INFA-FinOps for Cloud Data Integration

Atam Prakash Agrawal
Informatica
atagrawal@informatica.com

Anant Mittal
Informatica
amittal@informatica.com

Shivangi Srivastava
Informatica
ssrivastava@informatica.com

Michael Brevard
Informatica
mbrevard@informatica.com

Valentin Moskovich
Informatica
vmoskovich@informatica.com

Mosharaf Chowdhury
University of Michigan
mosharaf@umich.edu

***Abstract*—Over the past decade, businesses have migrated to the cloud for its simplicity, elasticity, and resilience. Cloud ecosystems offer a variety of computing and storage options, enabling customers to choose configurations that maximize productivity. However, determining the right configuration to minimize cost while maximizing performance is challenging, as workloads vary and cloud offerings constantly evolve. Many businesses are overwhelmed with choice overload and often end up making suboptimal choices that lead to inflated cloud spending and/or poor performance.**

In this paper, we describe INFA-FinOps, an automated system that helps Informatica customers strike a balance between cost efficiency and meeting SLAs for Informatica Advanced Data Integration (aka CDI-E) workloads. We first describe common workload patterns observed in CDI-E customers and show how INFA-FinOps selects optimal cloud resources and configurations for each workload, adjusting them as workloads and cloud ecosystems change. It also makes recommendations for actions that require user review or input. Finally, we present performance benchmarks on various enterprise use cases and conclude with lessons learned and potential future enhancements.

I. Introduction

Maximizing resource utilization while meeting service-level agreements (SLAs) is a perennial problem. Over the last decade, however, resource management complexity has increased significantly due to widespread transitions to cloud computing. This is because the cloud offers limitless possibilities for virtual resource instantiation and continues to expand its options. The ability to choose the best configuration for a given workload is powerful, but non-experts often cannot choose the right configurations for their workloads, nor can they predict or react to dynamic workload changes [1].

Informatica Advanced Data Integration (CDI-E [2]) users are not immune to this problem either. We hear questions like:

- How do we ensure SLA compliance while minimizing costs?
- Which instances and storage should we provision?
- How and when do we scale our resources up and down?
- How do we get visibility into our spending/performance?

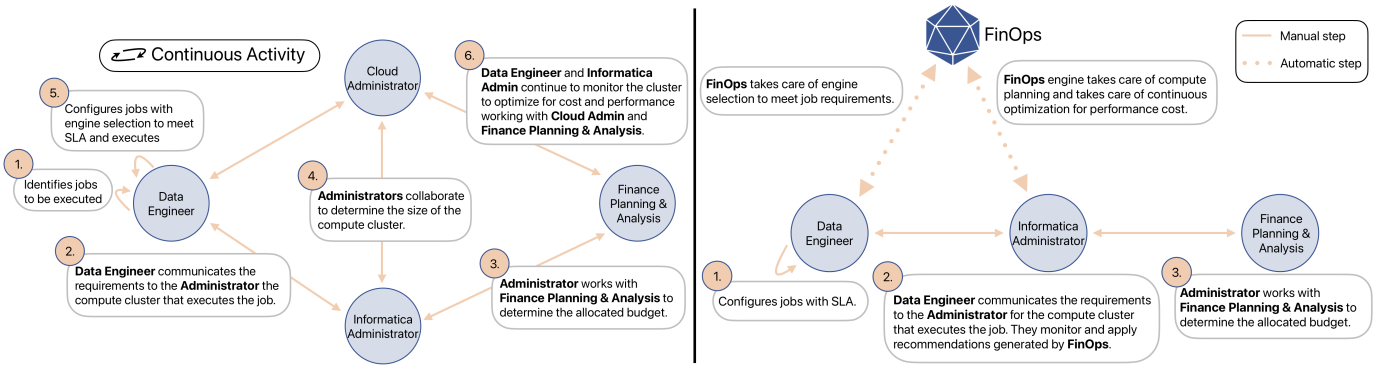
Without carefully tuning their configurations, customers end up largely over-provisioning resources and face runaway cloud bills.

There is no silver bullet because workloads vary widely across customers. They also change frequently, and customers cannot manually configure every time. Cloud ecosystems change as well. New instances and storage types are added, old ones are removed, and prices fluctuate. Spot Instances exacerbate these challenges. In short, it is non-trivial for customers to optimize workloads and keep them optimized over time. To understand the unique needs of CDI-E customers, we dedicated a year to extensively research and consult customer use cases. Based on customer feedback, we launched INFA-FinOps in October 2023 to reconcile their workload SLA requirements and budgetary constraints.

This paper focuses on the design and implementation of INFA-FinOps, which optimizes Informatica CDI-E resource management to enable our customers to find a balance between cost and performance. Informatica CDI-E is a fully-managed data engineering tool that utilizes Apache Spark on Kubernetes as the execution engine. INFA-FinOps hides the complexity of CDI-E resource management and performs continuous resource optimizations to minimize cost while maximizing performance of CDI-E workloads, all while adhering to customer SLAs and budgets. Specifically, INFA-FinOps implements three key use-inspired features:

- **Cluster tuning:** Boosting cluster efficiency by adaptively selecting instance types, resizing them, auto-scaling the cluster, and intelligently scheduling cluster resources.
- **Job tuning:** Fine-tuning individual jobs and recommending the right engine patterns to meet SLAs.
- **Recommendations:** Offering budget guidance and region co-location strategies, and providing job-specific recommendations such as choosing the optimal execution engine.

Outline. Section II provides a quick background on Informatica CDI-E and its workflow, along with FinOps challenges faced by our customers. Section III gives an overview of INFA-FinOps system architecture. Sections IV describes our cluster algorithms that automatically address most cost-related challenges. Section V describes how it provides recommendations when automated decision making is not practical. We present an analysis on the performance of common industry workloads using INFA-FinOps in Section VI and conclude in Section VII with lessons learned and our future vision.



(a) Without FinOps (b) With FinOps

Fig. 1: Informatica CDI-E life cycle comparison.

II. Background

Informatica, founded in 1993, is a cloud data engineering company focused on data engineering and warehousing. One of its core products, CDI-E, is designed for processing large-scale data in cloud environments using Apache Spark on Kubernetes for high performance and scalability.

A typical CDI-E deployment involves the following personas: the *Informatica Admin* manages Informatica tools; the *Data Engineer* develops the *mapping task*[1] for execution; the *Cloud Admin* manages cloud accounts such as the AWS administrative account; and the *Financial Planning and Accounting Team* is responsible for budgets and financial planning. Traditionally, they need to perform several actions before getting started "Fig. 1a". The process is iterative, as it is hard to determine the right cluster size upfront, and recalculations are needed to achieve a balance between cost and performance. Different teams find themselves in an ongoing loop of planning, estimating, deploying, optimizing, monitoring, and governing. This cycle must be revisited whenever any deployment factor changes, such as data volume, workload pattern, and cloud pricing.

INFA-FinOps simplifies this process so that an Informatica admin only needs to receive the budget from the financial planning and accounting team and the SLA from the data engineer. Then, it sets up and maintains the cluster "Fig. 1b". The design and implementation of the solution will be the focus of this paper.

A. Challenges

Balancing cloud costs and meeting SLAs presents complex challenges that organizations must navigate to ensure optimal performance and financial efficiency. To find a balance, customers must understand both the cloud ecosystem and the compute and storage requirements for the job. Unfortunately, they run into major issues:

Rightsizing resources: Rightsizing resources is crucial for balancing cost and performance. Over-provisioning wastes resources, while under-provisioning compromises SLAs. It involves multiple factors, such as selecting instance types, storage options, and the number of VMs, along with associated costs. Customers must also account for job scheduling, node scaling, Spot Instance usage, and other factors essential for cluster setup.

Changing workload patterns: As workloads evolve, initial resource allocations may become outdated, requiring ongoing adjustments for performance and cost-efficiency. Cloud workloads can vary greatly, with peak demand followed by low utilization. Meeting SLAs during peak times often requires additional capacity, which can be costly if not properly managed.

Complex pricing models: Cloud providers offer complex pricing models—On-Demand, Reserved, and Spot Instances—each affecting cost and performance differently. Predicting expenses and maintaining optimal performance can be challenging without the expertise to balance costs with SLAs.

TABLE I: INFA-FinOps customer success stories.

	Airline	Healthcare	Insurance Provider	Multinational Bank
Leveraging	FinOps tuning	FinOps tuning	FinOps tuning	Engine selection
Data format	JSON file	Flat file	Parquet, Avro	Delta lake
Mapping task	Hierarchical source and hierarchical parser	Intensive data shuffling and strict SLA	Complex queries	Relational operations: lookup, union, join, etc.
Concurrency	Medium	High	High	Medium
Data size	10 GB to 100 GB	1 TB	10 GB	10 GB to 100 GB
Performance gain	2x	1.8x	3x	2.72x
Cost savings	3x	2.2x	3.5x	2x

[1]The mapping task is the core abstraction of CDI-E [2]. The data engineer defines a Directed Acyclic Graph (DAG) that contains the source(s) of data to process, transformations that contain the processing logic, and the targets where the data is written. This DAG, along with any runtime bindings, constitutes the mapping task, which is executed as a job on the cluster configured by the Admin.

B. Customer Use Cases

The following use cases demonstrate a range of scenarios where Informatica customers have successfully used INFA-FinOps to address many of the aforementioned challenges "Tab. I".

An **airline company** with a fleet size of 150+ planes and 100+ hubs processes complex hierarchical jobs to analyze flight data. Their upstream system deposits files in a cloud object store, with data volumes evolving over time. Spark jobs are tuned based on data size and partitioning, requiring continuous adjustments to maintain performance and prevent job failures due to insufficient resources.

A large U.S. **healthcare provider** processes patient data and doctor's notes in near real time, with strict SLAs for timely updates. The concurrency requirement changes continuously based on the number of patients processed by the provider each day. Instead of manual intervention, cluster sizing to achieve target SLAs is now handled by the engine.

A major U.S. **life insurance providers** is modernizing its data platform, processing insurance files with high concurrency needs. Manually determining cluster size and node types to handle the concurrency is challenging due to the required expertise in Spark, Kubernetes, and cloud providers. INFA-FinOps simplifies this with a one-click solution, hiding the complexity.

A **multinational bank** has to process hierarchical data and transfer it from one table in a Databricks Delta Lake to another data store. Based on data-dependent factors, pushing the processing to Delta Lake might be a better choice versus processing it on a cluster to avoid ingress and egress charges to the customer. INFA-FinOps automatically identifies the most cost-effective option.

C. Design Goals

The primary objective of INFA-FinOps is to achieve financial accountability and cost efficiency while maintaining or improving an organization's job performance and SLA. It has three design goals:

Cost allocation and optimization: We must allocate cloud expenses accurately and efficiently to guarantee that cloud spending remains within budget while optimizing cost and performance to the fullest extent possible. In addition to automatic configurations, recommendations should be provided for additional ways to improve efficiency, such as co-locating data sources in the same region as the cluster and selecting preferred times for workload execution.

Cost transparency: We must provide clear visibility into cloud costs by tracking, analyzing, and reporting on expenditures at a granular level. Allowing users to set a budget and predict future cloud costs based on historical data and expected usage patterns is paramount to planning and budgeting.

Simplicity: Users should only have to keep track of budgets and SLAs. We must eliminate the complexity of understanding cloud ecosystems and job compute requirements. Users should have a one-click selection to optimize for cost or performance and provide the budget details.

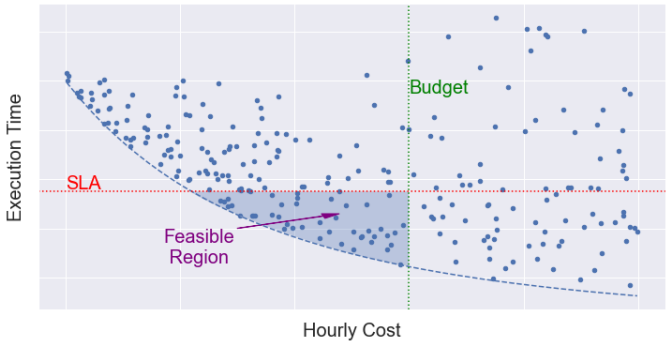


Fig. 2: Cost-vs-performance graph for different resource configurations of same workload. The region within the three lines contain the feasible configurations.

III. INFA-FinOps Overview

INFA-FinOps offers a user-friendly interface to address the FinOps challenges in CDI-E, making the setup process straightforward for the Informatica admin. The admin needs to provide only three values based on their business needs, and the cluster is automatically configured and optimized for them.

- **Cost limit:** Define the maximum dollar-per-hour limit.
- **Performance requirements:** Define performance SLA.
- **Cost vs. performance optimization:** Binary choice to prioritize cost-efficiency or cluster performance. Based on this value, INFA-FinOps tries to move toward the left or the bottom of the feasible region shown in "Fig. 2".

These options enable organizations to effectively optimize and streamline their workload execution with exceptional precision using FinOps capabilities. Automated analysis of cluster and job metrics, evaluation of workload patterns, and monitoring of resource utilization are used to get high levels of performance, scalability, and cost-efficiency in their cloud infrastructure.

INFA-FinOps utilizes cluster and job metrics to optimize the cluster within customer-provided constraints. "Fig. 2" demonstrates the high-level process using an example where the axes show the hourly cost and the total execution time for the job. Over time, INFA-FinOps collects metrics for the same workload running on a cluster with different resource configurations. Depending on the cost-vs-performance slider's value, it attempts to minimize one or the other while keeping both within the budget and the SLA.

A. Architecture

INFA-FinOps has different functionalities powered by dedicated components as show in "Fig. 3".

a) Automated Configuration: INFA-FinOps automatically configures many cluster properties to allow the Informatica admin to have more control over resource costs.

Cluster Tuner. Using values provided by the Informatica admin, the Cluster Tuner dynamically updates job configurations for optimal operation. The Cluster Metric Analyzer generates

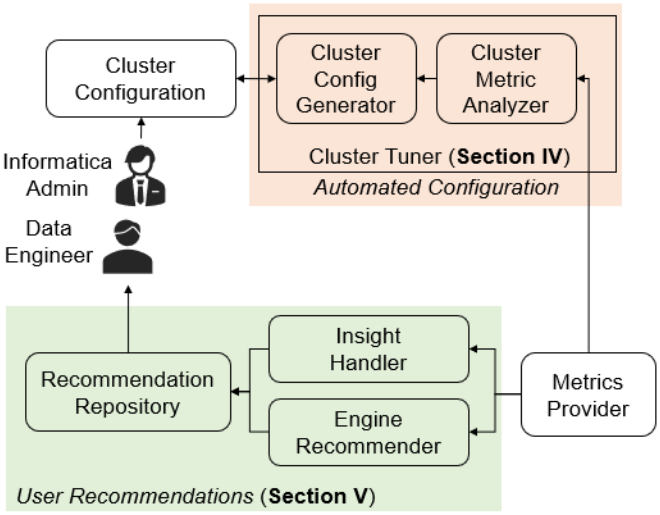


Fig. 3: INFA-FinOps architectural overview.

signals used by the Cluster Configuration Generator to decide optimal values for runtime properties such as the cluster node count and node types.

b) User Recommendations: INFA-FinOps generates recommendations to provide actionable items for users. Some recommendations are automatically applied but can be turned off by the user (e.g., the use of Spot Instances for cost-optimized clusters). Other recommendations have business implications, so the user is required to make explicit changes (e.g., increasing the minimum or the average budget).

Insight Handler. This component analyzes different metrics and generates resource usage insights that help the user make relevant changes. For example, if a cluster is often heavily-loaded during a specific time of day, the user is prompted to change the job schedule.

Engine Recommender. This component provides information to help the user choose the correct engine for a job by generating predictions about future runtime behavior. This is necessary because Informatica can run the same mapping task on different engines based on the sources, targets, and transformations used.

c) Metrics Provider: INFA-FinOps is powered by metrics collected from the cluster and the job. Some metrics such as the number of rows processed by a task are gathered directly from the ConfigMap in the Kubernetes cluster. It also deploys Prometheus to collect metrics on the cluster and job resources at regular intervals. Prometheus forms the major source of metrics that are used by INFA-FinOps. The Metrics Provider collects and stores all of these metrics.

IV. Cluster Tuner

The Cluster Tuner uses an AI-driven optimization model that can be customized to increase cost efficiency or enhance performance based on user preferences. At its core, it relies on the Cluster Config Generator, employing a metric-based algorithm to formalize the cluster configuration challenge and discover an optimal cluster configuration [3]–[5].

a) Optimization: Formally, the Cluster Tuner performs the following optimization:

$$\text{Minimize} \quad Cost(c) \quad \text{for } c \in C;$$

$$b(c) \leq \text{Budget} \quad \text{and} \quad p(c) \geq \text{Performance Threshold}$$

where $Cost(c)$ represents the cost metric associated with a particular configuration c within the space of all possible configurations C, accounting for the expenses incurred for cloud services, infrastructure, and other pertinent elements. Configuration space is searched under budget constraint ($b(c)$) and performance threshold (SLA) constraint ($p(c)$). These constraints ensure that the resulting configuration stays within customer requirements "Fig. 2".

b) Algorithm Overview: As shown in "Fig. 3", the Cluster Metric Analyzer gathers performance metrics from the Metric Provider to generate and save rolling cluster insights. The Cluster Configuration Generator uses most recent N rolling insights to guide the generation of new cluster configurations using the algorithm described in "Algorithm 1". The rolling insights are crucial to determine if the cluster configuration meets predefined constraints such as resource usage limits (e.g., CPU, memory), performance thresholds (e.g., latency, throughput), and budget constraints. If any constraints are not met, dynamic adjustment is done to cluster configuration parameters such as master and worker instance types, instance sizes, enabling Spot Instances, the scale set for worker nodes, disk types, disk size scale set, autoscaling settings and scheduler configurations. The adjusted configuration is then explored within the configuration space to alternative configurations that may better achieve the optimization goal. An adaptive parameter mechanism further enhances this process by enabling the algorithm to learn from previous iterations, refining its decision-making in subsequent runs.

This iterative process continues, interacting with new runs of the cluster until either the maximum number of iterations is reached or the convergence criteria are satisfied. The convergence criteria determine when the algorithm should stop by assessing whether the rolling insights indicate stabilization in the objective function or successful attainment of the optimization goal. Convergence is typically recognized through minimal changes in the objective function over several iterations, a plateau in performance improvement, or meeting the optimization goal within an acceptable tolerance range.

A. Adaptive Selection

Cloud providers offer a diverse array of infrastructure options. However, the inherent disparity between Spark workload characteristics and the underlying hardware often leads to performance degradation and resource inefficiency [6], [7]. Therefore, selecting the ideal combination is critical for achieving a harmonious balance between workload demands, performance, and cost efficiency [8].

Instance type selection: There is no one-size-fits-all instance type among the available options [9]–[11]. For example, in "Fig. 4a", the CPU-bound application showed better

Input: C, optimizationGoal, constraints
Output: ClusterConfig

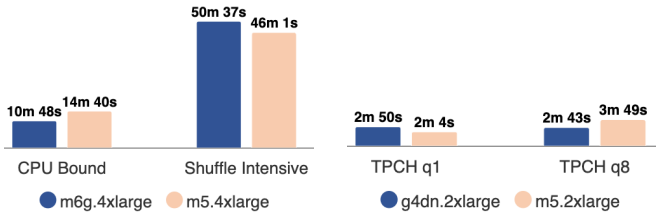
```
c* = InitialConfiguration(C, optimizationGoal);
i ← 0;
while i < MAX_ITERATIONS do
    insightsBuffer = CollectInsights(c*,currentTime);
    M(c*) = ObjectiveFunction(insightsBuffer);

    // Update rolling insights
    rollingInsights.enqueue(M(c*))
    if rollingInsights.size() ¿ N then
        rollingInsights.dequeue();
    end

    if ConstraintsNotSatisfied(c*, rollingInsights,
     constraints) then
        // Dynamic adjustment
        newConfiguration = DynamicAdjustment(c*,
         rollingInsights);
        // Explore configuration space
        c* = ExploreConfigurationSpace(C,
         newConfiguration, optimizationGoal);
        // Adaptive learning
        c* = UpdateAdaptiveParameters(c*);
        if Converged(rollingInsights) then
            break;
        end
    end
    sendClusterConfigrationForNewRun(c*);
    i ← i + 1
end
return c*;
```

Algorithm 1: MetricBasedClusterConfigTuner

performance on the ARM machine, AWS `m6g.4xlarge`, but the multi-stage shuffle-intensive job exhibited superior performance on the X86_64 machine, AWS `m5.4xlarge`. Likewise, in our comparison between GPU-based instances and non-GPU CPU instances "Fig. 4b", we found that there was no definitive winner.

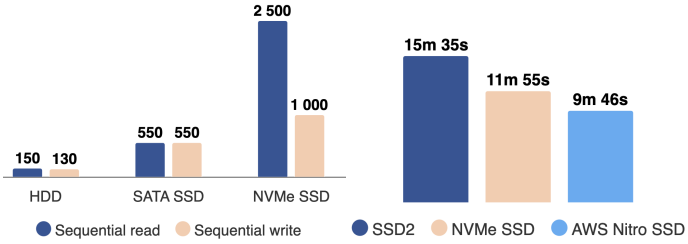


(a) Arm vs x86_64 (b) GPU vs CPU

Fig. 4: Instance type comparison.

The Cluster Tuner is trained using proprietary Informatica data, which comprises historical records of job executions. It calculates a mapping task's score on a specific instance type. The score is a function of the comprising transformation units, data volume, and computation complexity. The Cluster Tuner actively monitors historical usage patterns and discerns the nature of the user's workloads. Based on the workload, it assembles an adaptive and heterogeneous cluster governed by the user's budget and optimization strategy. It then intelligently routes each job to the appropriate node group types within this heterogeneous cluster.

Storage instance type selection: Among available options for the storage type, SSDs exhibit superior read/write performance compared to HDDs, and NVMe SSDs further excel when dealing with intensive workloads. Within AWS, the `i4i.2xlarge` instance features AWS Nitro SSDs, which bring about up to a 60% reduction in storage I/O latency compared to other options, and up to 75% lower latency than NVMe SSDs [12]. "Fig. 5a" provides insights into the variations in sequential read/write among storage variants.



(a) IOPS of different storage types. (b) Workload execution time.

Fig. 5: Storage performance.

Selecting the appropriate storage solution for a workload can significantly enhance performance. We conducted experiments to analyze cost and performance of different storage types. "Fig. 5b" shows the results for a shuffle-intensive job with rank and aggregate operations where NVMe SSDs exhibited 20-30% reduction in overall execution time compared to SSD2, and AWS Nitro SSDs achieved a 30-35% improvement in overall execution time compared to SSD2. In scenarios where the workload is shuffle-intensive or I/O bound, the system automatically matches the storage type or opts for an instance type with either NVMe SSD or AWS Nitro SSD [13].

Spot Instance type selection: Spot Instances are a challenge for efficient workload execution due to fluctuating prices and interruption rates. When interrupted, tasks halt abruptly leading to delays and latency in time-sensitive workloads as new Spot Instances are identified and tasks restarted. To strike a balance between cost efficiency and minimal interruption rates, insights from the Spot Advisor [14]are used by Cluster Tuner to select Spot Instance types by factoring in cost-effectiveness factor (CE) :

$$CE = \frac{\text{Predicted Price}}{(1 - \text{Predicted Interruption Rate})}$$

"Fig. 6" shows a sample assessment of the Spot Advisor's effectiveness, demonstrating the comparative analysis of running workloads with and without Spot Instances for

a life insurance provider. The entire execution process took approximately 2 hours and 17 minutes. Notably, there were no reported interruptions when using Spot Instances, although this outcome can be influenced by the specific timing of the workload execution. Our experiment resulted in a substantial cost reduction of 1.65 times, showcasing the financial benefits of leveraging Spot Instances effectively.

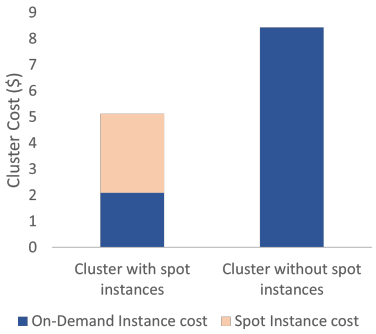


Fig. 6: Spot Instance price savings.

B. Resource Resizing

Resource sizing in Kubernetes involves determining the appropriate scale set of worker nodes and their respective node sizes to accommodate a given workload effectively.

Cluster resizing: When manually configuring a cluster, predicting the optimal cluster size for a specific workload can be challenging. The Kubernetes Autoscaler [15] allows users to set a maximum node limit. However, setting it too high can result in wasted resources and increased costs, especially for Spark workloads with dynamic allocation. Setting the maximum load limit can generate numerous short-lived executor Pods, overwhelming the cluster and increasing cloud infrastructure costs.

The Cluster Tuner uses insights on Pod requests and pending Pods throughout the cluster's life cycle. It predicts the optimal range for the minimum and maximum number of nodes and optimizes the configuration within budget constraints.

Node resizing: Selecting the right worker node size is crucial for scalable applications. For example, when aiming for a total of 384 GB memory and 96 CPUs in a cluster, admins have to choose between twelve 8 CPU/32 GB machines or two 48 CPU/192 GB machines.

Smaller and larger nodes exhibit their own advantages and drawbacks, with no definitive winner. An 8 CPU machines may be deemed optimal for irregular workloads, whereas 16-32 CPU machines are more suitable for consistent workloads. It is recommended to refrain from scheduling more than 20 Pods on a single worker node to prevent resource bottlenecks, especially for the healthcare provider use case (§II-B). In this specific scenario, involving 1 TB of input data and utilizing AWS instances such as `r6i.2xlarge` and `r6i.4xlarge`, the augmentation of vCPUs and memory did not yield proportional improvements in network and EBS bandwidth. This discrepancy resulted in bottlenecks during data-intensive shuffle operations, particularly on the `r6i.4xlarge` instance, as depicted in "Fig. 7".

Instance size	R6i.2xlarge	R6i.4xlarge
vCPU	8	16
Memory (GiB)	64	128
Network Bandwidth (Gbps)	Up to 12.5	Up to 12.5
EBS Bandwidth (Gbps)	Up to 10	Up to 10

Fig. 7: Shuffle-intensive job execution time.

The Cluster Tuner employs the coefficient of variation (CV), i.e., the ratio of standard deviation and mean, of resource requests made by Pods over time to determine the workload pattern. A lower value of CV indicates the number of Pods running over time was consistent; a higher value indicates inconsistency. Value of CV and the nature of the workload are used to determine the appropriate instance size.

C. Optimized Autoscaling Configuration

Optimized autoscaling involves fine-tuning system parameters to efficiently adapt to workload changes. This process aims to strike a balance between responsiveness and stability that will optimize resource allocation in changing environments.

Optimized cluster autoscaling: The current Kubernetes Autoscaler [15] responds to events and periodically checks for any unschedulable Pods every 10 seconds. When there is a pending request for a new Pod, it initiates a scale-up process. Similarly, when no Pods have been active on a node for the default duration of 10 minutes, it initiates a scale-out process. In situations where Spark execution is characterized by frequent spikes in short-lived Spark executor Pods, it tends to result in an increased number of nodes within the cluster, leading to suboptimal node utilization. In contrast, if a Pod becomes unschedulable during a heavy load, it is often too late to address the issue.

Cluster Tuner dynamically fine-tunes the Pod scale-up delay time at the job level, drawing insights from critical metrics such as the average execution time of Spark tasks, workload priority, and optimization model. This calibration manifests as:

$$ScaleUpDelayTime = k \times AvgTaskTime$$

$$k = f(WorkloadPriority, OptimizationModel)$$

This approach balances the number of worker nodes with the demand for executor pods for cost-optimized clusters. With a performance-optimized cluster, the Cluster Tuner employs a proactive scaling approach by adopting a predictive strategy. This strategy leverages a trained model that analyzes Pod request patterns during the execution of user workloads. This proactive approach involves adding instances to the Auto Scaling group before demand spikes are expected. This strategy enhances both availability and performance for workloads that exhibit predictable demand patterns. For example, if

an application takes 5 minutes to initialize, the Kubernetes Autoscaler will create new instances 5 minutes ahead of the anticipated load increase as illustrated in "Fig. 8".

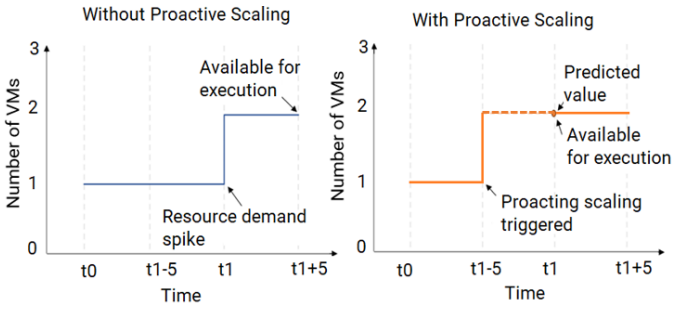


Fig. 8: With and without proactive scaling.

To assess the performance of our autoscaling mechanism, we established two distinct clusters: one cost-optimized and one performance-optimized. Both clusters were configured with an average and maximum hourly budget at $4 and $10, respectively.We subjected both clusters to an identical workload that a life insurance provider typically runs (§II-B).

Our findings, as shown in "Fig. 9", reveal that the performance-optimized cluster had aggressive scaling response to increasing resource demands and effectively downscaled when resource demand decreased. In contrast, the cost-optimized cluster adopted a more conservative scale-up approach. As a result, it achieved a remarkable 10-15% cost savings compared to the cluster that was configured with the default autoscaling settings. These results demonstrate the effectiveness of the cost-optimized configuration in achieving substantial cost savings while maintaining acceptable performance levels.

Summary	Cost optimized	Performance optimized
Initializing worker nodes	1	4
Scale-up strategy	Conservative	Proactive and aggressive
Peak duration	Shorter	Longer

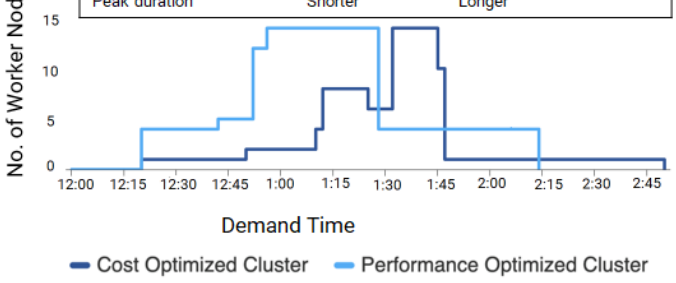


Fig. 9: Cluster life cycle using two approaches.

Optimized storage autoscaling: In addition to computing power, Spark jobs in a cloud environment need storage with some workloads being storage-heavy. For example, a Spark job that joins or ranks a large amount of data might require a large shuffling of data between worker processes. This shuffle data is kept in temporary storage on the local file system (in case it doesn't fit in memory) until its consumers have fully read it. When using temporary storage, we have two options:

- Add nodes to the cluster, providing additional storage.
- Increase storage on the same node where it is required.

Increasing storage is more cost-effective than adding nodes, making it the preferred solution for storage-intensive workloads. The Cluster Tuner, incorporating the storage scaler [2], functions as a daemon-set on Kubernetes worker nodes. It manages the provisioning of storage volumes for Spark jobs and identifies opportunities to streamline storage scale-down.

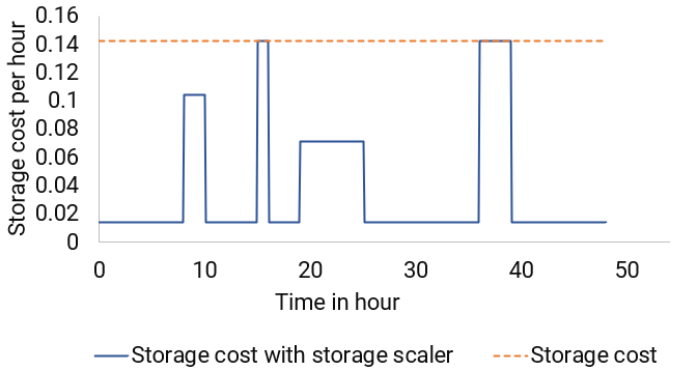


Fig. 10: Advantage of storage scaling.

Let us compare cost savings from upscaling and downscaling storage at fixed intervals versus no storage scaler at all, assuming a 48-hour cluster lifespan with varying storage needs up to 1 TB. "Fig. 10" shows that the total storage expenses for 48 hours will be $0.9594 with the storage scaler and $6.816 without, considering the cloud vendor's rate of $0.0139 per hour for 100 GB of storage. This represents a nearly 7-fold cost reduction. To illustrate the significance of this difference, consider a cluster with 100 nodes, each requiring 1 TB of storage for a specific time frame. Consequently, the impact of this cost differential becomes quite substantial.

D. Smart Cluster Scheduling

When deploying Spark applications on Kubernetes, efficient resource allocation is essential. By default, Kubernetes uses a first-in first-out queue for scheduling where the first task in the queue gets access to resources before others. Prioritization ensures that high-priority tasks promptly receive resources.

With INFA-FinOps, users can explicitly set job priorities and choose preemption or non-preemption execution mode. Our custom scheduler can avoid preempting Pods with specific labels, particularly critical in avoiding preemption of driver Pods, as it can disrupt the entire Spark job. To mitigate job starvation, we use an aging-based mechanism that gradually increases the priorities of low-priority tasks over time.

Furthermore, the default Kubernetes scheduler does not lend itself well to autoscaling. It uses the `LeastRequestedPriority` parameter as one of its scheduling strategies. This parameter prioritizes selecting a node with the least resource usage to place a pending Pod, resulting in a poor scale-down mechanism. Whereas our

custom scheduler uses the `MostRequestedPriority` [2] parameter as a scheduling strategy. This strategy aids in scheduling incoming Pods onto the most occupied nodes, resulting in efficient bin packing. To enhance further scheduling efficiency on a large cluster, it uses the `percentageOfNodesToScore` parameter to evaluate only a portion of nodes to find the most occupied node, resulting in time saved in the scheduling process. Through experiments across a range of workload types (§II-B), we consistently observed 15% to 35% savings. The degree of savings fluctuates due to variations in workload patterns.

V. User Recommendations

INFA-FinOps provides recommendation to help optimize Kubernetes clusters and mapping tasks. Recommendations are available during both the design and execution phases, ensuring a holistic approach to data integration.

A. Cluster Recommendations

To ensure that the Kubernetes cluster remains in its optimal state, some manual interventions might be necessary. These manual actions are available in the recommendation panel. The following are some examples of recommendations.

1) Cluster Budget Recommendations: INFA-Finops suggests increasing the maximum hourly budget allocations in response to increased resource demands detected by the Cluster Tuner[IV]. This aims to improve performance and scalability, ensuring optimal workload execution within constraints.

2) Cluster Region Recommendations: Cloud providers charge for inter-region data transfers, which increase expenses. The Cluster Tuner[IV] mitigates this by recommending a cost-effective region, typically the one where the majority of the data resides. As detailed in II-B, this approach results in significant savings, ranging from 10% to 30% of job costs. The extent of these savings is directly influenced by the volume of data being processed.

3) Peak Workload Recommendations: Customer use case analysis revealed that workloads suffer during peak cluster usage times, typically when users in the same region schedule mapping tasks concurrently. Identifying peak times allows mapping tasks to run in a more balanced distribution. Any critical jobs during peak times can be scheduled to start sooner to improve performance.

Peak workload detection uses a moving z-score of cluster data collected and bucketed into small time intervals. To calculate the peak during the day, multiple days' data is averaged over 24 hours. The z-score at each point i is calculated with a moving window of size w using the formula $z_i = \frac{x_i - \mu_w}{\sigma_w}$; where x_i, μ_w, and σ_w denote the value at point i, the moving average and the moving standard deviation in range [i-w,i].

A cutoff k is used as a peak detector. Any point i is defined to be the peak start such that $z_{i-1} < k$ and $z_i >= k$. We identify multiple potential starting points this way. Our initial approach to detect the end of peak j was to simply take the first point after i where $z_{j-1} > k$ and $z_j <= k$, but this approach failed to cover many cases. The availability of historical data

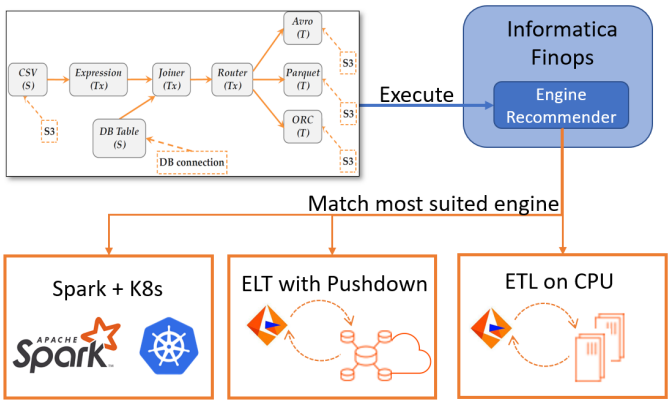


Fig. 11: INFA-FinOps engine recommendation.

allowed us to take a new approach to define the peak based on behavior beyond j. In this new approach, the peak start detection algorithm is used on the time-reversed series which gives us multiple candidates for peak endpoints.

Among the peak candidates, the final peak is determined by calculating total usage for each period. Then, we consolidate adjacent peaks [start1, end1] and [start2, end2] if the following condition is satisfied, as per the aggregation factor a:

$$usage_{avg}([start_1, end_2]) > a * max(usage_{avg}([start_1, end_1]), usage_{avg}([start_2, end_2]))$$

This process is repeated until no more merges are possible. After this, the peak with the highest total usage is selected as the final peak. For this algorithm, w, k, and a are hyperparameters that are tuned for sample workloads observed.

B. Engine Recommendation

INFA-FinOps provides recommendations to help user running the mapping task choose the right engine "Fig. 11". For example, Extract, Load, Transform (ELT) scenarios are suited for the pushdown engine where the processing happens in the data warehouse; while Extract, Transform, Load (ETL) is better addressed by executing the workload on an engine like Apache Spark or Informatica's proprietary native engine. Transformation logic and dataset characteristics determine the optimal engine.

The Engine Recommender uses a decision tree trained on Informatica's diverse use cases. It analyzes mapping task metadata and predicts the optimal engine if it has high confidence that a specific engine will optimize cost and/or performance. If the decision tree fails to determine a clear winner, the runtime prediction algorithm described in Section V-B1 predicts the runtime for each engine and selects the fastest one as the performance-optimal engine.

1) Mapping Task Runtime Prediction: INFA-FinOps uses past execution metadata of different mapping tasks to predict the runtime for a given mapping task. The problem statement is to calculate expected value of runtime duration t_x for given mapping task M_x using past runtime durations t_i for mapping

tasks M_i. The expected value needs to be calculated for given engine *E*. The first step is to filter out past executions on any other engine. Then all durations t_i are transformed based on their effective dataset sizes in M_i so that the values can be used for the dataset of M_x.

To calculate the runtime we needed a similarity function. We found Euclidean distance, Manhattan distance, and inner product to be too sensitive to mappings with larger vectors. Hamming distance and Cosine similarity were tried as they are immune to vector size issues. Hamming distance was found to be less accurate so Cosine similarity as defined below was finalized as the similarity function to use on the vector representation of the mapping tasks.

$$\text{similarity}(\mathrm{M_x}, \mathrm{M_i}) = \mathrm{s_{xi}} = \cos(\theta_{\mathrm{xi}}) = \frac{\mathrm{v_x} \cdot \mathrm{v_i}}{|\mathrm{v_x}|.|\mathrm{v_i}|}$$

This similarity function was used to calculate the similarity score s_{xi} of given mapping task M_x with vector $\mathrm{v_x}$ against each of the previous executions M_i. These scores were used as weights to get final value of the expected runtime for the new mapping task M_x. The time taken for this calculation was high as the iteration over the M_i mappings takes O(n) time with a high constant. For the results to be shown in the UI, this calculation has to be in real time. To reduce the algorithm runtime, the formula was rewritten as described below.

$$\text{estimate}(\mathrm{t_x}) = \frac{\sum_{\mathrm{i}} \mathrm{s_{xi}} \mathrm{t_i}}{\sum_{\mathrm{i}} \mathrm{s_{xi}}} = \frac{\sum_{\mathrm{i}} \frac{\mathrm{v_x} \cdot \mathrm{v_i}}{|\mathrm{v_x}|.|\mathrm{v_i}|} \mathrm{t_i}}{\sum_{\mathrm{i}} \frac{\mathrm{v_x} \cdot \mathrm{v_i}}{|\mathrm{v_x}|.|\mathrm{v_i}|}}$$

$$= \frac{\sum_{\mathrm{i}} \mathrm{v_x} \cdot (\frac{\mathrm{v_i} \mathrm{t_i}}{|\mathrm{v_i}|})}{\sum_{\mathrm{i}} \mathrm{v_x} \cdot (\frac{\mathrm{v_i}}{|\mathrm{v_i}|})} = \frac{\mathrm{v_x} \cdot (\sum_{\mathrm{i}} \frac{\mathrm{v_i} \mathrm{t_i}}{|\mathrm{v_i}|})}{\mathrm{v_x} \cdot (\sum_{\mathrm{i}} \frac{\mathrm{v_i}}{|\mathrm{v_i}|})}$$

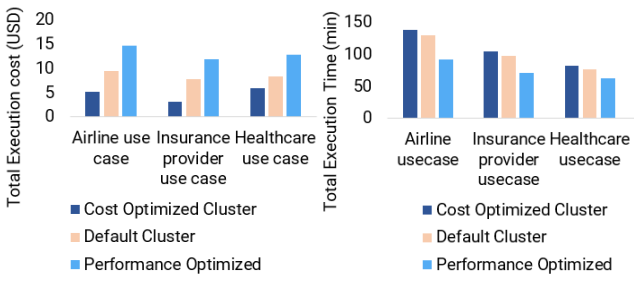
The final expression is more useful as the summations do not include $\mathrm{v_x}$, the vector of the test mapping task M_x. The iterated vector sum is precomputed and used when a prediction for any M_x needs to be made. This change reduced the computation time after the UI call from O(n) to O(1) even though the overall work remained the same. With this approach, every mapping task run updates the vector sums for use in future predictions.

VI. Performance and Use Case Tie-in

A. Experimental Setup

To provide a comprehensive evaluation of INFA-FinOps's performance, we conducted an extensive analysis across a wide spectrum of real-world use cases and scenarios detailed in Section II-B. To replicate these scenarios, we choose 4 distinct workload types "Tab. I". These workloads fall into 2 categories: those benefiting from FinOps tuning and those benefiting from Engine Selection. All experiments were conducted using the following cluster configurations:

- **Default cluster:** Includes 10 machines with the default Kubernetes setup, each having 8 cores and 32 GB RAM, totaling 80 cores and 320 GB RAM for worker nodes.
- **FinOps cluster (Cost and Performance Optimized Cluster):** The maximum hourly budget is set at 10, aligning with the cost of the default cluster configuration.



(a) Cost comparison (b) Execution time comparison

Fig. 12: Workload use cases.

B. Results

1) Workload-Level Improvements and Breakdowns: We executed our first category of workloads to assess the efficiency of our FinOps tuning with the data set illustrated in "Fig. I". We computed the execution time of the workload as well as the total execution cost reported by the cloud vendor, AWS. Our findings are shown in "Fig. 12a" and "Fig. 12b". In terms of cost optimization, our cluster configuration achieved cost savings ranging from 2x to 3.5x compared to the default cluster. These cost savings are achieved through the utilization of Spot Instances, optimized cluster scheduling, conservative cluster autoscaling, judicious instance type selection, and smart cluster shutdown. The observed average distribution of these savings is outlined in II.

Conversely, performance-optimized cluster demonstrated a significant performance boost, ranging from 1.5x to 1.7x in overall execution speed. The factors contributing to this acceleration include effective instance type selection, aggressive autoscaling, and smart cluster scheduling. The observed estimated distribution is outlined in II. Note that Spot Instances and Smart Shutdown are not utilized in the performance-optimized cluster.

TABLE II: Categorical benefit distribution

Category	Savings	Acceleration
Spot Instance selection	27%	N/A
Instance type selection	18%	48%
Cluster autoscaling	29%	34%
Smart scheduling	14%	18%
Smart shutdown	12%	N/A

2) Job-Level Improvements: During the evaluation of individual job runs, we observed performance enhancements of up to 3.1x and cost savings of up to 3.5x "Fig. 13". Interestingly, shuffle-intensive and IO-bound jobs on the cost-optimized cluster demonstrated similar or even superior performance compared to jobs executed on the default cluster. This is due to the fact that these jobs predominantly depend on network bandwidth and RAM speed rather than advanced CPU hardware. Leveraging more economical Spot machines allowed us to acquire a greater number of them at a reduced cost, enabling improved execution. However, it is important to note that the execution of CPU-bound, mixed-nature applications relies on

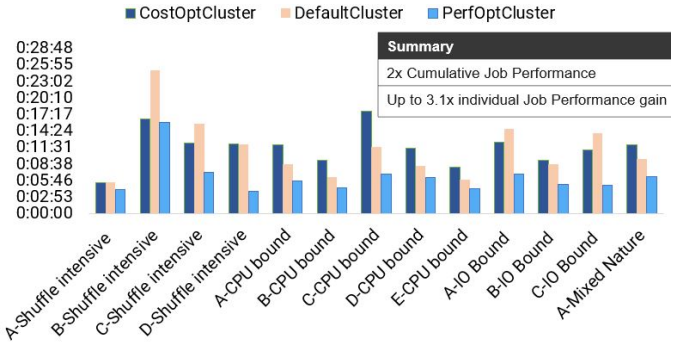


Fig. 13: Individual mapping task improvements.

advanced CPU hardware. So, the cost-optimized execution for such applications resulted in degraded performance.

3) Dynamic Adaptation: We conducted experiments to investigate the dynamic behavior of the cluster for the health care provider use case, where the Cluster Tuner continuously refines the cluster configuration (§IV). In this specific experiment, we introduced a workload characterized by high concurrency, a substantial data volume of 1 TB, and multi-stage shuffle operations. Initially, we executed the workload on both the cost-optimized and performance-optimized clusters having a max budget 10 to optimize job performance. Later, we conducted consecutive runs of the same workload over the performance-optimized cluster, given that our use case places a higher emphasis on SLA adherence.

We observed a notable 25% improvement across 3 consecutive runs along with cost savings of approximately 20%. Furthermore, after 3 iterations, the Cluster Tuner recognized that the initially allocated budget was insufficient to meet the demands of the user workload. This recognition prompted an adjustment in the allocated budget. As shown in "Fig. 14", when the user increased the maximum budget from 10 to 20, we witnessed a further performance improvement of 19%, resulting in an overall performance enhancement of 40%.

4) Impact of Engine Selection: For the second category of workloads was used to assess the impact of engine selection recommendations. When we implemented an engine selection recommendation for a multinational bank use case, particularly opting for the SQL ELT engine, we observed a significant improvement in workload execution time, achieving an impressive enhancement of 2.72×, while simultaneously reducing job costs by up to 2×. These cost savings can be attributed to several factors, including reductions in ingress and egress network expenses.

VII. Conclusions

In conclusion, this paper introduced INFA-FinOps, Informatica's FinOps-powered advanced data integration solution for addressing challenges in cloud infrastructure planning and budgeting. We discussed key production deployment issues and outlined Informatica's solutions, along with our performance observations.

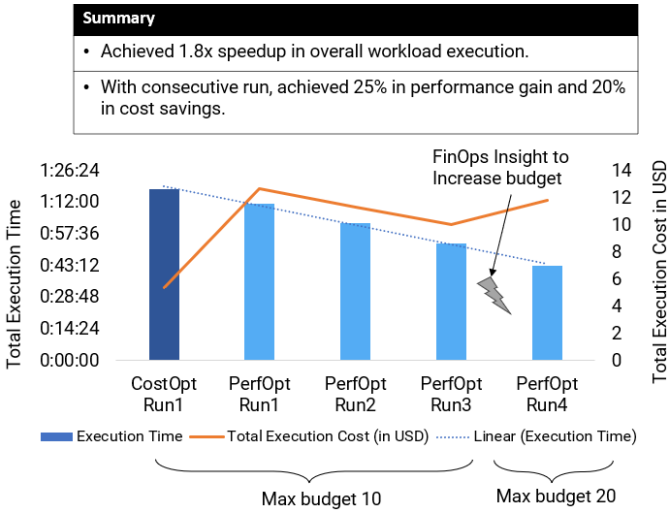


Fig. 14: Dynamic cluster (workflow with concurrent Mapping tasks with large data volumes).

While we've made progress in optimizing the system, there are still opportunities for further improvement. Our future roadmap includes integrating metric-aware and AI-based algorithms to enhance cost and performance tracking. We aim to expand beyond compute and storage metrics, improving cost efficiency and user optimization. We also plan to help users predict and plan future budgets by sharing usage patterns.

References

[1] Adam Ronthal and Kevin Gabbard, "Financial Governance for Successful Cloud Data and Analytics," *Gartner®*, Oct 25, 2023.
[2] P. Das, S. Srivastava, V. Moskovich, A. Chaturvedi, A. Mittal, Y. Xiao, and M. Chowdhury, "Cdi-e: An elastic cloud service for data engineering," *Proc. VLDB Endow.*, vol. 15, p. 3319–3331, aug 2022.
[3] H. Dou, K. Wang, Y. Zhang, and P. Chen, "Atconf: auto-tuning high dimensional configuration parameters for big data processing frameworks," *Cluster Computing*, vol. 26, pp. 1–19, 10 2022.
[4] C.-J. Hsu, V. Nair, T. Menzies, and V. Freeh, "Scout: An experienced guide to find the best cloud configuration," 03 2018.
[5] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song, and Y. Yang, "Bestconfig: Tapping the performance potential of systems via automatic configuration tuning," pp. 338–350, 09 2017.
[6] M. T. Islam, S. Srirama, S. Karunasekera, and R. Buyya, "Cost-efficient dynamic scheduling of big data applications in apache spark on cloud," *Journal of Systems and Software*, vol. 162, p. 110515, 12 2019.
[7] L. Xu, A. R. Butt, S.-H. Lim, and R. Kannan, "A heterogeneity-aware task scheduler for spark," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 245–256, 2018.
[8] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," vol. 49, pp. 127–144, 02 2014.
[9] Amazon Web Services, Inc., "Amazon EC2," Accessed October 2023.
[10] Google Cloud, Inc., "Google compute engine," Accessed October 2023.
[11] Microsft Azure, Inc., "Azure Virtual Machines," Accessed October 2023.
[12] Amazon Web Services, Inc., "AWS Nitro SSD – High Performance Storage for your I/O-Intensive Applications," Accessed October 2023.
[13] F. Pan, J. Xiong, Y. Shen, T. Wang, and D. Jiang, "H-scheduler: Storage-aware task scheduling for heterogeneous-storage spark clusters," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 1–9, 2018.
[14] "AWS Spot Advisor." https://aws.amazon.com/ec2/spot/instance-advisor/.
[15] Kubernetes, "Cluster Autoscaler," Accessed October 2023.