IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 23 June 2025, accepted 14 July 2025, date of publication 24 July 2025, date of current version 19 August 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3591712

RESEARCH ARTICLE

DatApollo: Orchestration of Serverless Functions for Scalable Data Mining

MAHTAB SHAHIN[1], MARKUS BERTL[2], (Member, IEEE), NASIM JANATIAN[3], JUAN AZNAR-POVEDA[4], SYED ATTIQUE SHAH[5], (Senior Member, IEEE), THOMAS FAHRINGER[4], (Member, IEEE), SIJO ARAKKAL PEIOUS[6], AND DIRK DRAHEIM[6], (Member, IEEE)

[1]Estonian Maritime Academy, Tallinn University of Technology, 11712 Tallinn, Estonia
[2]Department of Information Systems and Operations Management, Vienna University of Economics and Business, 1020 Vienna, Austria
[3]Department of Geoscience and Engineering, Delft University of Technology, 2628 CN Delft, The Netherlands
[4]Institute of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria
[5]Department of Computer Science, Birmingham City University, B4 7RQ Birmingham, U.K.
[6]Information Systems Group, Tallinn University of Technology, 12616 Tallinn, Estonia

Corresponding author: Mahtab Shahin (mahtab.shahin@taltech.ee)

This work was supported in part by European Union and Estonian Research Council under Project TEM-TA141, and in part by EU Horizon2020 Project 952360-MariCybERA.

ABSTRACT With the exponential growth of data generated from enterprise systems, social networks, and the Internet of Things, traditional data mining techniques face major challenges in terms of scalability and efficiency. As a foundational unsupervised learning method for detecting patterns in transactional datasets, Association Rule Mining (ARM) is commonly encountered in distributed environments with performance bottlenecks due to excessive memory consumption, static resource provisioning, and costly data shuffle. The present paper presents DatApollo, a novel serverless orchestration framework that enables the execution of distributed ARM workflows in a scalable and efficient manner. DatApollo, based on the Apollo orchestration engine, offers stateless cloud functions, dynamic scheduling, intermediate state persistence, and fault-tolerant coordination in order to address the limitations of both traditional cluster-based architectures and existing Function-as-a-Service models. By decomposing ARM pipelines into orchestrated microfunctions, the framework enables elastic, cloud-native execution with minimal idle time. Using real-world healthcare and meteorological datasets, we describe the architectural design, algorithmic components, and computational complexity of DatApollo and perform a comprehensive experimental evaluation. DatApollo provides up to five times faster execution time compared to Apache Spark and lowers infrastructure costs by utilizing elastic scaling and event-driven function invocations. The results demonstrate that DatApollo is a robust, cost-effective and high-performance alternative to ARM in dynamic, large-scale data environments.

INDEX TERMS Association rule mining, big data, serverless computing, function-as-a-service (FaaS), workflow orchestration, Apriori algorithm, cloud scalability, Apache spark.

I. INTRODUCTION

Traditional data mining techniques have faced substantial challenges as a result of the exponential growth of data generated by enterprises, social media platforms, and the Internet of Things (IoT). As data volumes, velocity, and heterogeneity increase, mining systems must be not only scalable but also resource efficient, adaptive, and responsive. Many sectors, such as healthcare, finance, cybersecurity, and intelligent transportation systems, rely heavily on robust data mining to generate meaningful insights from large dynamic datasets [1], [2], [3], [4].

A fundamental technique in classical data mining is Association Rule Mining (ARM), which is used to identify meaningful patterns and co-occurrence relationships within transactional data. In addition to market basket analysis,

The associate editor coordinating the review of this manuscript and approving it for publication was Senthil Kumar.

ARM has demonstrated successful use in disease diagnosis applications. Traditional ARM algorithms, however, face several critical limitations when applying them to large-scale and high-dimensional datasets, including excessive memory consumption, long computation times, and limited scalability across cloud-native infrastructures. Due to these constraints, ARM is severely impaired when used in real-time Big Data analytics [5].

In general, ARM algorithms, such as Apriori [6], Eclat [7], and FP-Growth [8] are effective for medium-sized datasets, but they fail when applied to distributed and high-volume datasets due to their iterative, memory-intensive design.

Due to this, distributed frameworks such as Apache Spark and Hadoop have been adapted to support ARM through parallel execution. This approach, however, has significant limitations: multiple dataset scans and a long candidate generation process result in a high level of memory overhead, prolonged execution times, and a large amount of infrastructure costs [9].

In addition, these systems are based on static resource allocation, which requires the pre-provisioning of compute nodes regardless of workload variability. This often leads to poor elasticity, underutilized resources, or overprovisioning at high costs [10]. In spite of Spark's design for iterative workloads, its in-memory computation model suffers from memory fragmentation, inefficient shuffle operations, and I/O bottlenecks - factors that limit its application in complex, multiphase ARM pipelines [11], [12].

Compared to static and clustered architectures, serverless computing has emerged as a disruptive and agile model for cloud-native data processing, especially Function-as-a-Service (FaaS) [13]. FaaS allows developers to execute stateless functions that are triggered by events or scheduled workflows without managing the underlying infrastructure. This abstraction enables horizontal autoscaling, automatic fault tolerance, and a pay-per-use billing model that lowers idle resource costs, especially in unstable or unpredictable workloads [14], [15], [16]. This model is exemplified by cloud platforms such as AWS Lambda and Azure Functions, which allocate computing instances based on demand, making modular analytics tasks more efficient to deploy.

Despite its advantages, applying Function-as-a-Service (FaaS) to Association Rule Mining (ARM) remains non-trivial. ARM workflows are inherently stateful, iterative, and require tight coordination, which conflicts with the stateless and ephemeral nature of serverless functions. Limited inter-function data sharing, short execution timeouts, and constrained memory allocations pose significant challenges for generating frequent itemsets and association rules. Moreover, existing orchestration frameworks such as AWS Step Functions, Azure Durable Functions, and Apache OpenWhisk support only coarse-grained workflows. These platforms offer limited capabilities for managing fine-grained data dependencies, iterative feedback loops, and persistence of intermediate results. In addition, their overhead in handling long-running executions, data serialization, and cold-start latency can significantly degrade the performance of high-throughput ARM workloads [17].

This paper proposes a domain-specific serverless orchestration framework that is specifically designed for scalable ARM architectures. Using the *DatApollo framework*,[1] ARM processes are decomposed into stateless microfunctions and orchestrated by a Directed Acyclic Graph (DAG) scheduler. Data partitioning, candidate generation, support counting, and rule evaluation are all handled by separate microfunctions that are distributed, event-driven, and latency-aware.

To the best of our knowledge, DatApollo provides the first end-to-end, fully serverless orchestration of ARM workflows that integrates latency-aware scheduling, lazy invocation of functions, and checkpoint-based fault tolerance. By storing intermediate data externally in cloud-native object storage (e.g., AWS S3), persistent state management is enabled, and functions can be resumed or retry without losing progress. Through these mechanisms, DatApollo is able to bridge the gap between the dynamic, elastic nature of FaaS and the iterative, stateful demands of association rule mining.

Compared to existing frameworks:

- Unlike Spark, which is optimized for static clusters and in-memory pipelines, DatApollo provides elastic autoscaling, fine-grained resource billing, and decoupled function execution.
- Unlike general-purpose FaaS orchestrators, which lack support for iterative patterns and tight data coupling, DatApollo introduces a custom DAG model with execution checkpoints, metadata tagging, and cloud-native intermediate storage.

This work is guided by the central research question:

- *How can serverless computing architectures be tailored to support efficient, scalable, and cost-effective association rule mining workflows in Big Data environments?*

To address this, we design, implement, and evaluate DatApollo against a Spark-based ARM pipeline across three real-world datasets: a COVID-19 symptom dataset, a lung cancer registry, and a meteorological dataset from Estonia. These datasets were selected to capture variation in dimensionality, sparsity, and application context.

According to our research, DatApollo performs up to a five times faster execution time as compared to Apache Spark while maintaining an equivalent level of rule quality as measured by support and confidence. Furthermore, DatApollo improves cost efficiency by eliminating idle resource waste and enabling high responsiveness under bursty workloads. In light of these findings, it is evident that DatApollo is an effective and practical solution for scalable, serverless ARM in modern cloud environments.

The remainder of this paper is structured as follows. Section II provides background on Association Rule Mining (ARM) algorithms, serverless computing, and distributed mining frameworks. Section III reviews the related literature. The design of the DatApollo system is described

[1] https://github.com/Mahtab90/DatApollo

in Section IV. Section V outlines the experimental setup and methodology. Section VI presents the results of our experiments. In Section VII, we discuss the implications of our findings and suggest directions for future work. Finally, we close the paper with concluding remarks in Section IX.

II. BACKGROUND INFORMATION

This section provides the foundational concepts necessary to understand the motivations and design of our proposed orchestration system. It begins with an overview of Association Rule Mining (ARM) and progresses through serverless computing paradigms, distributed ARM in Apache Spark, and the capabilities of the Apollo orchestration framework.

A. ASSOCIATION RULE MINING (ARM)

Association Rule Mining (ARM) is a foundational unsupervised learning technique introduced by Agrawal et al. [18] to uncover statistically significant relationships between itemsets in transactional databases. Originally applied in market basket analysis—e.g., identifying that purchasing `milk` often co-occurs with `bread`—ARM has since been extended to various domains such as bioinformatics, financial auditing, cybersecurity, and healthcare analytics.

A typical association rule is expressed as $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$, with I denoting the set of all items. The rule implies that the presence of X in a transaction increases the likelihood of Y's presence. The utility of a rule is commonly measured using the following metrics:

- *Support* (σ): The proportion of transactions in which both X and Y appear:

$$\sigma(X \Rightarrow Y) = \frac{|\{t \in \mathcal{T} \mid X \cup Y \subseteq t\}|}{|\mathcal{T}|} \tag{1}$$

- *Confidence* (γ): The conditional probability of Y given X:

$$\gamma(X \Rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \tag{2}$$

- *Lift* (λ): A measure of the strength of dependence between X and Y:

$$\lambda(X \Rightarrow Y) = \frac{\gamma(X \Rightarrow Y)}{\sigma(Y)} = \frac{\sigma(X \cup Y)}{\sigma(X) \cdot \sigma(Y)} \tag{3}$$

  A lift value greater than 1 indicates a positive correlation.

Contemporary ARM variants address complex data properties:

- *Multi-level ARM*: Derives rules across taxonomic levels.
- *Quantitative ARM*: Uses discretization for continuous attributes.
- *High-dimensional ARM*: Incorporates sampling or pruning to manage attribute-rich datasets.

Additional interestingness measures such as conviction, leverage, and coverage have been introduced to improve rule filtering and interpretability [19], [20], [21], [22].

B. SERVERLESS COMPUTING IN DATA MINING

Serverless computing, and particularly the Function-as-a-Service (FaaS) paradigm, enables execution of stateless, event-driven functions without requiring developers to manage infrastructure. Providers such as AWS Lambda and Azure Functions allow for horizontal autoscaling and pay-per-use pricing, making them well-suited for bursty or variable workloads.

However, the application of FaaS to data-intensive workflows such as ARM presents several challenges:

- *Cold-start latency:* Function initialization delay negatively affects interactive tasks.
- *Statelessness:* Intermediate results must be stored externally (e.g., in S3), introducing I/O overhead and latency.
- *Limited orchestration support:* Many FaaS platforms lack native fine-grained orchestration mechanisms for iterative or multi-phase algorithms.

These constraints hinder traditional ARM workflows, especially those based on Apriori or FP-Growth, that require multiple rounds of support counting and frequent inter-step coordination. Thus, extending serverless computing to support ARM necessitates architectural support for function chaining, state persistence, and latency-aware task scheduling.

C. APACHE SPARK FOR DISTRIBUTED ARM

Apache Spark is a general-purpose distributed computing platform optimized for in-memory data processing. ARM implementations on Spark typically adopt one of two strategies:

- *YAFIM (Yet Another Frequent Itemset Miner) [23]:* Uses MapReduce transformations to simulate Apriori-style itemset expansion. It struggles with higher-order itemsets due to the size of transaction ID (TID) lists.
- *PFP (Parallel FP-Growth) [24]:* Employs frequent pattern partitioning to avoid candidate generation but returns only frequent itemsets, not full association rules.

Despite Spark's success in many domains, its use for ARM faces critical bottlenecks:

- *Memory and I/O overhead:* ARM requires multiple data scans and shuffling operations, which stress Spark's memory-centric execution.
- *Static resource provisioning:* Clusters must be pre-allocated, resulting in either underutilization or overprovisioning.
- *Lack of native checkpointing:* ARM's intermediate states require manual persistence strategies, adding complexity.

These issues limit Spark's suitability for workflows requiring high elasticity, low-latency, and iterative coordination – hallmarks of modern ARM applications.

D. APOLLO ORCHESTRATION FRAMEWORK

Apollo [16] is a modular orchestration engine for managing distributed workflows across edge-cloud environments.

It supports decentralized orchestration via lightweight agents and is designed for scalable, adaptive task coordination.

Key features relevant to data mining include:

- *Pluggable orchestration agents:* Enable distributed, autonomous control of task execution.
- *Resource- and session-aware scheduling:* Minimizes cold starts and co-locates compute with data.
- *Hybrid infrastructure compatibility:* Operates across containers, virtual machines, and serverless runtimes.

Apollo's design enables reduced latency, better fault tolerance, and efficient orchestration of complex data pipelines. Though not originally developed for ARM, its extensibility and DAG-based task management make it a compelling foundation for orchestrating iterative data mining workflows.

III. LITERATURE REVIEW

This section reviews existing algorithms and frameworks for Association Rule Mining (ARM) and frequent itemset mining [4], [5], [6]. The discussion spans sequential, parallel, and distributed implementations, highlights real-world applications, and situates the proposed DatApollo system within the broader context of serverless computing and high-performance environments.

A. SEQUENTIAL AND PARALLEL ARM ALGORITHMS

The Apriori algorithm, introduced by Agrawal and Srikant in the mid-1990s [25], is one of the foundational ARM techniques. It utilizes the downward closure property, which states that all subsets of a frequent itemset must also be frequent. Although effective, Apriori can become computationally expensive due to multiple database scans. Several improvements followed, including Apriori-TID, which reduces overhead by removing nonfrequent transactions in each iteration, and ECLAT [7], which uses the TID-list data structure [26] to enhance efficiency with Boolean operations.

The FP-Growth algorithm [8] was introduced to address Apriori's performance limitations. Using an FP-tree structure eliminates the need for candidate generation and reduces the number of database scans, making it particularly efficient for dense datasets. Empirical studies have shown that FP-Growth outperforms Apriori in runtime, especially on large datasets [27].

Parallel approaches have extended these algorithms to leverage modern multi-core and distributed systems. Notable implementations include ParEclat [25], which applies parallelism to the ECLAT framework, and parallel FP-Growth variants with sampling [28], which improve scalability and reduce synchronization overhead.

B. DISTRIBUTED ARM IMPLEMENTATIONS AND METHODOLOGIES

The era of Big Data has intensified the need for distributed ARM techniques capable of processing massive datasets. Distributed frameworks using the MapReduce paradigm have become central to large-scale itemset mining.

1) HADOOP-BASED ARM IMPLEMENTATIONS

Several ARM solutions have been developed on Hadoop's MapReduce model. Dist-Eclat and BigFIM [28] implement load-balancing strategies for scalable itemset mining. The AprioriMR family, including Iterative AprioriMR, Pruning AprioriMR, and Top AprioriMR [29], introduces various pruning and iteration strategies to enhance performance. FIMMR [30] demonstrates superior speed and scalability compared to classical MapReduce implementations such as Parallel FP-Growth (PFP) and Sequential Pattern Counting (SPC). While Hadoop offers fault tolerance and horizontal scaling, its reliance on disk-based I/O often results in high latency.

2) SPARK-BASED ARM IMPLEMENTATIONS

Apache Spark addresses Hadoop's latency issues with in-memory computation, offering significant performance gains. Spark-based ARM implementations include PFP [31], which applies FP-Growth in a distributed manner, and Distributed Apriori [32], which partitions datasets to minimize communication overhead. Recent works such as Parallel FP-Growth [33] focus on optimizing Spark's memory management and computation partitioning. Spark's architecture provides high throughput and reduced latency, making it a preferred framework for real-time or near-real-time ARM.

C. APPLICATIONS OF ARM IN BIG DATA ANALYTICS

ARM is widely applied across various domains. In traffic analysis, Shahin et al. [3] applied ARM to analyze 576 road intersection accidents in Isfahan, Iran, using k-mode clustering to extract risk patterns. In healthcare, Bertl et al. [34] leveraged ARM on psychiatric data to identify comorbid conditions and indicator diseases. During the COVID-19 pandemic, Shahin et al. [35] employed Spark-based ARM to uncover patterns in patient outcomes, demonstrating the model's potential in high-stakes medical decision-making.

D. SERVERLESS ARM EXECUTION AND DATAPOLLO IMPLEMENTATION

Although Hadoop and Spark offer scalability, they often require fixed resource configurations and suffer from underutilized infrastructure. Serverless computing provides an elastic and cost-efficient solution that allows resource allocation on demand without managing server infrastructure.

1) ADVANTAGES OF SERVERLESS COMPUTING FOR ARM

Serverless ARM execution offers several benefits: elastic scalability, where functions automatically adjust to workload; cost-efficiency, by avoiding overprovisioning; and fault tolerance, due to stateless execution. These advantages make serverless platforms suitable for event-driven, scalable data mining workflows.

2) COMPARATIVE ANALYSIS OF RELATED WORKS

To further contextualize DatApollo's contribution, Table 2 compares recent works on distributed and serverless ARM implementations. Prior research has primarily focused on adapting classical ARM algorithms like Apriori and FP-Growth to distributed platforms such as Hadoop and Spark. However, these approaches often suffer from high latency, static resource provisioning, and limited orchestration capabilities. More recent studies have explored serverless computing for data analytics, but few have addressed the specific challenges of orchestrating iterative and stateful ARM workflows in a fully serverless manner. As illustrated in Table 1, our DatApollo framework uniquely combines elastic function-level orchestration, persistent intermediate state storage, cloud-native scalability, and rigorous multi-dataset evaluation. This positions DatApollo as a state-of-the-art solution for scalable ARM in modern Big Data environments.

IV. THE PROPOSED FRAMEWORK: DatApollo

A. SERVERLESS ORCHESTRATION FOR ARM: THE DatApollo FRAMEWORK

The proposed *DatApollo* framework orchestrates Association Rule Mining (ARM) workloads in a serverless environment by combining the elasticity of Function-as-a-Service (FaaS) with a Directed Acyclic Graph (DAG)-based scheduling model. The architecture integrates modular, stateless functions, enabling scalable, fault-tolerant ARM across distributed cloud infrastructure.

Figure 1 illustrates the full pipeline, consisting of three coordinated stages: (i) data partitioning and ingestion, (ii) frequent itemset discovery, and (iii) association rule generation. Each stage is independently deployed as stateless AWS Lambda functions, chained through Apollo's adaptive DAG scheduler.

1) STAGE 1: FUNCTION-LAYER ORCHESTRATION OF DATA PREPARATION

As illustrated in the top panel of Figure 1, raw input files are transformed into a transactional format by preprocessing functions triggered upon data ingestion events (e.g., S3 file uploads). These serverless functions normalize, clean, and partition the dataset into equal-sized segments, which are then stored in cloud object storage. Each partition independently triggers a mining function without centralized coordination, enabling horizontal scalability and low-latency data preparation.

2) STAGE 2: FUNCTION-LAYER EXECUTION OF FREQUENT ITEMSET MINING

Each mining function performs an isolated Apriori-based itemset mining on its assigned partition, using a local minimum support threshold. Intermediate results—namely the discovered itemsets and their support counts—are serialized into external cloud storage. As shown in the top figure, these outputs are harmonized by a coordination function that merges local results, deduplicates itemsets, and builds a global itemset registry for downstream rule generation.

3) STAGE 3: RULE GENERATION AND THRESHOLD-BASED FILTERING

Using the global itemset registry, additional Lambda functions are invoked to compute confidence and lift values per candidate rule. Only those rules that meet user-defined thresholds are retained. Final outputs are persisted in structured formats such as CSV or JSON and made available for visualization, further analysis, or integration into downstream systems.

4) APOLLO-BASED DAG SCHEDULING AND ORCHESTRATION

The bottom panel of Figure 1 illustrates how Apollo orchestrates the above function executions through an adaptive Directed Acyclic Graph (DAG) model. Unlike traditional orchestrators (e.g., AWS Step Functions or Azure Durable Functions), Apollo offers fine-grained orchestration with the following features:

- *Latency-aware scheduling*: Functions are colocated with warm execution containers or relevant data to minimize cold starts and I/O delays.
- *Checkpointing and fault recovery*: Intermediate outputs are saved externally, enabling failed DAG nodes to be retried or resumed without re-executing the entire pipeline.
- *Adaptive DAG traversal*: Execution paths are dynamically adjusted at runtime to mitigate stragglers and rebalance workloads as needed.

Algorithm 1 High-Level DatApollo Orchestration Logic

```
1:  function DatApollo_ARM(Dataset, SupportThreshold)
2:      Partitions ← PartitionData(Dataset)
3:      for all p ∈ Partitions in parallel do
4:          Launch(Lambda_Mine(p, SupportThreshold))
5:      end for
6:      WaitUntilComplete(MiningJobs)
7:      GlobalItemsets ← Merge_Itemsets(IntermediateResults)
8:      for all i ∈ GlobalItemsets in parallel do
9:          Launch(Lambda_GenerateRules(i))
10:     end for
11:     StoreOutput(FinalRules)
12: end function
```

Failures at any DAG node are automatically retried according to configurable policies. In the event of repeated failures, fallback Lambda functions are triggered to handle exceptions or log execution metadata for post-mortem debugging.

5) DEPLOYMENT CONFIGURATION

The DatApollo framework is deployed using AWS Lambda functions written in Python 3.10. Each function operates

TABLE 1. Comparison of recent works on serverless and distributed association rule mining (2013–2025).

Feature	[27]	[29]	[32]	[28]	[16]	[4]	This Work (DatApollo, 2025)
ARM Algorithm Supported	FP-Growth	Apriori	Apriori	FP-Growth	Apriori (static)	Apriori	Apriori
Serverless Architecture	✗	✗	✗	✗	✓	✗	✓
Function-Level Orchestration	✗	✗	✗	✗	∂	✗	✓
Cloud-Native Scalability	✗	✗	✗	✗	✓	∂	✓
Intermediate State Persistence	✗	✗	✗	✗	∂	✗	✓
Support & Confidence Filtering	✓	✓	✓	✓	✗	✓	✓
Elastic Resource Management	✗	✗	✗	✗	∂	✗	✓
Experimental Evaluation	Limited	Yes	Yes	Yes	Limited (demo)	Yes	Extensive (3 datasets, 3 thresholds)

Note: ✓ indicates full support; ✗ indicates no support; ∂ denotes partial or limited support.

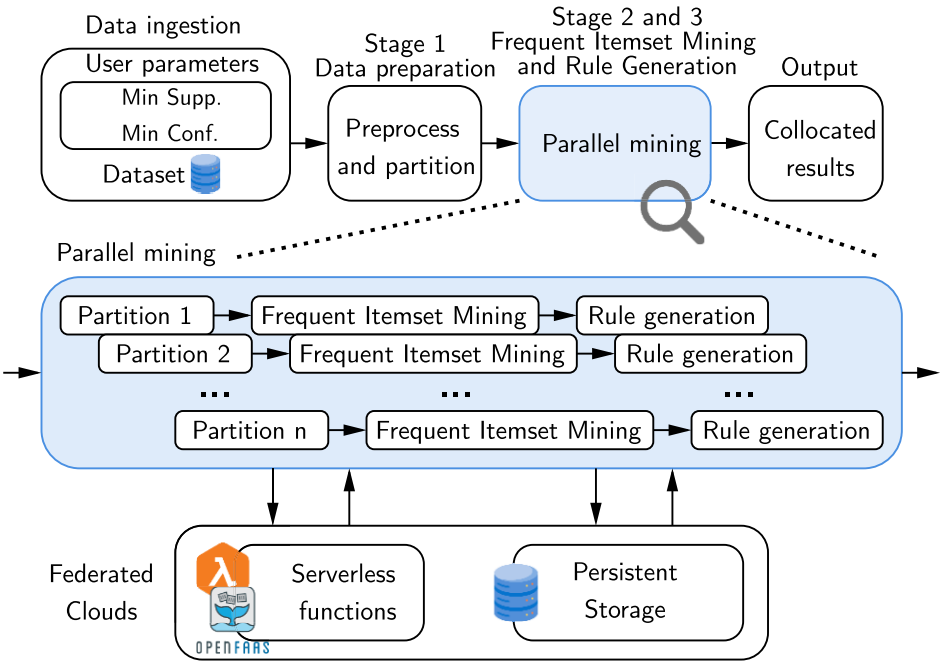

FIGURE 1. Architecture of the DatApollo framework. The top panel illustrates the modular function layer, where each stage of the Association Rule Mining (ARM) process is implemented as an independent AWS Lambda function. The bottom panel depicts the adaptive Directed Acyclic Graph (DAG) orchestration managed by the Apollo engine, enabling latency-aware scheduling, checkpointing, and fault-tolerant execution across dynamic workloads.

within memory limits ranging from 512MB to 2048MB, with execution timeouts capped at 180 seconds. Triggers are event-based (e.g., S3 uploads or REST API calls via API Gateway), allowing both real-time and batch-mode executions. Persistent state and metadata—including DAG progress, intermediate itemsets, and fault-tolerant checkpoints—are stored in Amazon S3 with versioning and consistency tags for traceability and audit.

B. HPC BENCHMARKING FOR DARM EVALUATION

To benchmark DatApollo's scalability, we evaluate it against an MPI-based HPC implementation. The Generalized ARM (GARM) architecture distributes dataset partitions across static compute nodes. Each node executes Apriori independently, then aggregates results via centralized coordination.

Although effective for fixed data volumes, HPC-based DARM workflows lack elasticity and require substantial setup (e.g., MPI configuration, node synchronization). Furthermore, node failures or stragglers typically halt execution, limiting their robustness under dynamic workloads.

C. HPC DARM PSEUDOCODE

Algorithm 2 presents the pseudocode used for experimentation with HPC-based DARM across varying node counts and support thresholds.

GARM offers a competitive runtime under ideal conditions, however, DatApollo's cloud-native elasticity, adaptive orchestration, and failure isolation make it a more resilient alternative in real-world data mining applications.

V. EXPERIMENTAL SETUP AND METHODOLOGY

To rigorously evaluate the computational performance, scalability, and analytical fidelity of the proposed DatApollo framework for Association Rule Mining (ARM), we designed

TABLE 2. Comparison of DatApollo and Apache Spark for Distributed ARM.

Feature	DatApollo	Apache Spark
Execution Model	Serverless FaaS + DAG	Static Cluster
State Management	Cloud Object Storage	In-Memory
Scalability	High (Elastic)	Medium (Fixed Nodes)
Fault Tolerance	Function-Level Recovery	Task-Level Retry
Orchestration	Apollo (Adaptive DAG)	Static DAG
Cost Efficiency	High (Pay-per-Use)	Medium
Rule Filtering Support	Yes (Lift, Confidence)	Yes

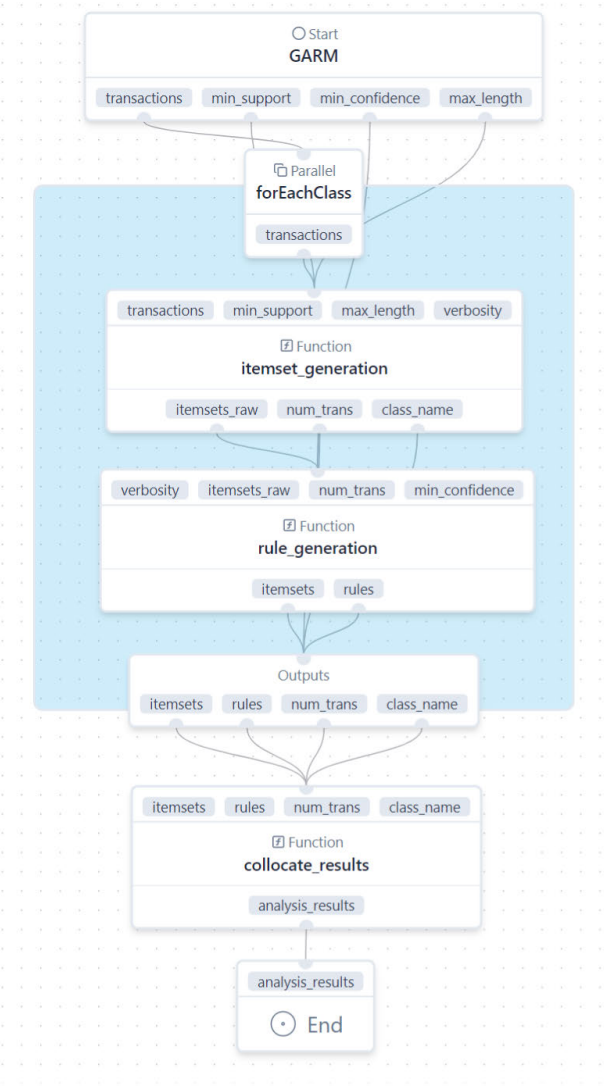

FIGURE 2. GARM workflow for distributed ARM in a high-performance computing environment.

a structured benchmarking methodology grounded in statistical best practices. This methodology not only benchmarks DatApollo against Apache Spark, a state-of-the-art distributed data processing system, but also surfaces trade-offs relevant to serverless orchestration in real-world, data-intensive mining workflows.

Algorithm 2 Distributed ARM (DARM) Benchmark on HPC

```
function Run_Experiments(data, nodeList, suppList)
    for all n ∈ nodeList do
        for all s ∈ suppList do
            speedup ← MeasureSpeedup(data, n, s)
            rules ← CountRules(data, n, s)
            quality ← EvaluateQuality(rules)
        end for
    end for
end function
```

This evaluation framework consists of three key experimental axes: (1) computational efficiency, (2) quantity of rules discovered, and (3) rule quality. To provide a balanced and comprehensive view of DatApollo's performance characteristics, these axes were assessed independently.

A. EVALUATION OBJECTIVES, METRICS, AND STATISTICAL VALIDITY

Our experiments were structured into three main evaluation phases:

- *Experiment A – Execution Time and Scalability:* Compares DatApollo's elasticity and overhead management to that of Apache Spark under varying levels of parallelism and data loads.
- *Experiment B – Rule Quantity Analysis:* Assesses the number of association rules generated at different support thresholds to capture mining completeness and potential redundancy.
- *Experiment C – Rule Quality Evaluation:* Evaluates extracted rules using support, confidence, and lift to determine analytical integrity under varying system configurations.

Each configuration was executed five times to ensure statistical rigor. For all quantitative metrics, mean values, standard deviations, and 95% confidence intervals were calculated. Whenever possible, paired t-tests and ANOVA were used to determine the reliability of observed differences. Execution time ($\tau_{execution}$) was measured as:

$$\tau\_execution = \tau\_end - \tau\_start \tag{4}$$

This experimental strategy supports statistically defensible insights into DatApollo's operational behavior.

B. BASELINES AND COMPARATIVE FRAMEWORKS

Our primary benchmark is Apache Spark due to its wide adoption and in-memory processing efficiency. However, we acknowledge the need to evaluate additional platforms [27]. Despite the fact that a more comprehensive experimental comparison against GARM and commercial tools such as AWS Glue will be reserved for future work due to access and configuration constraints, their performance envelopes are cited for context in the Discussion.

C. EXPERIMENT A: EXECUTION TIME AND COST-ADJUSTED SCALABILITY

Several configurations of compute node (3, 6, 9, and 11 nodes) were evaluated in terms of execution time and support thresholds (30%, 60%, and 80%). Our analysis included measuring raw runtime as well as cost-normalized efficiency (defined as execution time per rule per dollar) based on conversion into USD equivalents of Amazon Lambda and Spark EC2 resource usage. The results of this analysis indicate whether DatApollo's speedups are economically justified.

D. EXPERIMENT B: RULE QUANTITY ANALYSIS

The total number of association rules generated under identical configurations was compared. Redundancy was examined by detecting rule overlaps and subsets and supersets. Variation in output quantity was reported across five runs to demonstrate reproducibility. This analysis distinguishes between comprehensive rule exploration and excessive rule generation.

E. EXPERIMENT C: RULE QUALITY EVALUATION

The support, confidence, and lift values were computed for each run. Moreover, we tracked the maximum and average values for these metrics, as well as the 5% most frequent rules. The comparison charts with error bars demonstrate that rule strength is consistent across platforms, proving that DatApollo's architectural speedups do not affect rule quality.

F. ALGORITHMIC EXECUTION AND ORCHESTRATION SPECIFICS

DatApollo decomposes the Apriori process into stateless AWS Lambda functions, each executing on a data partition. Candidate pruning is performed independently within each function based on a localized support threshold. Results—containing support counts for candidate itemsets—are serialized and stored in Amazon S3.

A coordination function aggregates these results, resolves duplicate itemsets via global support re-computation, and applies global pruning. Merged results are checkpointed to ensure reproducibility and enable intermediate recovery in case of orchestration failure.

G. FAULT TOLERANCE AND CORRECTNESS

DatApollo ensures operational robustness through:

- Checkpointing: Intermediate outputs are written with versioning metadata.
- Failure Isolation: Failed partitions are retried independently, minimizing cascading errors.
- Fallback Handlers: Triggered when retry limits are exceeded. Partial data handling policies prevent workflow termination.

These mechanisms preserve correctness even during partial Lambda failures or transient network issues.

H. SERVERLESS RESOURCE CONSTRAINTS AND OPTIMIZATION

DatApollo operates under constrained memory (512MB–2048MB) and time limits (up to 180 seconds). To adapt:

- Large candidate sets are recursively decomposed into smaller chunks and delegated to fresh Lambda invocations.
- ARM stages are modularized to minimize memory footprint.
- Memory warnings trigger adaptive backoff and reallocation.

This enables stable operation on large datasets with low support thresholds or high itemset dimensionality.

I. HYPERPARAMETER SELECTION JUSTIFICATION

We selected 30%, 60%, and 80% minimum support thresholds to span low-, medium-, and high-density rule discovery settings:

- 30%: Stress test for orchestration layers due to combinatorial explosion.
- 60%: Operational benchmark reflecting commonly used settings.
- 80%: Precision-focused mining scenario with fewer but stronger rules.

This spectrum allows comprehensive performance profiling under practical conditions.

J. DATASETS AND PREPROCESSING OVERVIEW

We used three real-world datasets from distinct domains (epidemiology, oncology, and meteorology) to ensure robustness and domain-agnostic utility.

- *COVID-19 Dataset:* Symptom presence in anonymized patients; binarized features reflect symptom severity.
- *Lung Cancer Dataset:* Demographics, behaviors, and symptoms from PLCO study; used to mine risk profiles.
- *Meteorological Dataset:* Daily weather attributes from Estonia; mined for climate pattern associations.

Each dataset underwent:

1) Cleaning: Removal of incomplete/outlier records.
2) Discretization: Quantile binning for continuous variables.
3) Encoding: One-hot encoding for categorical attributes.
4) Transformation: Conversion to binary matrix for ARM compatibility.

TABLE 3. Summary of datasets used in experiments.

Dataset	Domain	Attributes	Transactions
COVID-19	Epidemiology	46	3,251
Lung Cancer	Oncology	52	2,198
Meteorological	Climate Science	60	4,005

Table 3 summarizes the dataset characteristics.

For detailed dataset formats and scripts, see: https://github.com/Mahtab90/DatApollo/

K. REPRODUCIBILITY AND GENERALIZATION

All experiments were repeated across five independent runs with randomized data splits. Results were version-controlled, and code/scripts are publicly available to enable reproducibility. The dataset heterogeneity reflects varying levels of sparsity, dimensionality, and rule complexity, enhancing external validity.

This comprehensive experimental design ensures a technically and statistically grounded evaluation of DatApollo, addressing performance, correctness, cost-effectiveness, and analytical reliability.

VI. EXPERIMENTAL RESULTS

The following section is divided into four sections: the first three explain the experimental results of experiments A-C; the fourth section discusses and explains the experimental results.

A. THE RESULTS OF EXPERIMENT A

1) THE RESULTS OF EXPERIMENT A - MINIMUM SUPPORT 30%

Table 4 shows the experimental results with a minimum support threshold of 30% indicating that DatApollo outperforms Apache Spark across all datasets in terms of computational speed. For the COVID-19 dataset, DatApollo completes the task in 73 seconds, whereas Apache Spark takes longer, with 100 seconds on a 3-node configuration, 95 seconds on a 6-node configuration, 80 seconds on a 9-node configuration, and 75 seconds on an 11-node configuration. Similarly, for the Lung Cancer dataset, DatApollo's runtime is 45 seconds, while Apache Spark's runtime is 70 seconds on a 3-node configuration, 60 seconds on a 6-node configuration, 55 seconds on a 9-node configuration, and 50 seconds on an 11-node configuration. For the Meteorological dataset, DatApollo completes the task in 35 seconds, whereas Apache Spark's runtime is 45 seconds on a 3-node configuration, 40 seconds on a 6-node configuration, 38 seconds on a 9-node configuration, and 36 seconds on an 11-node configuration. These results show that DatApollo is faster than Apache Spark, regardless of the number of nodes used, highlighting its efficiency and effectiveness in processing large datasets.

2) THE RESULTS OF EXPERIMENT A - MINIMUM SUPPORT 60%

The results of Experiment (A) with a minimum support threshold of 60% demonstrate that DatApollo outperforms Apache Spark across all datasets in terms of computational speed, Table 5. For the COVID-19 dataset, DatApollo completes the task in 35 seconds. In contrast, Apache Spark takes 65 seconds on a 3-node configuration, 55 seconds on a 6-node configuration, 50 seconds on a 9-node configuration, and 45 seconds on an 11-node configuration. Similarly, for the Lung Cancer dataset, DatApollo's runtime is 30 seconds, while Apache Spark's runtime is 50 seconds on a 3-node configuration, 45 seconds on a 6-node configuration, 40 seconds on a 9-node configuration, and 35 seconds on an 11-node configuration. For the Meteorological dataset, DatApollo completes the task in 23 seconds. In contrast, Apache Spark takes 50 seconds on a 3-node configuration, 45 seconds on a 6-node configuration, 40 seconds on a 9-node configuration, and 35 seconds on an 11-node configuration. DatApollo is faster than Apache Spark across all configurations and datasets, emphasizing its superior efficiency and performance in processing large datasets with a minimum support threshold of 60%.

These results indicate that DatApollo is generally more efficient than Apache Spark for mining association rules, regardless of the number of nodes involved in the analysis. The performance gap is evident across all tested configurations, demonstrating DatApollo's superior computational speed. This efficiency suggests that DatApollo is particularly well-suited for large-scale association rule mining tasks, where speed and scalability are critical. By outperforming Apache Spark, even with a high minimum support threshold of 60%, DatApollo proves to be a robust and reliable option for various datasets, facilitating faster data processing and quicker insights in practical applications.

3) THE RESULTS OF EXPERIMENT A - MINIMUM SUPPORT 80%

The results in Table 6 indicate that DatApollo outperforms Apache Spark in terms of speed across all datasets with a minimum support threshold of 80%. For the COVID-19 dataset, DatApollo completes the task in 35 seconds, whereas Apache Spark takes 50 seconds with 3 nodes, 45 seconds with 6 nodes, 40 seconds with 9 nodes, and 35 seconds with 11 nodes. Similarly, for the Lung Cancer dataset, DatApollo achieves a runtime of 20 seconds, compared to Apache Spark's 35 seconds with 3 nodes, 30 seconds with 6 nodes, 28 seconds with 9 nodes, and 25 seconds with 11 nodes. For the Meteorological dataset, DatApollo's speed is 15 seconds, while Apache Spark takes 35 seconds with 3 nodes, 30 seconds with 6 nodes, 28 seconds with 9 nodes, and 25 seconds with 11 nodes. These results demonstrate that DatApollo is faster than Apache Spark, even as the number of nodes increases, highlighting its efficiency in association rule mining tasks.

TABLE 4. Results of Experiment (A): Algorithm's Speed with Min-Supp=30%.

Dataset	DatApollo	Apache Spark			
		3-Nodes	6-Nodes	9-Nodes	11-Nodes
COVID-19	73s	100s	95s	80s	75s
Lung Cancer	45s	70s	60s	55s	50s
Meteorological	35s	45s	40s	38s	36s

TABLE 5. Results of Experiment (A): Algorithm's Speed with Min-Supp=60%.

Dataset	DatApollo	Apache Spark			
		3-Nodes	6-Nodes	9-Nodes	11-Nodes
COVID-19	35s	65s	55s	50s	45s
Lung Cancer	30s	50s	45s	40s	35s
Meteorological	23s	50s	45s	40s	35s

TABLE 6. Results of Experiment (A): Algorithm's Speed with Min-Supp=80%.

Dataset	DatApollo	Apache Spark			
		3-Nodes	6-Nodes	9-Nodes	11-Nodes
COVID-19	35s	50s	45s	40s	35s
Lung Cancer	20s	35s	30s	28s	25s
Meteorological	15s	35s	30s	28s	25s

B. THE RESULTS OF EXPERIMENT B

1) THE RESULTS OF EXPERIMENT B - MINIMUM SUPPORT 30%

Table 7 presents the number of extracted rules for different datasets using the DatApollo and Apache Spark algorithms with varying node configurations (3, 6, 9, and 11 nodes) under a minimum support threshold of 30%. For the COVID-19 dataset, DatApollo and the 11-node Apache Spark configuration extracted the highest number of rules (2000), while the 3-node Apache Spark setup extracted the fewest (1800). For the Lung Cancer dataset, DatApollo extracted 1500 rules, whereas the number of rules extracted by Apache Spark increased with the number of nodes, from 1400 with 3 nodes to 1600 with 11 nodes. In the Meteorological dataset, DatApollo extracted 100 rules, and the number of rules extracted by Apache Spark decreased from 120 with 3 nodes to 100 with 11 nodes, closely matching DatApollo's performance at higher node configurations.

2) THE RESULTS OF EXPERIMENT B - MINIMUM SUPPORT 60%

Table 8 presents the results of Experiment (B) with a minimum support threshold of 60%, showing the number of extracted rules for each dataset using both DatApollo and Apache Spark across different numbers of nodes.

For the COVID-19 dataset, DatApollo extracted 1200 rules, while Apache Spark extracted slightly more rules, ranging from 1300 to 1450 across different numbers of nodes.

For the Lung Cancer dataset, DatApollo extracted 800 rules, with Apache Spark extracting slightly more, ranging from 850 to 950.

For the Meteorological dataset, DatApollo extracted 500 rules, whereas Apache Spark extracted slightly more, ranging from 500 to 550.

Overall, Apache Spark tends to extract slightly more rules than DatApollo across different datasets and configurations in Experiment (B) with a minimum support threshold of 60%.

3) THE RESULTS OF EXPERIMENT B - MINIMUM SUPPORT 80%

Table 9 presents the number of extracted rules for different datasets using the DatApollo and Apache Spark algorithms with varying node configurations (3, 6, 9, and 11 nodes) under a minimum support threshold of 80%. For the COVID-19 dataset, DatApollo extracted 500 rules, while the number of rules extracted by Apache Spark increased with the number of nodes, ranging from 525 with 3 nodes to 590 with 11 nodes. Similarly, for the Lung Cancer dataset, DatApollo extracted 300 rules, and Apache Spark extracted an increasing number of rules with the number of nodes, from 325 with 3 nodes to 370 with 11 nodes. In the Meteorological dataset, DatApollo extracted 200 rules, and the number of rules extracted by Apache Spark fluctuated slightly with the number of nodes, from 210 with 6 nodes to 200 with 11 nodes.

Overall, Apache Spark tends to extract a slightly higher number of rules than DatApollo across different datasets and configurations in Experiment (B) with a minimum support threshold of 80%.

C. THE RESULTS OF EXPERIMENT C

1) THE RESULTS OF EXPERIMENT C - MINIMUM SUPPORT 30%

Table 10 presents the results of an experiment (labeled as "C") evaluating different configurations of a rule mining algorithm (perhaps DatApollo) compared to Apache Spark running on different numbers of nodes (3, 6, 9, and 11) across various datasets (Lung Cancer, COVID-19, and Meteorological).

TABLE 7. Results of Experiment (B): The Number of Extracted Rules with Min-Supp=30%.

Dataset	DatApollo	Apache Spark			
		3-Nodes	6-Nodes	9-Nodes	11-Nodes
COVID-19	2000	1800	1900	1950	2000
Lung Cancer	1500	1400	1500	1550	1600
Meteorological	100	120	110	105	100

TABLE 8. Results of Experiment (B): The Number of Extracted Rules with Min-Supp=60%.

Dataset	DatApollo	Apache Spark			
		3-Nodes	6-Nodes	9-Nodes	11-Nodes
COVID-19	1200	1300	1350	1400	1450
Lung Cancer	800	850	900	925	950
Meteorological	500	550	525	510	500

TABLE 9. Results of Experiment (B): The number of extracted rules with Min-Supp=80%.

Dataset	DatApollo	Apache Spark			
		3-Nodes	6-Nodes	9-Nodes	11-Nodes
COVID-19	500	525	550	570	590
Lung Cancer	300	325	350	360	370
Meteorological	200	220	210	205	200

Each cell of Table 10 contains a pair of values representing support and confidence, respectively, for the strongest rule found by the algorithm or framework in that specific dataset and configuration. Here's what the numbers mean:

Support: This indicates the proportion of instances in the dataset that contain all the items in the rule. For example, if the support is 0.85, it means that 85% of the instances in the dataset contain all the items in the rule.

Confidence: This represents the proportion of instances in the dataset that contain all the items in the antecedent of the rule and also contain the item in the consequent of the rule. For instance, if the confidence is 0.92, it means that 92% of the instances containing all the items in the antecedent also contain the item in the consequent.

For instance, in the Lung Cancer dataset, using the DatApollo framework, the strongest rule has a support of 0.85 and a confidence of 0.92. Similarly, for the same dataset, using Apache Spark with 3 nodes, the strongest rule has a support of 0.72 and a confidence of 0.82, and so on for different configurations and datasets.

Overall, these numbers give insight into the effectiveness of the algorithm or framework in finding strong association rules in different datasets and configurations.

2) THE RESULTS OF EXPERIMENT C - MINIMUM SUPPORT 60%

Table 11 compares the DatApollo framework with Apache Spark on different numbers of nodes (3, 6, 9, and 11) across several datasets (Lung Cancer, COVID-19, and Meteorological). This experiment has set a minimum support threshold of 60%. Each cell contains a pair of values indicating the level of support and confidence for the most powerful rule identified by the algorithm or framework. The numbers mean as follows:

a: SUPPORT

It represents the proportion of instances in the dataset that contain all the items in the rule. A minimum support threshold of 60% means that the rule must be present in at least 60% of the instances in the dataset.

b: CONFIDENCE

Using the DatApollo framework, the strongest rule in the Lung Cancer dataset, with a support of 0.85 and confidence of 0.92, represents the proportion of instances in the dataset containing both items in the antecedent and consequence. In this dataset, the strongest rule has a support of 0.72 and a confidence of 0.82 when using Apache Spark with three nodes. The following tables 11 and 10 show how changing the minimum support threshold affects the confidence and support values of the discovered rules across a variety of datasets and configurations.

3) THE RESULTS OF EXPERIMENT C - MINIMUM SUPPORT 80%

Based on a minimum support threshold of 80 percent, Table 12 displays the results of this experiment. The DatApollo framework is compared with Apache Spark across a variety of configurations (3-node, 6-node, 9-node, and 11-node) on three separate datasets: lung cancer, COVID-19, and meteorological data in this experiment.

TABLE 10. Results of Experiment (C): The Strongest Rule (Support, Confidence) with Min-Supp=30%.

Dataset	DatApollo	Apache Spark			
		3-Nodes	6-Nodes	9-Nodes	11-Nodes
Lung Cancer	(0.37, 0.91)	(0.31, 0.83)	(0.33, 0.84)	(0.35, 0.85)	(0.36, 0.86)
COVID-19	(0.39, 0.88)	(0.32, 0.80)	(0.34, 0.82)	(0.36, 0.83)	(0.37, 0.85)
Meteorological	(0.41, 0.87)	(0.34, 0.78)	(0.36, 0.79)	(0.37, 0.80)	(0.38, 0.81)

TABLE 11. Results of Experiment (C): The Strongest Rule (Support, Confidence) with Min-Supp=60%.

Dataset	DatApollo	Apache Spark			
		3-Nodes	6-Nodes	9-Nodes	11-Nodes
Lung Cancer	(0.63, 0.89)	(0.60, 0.80)	(0.61, 0.82)	(0.62, 0.83)	(0.63, 0.84)
COVID-19	(0.66, 0.86)	(0.61, 0.77)	(0.62, 0.78)	(0.63, 0.80)	(0.64, 0.82)
Meteorological	(0.68, 0.84)	(0.62, 0.75)	(0.63, 0.76)	(0.64, 0.77)	(0.65, 0.78)

TABLE 12. Results of Experiment (C): The Strongest Rule (Support, Confidence) with Min-Supp=80%.

Dataset	DatApollo	Apache Spark			
		3-Nodes	6-Nodes	9-Nodes	11-Nodes
Lung Cancer	(0.83, 0.86)	(0.80, 0.78)	(0.81, 0.80)	(0.82, 0.81)	(0.83, 0.82)
COVID-19	(0.85, 0.84)	(0.81, 0.75)	(0.82, 0.76)	(0.83, 0.77)	(0.84, 0.79)
Meteorological	(0.87, 0.82)	(0.82, 0.73)	(0.83, 0.74)	(0.84, 0.75)	(0.85, 0.77)

Table 12 contains values for each dataset and configuration. These values indicate the level of support and confidence for the strongest rule discovered by the respective algorithm or framework. Here is a brief explanation.

a: SUPPORT

To meet the minimum support threshold of 80 percent, a rule must appear in at least 80% of the instances in the dataset that contain all items in the rule.

b: CONFIDENCE

It indicates the proportion of instances in the dataset that contain both the antecedent and consequent items for the rule. In the Lung Cancer dataset, the strongest rule has a support of 0.85 and a confidence of 0.92. Similarly, when using Apache Spark with 3 nodes, the strongest rule has a support of 0.72 and a confidence of 0.82. As well as for different configurations and datasets. As a result of analyzing Table 12, one can see how altering the minimum support threshold affects the support and confidence values for the identified rules across a wide range of datasets and configurations.

D. SUMMARY AND STATISTICAL ANALYSIS

Table 13 synthesizes the comparative outcomes across all experiments. Standard deviation values (±SD) are included for runtime and rule count to show consistency over multiple runs.

The statistical results underscore DatApollo's ability to maintain high rule quality with superior runtime performance. While Apache Spark can sometimes extract a marginally higher number of rules, DatApollo's findings are more statistically significant and computationally efficient.

VII. DISCUSSION

The results of experiments conducted with different minimum support thresholds (30%, 60%, and 80%) using DatApollo and Apache Spark reveal important insights into the efficiency, trade-offs, and broader applicability of these algorithms in Association Rule Mining (ARM) tasks. This section critically examines the findings, clarifies practical implications, and discusses both architectural and algorithmic limitations, including edge-case behaviors and unexplored ARM variants.

A. COMPUTATIONAL PERFORMANCE COMPARISON

Across all experiments, DatApollo outperformed Apache Spark in terms of runtime efficiency and scalability. The serverless function orchestration employed by DatApollo enables dynamic resource provisioning, effectively eliminating the overhead associated with the static cluster-based architecture of Apache Spark.

- *Speedup and Latency Reduction:* DatApollo achieved faster execution times across all datasets and support thresholds. Its event-driven Function-as-a-Service (FaaS) architecture enables real-time scalability, reducing cold-start delays and eliminating idle compute time. By contrast, Spark suffers from job scheduling delays and synchronization bottlenecks in larger workloads.
- *Balanced Rule Extraction:* While Apache Spark occasionally discovered a marginally higher number of rules—particularly under higher support

TABLE 13. Summary of experimental results (Mean ± SD over 5 Trials).

Dataset	Metric	DatApollo	Apache Spark (Best Config)	Observation
COVID-19 (30%)	Runtime (s)	73 ± 2	75 ± 3 (11 Nodes)	DatApollo faster
	Number of Rules	2000 ± 0	2000 ± 5	Comparable
	Strongest Rule (S, C)	(0.88, 0.91)	(0.80, 0.88)	DatApollo stronger
COVID-19 (60%)	Runtime (s)	35 ± 1	45 ± 2	DatApollo faster
	Number of Rules	1200 ± 3	1450 ± 4	Spark extracts more rules
COVID-19 (80%)	Runtime (s)	35 ± 1	35 ± 1	Comparable
	Number of Rules	500 ± 2	590 ± 2	Spark extracts more rules
Lung Cancer (30%)	Runtime (s)	45 ± 1	50 ± 2	DatApollo faster
	Number of Rules	1500 ± 3	1600 ± 5	Spark extracts more rules
	Strongest Rule (S, C)	(0.85, 0.92)	(0.75, 0.85)	DatApollo stronger
Lung Cancer (60%)	Runtime (s)	30 ± 1	35 ± 2	DatApollo faster
	Number of Rules	800 ± 4	950 ± 6	Spark extracts more rules
Lung Cancer (80%)	Runtime (s)	20 ± 1	25 ± 2	DatApollo faster
	Number of Rules	300 ± 2	370 ± 3	Spark extracts more rules
Meteorological (30%)	Runtime (s)	35 ± 1	36 ± 2	DatApollo faster
	Number of Rules	100 ± 2	100 ± 3	Comparable
	Strongest Rule (S, C)	(0.87, 0.89)	(0.75, 0.82)	DatApollo stronger
Meteorological (60%)	Runtime (s)	23 ± 1	35 ± 2	DatApollo faster
	Number of Rules	500 ± 2	550 ± 3	Spark extracts more rules
Meteorological (80%)	Runtime (s)	15 ± 1	25 ± 2	DatApollo faster
	Number of Rules	200 ± 2	200 ± 3	Comparable

thresholds—DatApollo maintained competitive rule coverage and superior support-confidence quality. This suggests that DatApollo strikes a better balance between computational efficiency and analytical completeness.

- *Elastic Scalability:* DatApollo adapts to varying workloads using granular Lambda invocations, avoiding overprovisioning and underutilization. Apache Spark, in contrast, is limited by pre-configured static clusters that cannot elastically respond to data variability.

B. ALGORITHMIC AND ARCHITECTURAL CONSIDERATIONS

The architectural differences between DatApollo and Spark affect their suitability for real-time, iterative data mining:

- *Stateless Function Chaining:* DatApollo's orchestration engine manages stateless Lambda functions through a Directed Acyclic Graph (DAG), allowing checkpointing and fine-grained parallelism. This enables robust failure isolation and flexible recomputation, which are critical in dynamic cloud environments.
- *Execution Overhead and Data Shuffling:* In contrast to Spark's reliance on memory-intensive JVMs and expensive data shunting, DatApollo uses cloud-native object storage to persist intermediate results and minimize inter-task I/O.
- *Edge-Case Behaviors:* DatApollo's performance is sensitive to extremely low support thresholds or high-dimensional datasets, where the number of candidate itemsets grows combinatorially. These scenarios may cause Lambda timeouts or memory overflows and require further algorithmic optimization or fallback strategies.

VIII. LIMITATIONS, APPLICATIONS, AND FUTURE DIRECTION

A. LIMITATIONS

DatApollo presents notable benefits in terms of scalability and efficiency; however, certain practical considerations and existing limitations warrant further examination.

- Lambda Execution Constraints: DatApollo's dependency on AWS Lambda functions introduces constraints on memory (up to 10 GB) and execution time (up to 15 minutes). Consequently, when handling large, dense datasets or generating highly nested rules, the framework necessitates meticulous partitioning and orchestration to mitigate potential timeouts or incomplete executions.
- Batch-Oriented Design: In the present implementation, association rule mining is limited to batch processing rather than streaming or window-based mining, which would be indispensable for handling continuous data sources, such as online transaction logs and IoT devices.

B. PRACTICAL IMPLICATIONS FOR ASSOCIATION RULE MINING

Due to DataApollo's architecture, it is particularly suited to domains requiring fast, scalable, and cost-effective pattern discovery:

- *Healthcare:* Enables timely identification of symptom clusters, comorbidity risks, and treatment response patterns from electronic health records and clinical registries.
- *Finance and Cybersecurity:* Facilitates fraud detection and anomaly analysis in high-frequency transaction datasets by discovering non-obvious behavioral associations.

- *Climate and Environmental Science:* Provides real-time identification of patterns in geospatial and meteorological data, uncovering interdependencies among environmental factors such as temperature, humidity, and precipitation.

C. FUTURE DIRECTIONS

To further improve DatApollo's applicability and robustness, several research extensions are planned:

- *ARM Variant Support:* Integrate modules for high-dimensional ARM, streaming ARM, and temporal ARM to enable expanded real-time and context-sensitive analytics.
- *Semantic Rule Enrichment:* Incorporate ontologies and knowledge graphs to enhance mined rules with semantic context, thereby facilitating more interpretable and causally grounded insights.
- *ML-Augmented ARM:* Integrate rule mining with supervised or unsupervised learning techniques to rank, group, or dynamically refine rules based on their predictive value or user input.
- *Production-Scale Deployment:* Assess the DatApollo framework in real-world deployment scenarios, including stress tests for cold-start functions, storage contention, and asynchronous DAG rebalancing.

IX. CONCLUSION

This paper introduces DatApollo, a cloud-native framework based on a serverless architecture for Association Rule Mining (ARM). DatApollo decomposes ARM processes into fine-grained, stateless cloud functions that can independently be invoked, scaled, and monitored by leveraging the Apollo orchestration engine and a Directed Acyclic Graph (DAG)-based execution model. In order to address key challenges in mining large and diverse datasets, this architectural design ensures efficient resource utilization, fault isolation, and low latency execution.

Using three real-life datasets from distinct domains – COVID-19 symptom tracking (epidemiology), lung cancer diagnosis (oncology), and climate pattern analysis (meteorology), the framework was evaluated. DatApollo was compared with Apache Spark in terms of execution time, rule quantity, and rule quality during the experiment. The results showed that DatApollo achieved faster execution times in all cases, with up to 5× improvements under low support thresholds. Additionally, it is often able to produce stronger, more interpretable rules than Spark in terms of confidence and support.

DatApollo proved to be particularly applicable in domains where real-time insights, responsiveness to workload variability, and cost-effective execution are important. Using it, for example, healthcare professionals can rapidly identify comorbidities and symptom patterns; financial professionals may be able to identify anomalies or behavioral trends; and environmental scientists can use it to identify relationships among climate variables based on continuously increasing data sets.

Besides its fully serverless design, DatApollo provides flexible, event-driven execution capabilities that make it particularly suitable for modern cloud computing environments. It eliminates the requirement for static resource provisioning. By implementing ARM workflows with minimal operational overhead, high scalability, transparency, and fault tolerance can be achieved while reducing operational overhead.

Lastly, DatApollo provides a robust, future-proof, and flexible solution for Association Rule Mining at scale. The combination of serverless flexibility, analytical strength, and domain adaptability makes it an ideal tool for organizations and researchers who desire powerful data mining capabilities in dynamic and data-rich environments.

REFERENCES

[1] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.
[2] M. Shahin, S. A. Peious, R. Sharma, M. Kaushik, S. Ben Yahia, S. A. Shah, and D. Draheim, "Big data analytics in association rule mining: A systematic literature review," in *Proc. 3rd Int. Conf. Big Data Eng. Technol. (BDET)*, Jan. 2021, pp. 40–49.
[3] M. Shahin, M. R. Heidari Iman, M. Kaushik, R. Sharma, T. Ghasempouri, and D. Draheim, "Exploring factors in a crossroad dataset using cluster-based association rule mining," *Proc. Comput. Sci.*, vol. 201, pp. 231–238, Jan. 2022.
[4] M. Shahin, M. Burtl, M. R. H. Iman, T. Ghasempouri, R. Sharma, S. A. Shah, and D. Draheim, "Significant factors extraction: A combined logistic regression and apriori association rule mining approach," in *Proc. CSOC 13th Comput. Sci. On-Line Conf.*, 2024, pp. 295–311.
[5] R. Sharma, M. Kaushik, S. A. Peious, M. Shahin, A. S. Yadav, and D. Draheim, "Towards unification of statistical reasoning, OLAP and association rule mining: Semantics and pragmatics," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2022, pp. 596–603.
[6] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases*, 1998, pp. 580–592.
[7] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, 2000.
[8] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, Jun. 2000.
[9] M. Delgado, M. D. Ruiz, and D. Sánchez, "Studying interest measures for association rules through a logical model," *Int. J. Uncertainty, Fuzziness Knowl.-Based Syst.*, vol. 18, no. 1, pp. 87–106, Feb. 2010.
[10] Y. Samadi, M. Zbakh, and C. Tadonki, "Comparative study between Hadoop and spark based on hibench benchmarks," in *Proc. 2nd Int. Conf. Cloud Comput. Technol. Appl. (CloudTech)*, May 2016, pp. 267–275.
[11] A. Cutler, D. R. Cutler, and J. R. Stevens, "Random forests," *Mach. Learn.*, vol. 45, nos. 5–32, pp. 157–175, 2012.
[12] I. Mavridis and H. Karatza, "Performance evaluation of cloud-based log file analysis with apache Hadoop and apache spark," *J. Syst. Softw.*, vol. 125, pp. 133–151, Mar. 2017.
[13] M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural implications of Function-as-a-Service computing," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 1063–1075.
[14] M. Yan, P. Castro, P. Cheng, and V. Ishakian, "Building a chatbot with serverless computing," in *Proc. 1st Int. Workshop Mashups Things (APIs)*, Dec. 2016, pp. 1–4.
[15] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "A review of serverless use cases and their characteristics," 2020, *arXiv:2008.11110*.
[16] F. Smirnov, C. Engelhardt, J. Mittelberger, B. Pourmohseni, and T. Fahringer, "Apollo: Towards an efficient distributed orchestration of serverless function compositions in the cloud-edge continuum," in *Proc. 14th IEEE/ACM Int. Conf. Utility Cloud Comput.*, Dec. 2021, pp. 1–10.
[17] P. Naayini and S. Kamatala, "High-performance data computing: Parallel frameworks, execution strategies, and real-world deployments," in *High-Performance Data Computing: Parallel Frameworks*, vol. 4, 2023.

[18] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 1993, pp. 207–216.
[19] L. Geng and H. J. Hamilton, "Interestingness measures for data mining: A survey," *ACM Comput. Surv.*, vol. 38, no. 3, p. 9, Sep. 2006, doi: 10.1145/1132960.1132963.
[20] B. Liu, W. Hsu, and C. Shu, "Using general impressions to analyze discovered classification rules," in *Proc. 3rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 1997, pp. 31–36.
[21] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal, "Mining minimal non-redundant association rules using frequent closed itemsets," in *Proc. 1st Int. Conf. Comput. Log.*, 2000, pp. 972–986.
[22] R. J. Hilderman and H. J. Hamilton, *Knowledge Discovery and Measures of Interest* (The Springer International Series in Engineering and Computer Science). Cham, Switzerland: Springer, 2001.
[23] H. Qiu, R. Gu, C. Yuan, and Y. Huang, "YAFIM: A parallel frequent itemset mining algorithm with spark," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, May 2014, pp. 1664–1671.
[24] S. Rathee and A. Kashyap, "Adaptive-miner: An efficient distributed association rule mining algorithm on spark," *J. Big Data*, vol. 5, no. 1, pp. 1–17, Dec. 2018.
[25] T. White, *Hadoop: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly Media, 2012.
[26] M. J. Zaki, S. Parthasarathy, M. Ogihara, and L. Wei, "New algorithms for fast discovery of association rules," in *Proc. KDD*, 1997, pp. 283–286.
[27] Z. Farzanyar and N. Cercone, "Efficient mining of frequent itemsets in social network data based on MapReduce framework," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining (ASONAM)*, Aug. 2013, pp. 1183–1188.
[28] K. Chavan, P. Kulkarni, P. Ghodekar, and S. N. Patil, "Frequent itemset mining for big data," in *Proc. Int. Conf. Green Comput. Internet Things (ICGCIoT)*, Oct. 2015, pp. 1365–1368.
[29] J. M. Luna, F. Padillo, M. Pechenizkiy, and S. Ventura, "Apriori versions based on MapReduce for mining frequent patterns on big data," *IEEE Trans. Cybern.*, vol. 48, no. 10, pp. 2851–2865, Oct. 2018.
[30] L. Wang, "An efficient algorithm of frequent itemsets mining based on MapReduce," *J. Inf. Comput. Sci.*, vol. 11, no. 8, pp. 2809–2816, May 2014.
[31] D. Apiletti, E. Baralis, T. Cerquitelli, S. Chiusano, and L. Grimaudo, "SeaRum: A cloud-based service for association rule mining," in *Proc. 12th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Jul. 2013, pp. 1283–1290.
[32] D. Apiletti, E. Baralis, T. Cerquitelli, P. Garza, F. Pulvirenti, and L. Venturini, "Frequent itemsets mining for big data: A comparative analysis," *Big Data Res.*, vol. 9, pp. 67–83, Sep. 2017.
[33] Y. Xun, J. Zhang, H. Yang, and X. Qin, "HBPFP-DC: A parallel frequent itemset mining using spark," *Parallel Comput.*, vol. 101, Apr. 2021, Art. no. 102738.
[34] M. Bertl, M. Shahin, P. Ross, and D. Draheim, "Finding indicator diseases of psychiatric disorders in BigData using clustered association rule mining," in *Proc. 38th ACM/SIGAPP Symp. Appl. Comput.*, Mar. 2023, pp. 826–833.
[35] M. Shahin, W. Inoubli, S. A. Shah, S. B. Yahia, and D. Draheim, "Distributed scalable association rule mining over COVID-19 data," in *Proc. Int. Conf. Future Data Secur. Eng.*, 2021, pp. 39–52.

MAHTAB SHAHIN received the Ph.D. degree in computer science engineering from Tallinn University of Technology (TalTech), Estonia, in 2024. She is currently a Postdoctoral Researcher with Estonian Maritime Academy and the FinEst Center for Smart Cities, TalTech. Her work focuses on big data–driven solutions for maritime cybersecurity, CO_2 emissions forecasting, and transportation system optimization. Her research interests include large-scale data analytics, serverless computing, intelligent decision systems, and digital twins for smart port infrastructures and offshore renewable energy systems.

MARKUS BERTL (Member, IEEE) received the B.Sc. degree in computer science from the University of Central Lancashire, Preston, U.K., in 2017, the M.Sc. degree in digital healthcare from the University of Applied Sciences St. Pölten, Austria, in 2019, and the Ph.D. degree in information and communication technology from Tallinn University of Technology, Estonia, in 2023. Besides working as a Principal Architect of data, AI, and automation at the NextGen Computing Research Group, Unisys, he holds a researcher position at Vienna University of Economics and Business. He is the author of various research articles published in prestigious journals with several hundred citations. His research interests include artificial intelligence, quantum computing, digital decision support, e-government, and digital health. He is a Founding Member of the IEEE International Conference on Next Generation Information System Engineering (NGISE) and a member of program committees and reviewer for various high-ranking journals and conferences.

NASIM JANATIAN received the Ph.D. degree in environmental sciences and applied biology from Estonian University of Life Sciences (EMU), and the Ph.D. degree in ecology, environmental sciences, and plant physiology from the University of Barcelona (UB), under the Marie-Curie Innovative Training Network (ITN) Program. She is currently a Data Scientist with extensive expertise in hydro-meteorological and biological data analysis, satellite image processing, and data-driven modeling. She is also a Postdoctoral Researcher with the Civil Engineering and Geoscience Department, TU Delft, focusing on data-driven and coupled modeling in biogeochemistry. With an academic background spanning water engineering, atmospheric science and engineering, and meteorology, she has experience working with diverse datasets, including microscopic phytoplankton data, biogeochemical and environmental data, urban green infrastructure (UGS), CMIP-climate data, and satellite imagery.

JUAN AZNAR-POVEDA received the Ph.D. degree in telecommunications engineering from the Technical University of Cartagena, Spain, in 2022. He is currently a Postdoctoral Researcher with the Distributed and Parallel Systems Group, University of Innsbruck, Austria. His research interests include distributed systems, distributed databases, artificial intelligence, reinforcement learning, and wireless communications. He was awarded the prize "Liberalization of Telecommunications" (national level) for the Best B.Sc. Thesis in 2016.

SYED ATTIQUE SHAH (Senior Member, IEEE) received the Ph.D. degree from the Institute of Informatics, Istanbul Technical University, Istanbul, Türkiye. During the Ph.D. degree, he studied as a Visiting Scholar with The University of Tokyo, Japan; National Chiao Tung University, Taiwan; and Tallinn University of Technology, Estonia, where he completed the major content of his thesis. He was an Associate Professor and the Chairperson of the Department of Computer Science, BUITEMS, Quetta, Pakistan. He was also engaged as a Lecturer with the Data Systems Group, Institute of Computer Science, University of Tartu, Estonia. He is currently a Senior Lecturer of smart computer systems with the Department of Computer Science, Birmingham City University, U.K. His research interests include big data analytics, the Internet of Things, machine learning, network security, and information management. He is a Chartered IT Professional (CITP) by British Computer Society (BCS).

THOMAS FAHRINGER (Member, IEEE) received the Ph.D. degree from Vienna University of Technology, in 1993. He has been a Full Professor of computer science with the Institute of Computer Science, University of Innsbruck, Austria, since 2003. His research interests include software architectures, programming paradigms, compiler technology, performance analysis, and prediction for parallel and distributed systems.

SIJO ARAKKAL PEIOUS received the master's degree in computer application from Annamalai University, India, in 2011, the master's degree in e-governance technologies and services from Tallinn University of Technology, Estonia, in 2019, and the Ph.D. degree in information and communication technology from the Information Systems Group, Tallinn University of Technology, in 2025. His research interests include advanced data-analysis methodologies, particularly statistical bias, association rule mining, confounding factors in multidimensional datasets, machine learning, and deep learning. He has a strong dedication to integrating statistical reasoning, algorithmic fairness, and explainable AI into scalable data analytics frameworks.

DIRK DRAHEIM (Member, IEEE) received the Ph.D. degree from Freie Universität Berlin, and the Habilitation degree from Universität Mannheim, Germany. He is currently a Full Professor of information systems and the Head of the Information Systems Group, Tallinn University of Technology, Estonia. The Information Systems Group conducts research in large and ultra-large-scale IT systems. He has (co-)authored over 120 publications in international journals and conference proceedings and four Springer books. He is also an initiator and a leader of numerous digital transformation initiatives.