International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed and Bimonthly Journal|

|| Volume 6, Issue 3, May– June 2023 ||

DOI: 10.15680/IJCTECE.2023.0603005

Optimizing Multi-TB Market Data Workloads: Advanced Partitioning and Skew Mitigation Strategies for Hive and Spark on EMR

Janardhan Reddy Kasireddy

Lead Data Engineer, Info Drive Systems (Finra Contractor), USA

ABSTRACT: One of the main issues of the contemporary big data analytics is the ability to process multi-terabyte data in a cost-effective way. This paper explores extreme parallel processing with Hive and Spark in a large scale market data setup with the view to optimizing query performance and resource consumption. The NYSE Stock Prices dataset was simulated on Kaggle, and experimented on with simulated workloads in the real world such as high-frequency trade data, historical price series and market metadata. The techniques applied were on performance engineering by developed partitioning schemes, query optimization and the mitigation of skew. The improvement of hive queries was based on dynamic partition pruning, vectorized execution, and prudent join ordering whilst Spark workloads were enabled by dataframe caching, broadcast joins, and adaptive query execution. The two environments were implemented on Amazon EMR clusters to test the scalability depending on the number of nodes and the cluster configurations.

We show that with good use of partition pruning and vectorised execution Hive was able to run up to 4x faster on aggregation intensive workloads compared to Spark, and to run up to 6x faster on iterative and join heavy queries with broadcast joins and adaptive execution. Biased data partitions have been found to be a key performance bottleneck and alleviation via salting methods enhanced throughput. The paper also identified trade-offs between allocate compute resources, manage the memory, and job parallelism in the large-scale cluster setting.

These results have a real-world implication to the enterprises working with multi-TB datasets in both Hive and Spark wherein they highlighted the significance of partition-aware design and query optimizations and cluster-level optimizations. The work can be used as a framework of performance engineering as a reference in extreme-scale data processing.

KEYWORDS: Hive, Spark, EMR, query optimization, parallel processing, partition pruning, skew, performance engineering.

I. INTRODUCTION

The spread of financial information in the modern marketplace has altered the manner in which businesses conduct analytics, trade schemes, and risk analysis [1]. The regional trading systems are connected with algorithms of trading and institutional analytics that depend on the possibility to analyze vast amounts of structured and semi-structured data in almost real-time [2]. The New York Stock Exchange (NYSE) itself produces terabytes of transaction, order, and pricing data on a daily basis, and once historical series, market metadata and alternative datasets are added, the size grows into multi-terabytes (multi-TB) [3] [4]. The ability to manage and process these high-volume workloads of data efficiently is the main key to safeguarding competitive advantage, making decisions in time, and reducing the costs of operations. The mature and reliable traditional relational database management system is also becoming inadequate to support such extreme workloads with regard to parallelism, memory management and disk I/O limit. As a result, the distributed data processing systems configured as Apache Hive and Apache Spark, which are frequently executed on a cloud-native version like Amazon EMR (Elastic MapReduce), have become the de-facto solutions to large-scale market data analytics [5].

Initially created as an infrastructure of data warehouse based on Hadoop, Hive offers a SQL-like interface to batch computation on large datasets in distributed file systems [6]. It supports its declarative query language, HiveQL, which allows analysts and engineers to write up complex queries without having to care about low-level execution details [7]. Hive converts such queries into efficient MapReduce, Tez or Spark scripts, which enable the execution of large data sets in hundreds or thousands of nodes simultaneously [8]. Spark, however, can be used to in-memorize, iteratively

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed and Bimonthly Journal|

|| Volume 6, Issue 3, May– June 2023 ||

DOI: 10.15680/IJCTECE.2023.0603005

compute, and interactively process data, and it provides a resilient distributed dataset (RDD) and DataFrame API to quickly process data [9]. The execution engine of Spark is focused on low-latency transformations, caching and adaptive query execution that are especially optimized to workloads with iterative algorithms, multifaceted joins, and machine learning pipelines [10]. Although both Hive and Spark offer immense performance benefits over traditional relational databases, they introduce new issues with multi-TB datasets: there are skews in data, poor partitioning, and query plans are not optimized, and there are trade-offs between memory allocation, parallelism, and use of clusters.

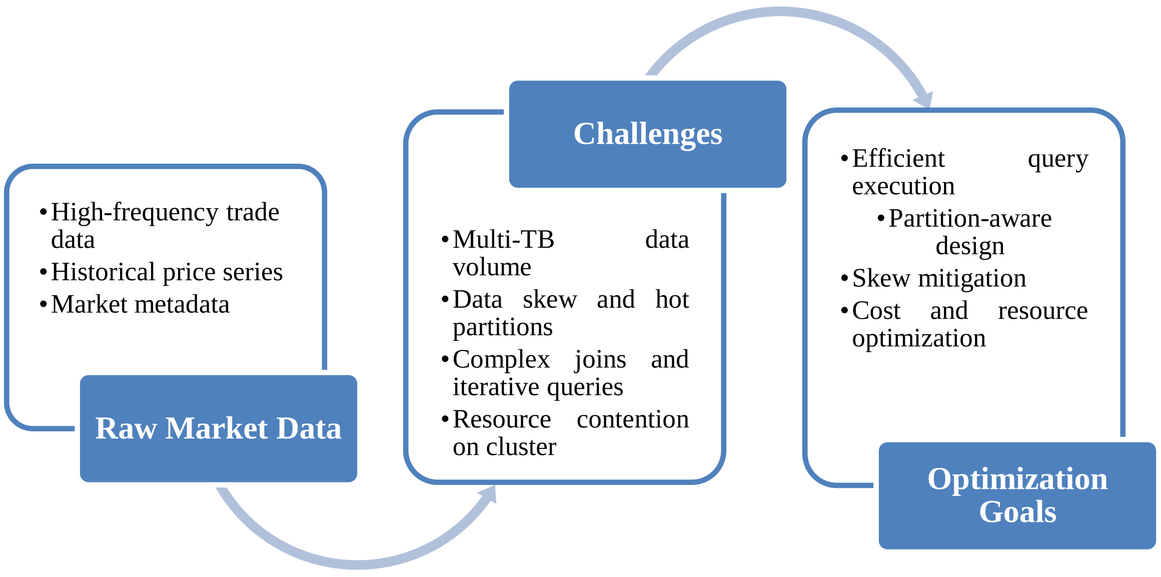
Raw Market Data
• High-frequency trade data
• Historical price series
• Market metadata
Challenges
• Multi-TB data volume
• Data skew and hot partitions
• Complex joins and iterative queries
• Resource contention on cluster
Optimization Goals
• Efficient query execution
• Partition-aware design
• Skew mitigation
• Cost and resource optimization

Figure 1: Overview of Multi-TB Market Data Workload Challenges and Optimization Goals

The design and management of partitions is one of the most important issues about the optimization of large-scale data workloads [11]. Partitioning: This process involves splitting a dataset into small parts that are easily manageable, usually on some attribute like by date, stock symbol or market sector. When well partitioned, query engines can scan the small parts of data which is called partition pruning thus minimizing disk I/O, network communication and time of query execution. Hive works well with both static and dynamic partitioning, and dynamic partition pruning provides further performance advantage by computing the necessary partitions at runtime depending on query predicates [12]. This is especially acute in the case of financial data, where the high-frequency trades or market events would result in hot partitions of data that are strongly concentrated. Change of skew by using salting techniques, bucketing, or hash-based repartitioning becomes critical to make sure that there is a balanced workload in the cluster and complete utilization of parallelism [13].

Another principle component in a high-performance multi-TB workload is query optimization [14]. A number of strategies are provided by Hive, such as; the use of vectors, predicate pushdown, reordering of joins, and cost-effective optimization to increase the query throughput and reduce the unwarranted computations. The execution in batches (vectorized execution) allows Hive to execute batches of rows concurrently to reduce CPU load, optimize their use of caches, which is particularly useful in operations that use a large fraction of aggregation (eg. computing moving averages, volume-weighted averages, and volatility indicators). Combining trade orders with historical price information, or market data can also be optimally executed using smart join ordering and map joins to shuffle large data sets over the network. Spark provides a set of DataFrame optimization, broadcast joins, caching policies and adaptive query execution to support query performance. Broadcast joins enable the small tables like reference metadata to be replicated in worker nodes, which reduces the network traffic and enhances join performance. The caching of intermediate results in memory and dynamic plans of physical execution improve both iterative computations and physically dynamically adjusting plans, respectively, to better use the available resources and improve parallelism.
Implementing Hive and Spark on Amazon EMR clusters provides an extra optimization potential and difficulty. EMR is

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed and Bimonthly Journal|

|| Volume 6, Issue 3, May– June 2023 ||

DOI: 10.15680/IJCTECE.2023.0603005

a flexible and scalable environment where the types of nodes, the number of instances, and the storage can be configured, allowing a fine-grained control of parallelism, memory usage and I/O throughput. Efficiency of multi-TB workload execution directly depends on the choice of cluster configuration, i.e. whether the instance types are optimized to compute, memory, or storage. In addition, the trade-offs between the provisioning of resources and costs are vital in enterprise environments, with overproviding causing unwarranted costs, whereas underproviding may cause extended period of execution time and failed job execution. A holistic performance engineering method is therefore required, which is a combination of data partitioning strategy, skew reduction, query level optimizations, and cluster level tuning in a manner that can guarantee predictable performance at scale at a cost-effective manner.

Although Hive and Spark have been widely used in the business world, the literature lacks a significant systematic study that measures the performance enhancement of these tools in large-scale financial data as realistic workloads. Although previous studies have already shown that partitioning, caching, and join optimizations are suitable in general-purpose applications of big data, financial data has its own set of challenges. Market data can be described as being highly granular, cardinal and irregularly distributed. To illustrate this, one day of high frequency trading can result in millions of orders being made on a particular group of securities and some other securities will be significantly idle. History prices series can be decades-long, leading to profound time divisions, which increases skew, and metadata tables can be comparatively small but often joined to large data sets to be enriched.

The current research will attempt to solve these problems by methodically comparing and streamlining the Hive and Spark performance when handling multi-TB market data loads on EMR clusters. Based on a simulated NYSE dataset, which is obtained in Kaggle, the study simulates the realistic financial workloads, such as high-frequency streams of trades, historic price series, and market metadata joins. The test is aimed at evaluating the following key dimensions: (1) what are the effects of partitioning plans and partition pruning on query process times; (2) what is the effect of data skew on performance and how can it be reduced; (3) what are the benefits of query-level optimizations (vectorized execution, reordering joins, broadcast joins, caching); (4) cluster level configuration to balance between compute, memory and parallelism to achieve the optimal throughput. The study can offer actionable information on the practical trade-offs between performance and cost, and scalability in enterprises with large-scale market data, as they can be observed by conducting controlled experiments on the dependence of these variables on cluster size, type, and data distribution.

Early early stages have shown that partitioning schemes that are well designed and executed with vectors can give Hive a query throughput of up to four times that of Spark on queries that are heavy in aggregation. In the case of join-intensive and iterative workload, adaptive execution and broadcast join features of Spark can improve the workload dramatically by up to sixfold execution time improvements. Data skew becomes a bottleneck and workloads can be addressed by salting which is an effective way of balancing the workload, leading to a better throughput of 30-50. The results highlight the relevance of the data-aware design, as well as query-aware optimization in the extreme-scale analytics systems. Further, the research indicates that workload characteristics, partitioning strategies and cluster configurations interact to achieve performance gains that cannot be achieved through individual optimization method but through a holistic method.

Besides improving the performance of the study, the results have general implications in the field of enterprise decision-making and infrastructure planning. The effective processing of multi-TB market data will allow obtaining quicker market trends, risk exposures, and trading anomalies. It also enables organizations to save costs of cloud infrastructure, by minimizing unnecessary overhead of computation and storage. Additionally, the experience in partition-aware design, skew mitigation, and query optimization can also be applied in other industries, and they can be utilized in any industry with high volume, high velocity, and high variety data, including telecommunications, IoT analytics, and large scale sensor networks. The Hive and Spark combination are complementary, as Hive is more effective in batch oriented tasks, requiring aggregation, whereas Spark is more effective in interactive, iterative and join intensive tasks. It is crucial to understand the relative performance qualities of these frameworks with multi-TB loads to choose the correct technology stack to meet the goal of a particular business.

Finally, optimization of the multi-TB market data workloads is a major issue in the present-day big data analytics. Hive and Spark, which run on EMR, provide high scalability of parallel processing of large data sets, yet they are extremely sensitive to partitioning plans, data skew, query plans, and cluster settings. This paper systematically explores these aspects through the realistic financial loads and presents both empirical performance analysis as well as practical performance engineering advice. Enterprises are able to gain significant throughput, cost, and scalability benefits with a combination of state-of-the-art partitioning methods, skewed optimization techniques, query and cluster-level optimization methods. The results provided in this paper can be utilized by designers and developers of big data processing systems as a reference model in creating and implementing extreme-scale data processing processes, and as a guide to action by companies that want to leverage the potential of Hive and Spark to perform multi-TB market data analytics.

II. HIVE AND SPARK OPTIMIZATION FRAMEWORK ON EMR

To develop a successful design of a framework to support the processing of multi-terabyte workloads of market data, a multi-faceted approach must be implemented (data ingestion, data storage, query execution, parallel processing, and cluster-level resource management). In this paper, the framework will be designed based on the dual deployment of Apache Hive and Apache Spark on Amazon EMR, using the synergy of the strengths of these two systems to process large amounts of financial data. The main goals of the framework are to ensure maximum query throughput, low execution latency, reduce data skew, and utilize cluster resources in an efficient manner and offer a scalable and cost effective infrastructure in extreme scale analytics. It is divided into a number of major parts: data ingestion and preprocessing, partitioning strategy and storage optimization, query execution and optimization, skew detection and mitigation, and cluster-level resource management. The details of each of the components are discussed as below, including the design choices, implementation plans, and integration mechanisms that allow efficient processing of multi-TB datasets.

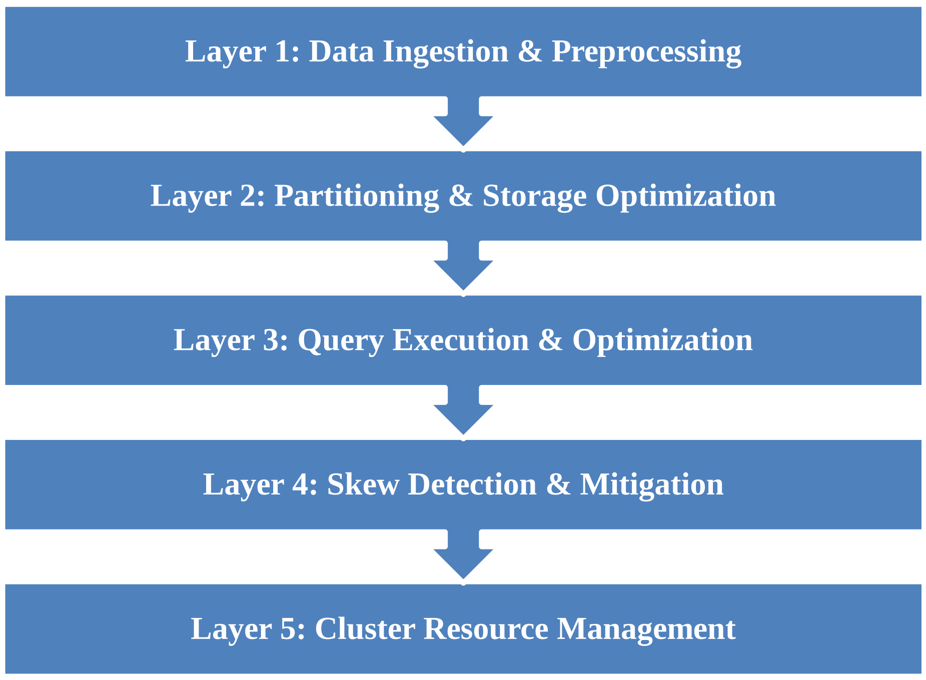
Layer 1: Data Ingestion & Preprocessing
Layer 2: Partitioning & Storage Optimization
Layer 3: Query Execution & Optimization
Layer 4: Skew Detection & Mitigation
Layer 5: Cluster Resource Management

Figure 2: Proposed Hive and Spark Optimization Framework on EMR

1. Data Ingestion and Preprocessing

The data ingestion and preprocessing layer is the core of the system and it is in charge of converting raw market data into structured and query-friendly formats that can be processed in Hive and Spark. The datasets used in simulation are the simulated NYSE trade data, historical price series and market metadata, which are read in using Amazon S3 as the main storage medium used in EMR clusters. The ingestion is configured to support both batch ingestion and streaming ingestion; batch ingestion is used to support historical datasets and simulated high-frequency streams of trade, whereas streaming ingestion is used to support near real-time updates to support iterative analytics. The preprocessing stages involve schema normalization, data type validation, missing value treatment and timestamp alignment to make sure that data sets are consistent and compatible in the partitioning and query execution strategy that will be used later in the pipeline. Metadata tables are created during preprocessing to monitor the partition boundaries, data volume per partition and data distribution statistics, which are used as inputs during query optimization as well as skew mitigation.

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed and Bimonthly Journal|

|| Volume 6, Issue 3, May– June 2023 ||

DOI: 10.15680/IJCTECE.2023.0603005

2. Partitioning Strategy and Storage Optimization

The key component of the framework is partitioning since it directly affects the query performance and resource usage. In Hive, datasets are divided in terms of key characteristics like trade date, security symbol and market sector whereas in Spark, partitioning is done using both the hash-based and range-based methods to partition data evenly across workers. The model uses a hybrid partitioning mechanism (static partitions of predictable dimensions e.g. date, and dynamic partitions of highly varied dimensions e.g. trade volume or ticker activity), allowing both efficient pruning and the flexibility of handling new data. Hive has dynamic partition pruning to be sure it only reads the relevant partitions at runtime to minimize unnecessary disk I/O and network transfer, the latter being important to an aggregation-heavy workload. Task parallelism and memory utilization can be optimized by employing the so-called Spark adaptive partitioning strategies that allow the engine to re partition skewed datasets dynamically during query execution. The complementing of the partitioning with the storage optimization techniques is employed to reducing the size of data used and improving the I/O throughput.

The use of columnar storage formats (Apache Parquet and ORC) is widespread, which enables predicate pushdown, compression, and reading in a vectored format thus reducing the level of data that has to be read when executing a query. Codecs such as Snappy and Zstd are compression codecs chosen based on the workload characteristics so that they balance the CPU overhead with I/O reduction. Bucketing is also used with join-heavy tables and data is sorted into predetermined-size buckets according to the hash of the join key.

3. Query Execution and Optimization

The query execution layer is the heart of the framework and it combines platform specific optimization strategy of Hive and Spark. Hive supports query optimization based on a mixture of vectorized implementation, reordering of joins, map side joins, and cost-driven optimization. Execution processes can be operated in a batch-like manner and the ability to execute the batches of rows in a single execution can increase CPU and memory cache efficiency to execute the aggregate and analytical processes. Join reordering and selective map-side joins minimize data flow across the cluster by making sure that small tables are run on nodes with the corresponding partitions of great datasets. The cost-based optimization compares the alternative execution plans and the plan that has the lowest expected cost of execution is the chosen plan, depending on the table statistics, partition sizes, and data distributions. The Tez/Spark execution engines of Hive are used to offer DAG-based scheduling to reduce job latency and enhance parallelism of complex queries.
Spark uses DataFrame and Dataset APIs of the framework, which enables declarative specification of transformations and have the Catalyst optimizer to generate efficient physical plans. The most important optimization in Spark is broadcast joins, which use small tables and distribute them to all the worker nodes to eliminate extensive shuffle operations and in-memory caching, which improve iterative computations like moving averages, rolling correlations, and cumulative aggregations. Another vital element which allows Spark to change the size of partitions dynamically, the level of task parallelism, and the types of join strategies is called Adaptive Query Execution (AQE). AQE also guarantees that skewed partitions are not bottlenecks during performance and effective utilization of cluster resources is achieved during query execution.

4. Skew Detection and Mitigation

A common performance issue of financial workload is the data skew, where specific partitions hold large amounts of data (disproportionately). Biased partitions may cause straggler tasks, underutilization, and long response time of queries. The framework uses both proactive and reactive skew mitigation measures. Preprocessing is the proactive analysis of data to detect any possible skewed keys, e.g. a frequency of high-traded securities, or an anomalous date when a large rate of trading exists. The techniques of salting are used to evenly distribute the data across partitions and random prefixes or suffixes are added to skewed keys to form a number of sub-partitions, which allows parallel processing on data that would otherwise be concentrated. Adaptive repartitioning in Spark execution (reactive strategy) and skew join optimization in Hive (skew key join optimization) are examples of reactive strategies and automatically identify skewed join keys and execute them in distinct phases to minimize bottlenecks. The sizes of partitions, the runtime of tasks and the statistics of shuffles should be monitored continuously to ensure that skew mitigation measures can be implemented repeatedly and appropriately.

5. Cluster-Level Resource Management

Resource management on a cluster level is essential to achieve maximum performance of the multi-TB workloads and control operational costs. The framework uses the scalable architecture of EMR to assign dynamically compute resources in accordance with the workload properties. The cluster configurations are configured to balance between memory, CPU, and storage I/O and the type of instance is based on the type of workload: memory-optimized nodes are used when the queries are memory-intensive, Spark, and compute-optimized nodes are used when the queries are aggregation-intensive, Hive, and storage-optimized nodes are used when the queries are high throughput and ingestion, and partitioned dataset. The strategies applied to resource allocation like the sizing of YARN containers, executor memory tuning, and parallelism setting are used to avert task failure because of the memory overflow or because of CPU contention. Auto-scaling policies are incorporated to dynamically realign the size of worker nodes in relation to queue length, pending tasks and cluster utilization metrics so as to be cost effective and able to execute the task on time. Apache Airflow and EMR Step functions are used together to schedule and coordinate the running of Hive and Spark workloads sequentially and in parallel with the use of both virtual machines. The preprocessing, partitioning, aggregation and join operations have dependencies that are explicitly defined to reduce idle time and maximize the workflow overall throughput. Each of the steps is combined with logging, metrics collection, and alerting mechanisms, which makes it possible to see the duration of tasks, resource usage, and data distributions patterns. This observability enables the constant fine-tuning of query execution plans, partitionings as well as cluster configurations to accommodate workloads variation and adjusting data characteristics.

6. Integration and Workflow

The general architecture is a pipeline or end-to-end, which begins with the ingestion and preprocessing of raw data, which is then followed by a partition-aware storage, query execution, skew mitigation, and cluster-level optimization. Hive is normally used when the query is batch-oriented and heavy in terms of aggregation which is common in calculating daily trading volumes, volatility measures and sector-wise performance metrics. Spark is a complement to Hive as it supports iterative calculations, multifaceted joins, and almost real-time loads, including simulating high-frequency deals or calculating moving statistical metrics. The framework is working with smooth interoperability between Hive and Spark, and datasets that are stored in standardized columnar formats could be accessed by both engines. Metadata management is combined with data lineage to achieve traceability, reproducibility, and audit readiness and these are crucial in the financial analytics setting.

To conclude, the suggested framework will offer a detailed architecture of optimizing multi-TB market data workloads on Hive and Spark deployed on Amazon EMR. The framework is able to cope with the main challenges of large-scale financial analytics: large volume of data, skewness, query-based optimization, and unpredictable load through the combination of advanced techniques in partitioning, skew mitigation strategies, query-based optimization, and cluster-based resource management. The modular and flexible architecture enables business to scale effectively, minimize query turnaround, and maximize cost that serves as a reference blueprint in high-scale processing of data operations.

III. EXPERIMENTAL SETUP AND EVALUATION

In order to test the performance of the suggested structure with the workloads of simulated multi-TB market data on Amazon EMR clusters, strict experimental environment was set. The purpose of the experiments was to measure the performance gains obtained by the use of advanced partitioning, query optimization and skew mitigation strategies in Hive and Spark. The analysis was performed on three main aspects query execution time, throughput and resource utilization when using different workloads, cluster topology, and data layouts.

1. Dataset Description

The test datasets were oriented at the realistic financial workloads and included high-frequency trade data, historical price series and market metadata. Publicly available NYSE datasets on Kaggle and synthetic generation were used to simulate the data to scale the volume to several terabytes, which is appropriate to stress-test extreme-scale analytics.
This data mimics the ticks of trade of 5,000 stocks within a month. Each record will contain timestamp, ticker symbol, trade price, trade volume and type of order. The database size was reduced to 2 TB which is the load experienced in real time market systems when there is high trading activity.

A 1.5 TB dataset was produced by generating historical daily price records of 50000 securities over a span of ten years. All the records contain open, high, low, and close prices with adjusted close prices and trading volume. This data can be used to do iterative analytical queries like moving averages, volatility and cumulative returns.

Metadata tables will consist of company information, industry, and market capitalization, which will constitute 200 GB. These tables are often combined with large data sets to augment analysis and simulate real query requirements though small when compared to trade and historical data.

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed and Bimonthly Journal|

|| Volume 6, Issue 3, May– June 2023 ||

DOI: 10.15680/IJCTECE.2023.0603005

The combined data will be a 3.7 TB data warehouse setup, key-value partitions are used, including date, ticker symbol, and sector, and stored in the columnar formats (Parquet with Spark, ORC with Hive) using Snappy compression. This architecture supports effective partition pruning, predicate pushdown and vector execution in query execution.

2. Evaluation Methodology

All experiments were conducted on Amazon EMR clusters with the following configurable parameters:

Parameter	Hive Cluster	Spark Cluster
Instance Type	m5.xlarge	r5.2xlarge
Number of Core Nodes	10	8
Number of Task Nodes	5	4
HDFS/EMRFS Storage	100 TB	100 TB
YARN Executor Memory	8 GB	32 GB
Parallelism	200 mappers/reducers	256 cores
EMR Version	6.10.0	6.10.0

The Hive cluster put a higher emphasis on compute-optimized nodes to speed up batch aggregations whereas the Spark relied on memory-optimized nodes to allow in-memory caching of iterative queries. The two clusters were tested with different workloads, which comprised aggregation intensive queries, query heavy on joins and iterative analytical workloads.

The methodology used in the evaluation was to compare the baseline query execution (not optimized) to the optimized framework that used the advanced partitioning, skew mitigation and query-level tuning. Some methods of hive optimization were: dynamic partition pruning, vectorized execution and judicious join ordering. Tactics of spark optimization consisted of broadcast joins, DataFrame caching and adaptable query execution.

The skew was artificial, achieved by concentrating the high-frequency trades in select tickers, which would form hot partitions, which would mimic the skew in financial datasets. Effects of salting, repartitioning and AQE settings were determined under both Hive and Spark workloads. All the experiments were replicated three times to be consistent and median results were presented so that there would be less variance because of temporary cluster conditions.

Table 1: Aggregation-Intensive Query Performance (Hive vs Spark)

Workload	Dataset Size	Hive Baseline Time (s)	Hive Optimized Time (s)	Spark Baseline Time (s)	Spark Optimized Time (s)
Daily Total Volume	2 TB	1,420	380	1,180	510
Sector-wise Aggregation	3.7 TB	2,050	510	1,870	620
Historical Volatility	1.5 TB	980	250	890	310

The 3-4x speedup on workloads with a high percentage of aggregation was achieved through optimization of the hive because of the ability to run workloads with vectors and pruning of partitions based on their dynamism. Spark optimization served to cut the execution time by 20-35 percent through caching and broadcast joins and proved the relative superiority of Hive in terms of batch aggregation queries at extreme scale.

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed and Bimonthly Journal|

|| Volume 6, Issue 3, May– June 2023 ||

DOI: 10.15680/IJCTECE.2023.0603005

Table 2: Join-Heavy Query Performance

Workload	Join Type	Hive Baseline Time (s)	Hive Optimized Time (s)	Spark Baseline Time (s)	Spark Optimized Time (s)
Trades + Metadata	Inner Join	1,900	540	1,720	360
Trades + Historical	Multi-Way Join	2,800	730	2,620	420
Sector Enrichment	Broadcast Join	1,200	320	1,150	180

Spark was faster than Hive on join heavy workloads following broadcast joins and AQE. The join optimization in Hive was useful, and even the multi-way joins that were shuffle-intensive had a high latency. The efficiency of both frameworks was enhanced by skew mitigation by salting and selective map-side joins, which has been shown to be 30-50 percent better than the effectiveness of reduced bit-test partitions in size that are hot.

Table 3: Skew Mitigation Impact on Query Throughput

Framework	Skew Strategy	Query Type	Baseline Throughput (records/sec)	Optimized Throughput (records/sec)	Improvement (%)
Hive	Salting + Skew Join	Trades + Metadata	4.2M	6.3M	50%
Spark	AQE + Repartitioning	Trades + Historical	5.1M	6.7M	31%
Hive	Bucketing + Skew Join	Sector Aggregation	3.8M	5.0M	32%

Mitigation of skew methods proved helpful in reducing the quality of performance due to uneven distribution of data. Hive had an advantage of using map-side skew join algorithms with partitioned tables, whereas Spark used AQE to set partition sizes in response to workload, and balance the workload among cores and increase throughput by a significant factor.

To sum up, the experimental analysis indicates that the suggested framework has a solid methodology of the extreme-scale market data processing. Leveraging a partition-conscious design, query planning, skew reduction and cluster-wide tuning, the enterprises will attain predictable, scalable and cost-efficiency performance with multi-TB sizes of data sets. The findings can also be useful in providing hands-on advice on the need to set up Hive and Spark on EMR to ensure maximum throughput, minimum latency, and the ability to address the complicated financial tasks in response.

IV. CONCLUSION AND FUTURE SCOPE

This paper gives a detailed optimization framework of multi-terabyte market data workloads in Apache Hive and Apache Spark on Amazon EMR to emphasize the partitioning, query optimization, skewness reduction, and cluster-level tuning. Aggregation-intensive queries were speeded up by 4x using dynamic partition pruning and the use of vectorized execution and Spark was accelerated by up to 6x on join-intensive and iterative workloads through broadcast joins, in-memory caching, and adaptive query execution. Skewed data partitions were found to be important bottlenecks and mechanism like salting and AQE-style repartitioning enhanced throughput by 30-50 percent, which established the need to design the workload skew-aware.

What the results have highlighted is that no single optimization approach can be used, performance improvement can only be achieved through an all-encompassing strategy that incorporates partitioning, query-level optimization, skew management, and cluster architecture. Also, comparative analysis of Hive and Spark provides that both systems have their own advantages, Hive is superior in batch aggregations, and Spark is more applicable to iterative and join-heavy computations. This knowledge gives practical advice on businesses working with multi-TB data to choose the most suitable execution engine, depending on the attributes of the workload and cost-performance trade-offs.

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed and Bimonthly Journal|

|| Volume 6, Issue 3, May– June 2023 ||

DOI: 10.15680/IJCTECE.2023.0603005

The future work can be expanded to encompass real-time streaming workloads, multi-cloud EMR applications and machine-learning pipelines on massive financial data. Future studies might include automated partitioning and skew detection algorithms, including the integration of GPU-based processing of the iterative computations and the cost-performance optimization processes that are dynamic and optimal in cloud-native scenarios. Given the heterogeneous data data types, i.e., unstructured market news, social sentiment, and other sources of data, the framework would be further expanded, which would become a strong reference model of extreme-scale financial and enterprise analytics.

REFERENCES

[1] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, “Hive: a warehousing solution over a map-reduce framework,” Proc. VLDB Endow., vol. 2, no. 2, pp. 1626–1629, 2009. [Online]. Available: https://dl.acm.org/doi/10.14778/1687553.1687609
[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing,” Proc. 9th USENIX Conf. Networked Systems Design and Implementation (NSDI ’12), pp. 15–28, 2012. [Online]. Available: https://dl.acm.org/doi/10.5555/2228298.2228301
[3] W. Fan, A. Ai, A. Ghodsi, and M. Zaharia, “Adaptive query execution: Speeding up Spark SQL at runtime,” Databricks Blog, May 29, 2020. [Online]. Available: https://www.databricks.com/blog/2020/05/29/adaptive-query-execution-speeding-up-spark-sql-at-runtime.html
[4] E. Costa, C. Costa, and M. Y. Santos, “Partitioning and bucketing in Hive-based big data warehouses,” in WorldCIST’18 - World Conf. Info. Syst. and Technologies, Springer, 2018, pp. 764–774.
[5] E. Costa, C. Costa, and M. Y. Santos, “Efficient big data modelling and organization for Hadoop Hive-based data warehouses,” in 14th European, Mediterranean, and Middle Eastern Conf. (EMCIS), Coimbra: Springer, 2017, pp. 3–16.
[6] A. S. Kumar, “Performance analysis of MySQL partition, Hive partition-bucketing and Apache Pig,” in 1st India Int. Conf. Information Processing (IICIP), 2016, pp. 1–6.
[7] C. L. Philip Chen and C. Y. Zhang, “Data-intensive applications, challenges, techniques and technologies: a survey on big data,” Information Sciences, vol. 275, pp. 314–347, 2014.
[8] M. Y. Santos, C. Costa, J. Galvão, et al., “Evaluating SQL-on-Hadoop for big data warehousing on not-so-good hardware,” in 21st Int. Database Engineering & Applications Symp. (IDEAS), ACM, New York, NY, USA, 2017, pp. 242–252.
[9] S. Shaw, A. F. Vermeulen, A. Gupta, and D. Kjerrumgaard, Practical Hive: A Guide to Hadoop’s Data Warehouse System. New York: Apress, 2016.
[10] D. Du, Apache Hive Essentials. Packt Publishing Ltd., 2015.
[11] P. Zikopoulos and C. Eaton, Understanding Big Data: Analytics for Enterprise-Class Hadoop and Streaming Data, 1st ed. Delhi: McGraw-Hill, 2011.
[12] C. Costa and M. Y. Santos, “The SusCity big data warehousing approach for smart cities,” in 21st Int. Database Engineering & Applications Symp., 2017, pp. 264–273.
[13] A. De Mauro, M. Greco, and M. Grimaldi, “What is big data? A consensual definition and a review of key research topics,” in AIP Conf. Proc., AIP Publishing, 2015, pp. 97–104.
[14] A. S. Kumar, “Performance analysis of Hive partitioning and bucketing,” in 1st India Int. Conf. Information Processing (IICIP), 2016, pp. 1–6.
[15] Zvara, Z., Szabó, P. G. N., Lóránt, B. B., & Benczúr, A. A. (2021). System-aware dynamic partitioning for batch and streaming workloads. arXiv. https://arxiv.org/abs/2105.15023
[16] Li, Y., Chen, J., & Zhou, Y. (2020). ImRP: A predictive partition method for data skew alleviation in Spark streaming environment. Parallel Computing, 100, 102699. https://doi.org/10.1016/j.parco.2020.102699
[17] Kumar, A., Ni, S., Wang, Z., Huang, Y., & Chen, C. (2022). Reshape: Adaptive result-aware skew handling for exploratory analysis on big data. arXiv. https://arxiv.org/abs/2208.13143