Throughput vs Performance: How to Make them Scale Elastically in a Sharding System

Nga Tran
Feb 2022

Throughput and ***Performance*** scalings are critical design topics for all distributed databases and ***Sharding*** is usually a part of the solution. However, a design that increases throughput does not always help with performance and vice versa; and even if a design supports both of them, making them **scale up and down easily at the same time** is not always easy. This post will describe these two types of scalings for both query and ingest workloads, and discuss sharding techniques that make them elastic. Before digging into the database world, let us see how the elastic throughput and performance scalings affect our daily life.

Scaling Effects in a Fast Food Restaurant

Nancy is opening a fast food restaurant and laying out the scenarios to optimize her operational costs on different days of the week, month, and year. Figure 1 illustrates her business on a quiet day. For the restaurant to be open, there are two lines which must remain open: drive-thru and walk-in. Each requires one employee to cover. On average, each person needs six minutes to process an order and her two employees should be able to cover the restaurant's expected throughput of 20 customers per hour.

With the food she offers, an order can also be processed in parallel by at most two people, one makes drinks and the other makes food. Her employees are trained to go and help with the other line if their line is empty. Doubling up on a single line reduces the order processing time to three minutes and helps keep the throughput steady when customers come to their lines at various spaces.

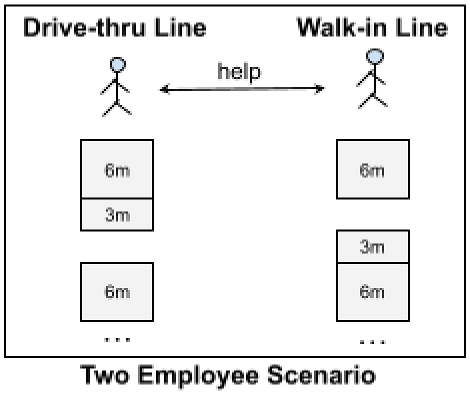

Figure 1: The restaurant operation on a quiet day

Figure 2 shows a busier day with around 50% more customers and adding an employee should cover the 50% *increase in throughput*. Nancy requests her team to be flexible:

- If only one customer comes to a line at a time, one person should run between two lines to help reduce the processing time so they will be available to help new customers immediately.
- If a few customers walk in at the same time, they should open a new line to help at least two walk-in customers at the same time because Nancy knows walk-in customers tend to be happier when their orders are taken immediately but very tolerant with the six minute processing.

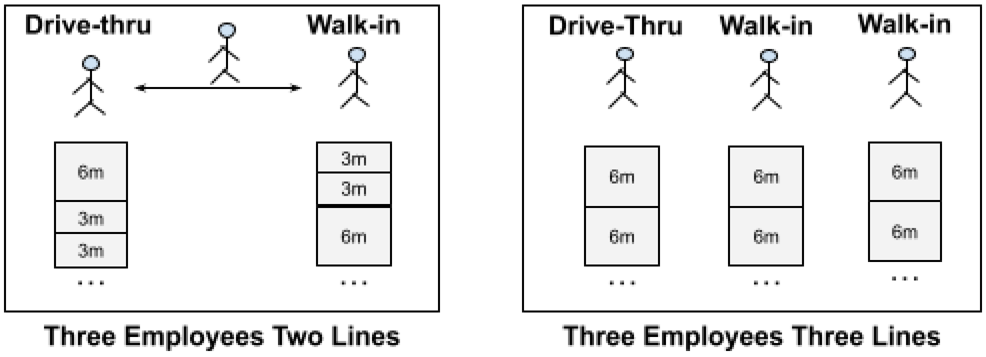

Figure 2: The operation that covers 50% more customers

To deal with the busiest days of the year smoothly, which she estimates around 80 customers per hour, she builds a total of four counters: one drive-thru and three walk-ins, as shown in Figure 3. Since adding a third person to help with an order won't help reduce the order time, she plans to put up to two employees per counter. A few days a year, when there is a big event that the town closes down her street which makes the drive-thru useless, she accepts her *max throughput* will be 60 customers per hour.

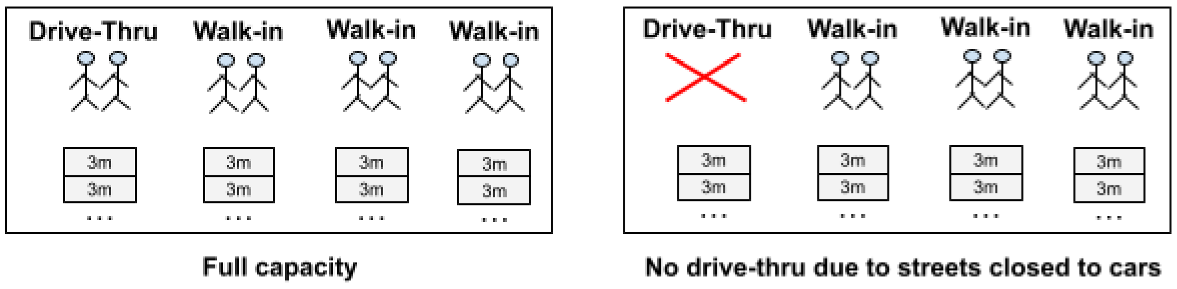

Figure 3: The operation on a busy day

Nancy's ordering strategy **elastically** (aka as needed) scales up and down customer **throughput** while also applying flexibility to make order processing time (aka **performance**) faster. Important points to notice:

1. The **max performance scaling factor** (max number of employees to help with one order) is two and Nancy cannot change it if she wants to stay with the same food offers.
2. The **max throughput** is 80 customers per hour due to the max number of counters being four. Nancy will be able to change this if she has room to add more fixed or removable counters to her restaurant.

Scaling Effects in a Sharding Database System

Similar to the operation at a fast food restaurant, a database system should be built to support elastic throughput and performance scaling for both query and ingest workloads.

Query Workload[1]

Term definition:

- **Query throughput scaling:** the ability to scale up and down the number of queries executed in a defined amount of time such as second or minute.
- **Query performance scaling**: the ability to make a query run faster or slower.
- **Elastic scaling**: the ability to scale (throughput or/and performance) up and down easily based on the needs or/and traffic.

Examples

Assuming the Sales Data is stored in an accessible storage such as a local disk or a remote disk or a cloud. Three teams in the company: Reporting, Marketing, and Sales, want to query this data frequently. Their first setup illustrated in Figure 4 is to have one Query Node to receive all queries from all three teams, read the data, and return the query results.

[1] Due to the scope of the post, other related technology such as Queue Management and Work Stealing are not discussed.

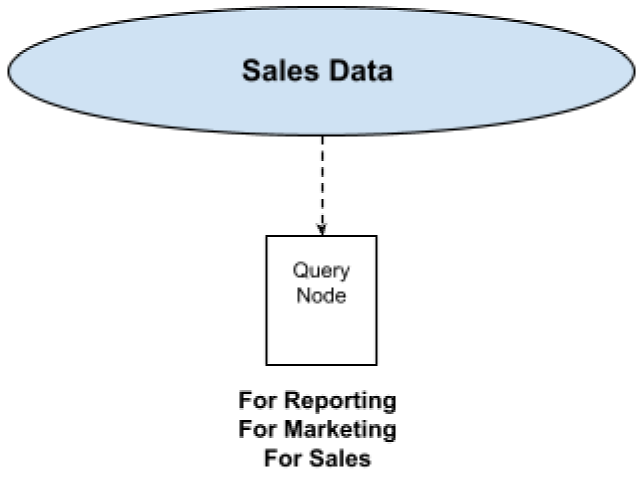

Figure 4: One Query Node handles all requests

At first the setup works well but when more and more queries are added, the wait time to get results back is quite large and many times the queries even get lost due to timeouts. To deal with this ***increasing query throughput requests***, a new setup shown in Figure 5 provides four Query Nodes each of which work independently for different business purposes, one for Reporting team, one for Marketing team, one for Sales team focusing on small customers, and one for Sales team focusing on large customers.

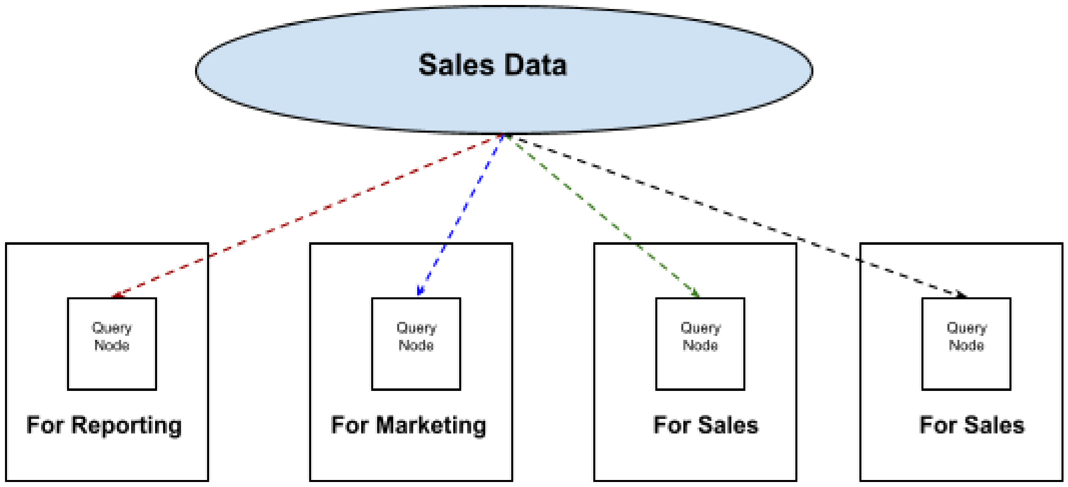

Figure 5: Add more Query Nodes, one for each purpose, to handle more throughput

The new setup catches up well with high volume of throughput and no queries get lost. However, for some sensitive queries the teams need to react to immediately, waiting for several minutes to get the result back is not good enough. To solve this problem, the data is split[2] equally into four shards; each shard contains data of 12 or 13 states, as demonstrated in Figure 6. Since the Report team runs most latency sensitive queries, a Query Cluster of 4 Nodes each

[2] How to split the data will be described in next section, Ingest Workload

handles a shard is built for them to ***perform a query four times faster***. The Marketing team is still happy with their one Query Node setup so data from all shards is directed to that node. The Sales team does not deal with time sensitive queries but as their team gets larger, the number of query requests keep increasing so they ***take advantage of this performance scaling to improve their throughput*** to avoid reaching max throughput in the near future. This is done by replacing two independent Query Nodes with two independent Query Clusters, one four nodes and the other two nodes, based on their respective growth.

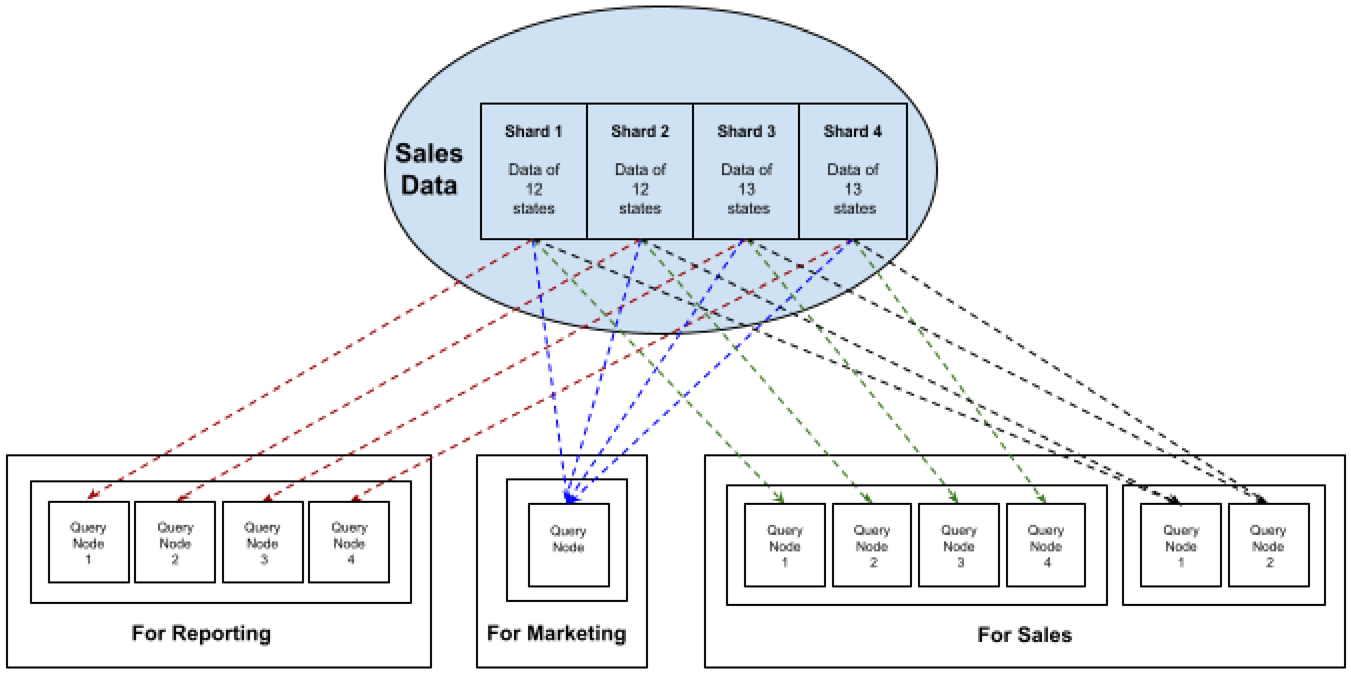

Figure 6: Shard data and add Query Nodes to handle sharded data in parallel

Many times of the year when the Reporting team does not need to handle time sensitive queries, two Query Nodes of its cluster are temporarily removed to save resources as shown in Figure 7. Similarly, when the Sales team does not need to handle high throughput workloads, it temporarily removes one of its clusters and directs all queries to the remaining one.

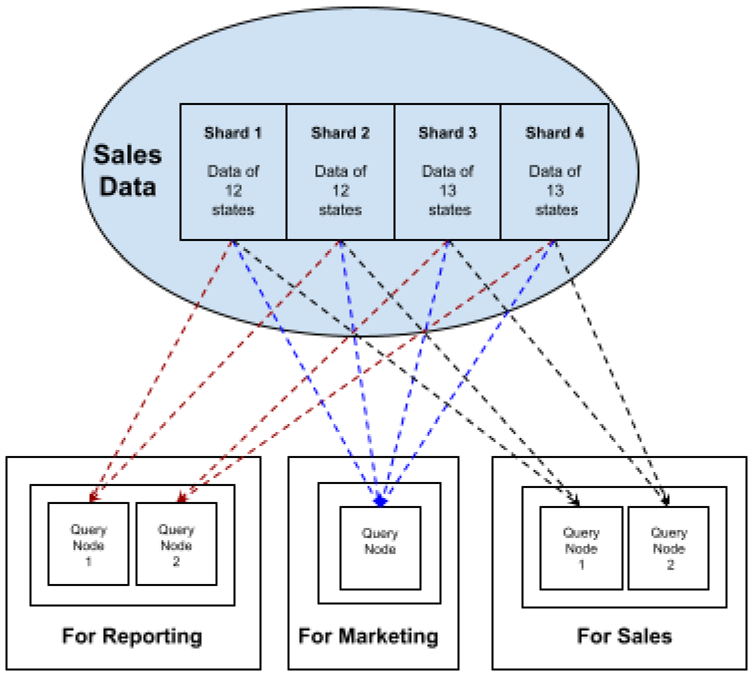

Figure 7: Adjust Reporting cluster size based on their performance needs and Shut down a Sale cluster based on their throughput needs

The teams are very happy with their elastic scaling setup. However, they notice that even though their current setup can scale up/down the throughput easily by adding/removing Query Clusters, performance scaling for the Report team cannot go up further when it reaches the limit factor of four (Query Nodes); scaling Query Nodes beyond that limit won't help. Thus they say their ***query throughput scaling is fully elastic, but their query performance scaling is only elastic to the scale factor of four***.

The only way they can scale the performance for a query up further is to split data into more and smaller shards which is not trivial and the subject to discuss next.

Ingest Workload

Term definition:

- **Ingest throughput scaling:** the ability to scale up and down the amount of ingested data in a defined amount of time such as second or minute.
- **Ingest performance scaling**: the ability to make a set of data ingested in the system faster or slower.

Examples

In order to have four-shard Sales Data as described above, the ingesting data must be sharded at load time. Figure 8 illustrates an Ingest Node that takes all ingest requests, shards them accordingly, handles pre-ingest work[3], and then saves data to the right Shard.

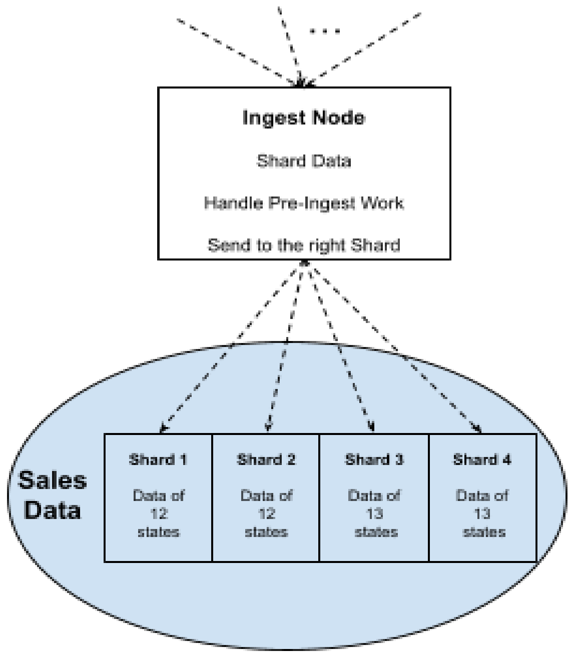

Figure 8: One Ingest Node handles all Ingesting Data

When the Ingest data increases, one Ingest Node no longer catches up with the requests and ingesting data gets lost, a new setup shown in Figure 9 is built to add more Ingest Nodes, each handles data for a different set of write requests to support higher ingest throughput.

[3] The pre-ingest work can be different depending on the needs of each system but they usually include sorting and compressing/encoding data which are beyond the scope of this post.

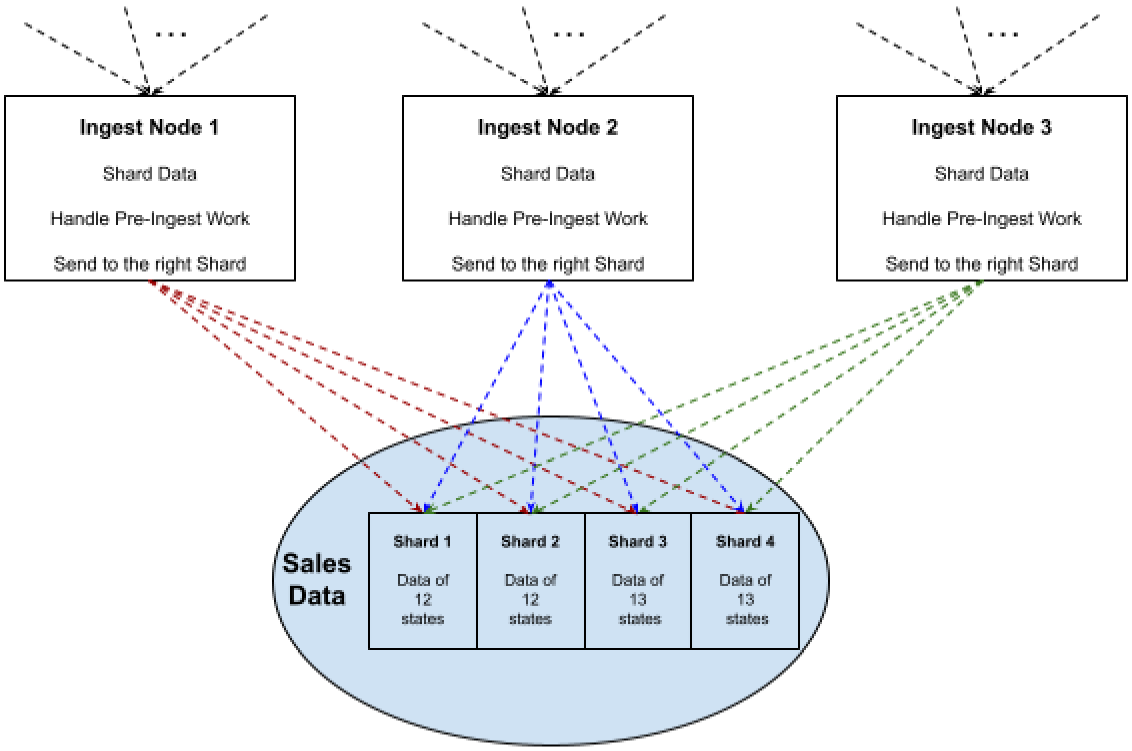

Figure 9: Add Ingest Nodes, one for a subset of write requests, to handle more throughput

Even though the new setup handles the high ingest volume of throughput and no data gets lost, the increasing demand of lower ingest latency makes the teams think they need to change the setup further. The Ingest Nodes that need smaller ingest latency are converted into Ingest Clusters demonstrated in Figure 10. Each cluster includes a Shard Node that is responsible for sharding the coming data and other Ingest Nodes each is responsible for processing Pre-Ingest work for data of its assigned shards and sending them to the right shard storage. The performance of Ingest Cluster 2 is twice faster as its latency is now around half of its previous setup and the Ingest Cluster 3 is around four times faster.

Many times of the year, when the latency is not critical, a couple of nodes are temporarily removed from the Ingester Cluster 3 to save resources. When ingest throughput is a lot less, Ingest Cluster 2 and Cluster 3 are even shut down and all write requests are directed to the first one for ingesting.

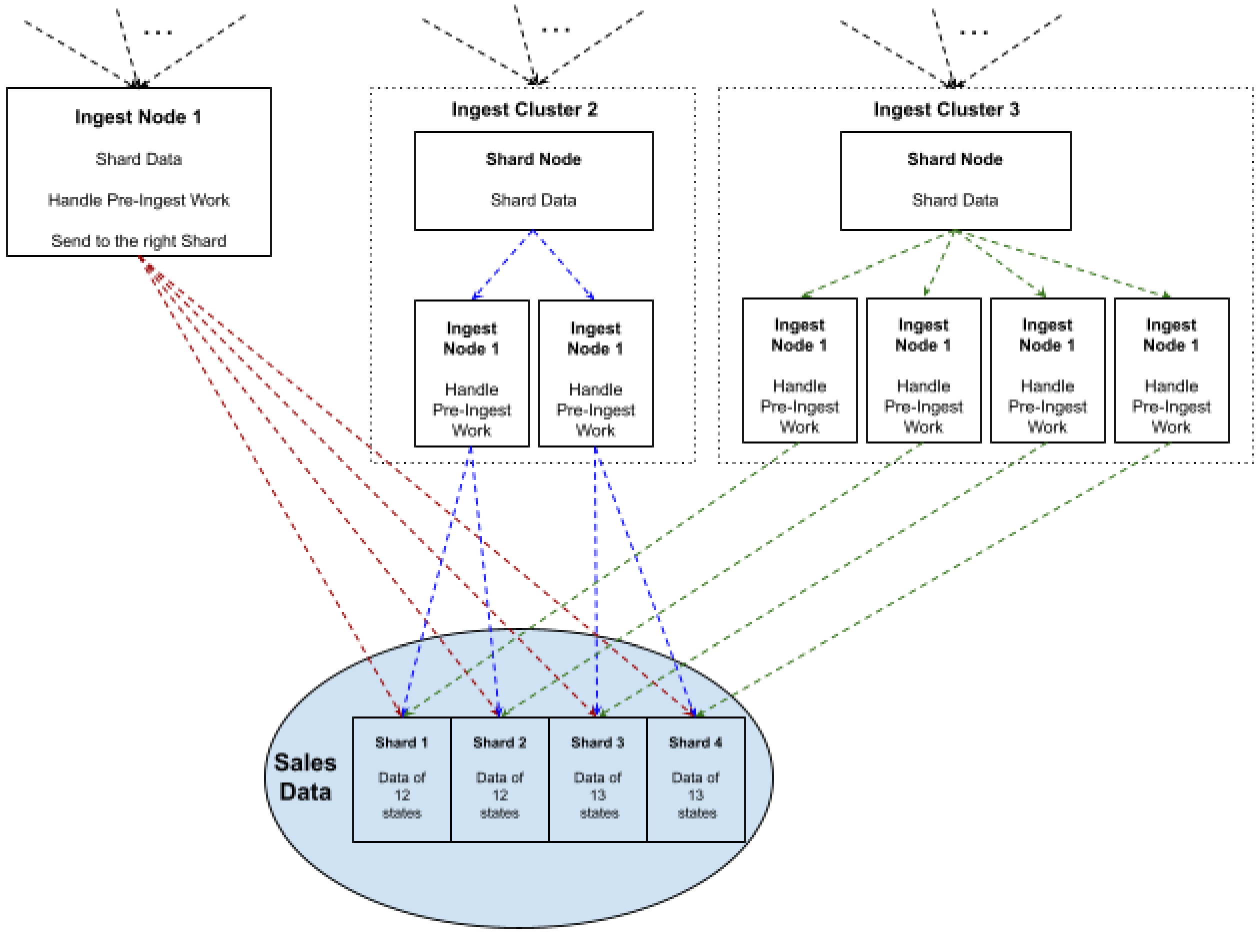

Figure 10: Convert Ingest Nodes to Ingest Cluster to handle faster data load.

Similar to the setup for their Query workloads, the teams are very happy with their elastic scaling setup for their Ingest workloads but notice that even though the setup can scale up/down the ingest throughput easily by adding/removing Ingest Clusters, Ingest Cluster 3 has reached its performance scale factor of four; adding more Ingest Nodes to its cluster won't help. Thus they say their ***ingest throughput scaling is fully elastic, but their ingest performance scaling is only elastic to the scale factor of four***.

Preparing for future Elasticity

As demonstrated in the examples, the query and ingest throughput scalings of the setups in Figure 6 and Figure 10 respectively are fully elastic, but their performance scaling is only elastic to the scale factor of four. To support higher **performance scaling factor**, the data should be split into smaller shards, e.g., one per state. However, in this case, when we go with a smaller scale factor many shards must be mapped to one Query Node in the Query Cluster; similarly, one Ingest Node must handle data of many shards.

A limitation of performance scaling is increasing scale factor (aka split data further to smaller shards) does not mean the system will scale as expected because of overhead or/and the limitation of each use case as specified in the Fast Food example.

Elastic Throughput and Performance Scalings described in this post are just examples for us to understand their role in a Database System. The real designs to support them are a lot more complicated and need to consider more factors.