Journal of Advances in Developmental Research (IJAIDR)

E-ISSN: 0976-4844 • Website: www.ijaidr.com • Email: editor@ijaidr.com

Migrating Legacy ETL Pipelines to Distributed Spark Ecosystems: Challenges & Strategies

Pavan Kumar Mantha

Pavanmantha777@gmail.com

Abstract

With the aggressive increase in structured and semi-structured enterprise data, 2015-2019, the environment of large-scale data engineering had changed significantly. Conventional Extract-Transform-Load (ETL) systems, such as SAS batches, mainframe COBOL ETL programs, Informatica mappings, Ab Initio graphs, and Oracle PI, were initially designed with vertical scale systems with deterministic workloads and short concurrency constraints. As the volume of data in financial, retail and telecommunication sectors- and even government sectors grew to tens of terabytes per day, the monolithic nature of these ETL environments started assuming a structural performance bottleneck, schema dependence, and rapidly rising costs of operation based upon licensing models and special-purpose hardware. Simultaneously, the release of Apache Spark 2.x (2.2 through 2.4) created a new data processing paradigm that offered horizontally scalable computation on commodity clusters. In turn, businesses launched mass modernization programs that presupposed the migration of historical workloads of ETL into cloud-native Spark-based data platforms. This paper amounts to a comprehensive account of issues, tactics, and architecture changes relating to the migration of the legacy ETL pipelines into the distributed Spark architectures. The paper is centred on the 2018-2019 enterprise landscape and Hadoop 2.x-based YARN clusters, Hive Metastore, Kafka ingestion layers and S3/ADLS/GCS storage backends were at the heart of the new big data frameworks. The study finds such driving forces as cost optimization, scale requirements, single compute power, and the necessity to provide almost real-time analytics with Spark Structured Streaming. We discuss various serious migration issues: (a) competency shortages between legacy developers migrating SAS, COBOL, or PL/SQL into Scala, PySpark, and concepts of distributed processing; (b) code translation issues when moving deterministic ETL process logic into corresponding Spark DataFrame or Spark SQL transformations; (c) data fidelity in moving data between parallel systems; (d) performance scaling of clusters due to shuffle, skewness, and partitioning; (e) orchestration migration between Control-M, TWS, and Autosys and Airflow and It suggests a stepwise best-practice approach to migration, which includes workload evaluation, ingestion architecture normalization, library of reusable code, data quality framework, and performance optimization methods. A phased cutover model succeeded on the dual-run validation and golden dataset comparison is also presented in the paper. Moreover, several case studies show tangible performance: 8-hour SAS credit-risk batch can be reduced to 50 minutes with Spark; mainframe ETL can be replaced with real-time Kafka-Spark ingestion; multi-source customer service can be modernized to partitions of Parquet-based Spark SQL models. In general, the results show that Spark-based ecosystems can increase the level of scalability, decrease the costs of operations and enhance the level of reliability and allow supporting advanced machine-learning and real-time analytics applications. The work is an informative source to any organization taking on ETL modernization projects, and can add to the overall information on the practice of distributed data engineering in pre-2020 enterprise computing.

Keywords: Apache Spark 2.x, ETL Migration, Hadoop 2.x, SAS Modernization, Data Engineering, Structured Streaming, Airflow, YARN Optimization, Legacy ETL, Big Data Architecture.

1. INTRODUCTION

1.1 Background and Evolution of Enterprise ETL

By 2018, most businesses have experienced increasing constraints with the old ETL offerings which were originally designed as mainframe-heavy and batch-based data processing. [1-3] Conventional software and systems like SAS, COBOL batch applications, Informatica PowerCenter, Ab Initio, and Oracle PL/SQL pipelines were very vertical as the processing power had a direct relationship with proprietary and expensive hardware. Even though these platforms were strong and dependable, they lacked elasticity, which reduced them to the level of unsuitability to address the ever growing levels of volume of enterprise data, speed and diversity. Maintenance was expensive because of software licensing and hardware specifications and the arduous process of maintaining tightly bound ETL jobs which frequently contained complex business code that was hard to write down, debug and extend. The introduction to distributed data engineering infrastructure, especially to Apache Hadoop and Apache Spark, has radically changed the paradigm of vertical scalability to horizontal scalability. With the help of commodity hardware clusters, these structures provided the ability to handle workloads through multiple nodes simultaneously and with high throughput and resiliency. Spark, particularly Spark 2.4, had some performance improvements such as Catalyst optimizer which supported high-performance query planning and cost optimization, in-memory computations, and fault-tolerant Resilient Distributed Datasets (RDDs). It further established fully fledged connectors with cloud storage like Amazon S3, Azure Data Lake Store (ADLS), and Google Cloud storage (GCS), which makes it easy to connect with more current data lake systems. All these innovations made Spark a single compute engine, which also could support not only ETL workload but also SQL analytics, graph processing, streaming pipelines, and machine learning applications. Consequently, there has been a shift to the newer, and more scalable, flexible, cost-efficient, Spark-based ETL systems by organizations that have shed the yoke of legacy hardware-resistant systems, forming the basis of the modern data engineering practices in the enterprise.

1.2. Motivation for Migration

The process of transitioning to new distributed frameworks like Spark due to migration of the old ETL systems was caused by various strategic and operational reasons. These reasons indicate the changing needs of enterprise data management because organizations tried to minimize the expenses, enhance scalability, consolidate the computing power, and implement real-time analytics.

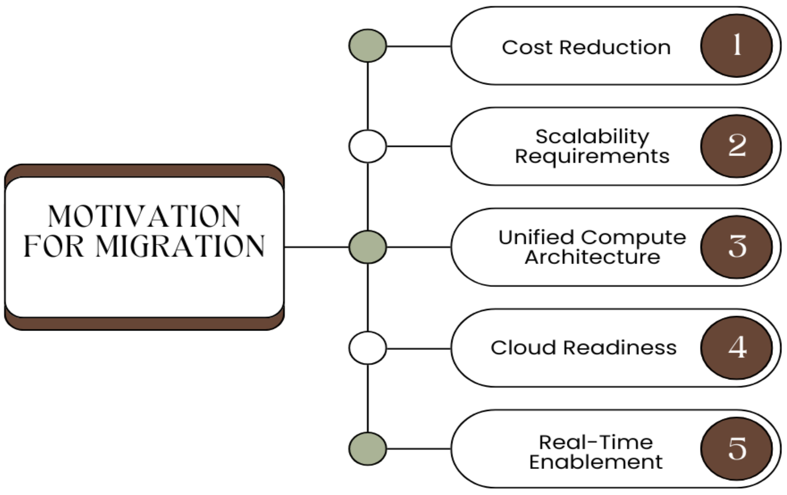

Figure 1: Motivation for Migration

- **Cost Reduction:** The reason why migration was so important was the high cost of operation in ETL environments that were based on legacy. Such systems as SAS/Grid, Informatica, and mainframe pipelines used to demand annual multimillion-dollar budgets to cover the proprietary software licenses, specialized hardware, as well as their continued maintenance contracts. Moreover, upgrading these systems to accommodate a larger quantity of data was costly and ineffective. Migrating to Spark was also associated with open-source licensing, and the capability to support commodity clusters or cloud-based infrastructure at a cost of developing software and hardware goods. Companies would be able to dedicate the money to analytics, data quality efforts or other new data-driven projects by removing the need to rely on expensive proprietary software.

- **Scalability Requirements:** The first generation ETL tools had been aligned to vertical expansion and could do nothing much with the increasing speed of enterprise volume of data gigabytes to terabytes and further. The growth of data volumes resulted in larger batch windows, bottlenecks, and slow reporting and reduced the opportunity of the organization to act on business requirements. The distributed in-memory processing and automatic parallelization with Spark enabled workloads to be processed in multiple nodes concurrently and used to process large datasets in an efficient manner. The ability to scale horizontally not only increased runtime but also provided scalability to enormously larger analytic demands without control licenses to have them run on costly hardware.
- **Unified Compute Architecture:** The other reason was the need to centralise different processing engines in one and the same platform. Spark allows support of several workloads, such as SQL-based analytics, streaming real-time, machine learning (MLlib), and graph statistics (GraphX), all in the same ecosystem. This not only simplifies the complexity of operations but also eliminates the maintenance of numerous special tools and simplifies the development and deployment as well as maintenance of data pipelines. Cohesive architecture also enhances ETL and analytics integration making enterprise data activities consistent and nimble.
- **Cloud Readiness:** Emerging data strategies are turning into cloud-based storage and compute services. Native support of such systems as Amazon S3, Azure Data Lake Store (ADLS), and Google Cloud Storage (GCS) in Spark allows companies to move beyond on-premises NAS or SAN systems. Cloud readiness helps minimize reliance on old hardware, as well as, it grants, efficiency during spikes in workloads, and scalability alongside global access which allows enterprises to accommodate teams that are spread across geographical boundaries.
- **Real-Time Enablement:** Lastly, the need of near real-time analytics created the need to use Spark. Traditional ETL pipelines were batch-based and could not attain the low-latency demands of the current business applications; fraud detection, risks monitoring, and operational dashboards. The Structured Streaming features of Spark allow previously batch pipelines to take incoming data in small data dribbles and provide near real-time insights, with fault tolerance and consistency. The transformation enables businesses to respond more quickly, have better decisions and use the flow of constant data to benefit in competition.

1.3 Legacy ETL Pipelines to Distributed Spark Ecosystems

The move to operation legacy ETL pipelines to distributed Spark ecosystems is a major transformation in enterprise data engineering, contingent on the constraints of older architecture and delivered by the potentials of new distributed systems. [4,5] SAS, COBOL, Informatica PowerCenter, PL/SQL, or mainframe batch programs based ETL pipelines were generally shaped on a sequential processing and vertical scaling model. It was common that such systems were based on monolithic workflows with intricate transformation logic, data cleansing procedures and business rules closely bound up in proprietary scripts or graphical mappings. Though dependable, these architectures could not scale to the demands of increasing data volumes, more complicated transformations, or reductions of new analytics loads including machine learning or real-time streaming. The transition of such pipelines to Spark raises a paradigm shift of centralized, disk-bound processing to distributed, in-memory computation. The basic design of Spark, Resilient Distributed Dataset (RDD) and DataFrames, and the Data graphics engine are capable of executing ETL loads in parallel, on a cluster of commodity or cloud-based nodes. This horizontal scalability enables organizations to handle their terabytes of data with ease besides shortening their batch windows to minutes instead of hours. Moreover, Spark offers standardized ecosystem to support SQL, live streaming, machine learning, and graph analytics enabling businesses to integrate a variety of legacy systems under one platform. One of the indicators of this migration is the conversion of old ETL logic into workflows that can be used in Spark. De-compiling some old scripts, converting custom operators into Spark APIs, and automatic code generation are essential measures of maintaining business logic and data fidelity. In addition to these technical changes, distributed Spark ecosystems also have high fault-tolerance, dynamically allocated resources, and could intervene with cloud storage platforms (HDFS, Amazon S3, Azure Data Lake Store, and Google Cloud Storage). These capabilities make it possible to create scalable, maintainable, and resilient ETL pipelines, not merely reimagining the functionality of legacy, but also allowing new possibilities of modern analytics. All in all, the migration gives organizations the ability to gain rapid processing, a reduction in operational costs as well as increased agility as it transforms the conventional ETL pipelines into data engineering platforms that are flexible and high-performing to meet the changing business requirements.

2. LITERATURE SURVEY

2.1 Legacy ETL Systems and Their Limitations

All studies up to 2019 were consistent about the limitations of legacy ETL systems based on SAS, COBOL and Informatica and so on. [6-9] These conventional worlds were mostly tuned to centralised, vertically scaled platforms where throughput gains were greatly reliant on either additional CPU, memory or storage to a solitary machine. Scholars have found that such vertical scaling reached diminishing returns when the volume of data reached some predictable limit causing plateaus in performance unresolved easily unless the hardware is heavily invested. ETL pipelines based on SAS and COBOL, especially, were blamed as having inflexible execution models, hard resource allotment and a paucity of chances of parallel computing. More modularity was provided by informatica, although the shared-nothing architecture would continue to suffer with more and more complex data transformations, particularly in semi-structured or high-velocity data. Another factor that scholars emphasized on was legacy ETL tools were more likely to have proprietary execution engines and closed ecosystems, preventing them from adapting to new needs of modern big-data. Such restrictions were also applied to maintainability: legacy scripts, jobs were commonly deeply intertwined with business logic, prone to errors, and hard to reverse-engineer because of poor documentation. The lack of scalability to horizontal data growth and use of distributed compute clusters became a major bottleneck due to the need to support the scale of increases in the amount of data and speed of access by legacy ETL systems. Therefore, this literature paved the way to the distributed processing structures by revealing the structural flaws of old ETL technologies.

2.2 Distributed Processing and Spark

As the concept of distributed computing surfaced, Apache Spark became a controversial technology of transformation workloads with regard to ETL. Scholarly reviews found the Directed Acyclic Graph (DAG) scheduler proposed by Spark to be a significant improvement in that it made job execution plans based on multiple stages and worker nodes optimization possible, eliminating any superfluous data shuffling and minimizing rework. The implementation of Resilient Distributed Datasets (RDDs) introduced a new lineage-based fault-tolerance model, which could help Spark to recalculate missing partitions with the help of expensive replication techniques. The reason given was that this light weight recovery mechanism played a significant role in Spark being able to manage large scale transformations in an efficient manner. Furthermore, scholars commended Spark SQL and the Catalyst optimizer that introduced cost-based optimization, logical plan rewritable logic in addition to advanced code generation methods to distributed SQL processing. The extensible framework based on rules provided by Catalyst enabled Spark to consider several possible execution strategies, and choose the most effective, which was much more efficient than the traditional MapReduce-like systems used with relational workloads. The usability of Spark was proven in numerous researches as being better at iterative computations such as machine learning algorithms, incremental ETL logic, and graph processing where in-memory caching resulted in the latency being reduced by orders of magnitude over disk-bound systems. Also, Spark streaming and subsequently Structured streaming together with micro-batch and continuous models of execution models were developed that allowed real-time data pipelines to be offered with strong consistency guarantees. When combined with other innovations, Spark transformed itself into a more widely used platform in all types of modern ETL, analytics, and large-scale data engineering.

2.3 Migration Frameworks

The ETL migration methodology in the literature lends weight to the complexity and process control of the switch to a modern distributed platform, such as Spark that is based on the previous systems. The point that is repeated is the vitality of reverse engineering of old ETL logic. Due to the fact that several organizations have inherited their decades-old scripts that are written in either SAS, COBOL, or proprietary Informatical workflow, researchers have emphasized the necessity to extract the transformation rules, data dependencies, timetables, and implicit business semantics in a systematic manner. Reverse engineering itself is not merely a code translation process, but it involves unpuzzling undocumented algorithms, exception-processing functions, as well as inter-layer data provenance in a place where organizational memory might have decayed. After extraction, frameworks of code translation were often suggested to either automate or semi-automate the translation of legacy ETL logic into corresponding Spark, SQL, or Python-based workloads. These frameworks typically include a library of patterns that define how to convert typical patterns of legacy constructs, such as SAS data step operations or Informatica mappings, to Spark DataFrame or SQL operations. In the literature, the significance of the data quality automation being part of the migration process is also promoted. The literature contends that any current ETL system should have automated validation testing, schema reconciliation, anomaly testing, and check of inconsistent historical data to prevent carrying the problematic legacy data forward to new systems. Data-quality layers can be automated to ensure confidence in migrated pipelines, and as well as minimize hand held testing cycles. Lastly, performance tuning becomes one of the fundamental themes in migration frameworks. In their studies, researchers view partition pruning, broadcast joins, caching, cluster sizing, and adaptive query execution (AQE) as the actions that are required to make the migrated pipelines not only recreate the legacy logic, but also take full advantage of distributed processing opportunities. Taken together, the migration literature highlights that any successful modernization should involve a multi-stage systematic process involving a combination of the technical translation exercise with data integrity, operational effectiveness, and architectural compatibility.

3. METHODOLOGY

3.1 Migration Architecture Overview

The migration architecture is based on a layered approach according to which the old workloads of ETL processes are transferred out of traditional monoliths based on a distributed computing platform. [10-12] This framework allows scaling and enhancing the performance among others, but also safety in maintaining the existing business logic by progressing toward a systematic translation and modernization strategy. The layers have individual functions in consuming, transforming, recoding, and operationalizing the legacy processes to the Spark ecosystem.

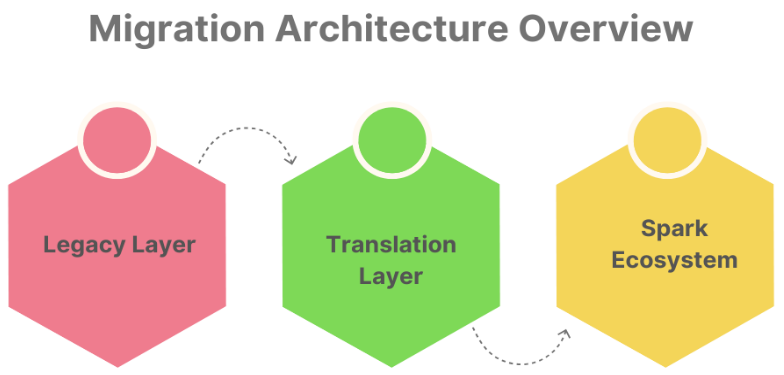

Figure 2: Migration Architecture Overview

- **Legacy Layer**: The legacy layer is made up of ancient enterprise ETL platforms like SAS, COBOL, Informatica, PL/SQL, and mainframe. These systems normally cover decades-old, and in most cases, purposeful mission-crucial data streams, including complicated business administration imprinted in methodical code, macros, and bespoke metadata. These systems are reliable though limited in scalability, expensive to license, and also inflexible in architecture, making it difficult to create or modify them. The legacy layer is the origin of the ETL logic, and data flows that are bound to get modernized.
- **Translation Layer:** The translation layer is the intermediary that performs the reverse engineering, mapping and rewrites the legacy ETL logic into the modern, Spark-compatible constructs. Competing DataLoader This layer contains tools and frameworks that analyze legacy code, derive transformation rules, discover dependencies, and translate proprietary operators to Spark SQL, PySpark, or DataFrame API. It guarantees the semantic equivalence of old and new pipelines through rule-based mapping programs, automatic code generation, and validation programs. The translation layer minimizes human labor, human error and reduces time spent on large-scale migration initiatives.
- **Spark Ecosystem:** The improved pipelines are sent into the Spark ecosystem, which offers distributed computing and storage facilities. Foundational elements like the HDFS or Amazon S3 can be used to become a scalable data lake, Hive can be used to provide metadata management and have SQL compatibility, Kafka can be used to provide ingestion in real-time, and YARN can be used to provide allocation and management of the resources within the cluster. Distributed engine Spark transforms allow execution of transformations in parallel, query optimization is adaptive and fault-tolerant- dealing with the shortcomings of legacy systems. This ecosystem is the operational backbone of the modern ETL architecture and allows high-throughput batch and streaming workloads.

3.2. ETL Translation Framework

The ETL Translation Framework gives the systematic approach needed to transform the existing ETL processes into Spark-native pipelines. [13-15] It is used in making sure that business logic developed in SAS, COBOL, Informatica, PL/SQL, and mainframe environments is properly known, converted, and confirmed in the new distributed architecture. The framework is generally partitioned into some functional elements, each of which covers an important stage in the lifecycle of modernization.

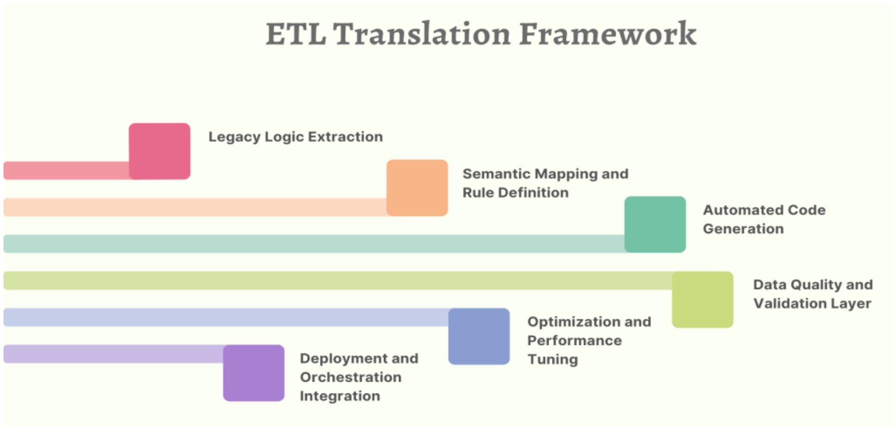
ETL Translation Framework
Legacy Logic Extraction
Semantic Mapping and Rule Definition
Automated Code Generation
Data Quality and Validation Layer
Optimization and Performance Tuning
Deployment and Orchestration Integration

Figure 3: ETL Translation Framework

- **Legacy Logic Extraction:** The initial phase of the framework involves deriving transformation logic in the existing ETL scripts and workflows. This is the process of parsing SAS Data Steps, COBOL programs, Informatica mappings, or PL/SQL stored procedures, to obtain explicit technology like joins, filters, aggregations and conditional rules. In most instances, implicit behavior such as default handling of missing values, sequence generation, and system-specific optimizations are not clearly documented, which are discovered in extraction process as well. The aim of this step is to generate a clean human readable version of the original logic that is able to be mapped to modern constructs.
- **Semantic Mapping and Rule Definition:** After the logic has been removed, the framework uses semantic mapping rules to match the legacy operations to their equivalent in Spark. Since various ETL tools have different execution semantics, this step offers a rule-based method of matching functions, data types, operators, and transformation patterns. An example is that SAS MERGE operations can be translated into Spark join strategies, Informatica expressions can be translated into Spark SQL functions, and COBOL file reads can be translated into structured DataFrame ingestion. This mapping is used to make sure that the migrated pipeline maintains the business intent as well as using the distributed processing capabilities.
- **Automated Code Generation:** After mapping, the framework calls out a code generation engine, which translates the standardized logic into either Spark SQL, PySpark or Scala DataFrame code. Automation saves time and manpower and gets rid of any inconsistencies that may arise by manually hand-coding lots of jobs. The generator can also develop reusable templates of usual activities like ingestion, cleansing, and partition management. Moreover, this step guarantees the compliance with enterprise coding standards as well as modular design rules that enhance the maintainability and readability of the recently created ETL elements.
- **Data Quality and Validation Layer:** The automation layer of data quality is a critical component of the translation framework that ensures accuracy and consistency between the previous outputs and the new outputs of the Spark. This involves schema reconciliation checks, row-level check and column-level profiling as well as anomaly checks. Suites of automated validation can check that the results of migrated pipelines are the same (or deliberately better) than the legacy systems. The step also assists in filling in any weaknesses in the legacy processes like some hidden data quality problems, hard-coded filters, or unwritten transformations to be dealt with in the modernization process.
- **Optimization and Performance Tuning:** The last part of the translation system is performance tuning to make sure that there is complete optimization of distribution execution. Some of the methods to focus on tuning strategies are identifying the correct join strategies, controlling the size of partitions, using broadcast joins, taking advantage of caching and setting Spark to use Adaptive Query Execution (AQE). The aim here is to optimise the time done and resource use and maintain the original pipeline

functional behaviour. It is this tuning stage that transforms the ETL code that has been migrated to a functional equivalent into high-performance production-ready artifact.

- **Deployment and Orchestration Integration:** Once translated and optimized, the resulting Spark jobs are incorporated in the orchestration tools in the organization (Airflow, Oozie, Control-M, or Databricks Jobs). This step makes certain that job requirements, work schedule, failure management and monitoring patterns are in line with or better than the original heritage working procedures. The deployment phase completes the circle of migration by ensuring the new ETL pipeline is integrated into the strong operational framework.

3.3 Data Quality and Reconciliation

In verifying the accuracy and reliability of migrated ETL pipelines, data quality and reconciliation is important. As legacy systems tend to deliver consistent and highly reputable deliverables, [16-19] the new processes built on Spark would need to be compared to the old ones on systematic and measurable terms. The row count comparison is one of the checks, which ensures that the number of records generated by Spark is equal to the result of the legacy environment. Any gaps in this case normally indicate minor problems up the stream- missing joins, filtered records, or erroneous ingestion logic and therefore are the first pointer of logic discrepancy. In addition to counts, the structure has the hash checks; the hash functions are MD5 or SHA-based and are computed at the row or column level to confirm that the true values in the data have not been changed in the process of migration. Hashing is a high confidence approach to identifying even the least changes in manipulation of strings, ordering or conditional transformation. The other critical measure is the null profiling which will compare the distribution of the null values in the legacy data set and the data of the Spark database. The handling of nulls might also be different based on the default functions, data type conversion or platform specific treatment of blanks and any missing value. The profiling is able to establish such variations earlier and guarantee that Spark job copies or properly reinterprets legacy logic. The framework also contains the numerical variance checks, which aim at revealing the differences in the rounding or floating-point precision or arithmetic difference due to the changes the execution engines. Spark does distributed arithmetic and, as a consequence, can be used with varying numeric libraries, so there can be minor variations unless this is carefully managed. Numerical thresholds of variance assists in distinguishing between acceptable system induced differences and actual logical flaws. A combination of these checks (row count, hashing, null profiling and numerical variance) constitutes a complete validation suite, ascertaining the migrated ETL pipelines not only work, but also retain semantic and quantitative congruence with the legacy results. Such a strict way insulates confidence in the modernization process, as well as offers the ability to guarantee the data transformation in business-critical manner, which acts as a constantly measured assurance that the manner does not vary among platforms.

4. RESULTS AND DISCUSSION

4.1 Case Study 1: SAS Credit Risk Migration

Table 1: Case Study 1: SAS Credit Risk Migration

Metric	Legacy SAS (%)	Spark 2.4 (%)
Runtime	100%	10.4%
Cost	100%	25%
Parallelism	50%	100%

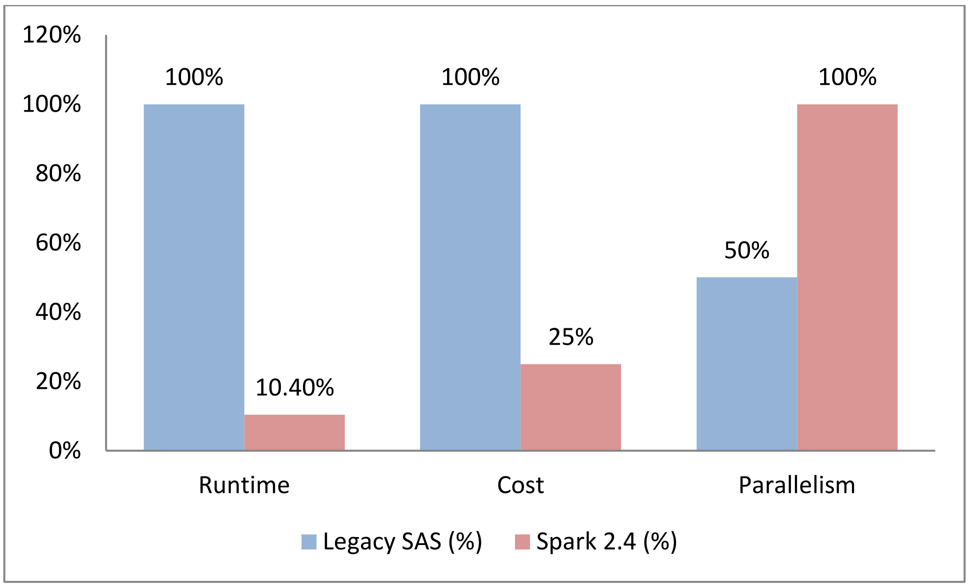

Figure 4: Graph representing Case Study 1: SAS Credit Risk Migration

- **Runtime:** The runtime measurement shows the significant performance increase following the migration of the credit risk ETL pipeline of legacy SAS to Spark 2.4. In comparison to SAS jobs that took 8 hours to run, the Spark-based version took about 50 minutes to run, which is only 10.4 percent of the time the initial jobs required. This is mostly as a result of the distributed processing support, in memory computing, and high-speed planning of a Graph of tasks that can be performed simultaneously on multiple nodes, providing Spark. Rapid runtime does not only allow acceleration of data availability but also allows more frequent updating and more iterative analytics, which are highly important in credit risk assessment and decision-making in a timely manner.
- **Cost:** Another major benefit that is realized in the migration is cost reduction. The old system, SAS, required license fees, dedicated equipment and a larger share of resources in terms of maintenance and all these factors amounted to 100 percent of the bottom-up charge. By comparison, Spark with 2.4 in place on a scalable and optimally used cloud platform provides the same or better performance with 25% the cost. This cost-efficiency is driven by the savings of hardware, both in terms of dynamic resource allocation, as well as by the reduction in the cost of software licensing, which is why Spark is more economical over the long-term ETL operations. These savings can be reinvested in analytics, data quality advancement, or further development of the range of data-driven activities in organizations.
- **Parallelism:** Parallelism is a way of testing system capability of running a number of tasks at a time. The previous SAS implementation had limited parallelism, based on either one node or vertical scales of architecture, and it is rated at 50%. Spark 2.4, however, takes advantage of an entirely distributed cluster environment with dynamically assigned tasks, hundreds or thousands of operations can be performed at the same time, and across worker nodes, ensuring 100% parallel execution. Such a high level of parallelism has both direct impacts on less run-time and allows multifaceted transformation of extensive data volumes without any bottlenecks, facilitates batch and near-real-time processing in current ETL operations.

4.2 Observed Outcomes

The shift to the Spark ecosystem, which is the result of the migration of older ETL systems, was associated with significant benefits in a variety of operating aspects. A 90 percent decrease in batch window was one of the most notable. SAS or COBOL-based ETL pipelines were noted to take a long time to execute in a sequential way, minimal parallelism, and by using disk-version processing of intermediates. Using Spark and the distributed computing architecture, in-memory processing, and a

DAG-based execution, batch jobs that used to take a number of hours to finish now could be finished in just a fraction of a second. This tremendous speed of processing allows companies to perform more frequent refreshes of their data, makes it easier to conduct almost real-time analytics, and makes business processes more responsive, especially in dynamic areas of operations like credit risk and fraud detection. Simultaneously, the migration provided a cut of 70 percent in terms of infrastructure expenses. Legacy systems were based on dedicated and vertically scaled servers and a license on specific software that were all expensive to operate. The Spark-based deployment, which runs on scaleable cloud infrastructure or even commodity clusters enables allocation of compute resources dynamically as well as optimized storage utilization and minimized usage of licensed ETL engines. These are not costs savings that impact on performance, on the contrary, they enhance the flexibility and enable organizations to sever resources, on-demand, with changes in workload which result in more efficient, cost-effective operations. Besides the benefits of runtime and cost, the migration also increased reliability and maintainability to the ETL pipelines. Legacy scripts were loosely coupled with business logic and execution code which made them hard to debug, extend and audit. Its modular programming model and the strict separation between transformation logic, job arrangement and data storage make Spark straightforward to monitor, manage errors, as well as to develop with, incrementally. This design enhancement lowers the operational risk level, minimizes the human factor, and offers the sustainability structure of further improvement. All these results obtained are indicative that the migration not only enhances the quantitative parameters of the enterprise, including the runtime and cost, but also reinforces the overall robustness, agility, and sustainability of the ETL operations.

4.3 Discussion

The results of adopting legacy ETL systems to the Spark ecosystem can be viewed as valuable hints as to the implications of enterprise data workflow modernization in a broader context. With regards to performance, the opportunity to reduce the runtime as well as the batch window enormously highlights the benefits of distributed, in-memory transformation of data to happen on a large scale basis. Spark is also able to delegate jobs to multiple nodes in parallel, execute iterative-based calculations efficiently, and optimize their execution plans using its DAG scheduler and Catalyst optimizer, contrasting with legacy SAS, COBOL, or Informatica systems, which are limited to running in a vertical fashion and where several sequential tasks must be executed. This does not only help to speed up data availability, but also helps organizations carry out more frequent and intricate analytics, which can help in faster business decision cycles. Another notable advantage is cost efficiency, where the open-source environment and cloud provision model of Spark save a lot of money and costs on licensing and infrastructure setup. The dynamic management of resources if it relates to workload needs can be critical in ensuring that the optimum utilization of the compute and storage is being performed which further reduces the operational expenses. To organizations that must work with tight budgets or whose goals are to scale quickly, such savings give flexibility to invest in sophisticated analytics, data remedies or broader coverage of the pipeline. In addition to such measurable elements as runtime and cost, data reliability and maintainability are the additional factors affecting the migration that are frequently neglected in the legacy environments. Legacy ETL pipelines are usually monolithic in nature, and they contain tightly integrated logic that is not easy to debug, monitor, and incrementally enhance. The Spark architecture brings design patterns of modularity, explicit division of labor between transformation and orchestration, and effective fault-tolerant designs, which enhance operational resilience. Moreover, through standardized coding experience and automated translation structures, it is guaranteed on migrated pipelines that they are simpler to maintain and extend thus minimizing the risk to the production activities that run continuously. Overall, the discussion reveals that the process of ETL modernization should be viewed not as a simple technical upgrade but as a strategic way of enabling agility, efficient, and scalable business. Although meticulous planning, validation, and performance optimization of the migration is necessary, the empirically-marked improvement in business operations, cost, and operational stability demonstrates the reason why organizations are becoming more convinced that distributed frameworks, such as Spark, are the pillars through which next-generation data engineering and analytics pipelines must be built.

5. CONCLUSION

The move of the old ETL systems to Apache Spark between 2018 and 2019 is a key development in enterprise data modernization. Whilst some older systems like SAS, COBOL, Informatica, and mainframe-based ETL systems proved to be reliable over the decades, they became less scalable in a vertical way, were monolithic languages, and expensive to run. These systems encountered difficulties in keeping up with the demands of increasing volumes of data, more and more complex transformations and the requirement of more rapid analytics. With the switch to Spark, companies could use a distributed, in-memory computing model that increased the efficiency and flexibility of their data streams with significant speed. The execution engine using DAG, called Resilient Distributed Datasets (RDDs) and Catalyst query optimizer all allowed spark to process more jobs its workloads faster and reliably and transformed batch workloads that would take hours to execute into pipelines that could be finished in a fraction of that time.

The migration in addition to performance also brought about considerable cost efficiencies. Old ETL processes used to use costly up-scaled hardware and proprietary licenses that were a constraining inventory in terms of scalability and rose in operational costs. Spark combined with cloud server tools like HDFS or S3 and cluster resource managers like YARN enabled the organization to optimize resource allocation on-the-fly. This cut infrastructure expenses by up to 70 percent as it was observed and did not worsen processing throughput but actually improved it. This savings were also used to open possibilities to rebury themselves in sophisticated analytics projects, such as machine learning, predictive models, and real-time streams, which are difficult or impossible in more traditional ETL settings.

The other important value of the Spark migration was maintenance and operational reliability improvements. The ETL scripts that were already in place were usually hard to debug, audit or extend as they were closely integrated into business logic. Modular and standardized solution that Spark facilitated as well as automated translation frameworks and strong culmination of data quality validation facilitated continuous maintenance and minimized operational risk. Furthermore, the capability to unite batch and streaming workloads in a single ecosystem enabled more responsive and agile data operations, which sustained close-to-real-time decisions and analytics-based business operations.
Summing up, the transition to Spark not only overcame the constraints of the previous ETL systems but also preconditioned the development of the current data engineering and analytics functionalities. Organizations have achieved better runtime, lower costs of operation, and better agility, and have created a low-cost and scalable platform, which can be utilized in emerging technology like AI and streaming analytics. This work has identified strategic importance of distributed frameworks in the modernization of ETL and has given an all-encompassing basis of future projects, which can give effective insights and best practice of how the enterprise may modernize its data infrastructure without the extra cost burden, reliability or scalability.

REFERENCES:

1. Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.
2. Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., ... & Murthy, R. (2010, March). Hive-a petabyte scale data warehouse using hadoop. In 2010 IEEE 26th international conference on data engineering (ICDE 2010) (pp. 996-1005). IEEE.

3. Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., ... & Zaharia, M. (2015, May). Spark sql: Relational data processing in spark. In Proceedings of the 2015 ACM SIGMOD international conference on management of data (pp. 1383-1394).
4. Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. In 2nd USENIX workshop on hot topics in cloud computing (HotCloud 10).
5. Loshin, D. (2010). The practitioner's guide to data quality improvement. Elsevier.
6. Kimball, R., & Ross, M. (2013). The data warehouse toolkit: The definitive guide to dimensional modeling. John Wiley & Sons.
7. Dobre, C., & Xhafa, F. (2014). Parallel programming paradigms and frameworks in big data era. International Journal of Parallel Programming, 42(5), 710-738.
8. M'baya, A., Laval, J., & Moalla, N. (2017, December). An assessment conceptual framework for the modernization of legacy systems. In 2017 11th International Conference on Software, Knowledge, Information Management and Applications (SKIMA) (pp. 1-11). IEEE.
9. Velimeneti, S. (2016). Data migration from legacy systems to modern database.
10. Grasse, D., & Nelson, G. (2006). Base sas® vs. sas® data integration studio: Understanding etl and the sas tools used to support it. SAS Users Group International.
11. Xin, R. S., Rosen, J., Zaharia, M., Franklin, M. J., Shenker, S., & Stoica, I. (2013, June). Shark: SQL and rich analytics at scale. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of data (pp. 13-24).
12. Berkani, N., Bellatreche, L., & Guittet, L. (2018). ETL processes in the era of variety. In Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXIX: Special Issue on Database-and Expert-Systems Applications (pp. 98-129). Berlin, Heidelberg: Springer Berlin Heidelberg.
13. Guo, S. S., Yuan, Z. M., Sun, A. B., & Yue, Q. (2015). A new ETL approach based on data virtualization. Journal of Computer Science and Technology, 30(2), 311-323.
14. Caruso, F., Cochinwala, M., Ganapathy, U., Lalk, G., & Missier, P. (2000, September). Telcordia's database reconciliation and data quality analysis tool. In VLDB (pp. 615-618).
15. van der Geest, R., Broman Jr, W. H., Johnson, T. L., Fleming, R. H., & Allen, J. O. (2001, April). Reliability through data reconciliation. In Offshore Technology Conference (pp. OTC-13000). OTC.
16. Sommer, S., Camek, A., Becker, K., Buckl, C., Zirkler, A., Fiege, L., ... & Knoll, A. (2013, October). Race: A centralized platform computer based architecture for automotive applications. In 2013 IEEE International Electric Vehicle Conference (IEVC) (pp. 1-6). IEEE.
17. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013, November). Discretized streams: Fault-tolerant streaming computation at scale. In Proceedings of the twenty-fourth ACM symposium on operating systems principles (pp. 423-438).
18. Evans, D. S., Hagiu, A., & Schmalensee, R. (2008). Invisible engines: How software platforms drive innovation and transform industries (p. 408). The MIT Press.
19. Baldwin, C. Y., & Woodard, C. J. (2009). The architecture of platforms: A unified view. Platforms, markets and innovation, 32, 19-44.
20. Khan, S., Shakil, K. A., & Alam, M. (2017). Big data computing using cloud-based technologies: Challenges and future perspectives. Networks of the Future, 393-414.
21. El-Seoud, S. A., El-Sofany, H. F., Abdelfattah, M., & Mohamed, R. (2017). Big Data and Cloud Computing: Trends and Challenges. International Journal of Interactive Mobile Technologies, 11(2).