Ask a Question

Trade-offs in Databases

Learn when to use horizontal sharding instead of vertical sharding and vice versa.

We'll cover the following

Which is the best database sharding approach?

Both horizontal and vertical sharding involve adding resources to our computing infrastructure. Our business stakeholders must decide which is suitable for our organization. We must scale our resources accordingly for our organization and business to grow, to prevent downtime, and to reduce latency. We can scale these resources through a combination of adjustments to CPU, physical memory requirements, hard disk adjustments, and network bandwidth.

The following sections explain the pros and cons of no sharding versus sharding.

Advantages and disadvantages of a centralized database

Advantages

- Data maintenance, such as updating and taking backups of a centralized database, is easy.
- Centralized databases provide stronger consistency and ACID transactions than distributed databases.
- Centralized databases provide a much simpler programming model for the end programmers as compared to distributed databases.
- It’s more efficient for businesses that have a small amount of data to store that can reside on a single node.

Disadvantages

- A centralized database can slow down, causing high latency for end users, when the number of queries per second accessing the centralized database is approaching single-node limits.
- A centralized database has a single point of failure. Because of this, its probability of not being accessible is much higher.

Advantages and disadvantages of a distributed database

Advantages

- It’s fast and easy to access data in a distributed database because data is retrieved from the nearest database shard or the one frequently used.

- Data with different **levels of distribution transparency↴** can be stored in separate places.
- Intensive transactions consisting of queries can be divided into multiple optimized subqueries, which can be processed in a parallel fashion.

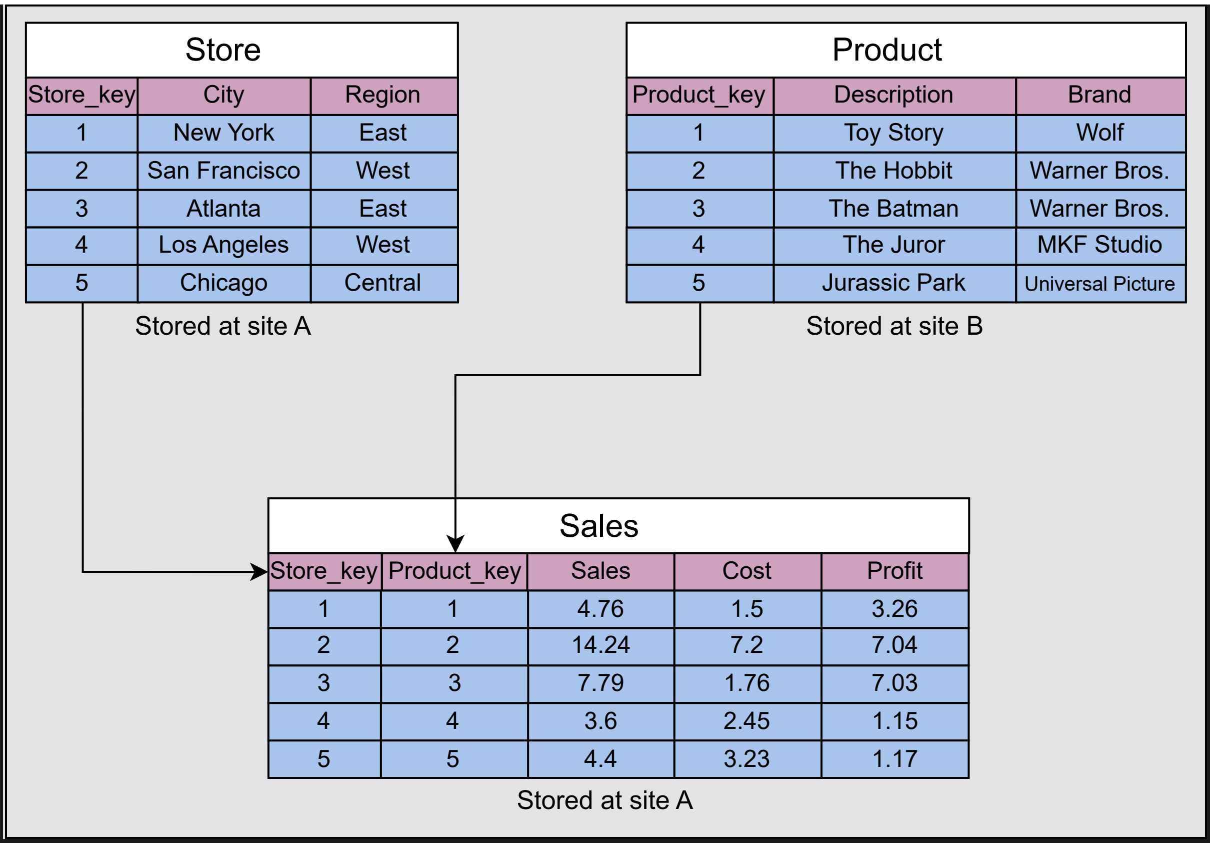
Disadvantages

- Sometimes, data is required from multiple sites, which takes more time than expected.
- Relations are partitioned vertically or horizontally among different nodes. Therefore, operations such as joins need to reconstruct complete relations by carefully fetching data. These operations can become much more expensive and complex.
- It’s difficult to maintain consistency of data across sites in the distributed database, and it requires extra measures.
- Updations and backups in distributed databases take time to synchronize data.

Query optimization and processing speed in a distributed database

A transaction in the distributed database depends on the type of query, number of sites (shards) involved, communication speed, and other factors, such as underlying hardware and the type of database used. However, as an example, let’s assume a query accessing three tables, `Store`, `Product`, and `Sales`, residing on different sites.

The number of attributes in each table is given in the following figure:



Store (Stored at site A)

Store_key	City	Region
1	New York	East
2	San Francisco	West
3	Atlanta	East
4	Los Angeles	West
5	Chicago	Central

Product (Stored at site B)

Product_key	Description	Brand
1	Toy Story	Wolf
2	The Hobbit	Warner Bros.
3	The Batman	Warner Bros.
4	The Juror	MKF Studio
5	Jurassic Park	Universal Picture

Sales (Stored at site A)

Store_key	Product_key	Sales	Cost	Profit
1	1	4.76	1.5	3.26
2	2	14.24	7.2	7.04
3	3	7.79	1.76	7.03
4	4	3.6	2.45	1.15
5	5	4.4	3.23	1.17

Database schema consisting of three tables: Store, Product, and Sales

Let's assume the distribution of both tables on different sites is the following:

- The `Store` table has 10,000 tuples stored at site A.
- The `Product` table has 100,000 tuples stored at site B.
- The `Sales` table has one million tuples stored at site A.

Now, assume that we need to process the following query:

```
Select Store_key from (Store JOIN Sales JOIN Product)
where Region= 'East' AND Brand='Wolf';
```

The above query performs the join operations on the `Store`, `Sales`, and `Product` tables and retrieves the `Store_key` values from the table generated in the result c join operations.

Next, assume every stored tuple is 200 bits long. That's equal to 25 Bytes. Furthermore, estimated cardinalities of certain intermediate results are as follows

- The number of the `Wolf` brand is 10.
- The number of `East` region stores is 3000 (since there are 10,000 rows in the store table, and 3000 have region as east).

Communication assumptions are the following:

- Data rate $=$ 50M bits per second
- Access delay $=$ 0.1 second

educative

Before processing the query using different approaches, let's define some parameters:

$a =$ Total access delay

$b =$ Data rate

$v =$ Total data volume

Now, let's compute the total communication time, T, according to the following formula:

$$T = \text{a} + \frac{v}{b}$$

Let's try the following possible approaches to execute the query.

Possible approaches

- Move the `Product` table to site A and process the query at A.

  $$T = 0.1 + \frac{100{,}000 \times 200}{50{,}000{,}000} = 0.5\ seconds$$

  Here, 0.1 is the access delay of the table on site A, and 100,000 is the number of tuples in the `Product` table. The size of each tuple in bits is 200, and 50,000,000 is the data rate. The 200 and 50,000,000 figures are the same for all of the following calculations.

- Move `Store` and `Sales` to site B and process the query at B:

  $$T = 0.2 + \frac{(10{,}000+1{,}000{,}000)\times 200}{50{,}000{,}000} = 4.24\ seconds$$

  Here, 0.2 is the access delay of the `Store` and `Sales` tables. The numbers 10,000 and 1,000,000 are the number of tuples in the `Store` and `Sales` tables, respectively.

- Restrict `Brand` at site B to `Wolf` (called selection) and move the result to site A:

  $$T = 0.1 + \frac{10\times 200}{50{,}000{,}000} \approx 0.1\ seconds$$

  Here, 0.1 is the access delay of the `Product` table. The number of the `Wolf` brand is 10, hence the number of tuples.

When we compare the three approaches, the third approach provides us the least latency (0.1 seconds). We didn't calculate filtering at site A because the number of rows will be much larger, and hence data volume will be more than the third case (filtering at the site B and then fetching data). This example shows that careful query optimization is also critical in the distributed database.

Conclusion

Data distribution (vertical and horizontal sharding) across multiple nodes aims to improve the following features, considering that the queries are optimized:

- Reliability (fault-tolerance)
- Performance
- Balanced storage capacity and dollar costs

Both centralized and distributed databases have their pros and cons. We should