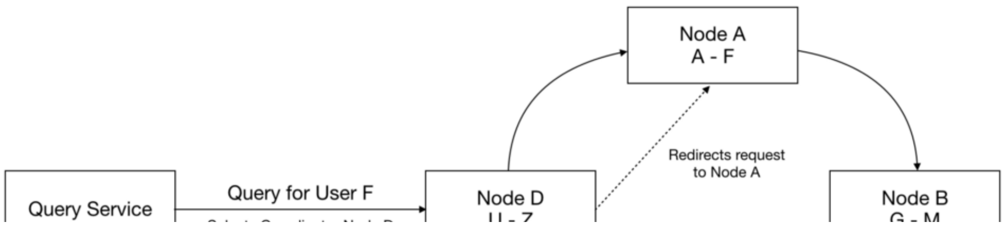
In a typical architecture, a cluster proxy is connected to multiple shards. The proxy has access to a configuration service (such as Zookeeper) that keeps track of shards and their indices. The cluster proxy looks at a query and the configuration to know which shard to hit. If shards are large, each shard may also have its own proxy that can do caching and monitoring. With hierarchical sharding, each shard may be sharded further.

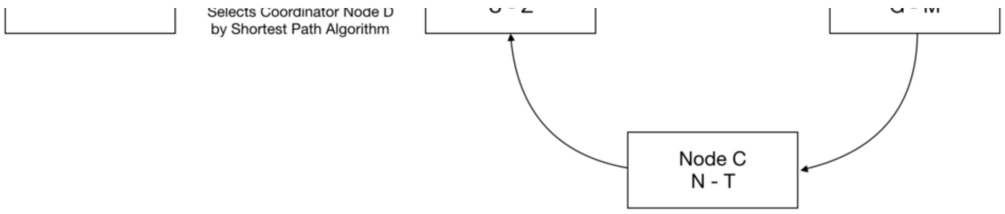
In Oracle Sharding, Shard Catalog contains the configuration. Routing is done by Shard Director. All shards are in the same region (data center). For high availability, replicated shards are in the same region as the primary shard. For disaster recovery, replicated shards are in another region.

- How is sharding different between SQL and NoSQL databases?



(a) SQL: sharding with master-slave replication





(b) NoSQL: coordinator routes query to correct node

Comparing SQL and NoSQL sharding architectures. Source: Adapted from Lim 2020.

SQL databases were designed with ACID transactions in mind, not scalability. NoSQL databases were designed with scalability in mind. Sharding in NoSQL databases happens almost transparent to the user. Availability is prioritized over consistency.

In SQL tables, shard key is based on one or more columns. In NoSQL records, one or more fields are used instead.

Since SQL databases combine sharding with replication, it's a master-slave architecture. In NoSQL world, shards are referred to as **nodes**. All nodes are equal, with one node connected to a few others in a graph-like structure. When a query comes in, a coordinator node is selected. If the coordinator can respond directly, it will. Otherwise, it forwards the query to another node that has the data. Nodes frequently share information about the data they store via what's called the **gossip protocol**.

- What are the pros and cons of database sharding?
  The main benefits of sharding include horizontal scaling, better performance, and higher availability. Processing and storage are distributed. An outage will most likely affect only one shard.

  Sharding can lower cost by avoiding high-end hardware or expensive software. Commodity hardware and open-source software can be used.

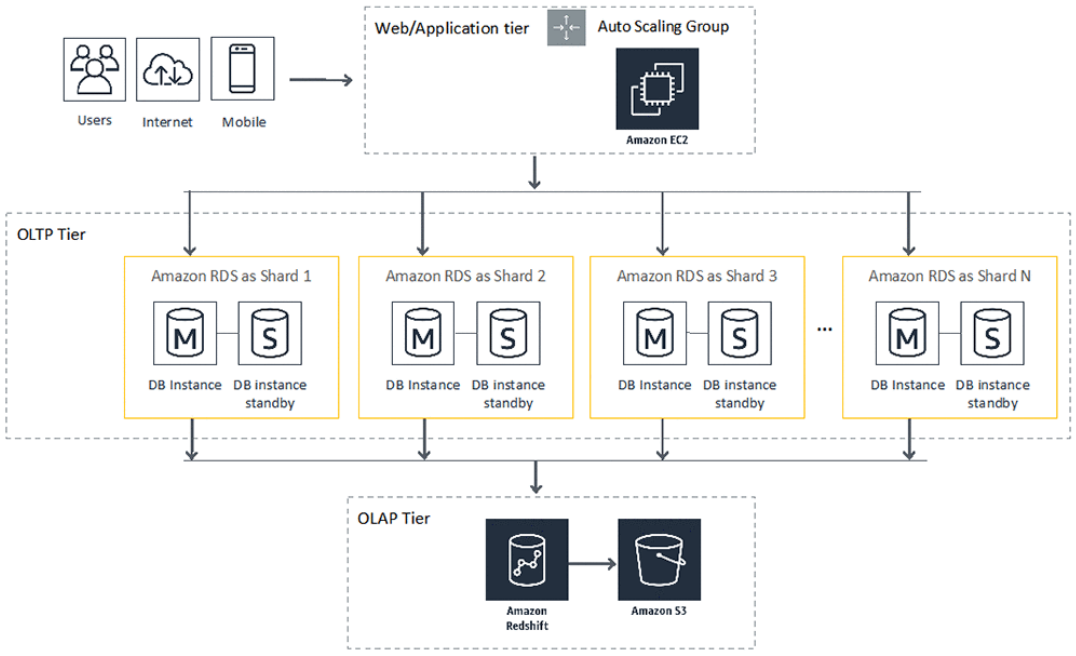
  Shards are suited for cloud deployments. Any upgrade can be tested on one shard before rolling it out to the rest. Shards can be geographically distributed to meet regulatory requirements or be closer to customers.

The complexity inherent with sharding is the main drawback. Improper implementation can lead to loss of data or data corruption. Maintaining multiple shards is not trivial and teams have to be trained in new workflows. Once sharded, it's hard to return to the unsharded architecture. Databases without native support for sharding will need custom solutions.

Since data is distributed across many servers, a `JOIN` operation easily done in a single SQL database becomes difficult. Such an operation may hit many shards and require merging the responses.

Shards may become unbalanced over time. In other words, some shards grow faster than others, thus becoming **database hotspots**. Resharding is the solution.

- What resources or tools are available to facilitate database sharding?



Sharding in Amazon RDS for OLTP. Source: Zeng 2019.

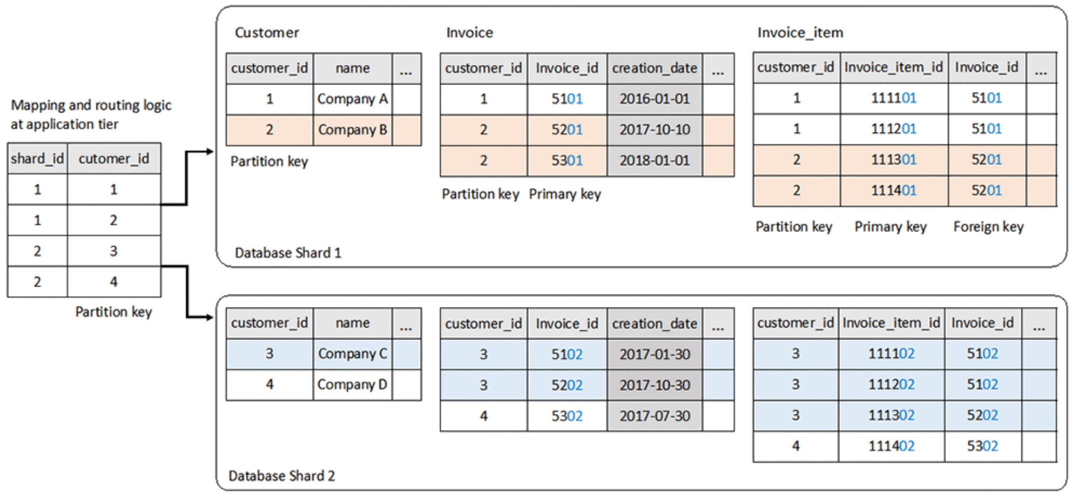
Cassandra, HBase, HDFS, MongoDB and Redis are databases that support sharding. Sqlite, Memcached, Zookeeper, MySQL and PostgreSQL are databases that don't natively support sharding at the database layer.

For databases that don't offer built-in support, sharding logic has to reside in the application. To avoid this complexity in application code, middleware can help. Two open-source middleware are Apache ShardingSphere and Vitess. Plugins for ShardingSphere offer support for MySQL, PostgreSQL, SQLServer, and Oracle. Vitess supports MySQL and MariaDB.

MySQL itself doesn't offer sharding but MySQL NDB Cluster and MySQL Fabric can be used to achieve sharding. For sharding PostgreSQL, PL/Proxy, Postgres-XC/XL and Citus can be used.

On Google Cloud Platform, Cloud SQL and ProxySQL services can be used to shard PostgreSQL and MySQL databases. On AWS, Amazon RDS is a service that can implement a sharded database architecture. The DB engine can be MySQL, MariaDB, PostgreSQL, Oracle, SQL Server, and Amazon Aurora. For data analytics, Amazon Redshift can store data from all shards. Amazon RDS also facilitates resharding.

- Could you share some best practices for database sharding?



shard_id	cutomer_id
1	1
1	2
2	3
2	4

Database Shard 1

Customer

customer_id	name	...
1	Company A	
2	Company B	

Invoice

customer_id	Invoice_id	creation_date	...
1	5101	2016-01-01	
2	5201	2017-10-10	
2	5301	2018-01-01	

Invoice_item

customer_id	Invoice_item_id	Invoice_id	...
1	111101	5101	
1	111201	5101	
2	111301	5201	
2	111401	5201	

Database Shard 2

customer_id	name	...
3	Company C	
4	Company D	

customer_id	Invoice_id	creation_date	...
3	5102	2017-01-30	
3	5202	2017-10-30	
4	5302	2017-07-30	

customer_id	Invoice_item_id	Invoice_id	...
3	111102	5102	
3	111202	5102	
3	111302	5202	
4	111402	5302	

Shard ID is suffixed to primary keys to keep them unique across shards. Source: Zeng 2019.

Consider sharding only when other approaches have failed. Main reasons for sharding include constraints on storage, processing, network bandwidth, regulations and geographic proximity.

Data analytics is usually performed on the entire dataset. Hence, sharding is more suited for Online Transaction Processing (OLTP) rather than for Online Analytical Processing (OLAP).

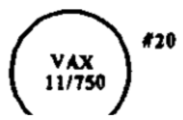
Tables with foreign key relationships can all share the same shard key. To keep primary keys unique across all shards for later OLAP, shard IDs can be suffixed to primary keys.

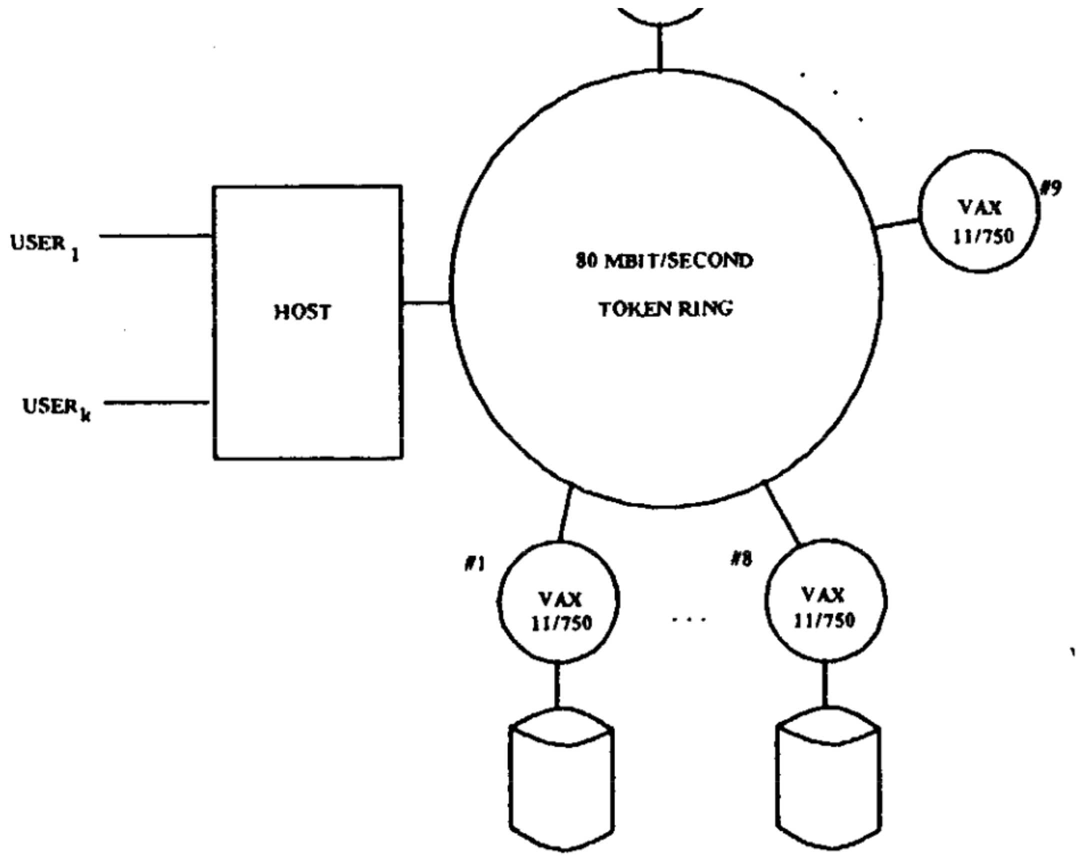
Sharding can be combined with replication. In fact, for high availability it's common to replicate shards. We could even duplicate some data across shards, such as duplicating chat messages in sender's and recipient's shards.

Monitor the performance of all shards in terms of CPU utilization, memory usage, and read/write performance. Consider resharding if there are hotspots.

Milestones

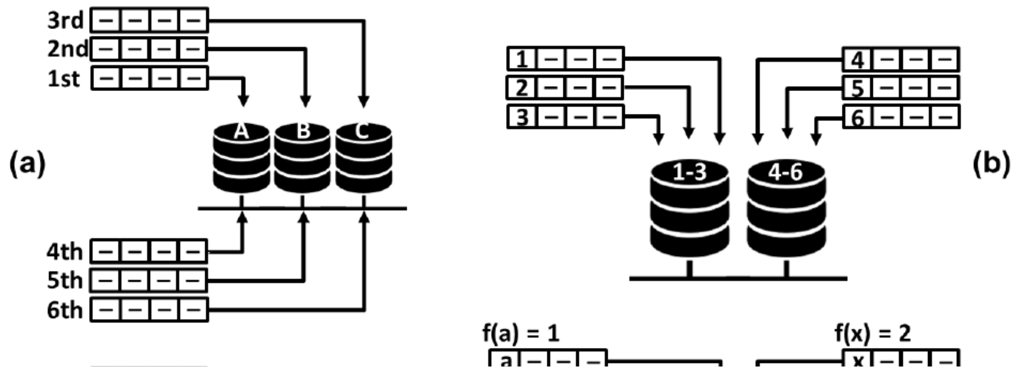
It's predicted that distributed databases with specialized hardware will die. Multiprocessor systems are seen to be I/O limited. However, later years prove this prediction to be false. This is due to the growth of relational databases through the 1980s. These can be easily parallelized. High-speed networking to interconnect parallel processors become available. Teradata, Tandem and other startups successfully market parallel machines.

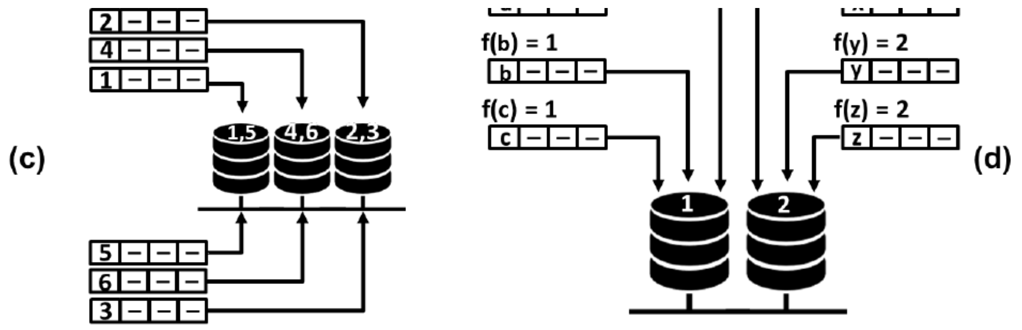




GAMMA is an early multiprocessor parallel database system. Source: DeWitt et al. 1986, fig. 3.

DeWitt et al. present a parallelized database they call *GAMMA*. It consists of 20 VAX 11/750 processors, each with dedicated memory. Eight of them have disk drives for database storage. The machines are connected by 80Mbps token ring. They note four ways to split data: round-robin, hashed, range (user specified), and range (uniform distribution). In a later version of GAMMA (1990), each processor has its own disk storage, giving rise to the term **shared-nothing architecture**.





Sharding strategies: (a) round-robin, (b) range, (c) list, and (d) hash. Source: Costa et al. 2015.

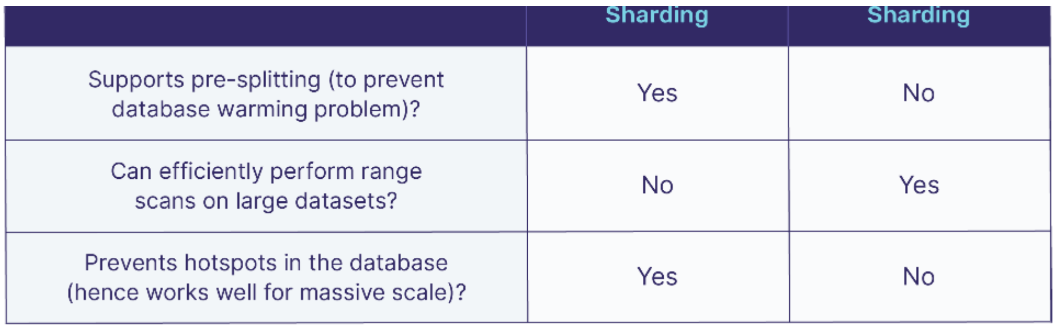
DeWitt and Gray apply dataflow programming paradigm to propose a distributed database architecture they call **partitioned parallelism**. Input data is partitioned and sent to multiple processors and memories. They propose four sharding strategies: round-robin, range, list and hash.

The use of the term "shard" in a computing and storage context probably originates with the massively multiplayer online game called *Ultima Online*. The story is that the fictional world Sosaria was trapped in a crystal. When the crystal shattered, each **shard** contained a copy of Sosaria. Each copy of the world ran on its own server and database.

In a presentation, Fitzpatrick explains how sharding helped LiveJournal scale their backend. The site launched in 1999 but grew to 7.9 million accounts by August 2005. With thousands of hits per second, sharding is adopted as the solution. The term "shard" is not used. Instead, the term **user cluster** is used. Within a cluster, replication is done.

Google's *Bigtable* is a distributed storage that can handle petabytes of data across commodity servers. Web indexing, Google Earth, Google Analytics, Orkut, and Google Finance are some projects at Google using Bigtable.

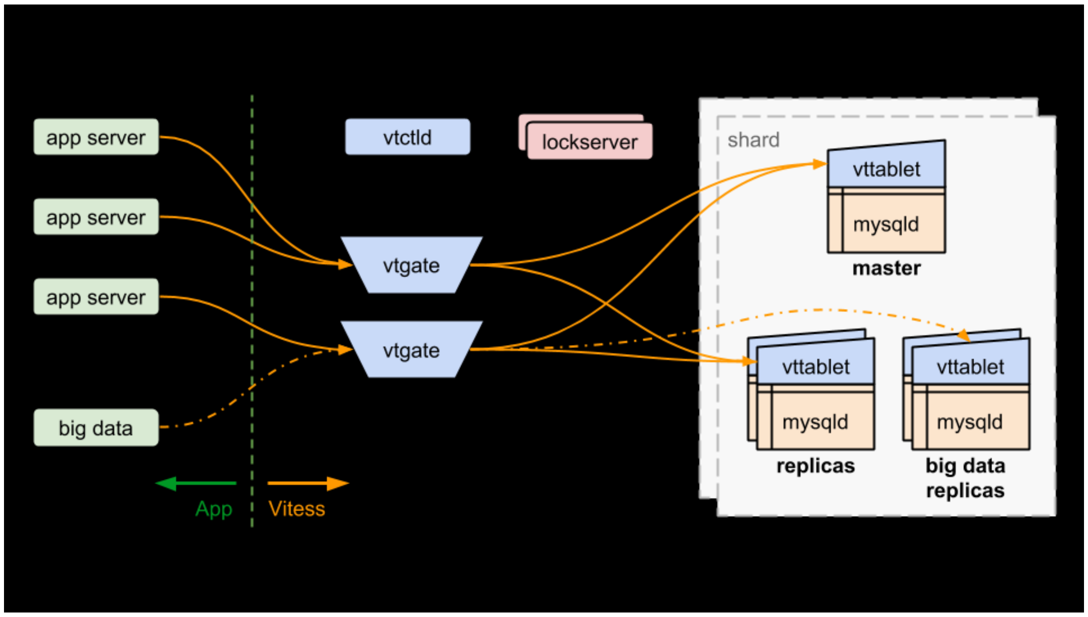
	Consistent Hash	Range



	Sharding	Sharding
Supports pre-splitting (to prevent database warming problem)?	Yes	No
Can efficiently perform range scans on large datasets?	No	Yes
Prevents hotspots in the database (hence works well for massive scale)?	Yes	No

Comparing consistent hash sharding and range sharding. Source: Ranganathan 2020.

At Facebook, engineers start work on an NoSQL database called *Cassandra*. They're inspired by **range sharding** of Google's Bigtable and **consistent hash sharding** of Amazon's DynamoDB. They adopt linear hash sharding that is a hybrid of the two. It preserves the sort order but distribution is uneven. This is later changed to consistent hash sharding at the expense of range queries.



The architecture of Vitess. Source: Constine 2018.

A team in YouTube creates *Vitess* to solve scalability challenges to its MySQL data storage. Before Vitess, YouTube tried replication to speed up reads, and then application-level sharding to speed up writes. With Vitess, routing logic can be removed from application code. Vitess sits between the application and the database. In February 2018, Vitess becomes a CNCF incubation project, graduating in November 2019.

References

1. Apache Software Foundation. 2020. "Apache ShardingSphere." Apache Software Foundation. Accessed 2020-10-28.
2. Chang, Fay, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2006. "Bigtable: A Distributed Storage System for Structured Data." 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI), {USENIX}, pp. 205-218. Accessed 2020-10-28.
3. Choudhury, Sid. 2019. "How Data Sharding Works in a Distributed SQL Database." Blog, Yugabyte, June 6. Updated 2020-08-28. Accessed 2020-10-28.
4. Constine, Josh. 2018. "They scaled YouTube — now they'll shard everyone with PlanetScale." TechCrunch, December 14. Accessed 2020-10-28.
5. Costa, Caio H., João Vianney B. M. Filho, Paulo Henrique M. Maia, and Francisco Carlos M. B. Oliveira. 2015. "Sharding by Hash Partitioning - A Database Scalability Pattern to Achieve Evenly Sharded Database Clusters." Proceedings of the 17th International Conference on Enterprise Information Systems - Volume 2: ICEIS, pp. 313-320, Barcelona, Spain. Accessed 2020-10-28.
6. Dave, Pinal. 2016. "Database Sharding – How to Identify a Shard Key?" Blog, SQL Authority, August 17. Accessed 2020-10-28.
7. DeWitt, D. and J. Gray. 1992. "Parallel database systems: The future of high performance database systems." Communications of the ACM, vol. 35, no. 6, pp. 85-98, June. Accessed 2020-10-28.

8. DeWitt, David J., Robert H. Gerber, Goetz Graefe, Michael L. Heytens, Krishna B. Kumar, and M. Muralikrishna. 1986. "GAMMA - A High Performance Dataflow Database Machine." Proceedings of the Twelfth International Conference on Very Large Data Bases, August, pp. 228-237. Accessed 2020-10-28.
9. Drake, Mike. 2019. "Understanding Database Sharding." Tutorial, Digital Ocean, February 7. Accessed 2020-10-28.
10. Fitzpatrick, Brad. 2005. "LiveJournal's Backend: A History of Scaling." O'Reilly Open Source Convention (OSCON), August 1-5. Accessed 2020-10-28.
11. Hazelcast. 2020. "What Is Sharding?" Glossary, Hazelcast. Accessed 2020-10-28.
12. Isaacson, Cory. 2009. "Database Sharding: The Key to Database Scalability." Database Trends and Applications, Information Today Inc., August 14. Accessed 2020-10-28.
13. jynus. 2014. "What's the difference between MySQL Fabric and MySQL Cluster?" Database Administrators, StackExchange, October 11. Accessed 2020-10-28.
14. Kim, Jeeyoung. 2014. "How Sharding Works." Medium, December 5. Accessed 2020-10-28.
15. Koster, Raph. 2009. "Database “sharding” came from UO?" January 8. Accessed 2020-10-28.
16. Lim, Zeng Hou. 2020. "How to Scale SQL and NoSQL Databases." Better Programming, on Medium, April 28. Accessed 2020-10-28.
17. Microsoft Docs. 2017. "Sharding pattern." Azure, Microsoft Docs, June 23. Accessed 2020-10-28.
18. Momjian, Bruce, and Alexander Korotkov. 2019. "The Future of Postgres Sharding." PGConf.EU 2019, October 17. Accessed 2020-10-28.

19. MongoDB Docs. 2020a. "Sharding." MongoDB Manual, Version 4.4, MongoDB. Accessed 2020-10-28.

20. MongoDB Docs. 2020b. "Shard Keys." MongoDB Manual, Version 4.4, MongoDB. Accessed 2020-10-28.

21. Mullins, Craig S. 2018. "What are the Database Scalability methods?" Blog, NuoDB, February 13. Accessed 2020-10-28.

22. Nasser, Hussein. 2020. "When should you shard your database?" YouTube, April 28. Accessed 2020-10-28.

23. Oracle Docs. 2020a. "Design Considerations for Sharded Database Applications." Chapter 5 in: Using Oracle Sharding, Oracle Database, Release 19. Accessed 2020-10-28.

24. Oracle Docs. 2020b. "Oracle Sharding Architecture." Chapter 17 in: Database Concepts, Oracle Database, Release 12.2. Accessed 2020-10-28.

25. Oracle Docs. 2020c. "Overview of Oracle Sharding." Chapter 49 in: Database Administrator's Guide, Oracle Database, Release 12.2. Accessed 2020-10-28.

26. Ranganathan, Karthik. 2020. "Four Data Sharding Strategies We Analyzed in Building a Distributed SQL Database." Blog, Yugabyte, January 14. Accessed 2020-10-28.

27. Schneider, Donovan A., and David J. DeWitt. 1990. "Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines." Proceedings of the 16th VLDB Conference , pp. 469-480. Accessed 2020-10-28.

28. Shastri, Tripti. 2018. "Application Design Considerations for Sharding High Volume Databases." PayPal Engineering, on Medium, September 10. Accessed 2020-10-28.

29. Singh, Vivek Kumar. 2019. "Database Sharding." System Design Blog, on Medium, April 22. Accessed 2020-10-28.

30. Udacity. 2012. "Scaling Databases." Web Development, Udacity, on YouTube, April 26. Accessed 2020-10-28.
31. Vitess. 2019. "History." Vitess, November 23. Accessed 2020-10-28.
32. Vitess. 2020. "What Is Vitess?" Vitess, October 24. Accessed 2020-10-28.
33. Yu, Chen. 2018. "Horizontally scaling a MySQL database backend with Cloud SQL and ProxySQL." Tutorial, Google Cloud, January 12. Accessed 2020-10-28.
34. Zeng, Lei. 2019. "Sharding with Amazon Relational Database Service." AWS Database Blog, AWS, March 20. Accessed 2020-10-28.

Further Reading

1. Drake, Mike. 2019. "Understanding Database Sharding." Tutorial, Digital Ocean, February 7. Accessed 2020-10-28.
2. Microsoft Docs. 2017. "Sharding pattern." Azure, Microsoft Docs, June 23. Accessed 2020-10-28.
3. Chang, Fay, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2006. "Bigtable: A Distributed Storage System for Structured Data." 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI), {USENIX}, pp. 205-218. Accessed 2020-10-28.
4. Ries, Eric. 2009. "Sharding for startups." Startup Lessons Learned, January 4. Accessed 2020-10-28.
5. Kerstiens, Craig. 2017. "Five sharding data models and which is right." Blog, Citus Data, August 28. Accessed 2020-10-28.
6. Polosatov, Andrew. 2020. "MongoDB Sharding: How We Saved

Our Client €100k A Year." Case Study, Krusche & Company GmbH, September 9. Accessed 2020-10-28.

Article Stats

Author-wise Stats for Article Edits

Author	No. of Edits	No. of Chats	DevCoins

Cite As

Devopedia. 2020. "Database Sharding." Version 4, October 31. Accessed 2022-06-15. https://devopedia.org/database-sharding