Summarization of Solutions to Data Skew in Apache Spark

Jiashu Chen

March 6th, 2024

Abstract

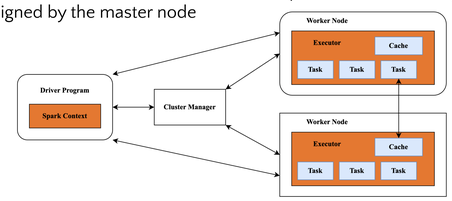
Apache Spark is a big data computing framework for parallel data processing on computer clusters. Spark's in-memory computing capability makes it ten to a hundred times faster than the disk-based Hadoop MapReduce system, especially for iterative algorithms and interactive data mining tasks. Despite its reputation for fast in-memory computation, its performance could degrade due to data skew.

This project first provides a summarization of solutions to address data skew in distributed computing systems based on previous studies. In the implementation phase, 9 join queries with skew issues are tested in Spark. Following the analysis of experiment results, insights on skew mitigation are discussed from the viewpoint of a Spark user.

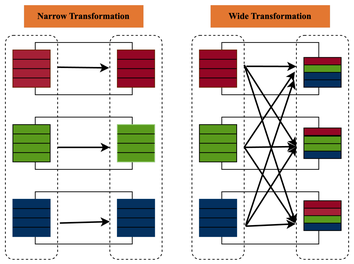
Introduction

I. What is Apache Spark

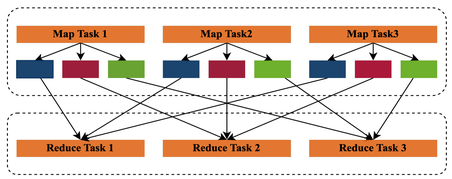
- *Spark Cluster Computing Architecture:* Spark cluster uses the master – worker architecture. Driver is the master node that manages a cluster of workers. Worker nodes are responsible for executing tasks assigned by the master node



- *Spark RDD*: Resilient Distributed Dataset (RDD) is the basic data structure in Spark. RDD uses coarse-grained transformation that enables Spark to evaluate transformations lazily as it could keep a record of dependencies between RDD transformations and implement actual transformations when an action is triggered.
- *Narrow/Wide Transformations:* In narrow transformations, one partition in the child RDD depends on one partition in the parent RDD. It includes operators such as map, filter. In wide transformations, one partition in the child RDD depends on multiple partitions in the parent RDD. It include operators such as groupBy, join, etc.



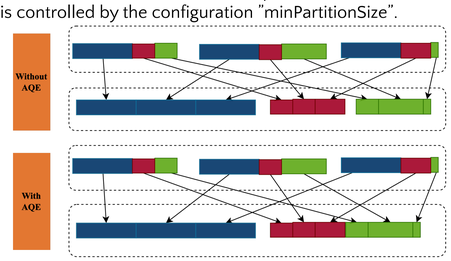
- *Shuffle:* Shuffling redistributes data among partitions. In shuffle write, data is partitioned and persisted to disks. Each shuffle map task has R buckets, which equals to the number of reducers. Buckets refers to an in-memory buffer, which are written to local disks files called shuffle block file. In shuffle read, reducers fetch shuffle block files and perform aggregate computations. Shuffle is an expensive operation as it involves disk I/O, data serialization, and network I/O



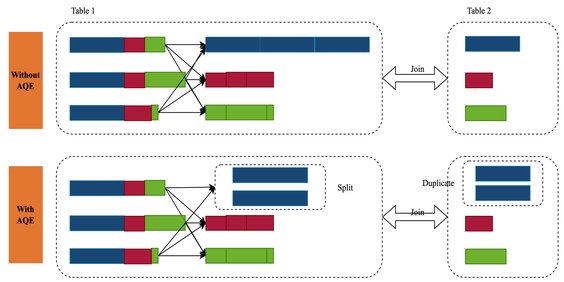
Introduction

II. Data Skew Solutions

- *Skew Mitigation for Parallel Databases:* Range partitioning split data based on the range of the key value. Each processor is assigned with a subrange of key value. If there are k processors, the entire key value range could be split into k subranges. The (k-1) splitting boundaries could be determined based on distribution of key values to equalize the number of tuples assigned to each processor. (DeWitt et al. (1992)).
- *Skew Mitigation in Spark:* Adaptive Query Execution: Adaptive Query Execution (AQE) is an optimization technique introduced in Spark 3.0, which is to re-optimize query plan based on runtime statistics
  - Coalescing Post Shuffle Partitions: The best number of shuffle partitions is data-dependent. Dynamically coalescing shuffle partitions is to coalesce small partitions after shuffling. The threshold to determine whether a partition needs to be coalesced is controlled by the configuration "minPartitionSize".



  - Dynamically Switching Join Strategies: AQE could adjust the join strategy based on runtime statistics. If the materialized join relation is overestimated and it is smaller than the broadcast threshold, AQE would switch to broadcast join.
  - Splitting Skewed Shuffle Partitions: AQE skew join optimization could detect skewed partition based on shuffle file statistics. If some partition is too big, it would be split into smaller partitions.



Experiment

I. Spark Environment

Spark is installed locally and all the computations are performed locally using 10 cores. Query plans and computing are obtained from Spark UI.

II. Two Datasets

- *New York Taxi Trips Data:* One table is taxi trip records, which is a big table with 19,817,583 records. Another table is the taxi zone look up table, which is a small table with only 265 records.
- *IMDB Data:* One table is name.basics, which is a big table with 13,302,452 records. Another table is title.principals, which is also a big table with 60,717,362 records.

III. Skewness Manipulation

Both datasets are manipulated to increase the skewness level. For instance, In the taxi_trip table, a significant number of records in the PULocation field are altered to a specific locationID, resulting in one partition containing significantly more records than others during shuffling

Experiment

IV. Four Configurations & Four Join Commands

Configuration	Broadcast Join	AQE	AQE Parameters
Conf1	Disabled	Disabled	Disabled
Conf2	Enabled	Disabled	Disabled
Conf3	Disabled	Enabled	Default: minPartitionSize(1MB); skewedPartitionFactor(5); skewedPartitionThresholdInBytes(256MB)
Conf4	Disabled	Enabled	Adjusted: minPartitionSize(128K); skewedPartitionFactor(3); skewedPartitionThresholdInBytes(256K)

Join Command	Data	Key Salting
Join Command 1	Taxi Trips Data	No
Join Command 2	Taxi Trips Data	Yes
Join Command 3	IMBD Data	No
Join Command 4	IMBD Data	Yes

Results

I. Join Results of Taxi Trip Data

6 join queries are tested on the taxi trip data. These join queries represent join between a large table and a small table.

Query	Configuration	Join Command	Key Salting	Query Plan	Straggling Task	Join Stage Duration	Median/75th Percentile/Max Join Task Duration
Query 1	Conf1	Join Command 1	No	Sort Merge Join	Yes	9s	1s; 1s; 9s
Query 2	Conf1	Join Command 2	Yes (20)	Sort Merge Join	No	4s	2s; 2s; 4s
Query 3	Conf1	Join Command 2	Yes (40)	Sort Merge Join	No	2s	1s; 2s; 2s
Query 4	Conf2	Join Command 1	No	Broadcast Hash Join	No	~0.8s	0.5s; 0.7s; 0.7s
Query 5	Conf3	Join Command 1	No	Sort Merge Join	Yes	8s	1s; 8s; 8s
Query 6	Conf4	Join Command 1	No	Sort Merge Join	No	5s	4s; 5s; 5s

- *Query1*: Broadcast Join is disabled in configuration 1. Salting technique is not used and AQE is disabled. Therefore, there is a straggling task that takes much longer than other tasks.
- *Query2*: Broadcast Join is disabled. Salting technique is used and AQE is disabled. Compared with Query1, we could see that key salting could effectively balance workload

Query1 Query2

- *Query3*: Query3 is the same as Query2, except that the splitting number is increased from 20 to 40, which means that each partition is split into more small partitions. The tasks are even more balanced compared to Query 2.
- *Query4*: Broadcast Join is enabled. Since the smaller table is less than the broadcast threshold, broadcast hash join is used. The join stage uses only 0.8s.

Query3 Query4

- *Query5*: Adaptive Query Execution is used. Since the largest partition (122.6MB) is less than this threshold of skewed partition, it remains unsplit. Therefore, even though two small partitions are coalesced after shuffling, the workload remains unbalanced.
- *Query6*: Query6 is the same as Query5, except that the AQE parameters are adjusted. Since the data size is relatively small, the "skewedPartitionThresholdInBytes" is set to 256K so that the largest partition in Query 5 would be detected as skew partition. The workload is slightly more balanced compared to Query 5.

Query5 Query6

II. Join Results of IMDB Data

3 join queries are tested on the IMDB data. These join queries represent join between two large tables.

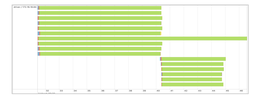
Query	Configuration	Join Command	Key Salting	Query Plan	Straggling Task	Join Stage Duration	Median/75th Percentile/Max Join Task Duration
Query 9	Conf2	Join Command 3	No	Sort Merge Join	Yes	13s	6s; 7s; 12s
Query 10	Conf2	Join Command 4	Yes (5)	Sort Merge Join	No	23s	15s; 15s; 16s
Query 11	Con3	Join Command 2	Yes	Sort Merge Join	Yes	15s	9s; 9s; 15s

Results

- *Query9:* Sort Merge Join is used since both tables exceed the broadcast join threshold. There is one task that takes relatively longer time than other tasks.
- *Query10:* Salting technique is used and AQE is disabled for partition distribution. However, as salting involves duplicating one of the join relation and both relations in this query are large. In this case, the name table has 13,302,452 rows originally. After salting, it has 66,512,260 (5 * 13,302,452) rows. The join stage takes more time compared to Query9. Moreover, the query failed several times due to out of memory error.

Query9 Query10

- *Query11*: AQE is used. From the query plan, no partition is split or coalesced after shuffling. The workload distribution is almost the same as Query9 as AQE doesn't flag the largest partition as a skew task. The join stage takes even longer time due to additional overhead introduced by AQE.



Query11

Conclusion

Based on experiment results, some insights are summarized from the viewpoint of a Spark user.

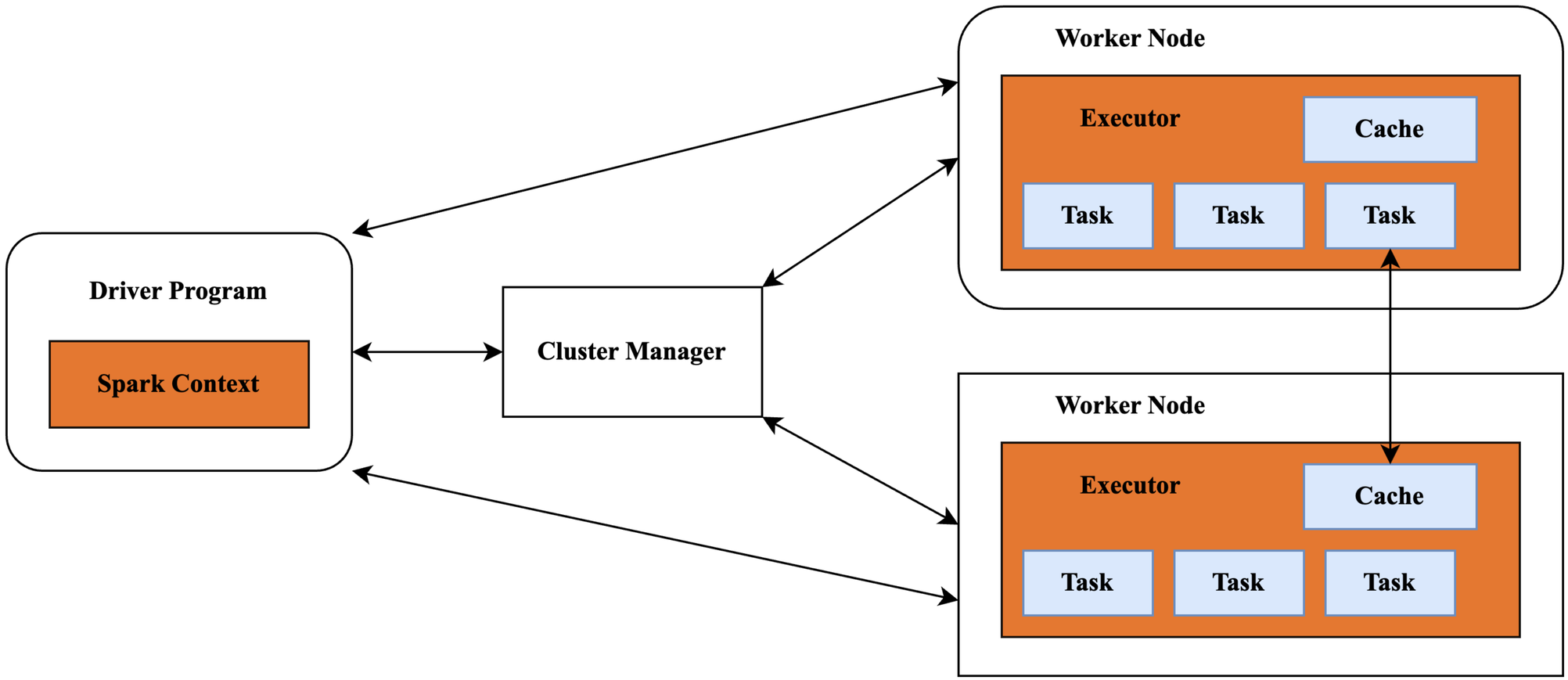
- If one relation in the join query is small, broadcast hash join would be the most efficient method.
- If one relation is small, but its size exceeds the broadcast hash join threshold, users could adjust the broadcast threshold configuration or use salting technique to distribute data evenly across partitions.
- In the case of key salting, a larger splitting number indicates a finer partitioning of the skewed partition, but it also requires increased replication of one partition.
- Adaptive Query Execution (AQE) does not guarantee skew mitigation. Its effectiveness depends on whether the configuration thresholds align with the data.
- AQE may not maximize parallelism. Executors and cores might not be fully utilized.
- Salting technique involves data duplication. Its use should be carefully considered, especially when dealing with large join relations.

In conclusion, while there are various ways to address data skew in Spark, there lacks a fully transparent and automatic solution for skew mitigation. Existing techniques may require extra effort from users in certain scenarios.

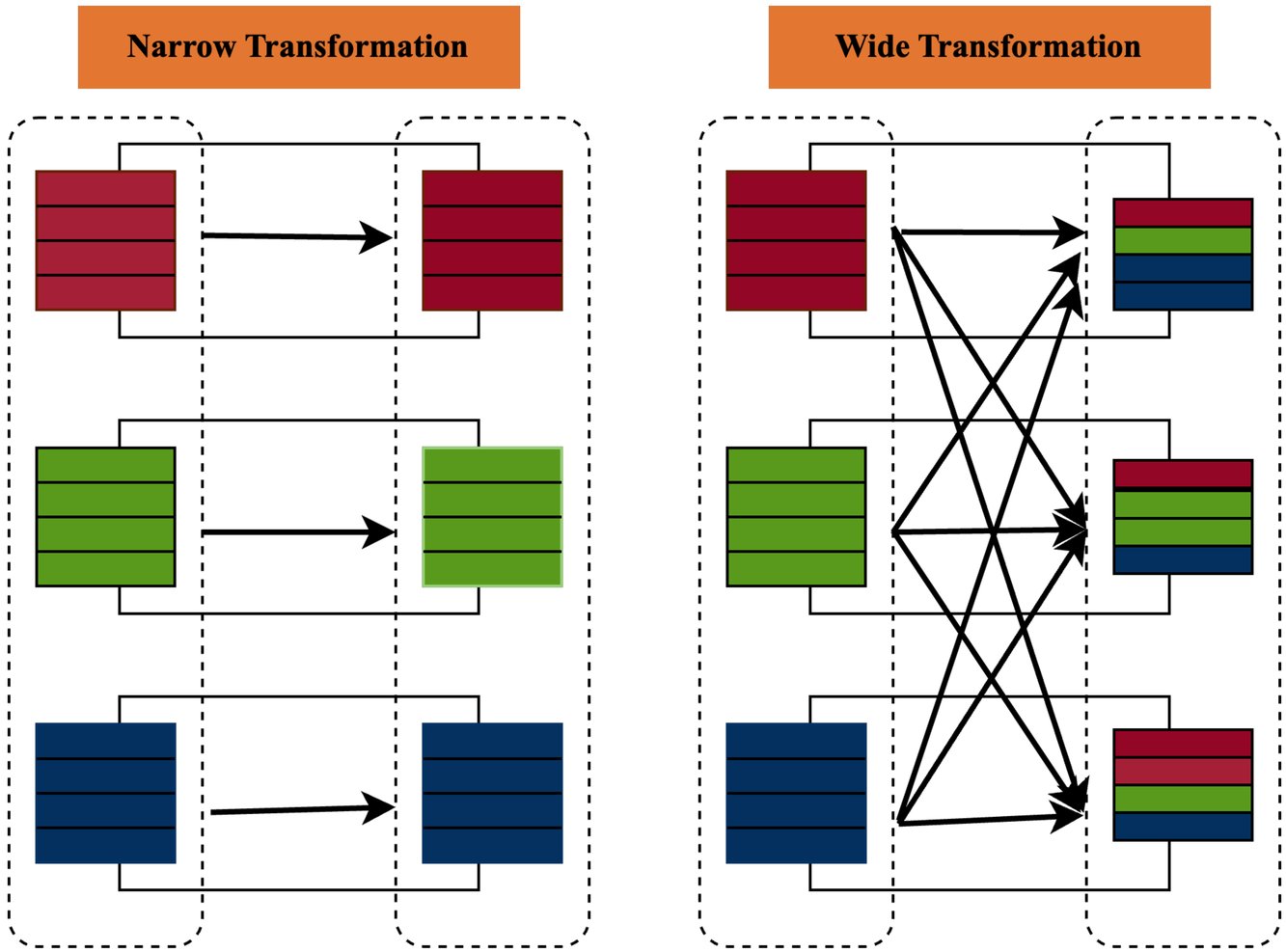
Limitations

- The experiment primarily focuses on the user's perspective. However, further experiments could provide valuable insights from a developer's viewpoint, such as exploring the integration of range partitioning in Spark.
- The experiment is conducted on a local machine, utilizing a single node with 10 cores. This setup may not accurately represent the complexities of cluster computing environments.
- The datasets utilized in the experiments remain relatively small in size.

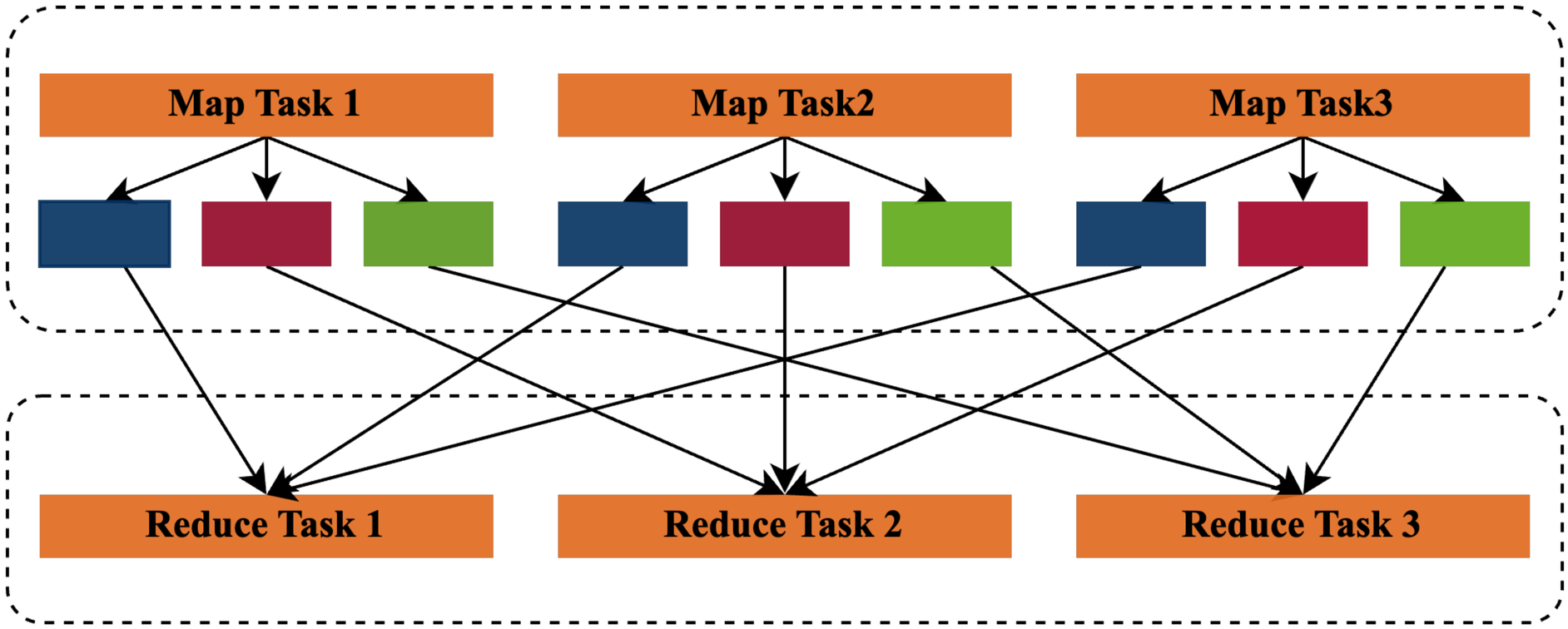
Spark Cluster Computing Framework



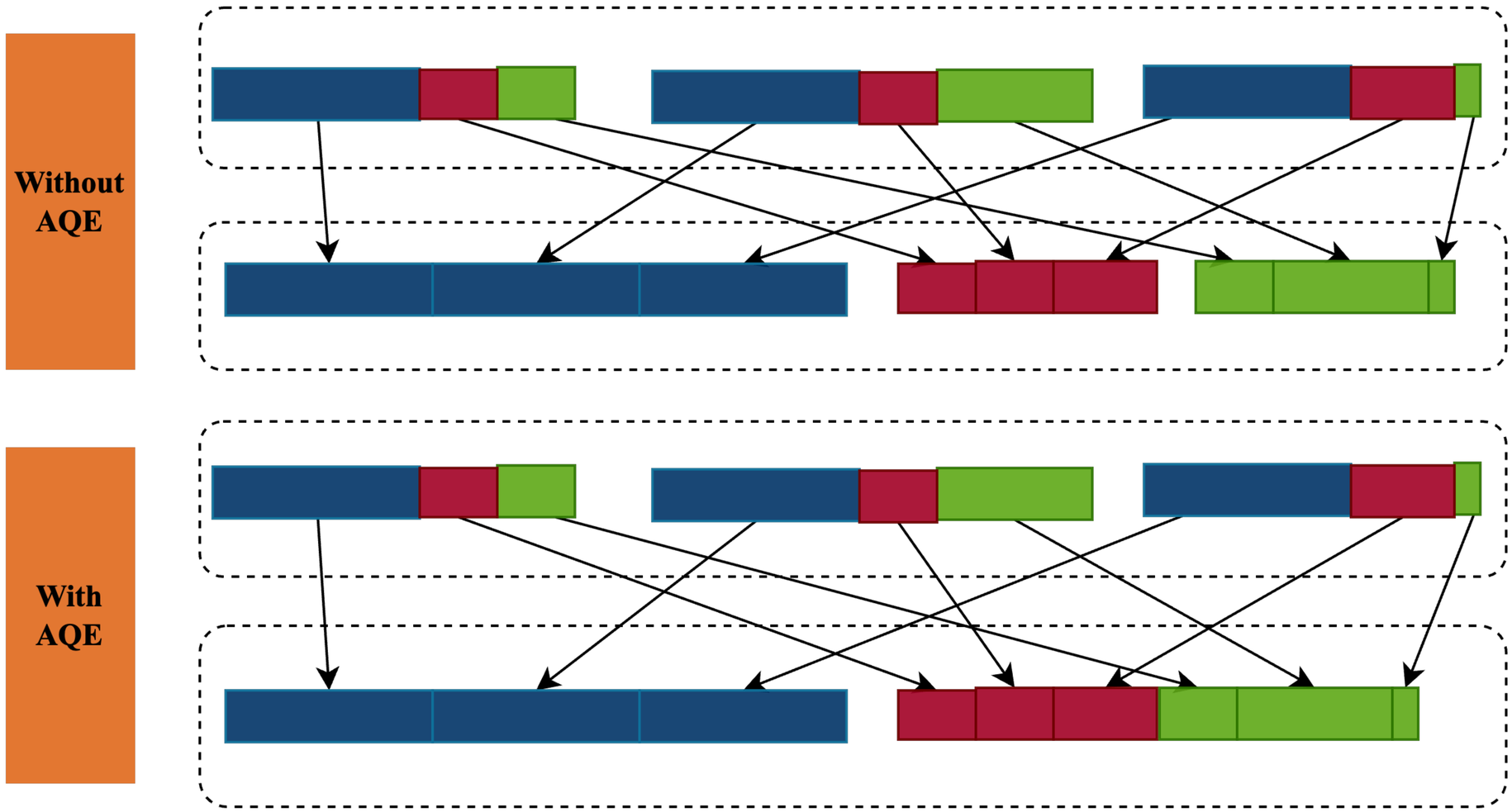
Narrow Wide Transformation



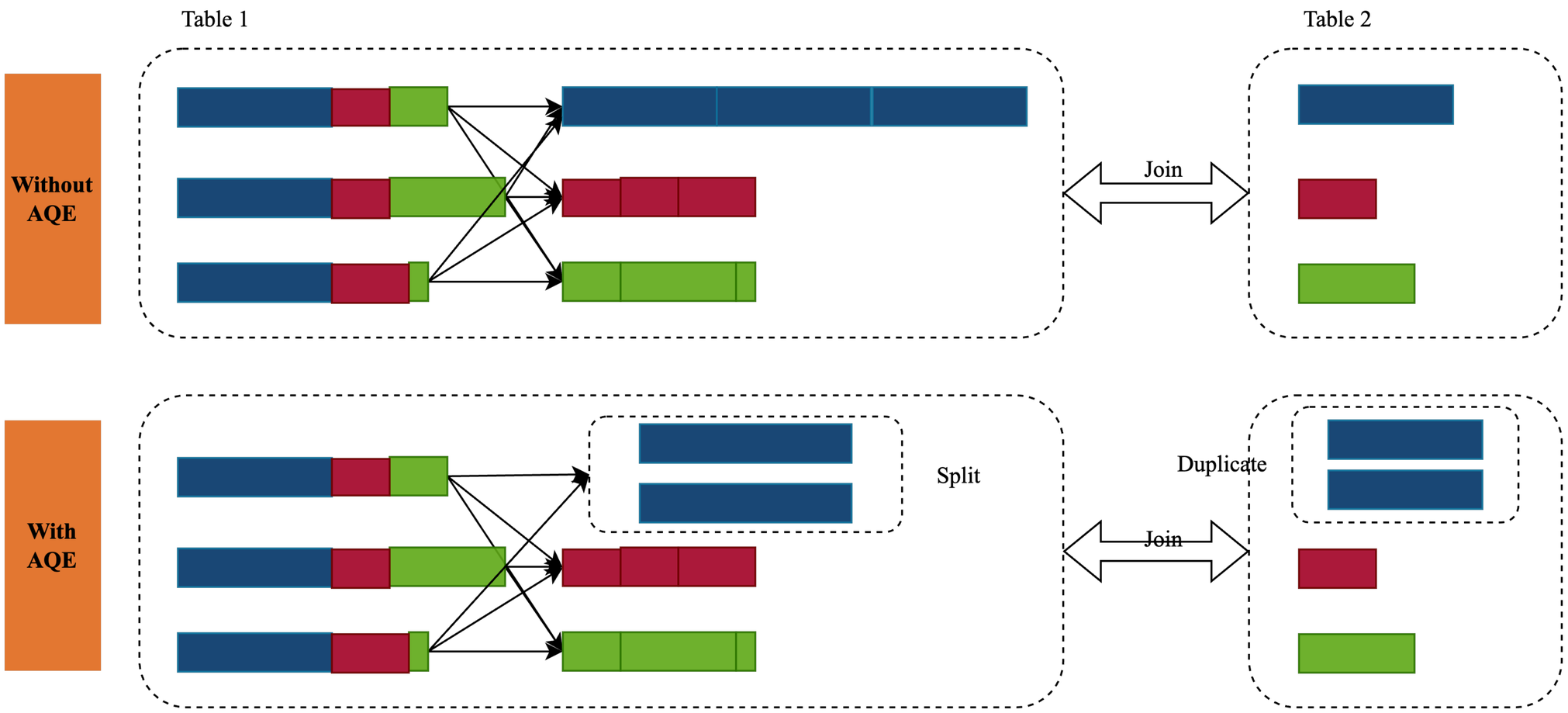
Shuffling


Map Task 1
Map Task2
Map Task3
Reduce Task 1
Reduce Task 2
Reduce Task 3

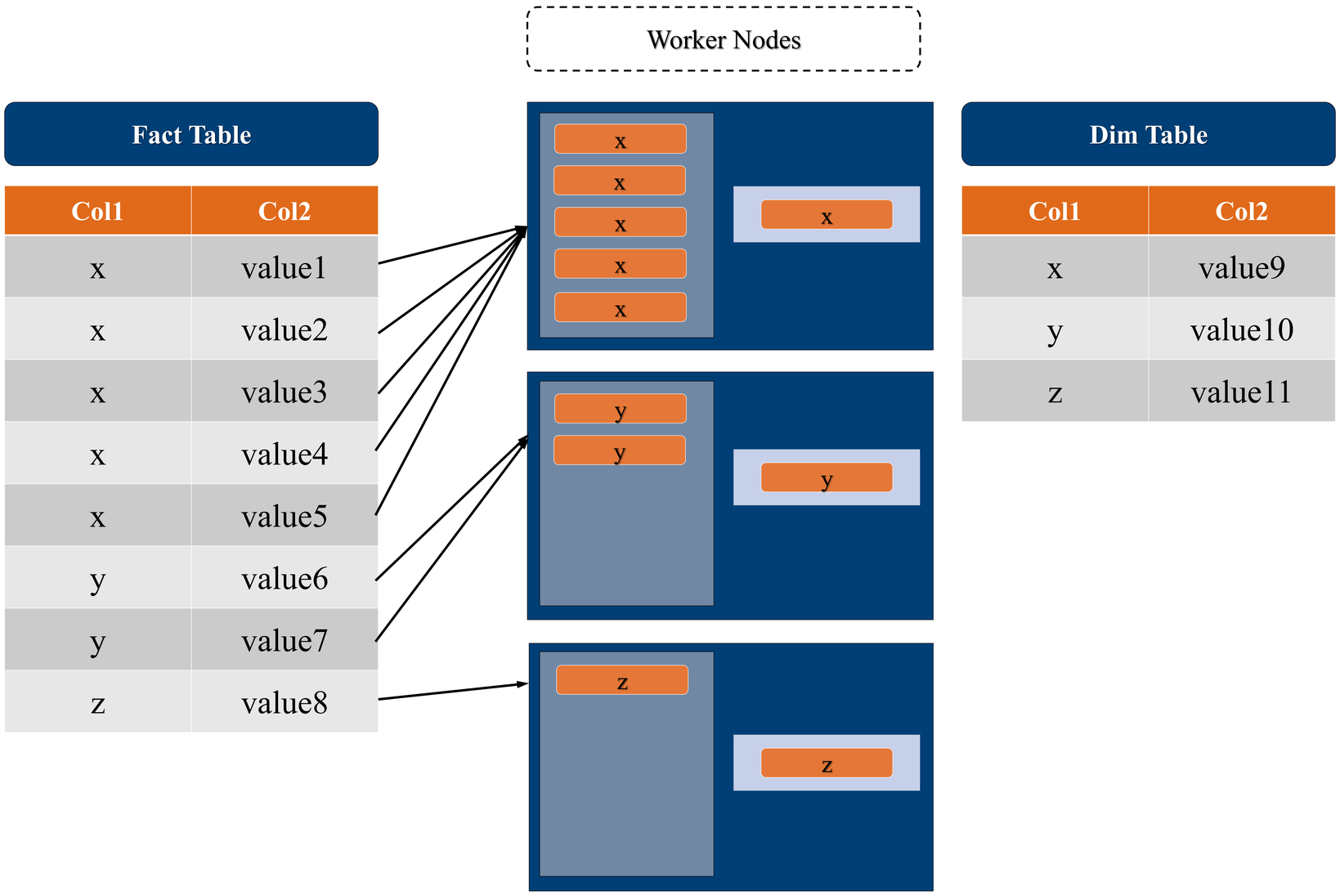
AQE: Coalescing Post Shuffle Partitions



AQE: Splitting Skewed Shuffle Partitions



Skewed Join


Worker Nodes
Fact Table
Col1
Col2
x
value1
x
value2
x
value3
x
value4
x
value5
y
value6
y
value7
z
value8
x
x
x
x
x
x
y
y
y
z
z
Dim Table
Col1
Col2
x
value9
y
value10
z
value11

Skewed Join using Salting

Fact Table

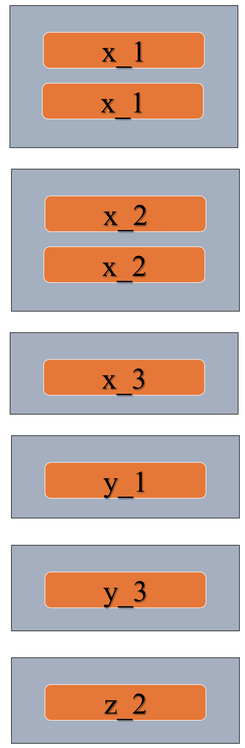
Col1	Col2
x	value1
x	value2
x	value3
x	value4
x	value5
y	value6
y	value7
z	value8

Salted Fact Table

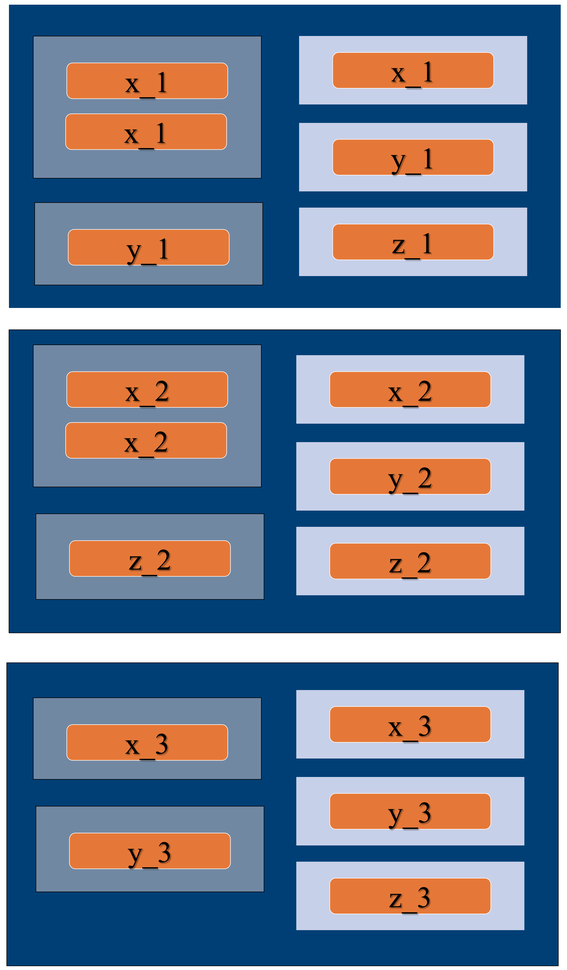
salting_num = 3

Col1	Col2	Salted_key
x	value1	x_1
x	value2	x_2
x	value3	x_3
x	value4	x_1
x	value5	x_2
y	value6	y_3
y	value7	y_1
z	value8	z_2

Partitions



Worker Nodes



Duplicated Dim Table

salting_num = 3

Col1	Col2	Salted_key
x	value9	x_1
x	value9	x_2
x	value9	x_3
y	value10	y_1
y	value10	y_2
y	value10	y_3
z	value11	z_1
z	value11	z_2
z	value11	z_3

Dim Table

Col1	Col2
x	value9
y	value10
z	value11