Optimizing Spark History Server for Large-Scale ML Workloads in Financial Systems

Prasanth Sasidharan

College of Engineering Trivandrum, Kerala

***Abstract*— Even basic procedures of a financial apparatus like fraud detection, credit scoring, and risk modelling utilizing machine learning (ML) processes are extremely complex, and the versatility of data infrastructure is put in the limelight. The Apache Spark platform of big data is now fashionable for executing big ML loads, owing to the fact that processing has been made in-memory and the ecosystem has been extended. Nevertheless, the performance and observability of every component—which in most cases is a bottleneck—is the Spark History Server (SHS), which is incidentally the component involved in post-execution diagnosis and monitoring. High-frequency financial settings may lead to low SHS performance and difficulty in the speed of troubleshooting, auditability, and understandability of the systems.**

The critical analysis of the ways through which the Spark History Server can be optimized with references to large-scale operations in financial systems connected to ML is devoted to the review. The paper is based on the literature available in the sphere of distributed computing, cloud infrastructure, hybrid data processing pipelines, and real-time analytics, and proposes architectural, operational, and performance-based solutions to SHS improvement. Among the most important ones are log compaction, memory management, streaming integration, and the importance of resilient cloud environments as tools to scale SHS. The review and recommendations of how this can be improved in the future also cover future limitations to implementations of SHS. Financial institutions will have an opportunity to increase the speed of their working activity, address new requirements, and provide the further delivery of ML-based insights to the further improvement of SHS functionality.

***Index Terms*— Spark History Server, Machine Learning, Financial Systems, Log Optimization**

1. Introduction

The giant machine learning (ML) training of existing financial systems has become an inseparable component of such delicate operations as fraud identification, algorithmic trading, risk management, and credit rating. Distributed computing structures like Apache Spark are still depended on by financial institutions as they need increased capacity of data infrastructure to accommodate the increasing volume of structured and unstructured data. In-memory computation in Spark specifically is applicable in calculations of an infinite number of iterations in ML loads. Nevertheless, among the features that cannot be disregarded when performing performance tuning, Spark History Server (SHS) is one of the most important elements of job diagnostics, monitoring, and debugging.

The SHS is of paramount importance in financial settings in which high throughput and low latency operations are mandatory and operational transparency must be ensured.

The paper consists of a critical analysis of the means and technologies to scale out the Spark History Server for mass-scale ML loads in financial systems. The prospects of Spark job logs and memory usage, I/O bottlenecks, and UI responsiveness management are to be examined within the framework of the specified review, in accordance with recent findings in cloud computing, distributed systems, and data engineering. In addition to that, it relates these optimizations with resilient cloud architecture, event-driven systems, and hybrid data processing pipelines, all of which are integral to financial technology infrastructures.

2. The Need for Optimizing SHS in Financial ML Pipelines

The financial institution is under high demand in its employment that requires the performance and compliance of the financial institution. With the help of the retrospective knowledge that the Spark History Server can offer, one can ensure tracing, reproducibility, and compliance auditing of ML workflows. Nevertheless, the SHS may be slow, non-responsive, or resource-intensive due to the volume of job history logs over time, especially in a high-frequency trading environment or in a fraud detection system which is activated in real-time. These bottlenecks may cause inefficiency in diagnostics, ineffective job tuning, and as a result, service breaks in the production level of ML [1].

Banking loads of financial information (volume change, speed, and complexity) may vary; this requires a strong logging system. The SHS, however, stores its information in event logs that might be too large, considering the task-level indicators that are huge. It is observed that both UI serving and metadata serving are affected by accrued overhead when there are thousands of running Spark jobs [1].

In addition, the SHS cannot distribute processing power like Spark; it needs one node and, therefore, cannot scale its performance across multiple points. The rate of exponential growth and size of job submissions is larger with the complexity of ML workflows that include multiple feature engineering steps, model selection, and hyperparameter tuning. Without SHS optimization, engineers and data scientists will have to dedicate more time to accessing past job data required for model validation and performance benchmarking [2].

3. Challenges in SHS Optimization for ML Workloads

The policy and size of retention of event logs are also among the main issues. The traditional Spark setup retains Spark logs indefinitely, which ultimately increases the storage and loading time of SHS. These issues could be partially addressed through ephemeral storage reduction or log rotation policies, although they must be balanced with audit and compliance policies typical of the financial industry [1]. There is also another cure that has been proposed, which is the introduction of scalable log-compaction as well as incremental loading operations—not to make the visibility at scale not visible, but to make it not get out of control.

The second issue is that SHS does not particularly achieve high efficiency in its memory usage in the SparkUI rendition of huge tasks. It is especially problematic in the cases of ML when the job DAGs (Directed Acyclic Graphs) are multi-staged and nested because of computation duplications. All these complications add to the increased time needed to generate and parse the JSON in the SHS web interface [3].

Moreover, the default configurations of Spark cannot handle bulky ML systems with high throughput. Parameters associated with SHS, such as spark.history.fs.cleaner.enabled, spark.history.fs.cleaner.interval, and spark.history.fs.logDirectory, must be tuned based on the magnitude and frequency of job submissions. The general history server has been identified as a bottleneck, especially when actively accessed by multiple users simultaneously [4].

Financial structures are also not secure by default because sensitive transaction or client information might appear in job logs. Optimization measures would include encryption and secure storage of log files, ideally under role-based access controls to restrict access to specific jobs or user levels [4].

4. Architectural Considerations and Integration Strategies

Optimization of SHS cannot be a solitary event. Rather, it should be tied to broader ideas of cloud-based ML architecture. An example of this is hybrid ML pipelines, i.e., pipelines with both batch and real-time components, which subject the Spark ecosystem to variable loads. An alternative that can substitute SHS in such cases is to store past logs in cold storage such as Amazon S3 or Google Cloud Storage, and store metadata indexes in memory to enable queries within seconds [2].

In addition, systems should be launched alongside log aggregation, where Spark is configured with distributed storage systems such as Hadoop Distributed File System (HDFS) or object stores. In cases where logging is done in a remote storage system, network throughput and latency directly affect the loading speed of SHS logs. One way to minimize the time taken to access job history files—particularly for long-running ML jobs—is to cache at least one data grid of job modeling information [3].

Containerization and orchestration approaches may also be used for SHS architectural improvements. SHS containers integrated into Kubernetes pods can have resource limits, and autoscaling can be applied to ensure effective memory management and fault tolerance. This is particularly effective in multi-tenant environments, which are common in large financial institutions where different teams operate different workloads under SHS [5].

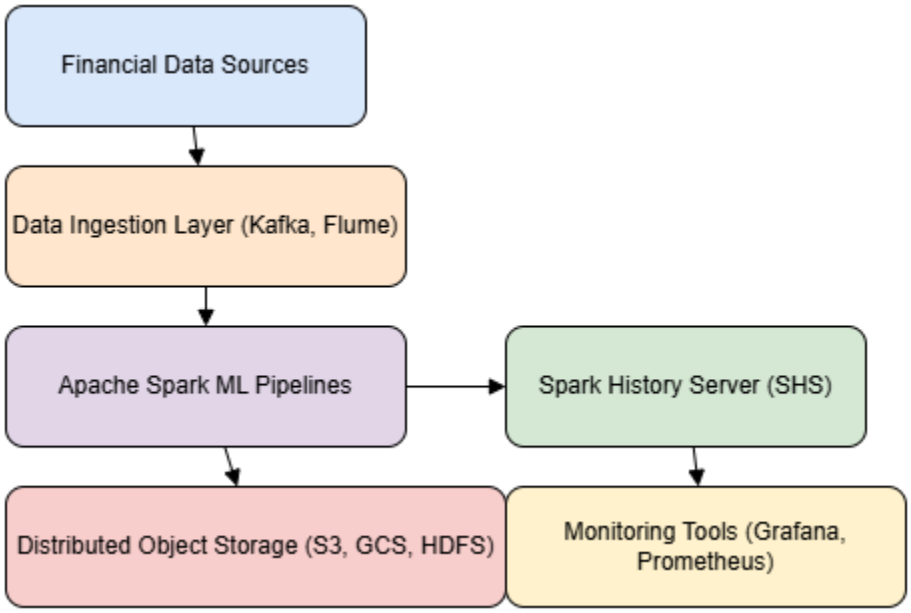

Figure 1: A layered architecture showing Spark ML pipeline integration with SHS and distributed storage in a financial system.

Figure 1: Optimized Spark-based architecture for large-scale ML in financial environments [5] [8].

Spark in this layered architecture uses ML Pipelines that contain stages of data ingestion, feature engineering, model training, and system evaluation. SHS is associated with the observability layer, which involves drawing job logs from distributed storage using data lake infrastructure. To scale up: performance tuning parameters can be used. To reduce latency: containerized deployment can be applied.

5. Techniques for SHS Memory and Log Optimization

Processes that can reduce memory load and improve log management in SHS include log compression, log truncation with the assistance of JSON, and event-based sampling. Even disk I/O time can often be optimized by using language-independent log encoding (e.g., LZ4 or ZSTD), especially when handling large Spark jobs with terabytes of input data [2]. A trade-off in compression exists, however, between disk space savings and CPU consumption, which must be adjusted based on cluster requirements.

Another potential remedy is JSON truncation or selective ingestion of logs, in which only task summary metrics are retained while execution traces are omitted. This greatly reduces the cost of parsing and significantly accelerates Spark UI rendering without affecting diagnostic value, making it highly cost-effective for rendering Spark UIs. It is an excellent option for financial ML tasks that are executed in predictable conditions, as they are often repeatable and only require delta-based monitoring [3].

In addition, data scientists frequently perform tasks with only slight variations, such as hyperparameter tuning or data preprocessing. Algorithms such as log deduplication or metadata indexing (e.g., Spark EventLog Indexer) can be applied to minimize redundancy and improve UI performance. For memory-bound SHS instances, JVM heap and garbage collection optimization may be necessary, for example using the parameter -XX:+UseG1GC and a controlled heap size [6].

Table 1: Comparative overview of SHS optimization strategies in financial ML workloads

Optimization Strategy	**Performance Benefit**	**Trade-Offs**	**Applicability in Financial Systems**
Log Compression (LZ4/ZSTD)	Reduces storage & I/O latency	Higher CPU usage during decompression	High-throughput environments
JSON Truncation	Faster UI rendering	Partial visibility into job tasks	Repetitive job structures
Metadata Indexing	Faster metadata retrieval	Additional storage overhead	Large-scale parallel ML workloads
Cold Storage Offloading	Reduced on-premise storage costs	Increased access latency	Long-term audit/compliance scenarios
JVM Heap Tuning	Improved memory efficiency	Requires monitoring and tuning effort	Critical for large DAG visualization

6. Real-Time SHS Adaptations for Stream-Driven ML

Spark streaming jobs have also received more consideration in real time because event-driven architecture has been adopted in financial systems, specifically to aid in the development of fraud detection and customer analytics behavior. Such employment consists of close to real-time control and feedback loops. The former SHS was not designed to handle real-time loads, hence its inability to meet the performance requirements of streaming pipelines [6].

SHS event-driven adaptations will include SHS metrics sinks such as Prometheus or Grafana, where major Spark metrics are emitted intentionally, and not queried randomly. This not only reduces pressure on SHS but also allows real-time visualization. It is a better hybrid monitoring mechanism with respect to performance and reliability in financial applications where anomalies in transactions must be identified as soon as possible [6].

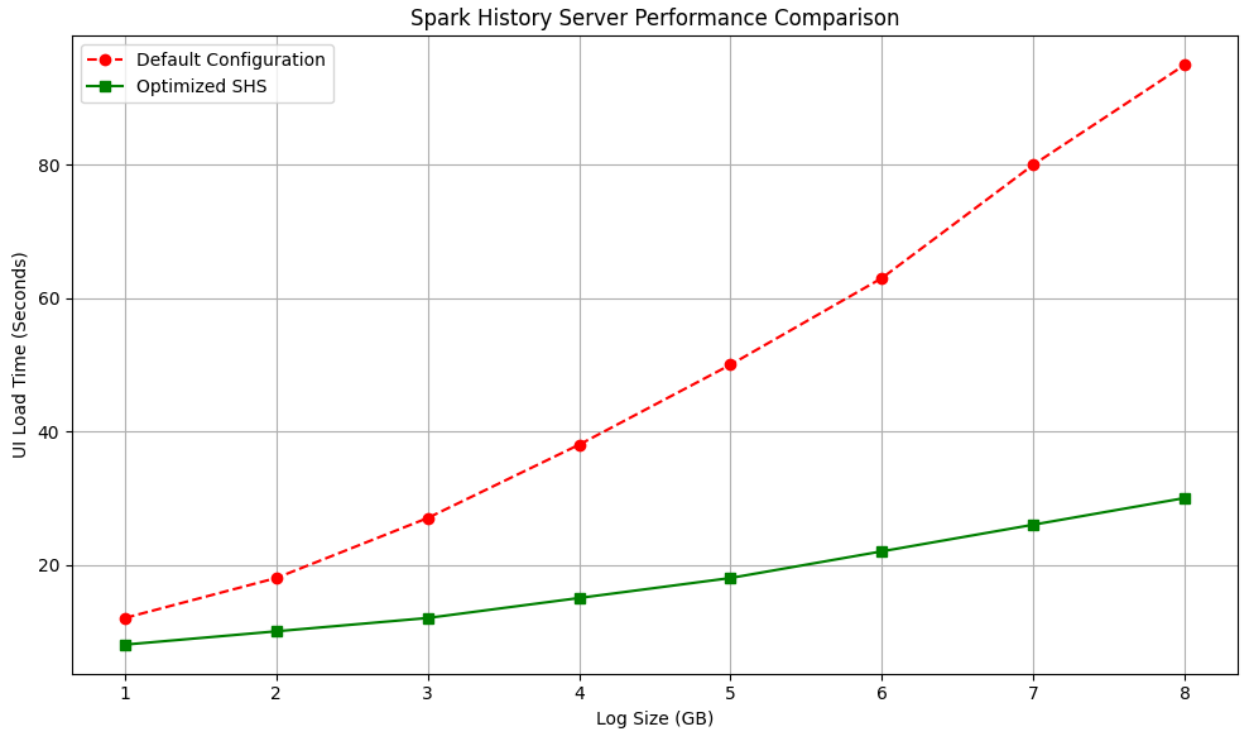

Figure 2: Performance comparison graph of SHS with and without log compression in high-volume ML jobs.

Figure 2: Benchmark results comparing SHS performance with and without optimization in terms of UI load time [2], [3] [6].

The performance graph in Figure 2 demonstrates the substantial improvement in UI responsiveness and log parsing speed when employing log compression and JSON truncation. The y-axis represents the time (in seconds) required to load job UIs, while the x-axis shows different log sizes in gigabytes. The aggressive settings do not get any worse than the default settings even beyond the size of logs of 5 GB.

7. SHS Performance in Hybrid ML Pipelines

Hybrid ML pipelines (combination of real-time analytics and batch processing) are gaining popularity in financial systems that require both quick decision-making and long-term historical analysis. The Spark History Server is a relevant component of post-mortem machine learning drift investigation, feature drift detection, and resource consumption tracking in such systems. Nonetheless, many issues arise when making SHS production-ready in these hybrid environments, especially with mixed job durations, and intermingled streaming and batch processes [7].

Multi-level logging and multi-tiered SHS deployments under hybrid conditions are necessary for optimization. Excessive output from short-lived jobs can result in many small logs, while batch jobs produce large logs due to intensive computations. Therefore, the segregation of batch and streaming workload pipelines is necessary to ensure stable SHS operations. In addition, adaptive log storage systems improve performance and cost-effectiveness by caching the most frequently accessed logs on SSDs while archiving older logs to object storage [8].

The Spark event logs can also be externalized and, potentially, they can be interrogated with the help of other systems, such as Apache Hive or Delta Lake, to access event logs more effectively regarding audit. These systems may also be applied in schema development and partitioning that is laudable in a large banking institution with a colossal ML audit trail [7]. This will enable data scientists and compliance officers to be in better control and have visibility of past job executions without affecting SHS performance and metadata catalog usability.

The existence of parallel SHS instances by business unit or job category is one of the optimization strategies. Individual SHSs can be given credit scoring, anti-money laundering, and fraud detection pipelines. This kind of segmentation will minimize the load the memory will bear on every instance of SHS, as well as avoid the performance of one unit influencing the performance of another unit [9].

8. Role of Resilient Cloud Architectures

The strong cloud architecture is arguably the reason behind the upgrading of the existing financial data processing systems. Apache Spark can be integrated with orchestration tools such as Kubernetes or Apache Mesos, and storage tools such as Amazon S3, Azure Blob storage, or data lakes based on Hadoop compatibility could be utilized. In that regard, SHS will have the responsibility to address a range of components, such as performance data and constraints [8].

Depending on various ways, even SHS optimization has advantages of resilient architectures. One, containerization provides resource isolation of SHS instances that can also be automatically scaled to demand. This scaling is especially required when workloads of ML are large, e.g., retraining or spiking a large scale of traffic. Second, the cloud-native logging and processing applications, such as Fluentd, Logstash, or AWS CloudWatch, can pre-process Spark logs and deliver them to SHS instances to minimize ingestion latency and unnecessary disk I/O [9].

It is also possible to checkpoint and trace lineage. Fault-tolerant architecture checkpointing and tracing lineage can also be done so that the job history is not lost to node failure or service failure. SHS metadata may be combined with checkpoints in a manner that recovery time may be minimized, and to make diagnosis as accurate as possible. This integration is required when high-stakes financial applications are involved, and the time of loss could cost the organization a fortune and when strength and observability are required [10].

Another important factor is security. Access control and cloud-native encryption protocols, including AWS IAM or Azure RBAC, could be implemented to reduce access to sensitive logs. Jurisdictional data handling regulations should also be complied with in SHS compliance audits of multi-regulatory jurisdictions like GDPR, PCI-DSS, or Basel III. This type of architecture will allow access to the secure and compliant logs without losing performance [10].

9. Streaming ETL and SHS Integration

Among the notable changes in the process of data pipeline optimization, it is possible to mention the substitution of the Extract-Transform-Load (ETL) process with real-time or streaming ETL. Real-time credit scoring, fraud detection, and sentiment analysis of social media feeds used in banking systems are done through streaming ETL. These streaming pipelines are on Spark Structured Streaming. Nevertheless, the combination of these workloads and SHS extends the novel performance factors [11].

SHS is forced to operate within a dynamic environment where it has to deal with a large number of temporary jobs. Regrettably, such logs may clog SHS, retard the UI, and stall job diagnostics. Such methods as the usage of low log granularity, which is gained by filtering logs based on streaming identifiers or aggregating them into daily roll-ups, may help. These optimizations give the system operators the view of the job frequency and latency in the system without slowing down with the micro-batch records [11].

In addition, third-party supervising infrastructures might be demanded to broadcast the Spark job measures to optimize streaming ETL and Spark jobs. Prometheus exporters can trace and reveal Spark measures in time-series forms. The Grafana type of such visualization tools can be implemented as a supplement to SHS and provide an operational overview (dashboards).

The other strength of SHS streaming ETL is that the tool can be used to analyze behavior over time as well as examine the performance of jobs. The financial institutions can be better placed to recognize the fusion of infrastructure and domain-specific effects due to the equalization of job measurements with business events, e.g., alterations of the market, or a major news release. This type of analytics process will increase operational efficiency and the use of data-driven behavior.

10. Limitations and Future Directions

Inasmuch as there is an abundance of optimization methods, it is not Spark History Server that is optimized to the maximum. Its one-thread log parsing and UI with its design is a highly limiting feature. These are exaggerated more in financial environments where ML pipelines create hundreds of job executions per day with complicated implementation plans.

The second tier of SHS optimization can be a distributed SHS architecture where log parsing and rendering can be done on multiple nodes. This would bring the problem of the difficulty of the coordination of states but might greatly decrease the latency of the UI in the presence of large logs. In addition, AI-generated semantic representation of job executions could make available a summary of logs to enable users to understand more easily the complex historical operations.

Another direction also in the future is the incorporation of more data lineage tools and metadata catalogs. The introduction of SHS logs into a larger observability system will enable companies with end-to-end data and ML pipeline visibility. It will also allow identifying anomalies in performance at an earlier stage and automatically tuning the system using machine learning.

Finally, the SHS interfaces can possibly be reformed to a more accessible format upon the emergence of no-code and low-code data science interfaces. These interfaces still contain vital information but will be abstracted from the Spark internals, and SHS would be beneficial to engineers, compliance officers, as well as business analysts.

11. Conclusion

Among such areas of the system that currently receive little attention is Spark History Server tuning, but it is a major contribution to the performance of the large-scale ML workloads that financial systems are making available. The next in the sequence of SHS optimization is the direction towards the efficiency and compliance of the operational processes as the complexity of the latter intensifies, and the frequency of the ML job augmentation speeds up, i.e., with the hybrid and real-time processing environment under consideration. Through the enactment of memory, cloud integration, and real-time monitoring tools, SHS can be made an effective part of the analysis to ensure it does not become a bottleneck.

Moreover, SHS will have to cope with these changes since financial institutions will gradually transform their systems into cloud and event-based architectures. Things such as distributed log parsing, AI-controlled summarization, and metadata integration will lead to SHS becoming scalable and intelligent and form the future of the system. In this way, the paper claims that optimization of Spark History Server will be capable of not only providing the financial sector with benefits but also helping organizations to make quicker, more accurate, audit-ready, as well as information-based decisions.

References

[1] Ramamoorthy, L. AI-Powered Infrastructure & Tools for Large-Scale Financial Systems: Challenges, Best Practices, and Standardization.
[2] Maria, R. Optimizing Cloud-Based Machine Learning Pipelines: A Hybrid Approach Using Big Data Processing and AI Models.
[3] Noor, S., Awan, H. H., Hashmi, A. S., Saeed, A., Khan, S., & AlQahtani, S. A. (2025). Optimizing performance of parallel computing platforms for large-scale genome data analysis. Computing, 107(3), 1-22.
[4] Ayyadurai, R., Parthasarathy, K., & Habib, M. (2025). The Optimizing Financial Data Transfers in the Cloud: A Comparative Analysis of Encryption and Machine Learning Algorithms: Financial Data Transfers in the Cloud Data. International Journal of Digital Innovation, Insight, and Information, 1(01), 01-13.
[5] Guntupalli, B. (2025). From SQL to Spark: My Journey into Big Data and Scalable Systems How I Debug Complex Issues in Large Codebases. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 6(1), 174-185.
[6] Kumar, R. Event-Driven Architectures for Real-Time Data Processing: A Deep Dive into System Design and Optimization. evolution, 7, 8.
[7] Sai, M. K. V. S. P. (2025). Integrating Machine Learning and Real-Time Analytics for Risk Management in Cloud-Based Insurance Platforms. Journal of Computer Science and Technology Studies, 7(6), 533-539.
[8] Bajwa, M. T. T., Rasool, A., Kiran, Z., & Latif, A. (2025). Resilient cloud architectures for optimized big data storage and real-time processing. International Journal of Advanced Computing & Emerging Technologies, 1(2), 54-68.
[9] Lopez, G. Real-Time Data Processing Using Apache Flink, Databricks, and Spark Integration.
[10] Shermy, R. P., & Saranya, N. (2025). Cloud-Based Big Data Architecture and Infrastructure. Resilient Community Microgrids, 131-188.
[11] Olaoye, G., Johnson, S., & Blessing, M. (2025). Batch to Real-Time: Leveraging AI for Streaming ETL Pipelines.