Log In Join

Back To Module Home

Distributed Systems

0% completed

Search Module

Conclusion

Mark Module as Completed

Algorithms for Horizontal Partitioning

Let's look into the algorithms used for horizontal partitioning.

We'll cover the following

There are a lot of different algorithms we can use to perform horizontal partitioning. We will study some of these algorithms, and discuss their advantages and drawbacks.

Range partitioning#

Range partitioning is a technique where we split a dataset into ranges according to the value of a specific attribute. We then store each range in a separate node. The case we described earlier-with the alphabetical split-is an example of range partitioning.

Range partitioning

Of course, the system should store and maintain a list of all these ranges and map which node stores a specific range. In this way, the system consults this node map whenever the system receives a request for a specific value (or a range of values) to identify which node (or nodes, respectively) the request should be redirected to.

Advantages of range partitioning#

Some advantages of range partitioning include:

- Simplicity and ease of implementation.
- The ability to perform range queries using the partitioning key value.
- A good performance for range queries that use the partitioning key, when the queried range is small and resides in a single node.
- Makes adjusting ranges (re-partitioning) easier and more efficient. One range can be increased or decreased, which exchanges data only between two nodes.

Disadvantages of range partitioning#

Some disadvantages of range partitioning include:

- The inability to perform range queries using keys other than the partitioning key
- A bad performance for range queries that use the partitioning key when the queried range is big and resides in multiple nodes

- An uneven distribution of the traffic or data, which causes some nodes to overload. For example, while range partitioning through an alphabetical split, we may find that some alphabetical letters appear as the initial letters in surnames more frequently than other letters. This means some nodes may have to store more data and process more requests than others.

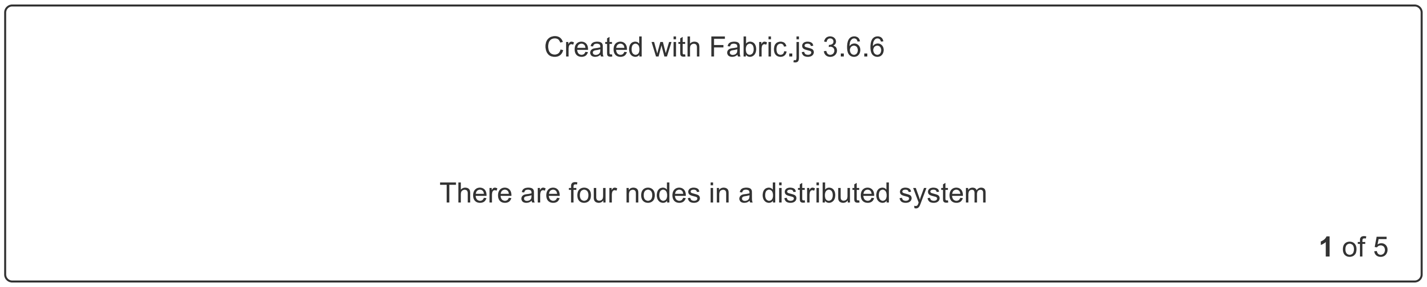
Some systems that leverage a range partitioning technique are Google's BigTable, and Apache HBase.

Hash partitioning#

Hash partitioning is a technique where we apply a hash function to a specific attribute of each row. This results in a number that determines which partition—and, thus, node—this row belongs to.

For the sake of simplicity, let's assume we have one partition per node, as in the previous example, and a hash function that returns an integer. If we have `n` number of nodes in our system and try to identify which node locates a student record with a surname `s`, we'll calculate it with the formula `hash(s) mod n`.

This mapping process will take place both when we write a new record, and when we receive a request to find a record for a specific value of this attribute.



There are four nodes in a distributed system

Advantages of hash partitioning#

Some advantages of hash partitioning include:

- The ability to calculate the partitioning mapping at runtime with no need to store and maintain the mapping. This is beneficial both in terms of data storage needs and performance, as we don't need any additional requests to find the mapping
- A greater chance that the hash function will uniformly distribute the data across our system's nodes, and prevent some nodes from overloading

Disadvantages of hash partitioning#

Some disadvantages of hash partitioning include:

- The inability to perform range queries at all—even for the attribute we use as a partitioning key—without storing additional data or querying all the nodes
- Adding or removing nodes from the system causes it to re-partition. This results in significant data movement across all nodes of the system

Consistent hashing#

Consistent hashing is a partitioning technique that is very similar to hash partitioning, but solves the increased data movement problem caused by hash partitioning.

This is how it works: each node in the system is randomly assigned an integer in a range of `[0, L]`. This range is called ring (for example, `[0, 360]`). Then, the system uses a record with an attribute value `s` as a partitioning key to locating the node after the point `hash(s) mod L` in the ring.

As a result, when a new node enters the ring, it receives data only from the previous node in the ring. The other nodes don't need to exchange any more data. Similarly, when a node leaves the ring, its data transfer to the next node in the ring.

The following illustration depicts this behavior and the difference between these two different algorithms.

Created with Fabric.js 3.6.6

There are four nodes in the system, we randomly assign an integer to each node in a range [0, 360], called ring

1 of 5

Advantages of consistent hashing#

Consistent hashing has one main advantage, when compared to hash partitioning:

- Reduced data movement when nodes are added or removed in the system

Disadvantages of consistent hashing#

Some disadvantages of consistent hashing include:

- The potential for the data's non-uniform distribution because of the random assignment of nodes in the ring
- The potential for more imbalanced data distribution as nodes are added or removed. E.g., a node's dataset is not distributed evenly across the system when it is removed but is instead transferred to a single node

We can mitigate these issues through the concept of "virtual nodes," where we assign each physical node multiple locations in the ring. These locations are known as **virtual nodes**.

> For further discussion on this concept, feel free to read the Dynamo paper.
> Another widely-used system that uses consistent hashing is Apache Cassandra.

Partitioning

Next

Replication

Mark as Completed

Report an Issue