WHITE PAPER

The Cloud Cost Optimization Imperative for Data-Intensive Workloads on Amazon EMR and Amazon EKS

Top challenges and solutions for modern, large-scale data environments

pepperdata

aws PARTNER

Table of Contents

Introduction: The Challenge of Cloud Cost Optimization for Data-Intensive Workloads 1

The Fundamental Problem at the Application Level: Waste *Inside* the Application 2

How Pepperdata Cost Optimization Automatically Mitigates Cloud Waste *Inside* Applications 4

Pepperdata Capacity Optimizer: Automated Instance Hour Reduction in Real Time 6

Pepperdata Observability 9

Pepperdata Security Policies, Procedures, and Certifications 10

Conclusion: Solving the Challenge of Cloud Cost Control 11

Appendices 12

Resources 15

The Challenge of Cloud Cost Optimization for Data-Intensive Workloads

The advantages of migrating compute-intensive workloads to the cloud have become abundantly clear. Organizations can enjoy near-infinite scalability on demand, agility in deploying new applications, and enhanced security and analytics, all combined with pay-as-you-go pricing. Gartner predicts that worldwide spending on the public cloud will reach nearly $700 billion in 2024[1], a 20.4 percent increase over 2023, as spending in key segments shifts to the public cloud.

While the cloud provides on-demand, elastic, and highly scalable computing resources, cost overruns do occur, often as a result of overprovisioning. Overprovisioning, especially with data-intensive workloads such as Apache Spark, invariably results in resource underutilization and waste, which can be difficult to identify and eliminate.

Not surprisingly, a recent survey found a 39 percent year-over-year increase in the amount of cloud spend over budget[2], a widespread problem that affects even the most sophisticated IT teams. It has been further estimated that 30 percent or more of cloud computing services will end up wasted[3]. The FinOps Foundation even reported that reducing waste has become the highest key priority for FinOps practitioners[4].

To address these issues and realize the promise of running data-intensive workloads in the cloud, organizations are adopting a variety of strategies to minimize waste, control costs, and rein in budgets:

Business Level

- Tagging
- Cost allocation
- Chargebacks/showbacks
- Usage monitoring

Application Platform Level

- Cluster Autoscaling
- Migrating to a serverless architecture

Infrastructure Level

- Rightsizing instances
- Deploying AWS Karpenter
- Implementing financial optimizations, such as Spot Instances, Reserved Instances and Savings Plans

Application Level

- Manually tuning applications and configurations
- Enabling Spark Dynamic Resource Allocation

[1] Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach $679 Billion in 2024

[2,3] Flexera's State of the Cloud 2024 Report

[4] FinOps Foundation's State of FinOps 2024 Report

The Fundamental Problem at the Application Level: Waste *Inside* the Application

Many practitioners believe that some combination of these strategies will minimize the cost of running data-intensive workloads such as Spark in the cloud. However, the fundamental problem with all of these options is that they are not able to address a specific dilemma: **Apache Spark executors waste requested resources inside an application**.

Waste inside the application occurs because application provisioning is static and set for peak usage throughout an application's run, even though application resource usage **actually varies dramatically over time**. Even an optimally-chosen allocation level set at the peak of the application's utilization will result in unused resources and waste since most applications typically run below peak levels most of the time.

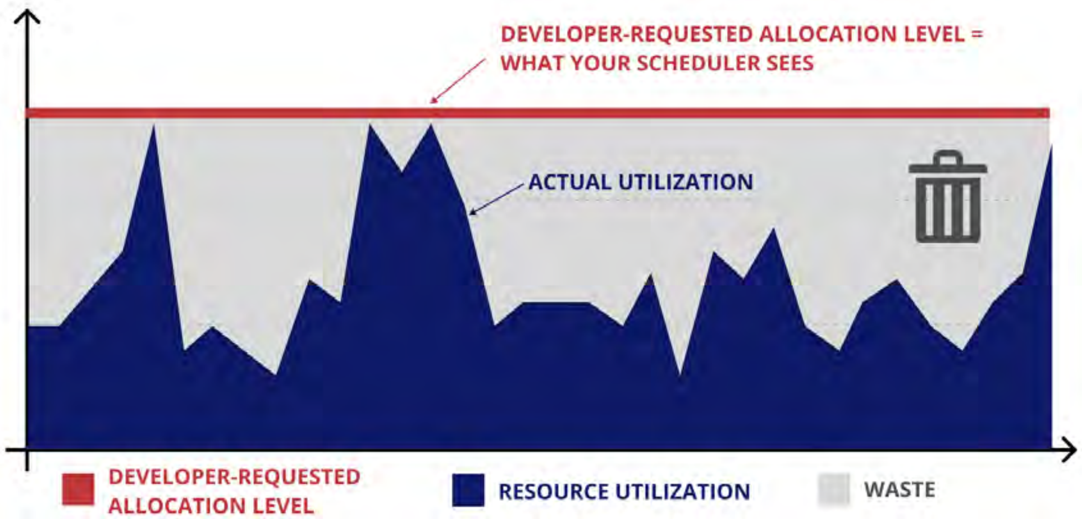


Figure 1: Most applications run at peak provisioning levels for only a small fraction of time, resulting in waste. Even if a developer requests resources accurately for peak, the autoscaler will grant the resources based on the request rather than for the actual utilization of resources required to run the application.

When developers request peak levels of resources, the scheduler gives them exactly what they asked for. The scheduler distributes resources based on allocation requests rather than actual usage, and **has no insight into the fact that the allocated resources are not being used**. As far as the scheduler is concerned, the developer asked for a certain level of resources, so that level of resources will be utilized.

**At a Cluster Level, the Scheduler Only Sees Allocations.
Instances Appear Almost Fully Utilized**

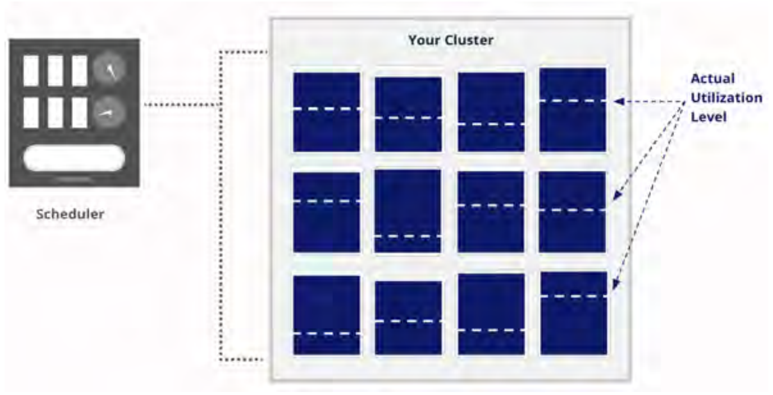


Figure 2: Within a cluster, the scheduler only considers the allocated resource levels, not those that are actually being used at any given time. The allocated but unused resources represent waste.

As a result, the scheduler "sees" a fully saturated cluster and interprets that as a cluster unable to accept any additional workloads. Then, when more applications come along in an environment that is already seen by the scheduler as fully saturated, the scheduler has only two options:

1. The scheduler can put the new workloads or applications into a queue or pending state until resources free up, **reducing throughput and performance**.
2. The scheduler can direct the autoscaler to spin up new instances at additional cost, even though existing instances are actually not fully utilized, **resulting in unneeded spend for unneeded resources**.

**Because Instances Appear Fully Utilized,
They're Put In Queue When New Apps Come Along**

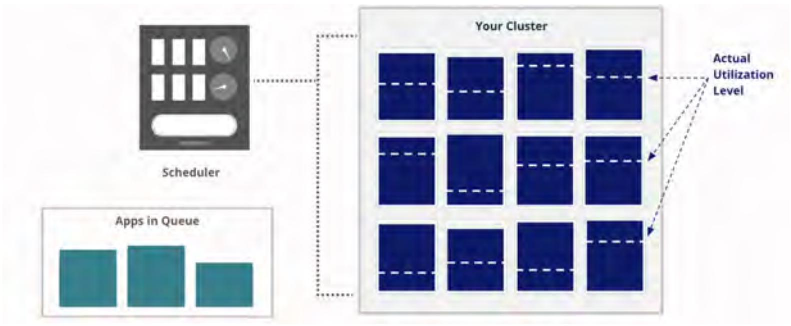


Figure 3: When the system appears to the scheduler to be fully saturated, the scheduler might leave new applications waiting in queue, delaying their execution and leading to reduced throughput and unnecessary cost for the unused capacity.

**Or New Instances Are Launched (At Extra Cost)
To Accommodate Those Apps**

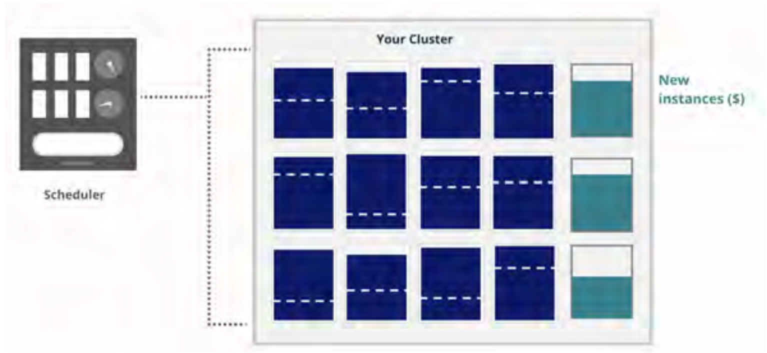


Figure 4: When the system appears to the scheduler to be fully saturated, the scheduler might spin up new instances—at additional and unnecessary cost—to process them.

In either case, the result is inefficiency and waste in the system as a result of the inherent inefficiencies inside Spark applications. This waste cannot be optimized out with any of the common strategies previously mentioned.

While the common strategies used to optimize Spark—such as Cluster Autoscaling, instance rightsizing, Spark Dynamic Allocation, and manual tuning—provide a level of savings or optimization, **none of these solve the fundamental problem of Spark overprovisioning and waste inside executors** when the application is not at peak. None of these addresses the issue of resources being assigned based on allocation rather than usage levels.

Even the most efficient infrastructure in the world—one in which every waste mitigation strategy is implemented—will then continue to run with waste since overprovisioned applications use infrastructure inefficiently. And these inefficiencies can be significant. On average, typical Spark applications can be overprovisioned by up to 47 percent or sometimes more.

How Pepperdata Cost Optimization Automatically Mitigates Cloud Waste *Inside* Applications

Pepperdata Capacity Optimizer reduces waste inside applications by addressing the **critical gap between allocated and actual resource utilization** inside an application.

Capacity Optimizer's **Continuous Intelligent Tuning** provides a **real-time stream of data** which enables cluster scheduler decision-making to be based on **actual usage instead of allocated usage**. This ensures that clusters use all available resources for each workload before adding new nodes or pods. More workloads can run on the same node or pod, increasing efficiency.

Working autonomously and continuously in the background, Capacity Optimizer uses patented algorithms to:

- **Immediately Reduce Instance Hours and Cost:** Only pay for what you use when CPU and memory are optimized in real time
- **Save Engineering Time With No Manual Tuning:** Reclaim hours of engineering time that can be reallocated to new projects and business growth initiatives
- **Improve Application Efficiency Without Code Changes:** Apply optimization easily without disruption to meet price/performance SLAs

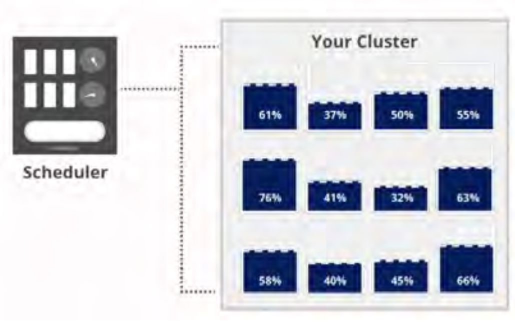
As a simple example, consider an overprovisioned application that requires 10 GB of memory at peak, but that peak time may only represent 20 percent of the application's runtime. The other 80 percent of the time, Capacity Optimizer can reclaim those unused resources for other applications. By finding the difference between allocated and used capacity in real time, Capacity Optimizer can instruct the YARN or Kubernetes scheduler to process waiting applications, thus providing greater throughput across the cluster.

Augmented Autoscaling

Capacity Optimizer also augments Cluster Autoscaling by ensuring that new instances are launched only when the existing instances are fully utilized. The result: **CPU and memory are autonomously optimized to run more workloads to increase savings**.

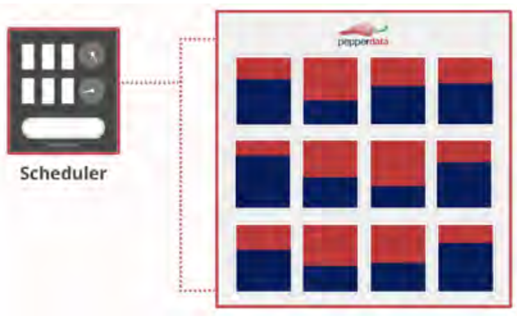
Capacity Optimizer eliminates the need to change applications based on system-generated recommendations by working immediately in real time to keep instances at their optimal utilization with no code changes required.

Nodes running suboptimally WITHOUT Pepperdata



Without Pepperdata, instances may run at a fraction of their potential utilization. This is because the scheduler is only aware of allocated resources and is not aware of what is actually being used, and therefore it cannot pack additional jobs onto nodes/pods that appear to be fully allocated.

Nodes running at greatest capacity WITH Pepperdata



Pepperdata removes the blinders from the scheduler to empower it to pack instances according to their actual utilization. Pepperdata's Continuous Intelligent Tuning empowers the scheduler to make full use of existing instances, increasing capacity and throughput and reducing cost.

Figure 5: Pepperdata enables nodes to run at the greatest capacity and highest efficiency on Amazon EMR and Amazon EKS.

Installed via a simple bootstrap script into a customer's Amazon EMR environment or via Helm chart in an Amazon EKS environment, Capacity Optimizer runs autonomously and safely in the customer's virtual private cloud (VPC) in the background, providing a dashboard view of the utilization level of each cluster.

Capacity Optimizer thus provides two immediate cost-cutting and time-saving benefits:

1. **Cost Savings: Decreased Instance Hour Consumption**
   By maintaining clusters in their sweet spot of optimal utilization based on actual resource usage, Capacity Optimizer reduces hardware usage for Amazon EMR and EKS an average of 30 percent. The decreased instance hours translate directly to greater efficiency and reduced costs.

2. **Time Savings: No Manual Tuning, No Recommendations, and No Application Code Changes**
   Capacity Optimizer works autonomously in real time, so no application changes or manual interventions are ever necessary. Development teams are freed from tuning applications so they can focus on high-value, innovative activities to grow their organizations.

Which Specific Costs are Reduced for Amazon EMR?

Let's take a look at what we mean when we say Pepperdata Capacity Optimizer reduces instance hour costs. For Amazon EMR, instance hour costs are the sum of Amazon EC2 costs and Amazon EMR service costs. This cost is split approximately 80/20, as measured by Amazon's published per-hour rates by instance type[5], with EC2 comprising 80 percent of the total instance hour costs and EMR comprising 20 percent. The cost reduction delivered by Capacity Optimizer is proportional to the overall instance hour cost, of which approximately 80 percent is Amazon EC2.

Pepperdata Capacity Optimizer: Automated Instance Hour Reduction in Real Time

As we have seen, Pepperdata Capacity Optimizer delivers 30-47 percent greater cost savings for data-intensive workloads such as Apache Spark on Amazon EMR with no application changes. Let's drill down a bit more into how it works.

[5] Amazon EMR Pricing

How Pepperdata Capacity Optimizer Works Inside AWS

Pepperdata Capacity Optimizer works by installing a proprietary, small-footprint PepAgent within the customer environment. PepAgent continuously collects and correlates hundreds of real-time operational metrics, including host-level CPU, RAM, disk I/O, and network metrics, as well as job, task, queue, workflow, and user info. This data is then aggregated to inform Capacity Optimizer's decision making and to power the graphs and insight within Pepperdata's observability dashboard.

In Amazon EMR environments, PepAgent is installed inside each node in the cluster. The PepAgent footprint is approximately one percent of one core and 300 MB of memory. PepAgent collects hundreds of metrics every five seconds and provides these metrics to Capacity Optimizer. These metrics serve as the source of the granular cluster-level and application-level data presented in Pepperdata's customer dashboard.

In Amazon EKS environments, PepAgent runs as a service, and makes use of Prometheus[6] as a bundled component to provide key optimization metrics to Capacity Optimizer. Pepperdata developed a specialized configuration for Prometheus that reduces memory footprint by nearly tenfold so that, even in highly-scaled customer environments, Pepperdata Capacity Optimizer with Prometheus as a bundled component exhibits a very small memory footprint, which supports stability and reliability.

PepperdataSupervisor Agents then write metrics data to a Local File System. PepCollector Agents read those files and send them to the Pepperdata Dashboard via the Pepperdata Backend.

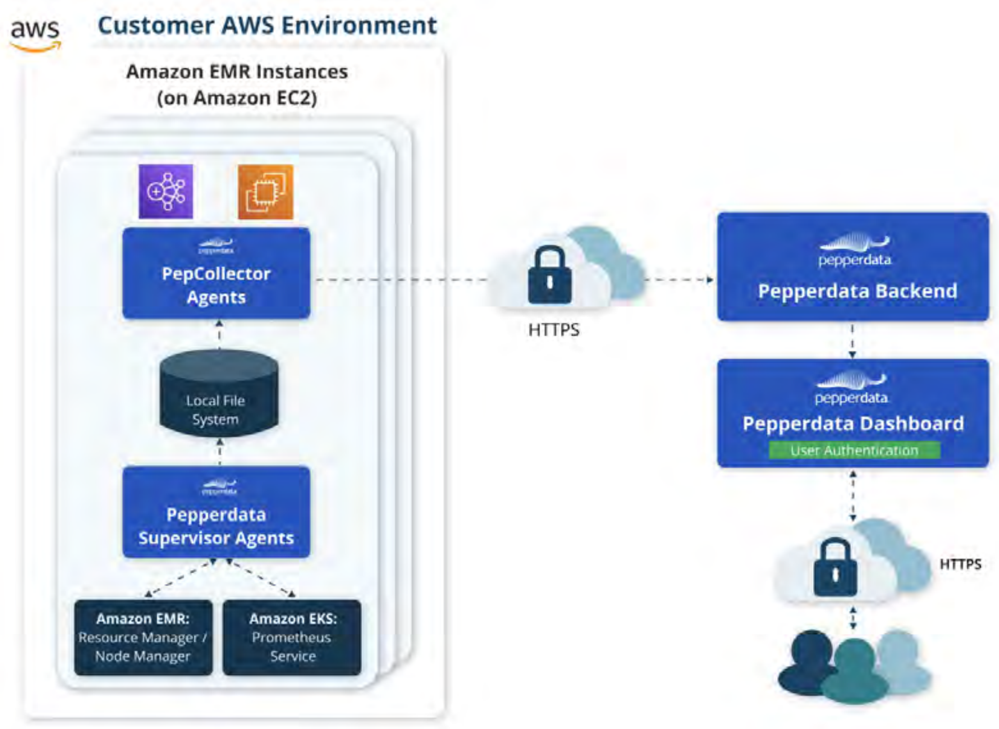


Figure 6: Pepperdata's architecture. Data collected by PepAgents is used to inform Capacity Optimizer's decision making as well as power the graphs and insight within Pepperdata's observability dashboard.

[6] Prometheus.io

The Power of Capacity Optimizer With a Real-World Application

Consider the power of Pepperdata Capacity Optimizer when applied to just a single application. The following chart taken from Pepperdata's dashboard highlights the waste that occurs in one example Spark application selected at random from among the millions that Capacity Optimizer supports every day:

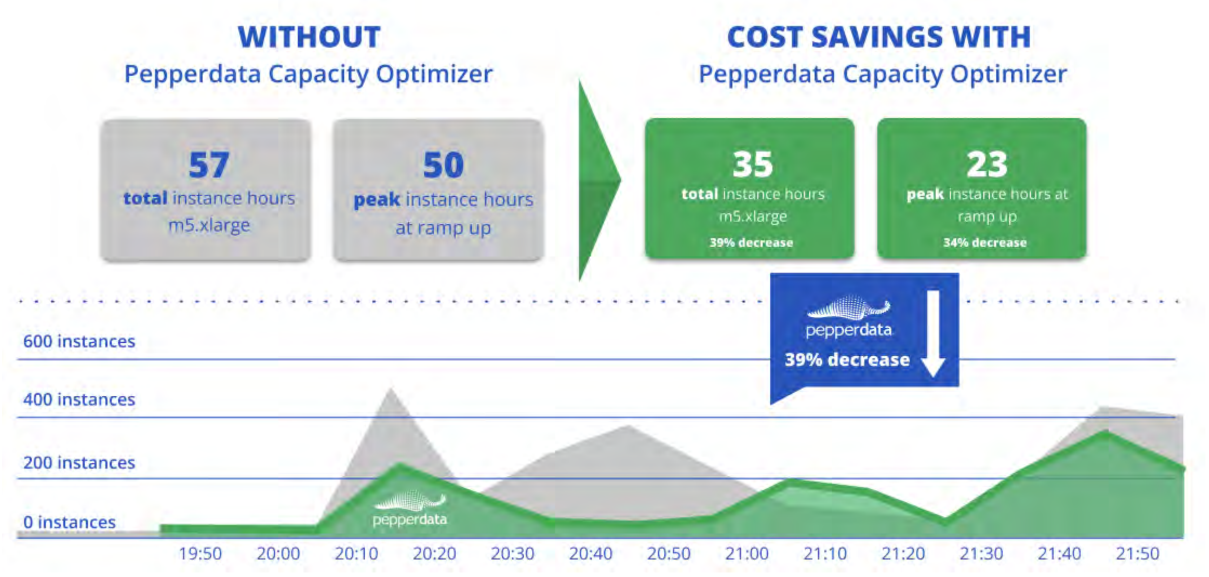


Figure 7: Pepperdata's dashboard highlights the power of Capacity Optimizer in reducing both overall instance hours and the peak instance hour requirement at maximum ramp up for a single application.

Without Capacity Optimizer, this sample application required 57 total hours of the m5.xlarge instance to run, with a maximum ramp up to 50 instance hours. When the same application was run again with Capacity Optimizer enabled, the total instance hours dropped to 35—a 39 percent decrease—and the peak dropped in half, to 23.

This single application with its modest resource requirements illustrates how similar results occur at scale across the millions of applications whose performance Pepperdata optimizes every year.

Environmental Considerations

Pepperdata provides the greatest benefit for large, complex, multi-tenant clusters where costs run high and where manual tuning effort is limited by the scale and scope of the environment. In such environments, with large numbers of concurrently running applications, Pepperdata's Continuous Intelligent Tuning remediates application-level waste autonomously and in real time, contributing an average of 30 percent additional savings for most all organizations running Apache Spark.

In environments where only a single executor exists per node, and where applications are designed to be single tenant down to the executor level, Capacity Optimizer may not have the necessary conditions to do its job effectively. This is because no other application can be serviced with the leftover, unused resources from that single application. Implementing a cost optimization solution such as instance rightsizing or manual tuning can reduce infrastructure-level waste for these single-tenant environments. However, applications in this type of environment will continue to generate the waste unique to the problem of application overprovisioning.

Serverless environments, which are designed for practitioners seeking minimal operational overhead, are currently not supported by Capacity Optimizer.

Pepperdata Observability

Alongside its autonomous cost optimization, Pepperdata also provides full-stack observability and real-time insights into Apache Spark workloads on prem or in the cloud. Pepperdata's observability dashboard offers deep insights into application performance, cluster health, and cost metrics that are not readily available through general-purpose monitoring or out-of-the-box performance management tools. Executives, platform teams, and applications teams can use this knowledge to improve decision making about their Amazon EMR and EKS environments.

The deep observability features included with Pepperdata Capacity Optimizer work by instrumenting every node of a cluster with the small-footprint PepAgent, which continuously collects and correlates hundreds of real-time operational metrics, including host-level CPU, RAM, disk I/O, and network metrics, as well as job, task, queue, workflow, and user info. The single pane of glass of the Pepperdata dashboard enables users to make intelligent decisions based on these correlated real-time operational metrics, and to quickly diagnose, troubleshoot, and resolve both cluster-wide and low-level application issues.

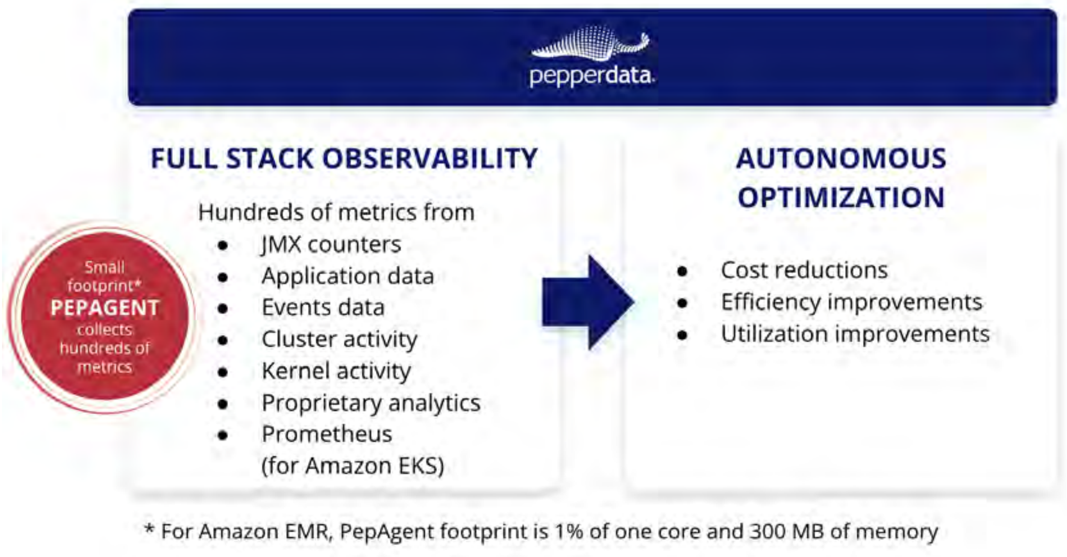


Figure 8: Pepperdata's full-stack observability complements its autonomous optimization to obtain deep insight into your Amazon EMR and Amazon EKS environments.

As compared to general-purpose monitoring or out-of-the-box performance management tools, the Pepperdata Dashboard provides everything needed for holistic, enterprise-wide platform observability—and therefore insight and control—at both the cluster and the application level.

The combination of Pepperdata's autonomous cost optimization plus additional observability features empowers customers to both optimize and understand the performance and cost of large-scale data analytics clusters and applications.

Pepperdata Security Policies, Procedures, and Certifications

As a SOC 2® Type 1[7] compliant company, Pepperdata maintains a comprehensive, written information security program that actively monitors and manages vulnerabilities to ensure all software is compliant with SOC reporting requirements.

Pepperdata's security program is based on a strict set of Data Protection guidelines and Security Policies:

Data Protection

- Pepperdata has no access to data or applications inside a customer's environment.
- Transport-level encryption is always in effect between a customer's cluster and Pepperdata. Communication between users and the Pepperdata dashboard requires secure HTTPS access. Sensitive data is encrypted both at rest and in transit to prevent unauthorized access or interception.
- Pepperdata's dashboard is read only. Underlying data and applications are never changed in any way by Pepperdata.
- Access to Pepperdata's dashboard is via secure user authentication. Pepperdata collects an audit trail of which pages are viewed whenever a user logs in.

Security Policy Implementation and Ongoing Audits

- Pepperdata's internal security policies include procedures and strict guidelines for employees regarding access control, incident response, and acceptable use of company resources.
- Pepperdata performs a variety of security-related testing to keep current with security recommendations and evaluates and implements new security services and features regularly. These include regular security audits to identify and assess potential security risks, vulnerabilities, and threats and make continuous improvements to Pepperdata's security program.

For more information about Pepperdata Security Policies, please contact us at info@pepperdata.com to request a copy of our Security Whitepaper.

[7] SOC 2® - SOC for Service Organizations: Trust Services Criteria

CONCLUSION

Solving the Challenge of Cloud Cost Control

Given that 30 percent (or more) of cloud computing services regularly goes to waste[8], and as minimizing that waste has become the top priority for FinOps practitioners[9], cloud cost optimization has become essential. Cloud cost optimization enables practitioners to mitigate the waste inherent in many applications and extract the most value out of their cloud investment.

Pepperdata Capacity Optimizer is a lightweight, highly scaled cost optimization solution that remediates waste and cost overruns in some of the world's largest, most complex cloud and on-premises clusters. By automatically packing additional pending jobs on underutilized nodes/pods through autonomous tuning, Capacity Optimizer increases node/pod utilization and maximizes the efficiency of thousands of workloads without requiring an investment in additional resources. This translates directly to reduced instance hours and cost, greater efficiency, improved performance, and ultimately ongoing customer satisfaction.

Hardened and battle tested since 2012, Pepperdata has saved its customers an estimated $200 million in that time. Pepperdata is currently deployed on 20,000+ clusters per month for both optimization and observability on some of the largest and most complex cloud deployments in the world.

For a demo or to learn more about utilizing Pepperdata in your environment, please contact us at info@pepperdata.com.

[8] Flexera's State of the Cloud 2024 Report

[9] FinOps Foundation's State of FinOps 2024 Report

Appendix 1: Benchmarking with Amazon EMR 7.0 May 2023

Methodology

Pepperdata selected TPC-DS[9], the decision support benchmark from the Transaction Processing Performance Council (TPC), as the standardized workload model. TPC-DS is a commonly used benchmark for measuring compute performance. Pepperdata used a GitHub repository recommended by Amazon Web Services (AWS) that closely adheres to the TPC-DS model for benchmarking purposes. Pepperdata ran this benchmark "out of the box" and did not modify or recompile them using any special libraries. (Note: This work is not an official audited benchmark as defined by TPC.)

Parameters

Pepperdata set up the Spark benchmarking test environment on Amazon EMR 7.0 using Graviton instance types to test Pepperdata Capacity Optimizer.

Pepperdata ran this workload three times both with and without Capacity Optimizer to capture the "before and after" effects of Capacity Optimizer.

Results

The benchmark results show that for Spark workloads running on Amazon EMR 7.0, Pepperdata Capacity Optimizer:

- Decreased total instance hours consumed by 51 percent
- Increased overall throughput, increasing concurrent container count by 57 percent
- Improved memory utilization by 65 percent, thus increasing performance
- Increased CPU utilization by 82 percent to enable more efficient processing

Further detail is available in the full report, Pepperdata Reduces the Cost of Running Spark Workloads on Amazon EMR 7.0 by 51%.

[9] TPC-DS is a Decision Support Benchmark

Appendix 2: Pepperdata Benchmarking on Amazon EKS October 2023

Methodology

Pepperdata selected TPC-DS[10], the decision support benchmark from the Transaction Processing Performance Council (TPC), as the standardized workload model. TPC-DS is a commonly used benchmark for measuring compute performance. Pepperdata used a GitHub repository recommended by Amazon Web Services (AWS) that closely adheres to the TPC-DS model for benchmarking purposes. Pepperdata ran this benchmark “out of the box” and did not modify or recompile them using any special libraries. (Note: This work is not an official audited benchmark as defined by TPC.)

Parameters

Pepperdata set up a Spark benchmarking test environment on Amazon EKS to test Pepperdata Capacity Optimizer for large-scale workloads.

Pepperdata ran this workload both with and without Capacity Optimizer to capture the “before and after” effect of Capacity Optimizer.

Results

On Amazon EKS, Capacity Optimizer decreased instance hours and cost by 41.8 percent. Additionally, Pepperdata Capacity Optimizer also improved average memory and core utilization, which enables greater throughput. Average memory utilization increased 6.4 percent while average core utilization increased 87.5 percent, a significant uplift.

Further detail is available in the full report Pepperdata Reduces the Cost of Running Spark Workloads on Amazon EKS at Scale by 41.8%.

[10] TPC-DS is a Decision Support Benchmark

Appendix 3: Dramatic Results at Scale in Real-World Customer Environments

	CUSTOMER ENVIRONMENT	CASE STUDY
AUTODESK	Amazon EMR	Autodesk Reduces Amazon EMR Costs by 50% and Boosts Performance with Pepperdata
EXTOLE	Amazon EMR	Within Five Days, Pepperdata Reduced Extole's Amazon EC2 Compute Cost by an Additional 30 Percent
FORTUNE 10 TECH COMPANY	Amazon EMR on EKS	Pepperdata Reduced the Cost of Amazon EMR on EKS by 42.5%
FORTUNE 500 CONSUMER BRAND	Amazon EKS with special focus on • Karpenter for advanced autoscaling • YuniKorn for advanced scheduling	Pepperdata Saves Large Consumer Internet Company 33% for Apache Spark on Amazon EKS
FORTUNE 50 BANK	On Premises	Financial Services Giant Saves $20M with Pepperdata Real-Time Cost Optimization
FORTUNE 10 TECH COMPANY	Migration from On Premises to Amazon EMR	Global Technology Brand Name Realizes up to 25% Cost Savings on Amazon EMR with Pepperdata

Resources

1. **Blog Series:** The Five Myths of Apache Spark Optimization
    a. Myth 1: Observability & Monitoring
    b. Myth 2: Cluster Autoscaling
    c. Myth 3: Instance Rightsizing
    d. Myth 4: Manual Application Tuning
    e. Myth 5: Spark Dynamic Allocation
    f. I've Done All I Can: A Bonus Myth and a Solution
2. **Benchmark:** Pepperdata Reduces the Cost of Running Spark Workloads on Amazon EMR 7.0 by 51%
3. **Benchmark:** Pepperdata Reduces the Cost of Running Spark Workloads on Amazon EKS at Scale by 41.8%
4. **Case Study:** Financial Services Giant Saves $20M with Pepperdata Real-Time Cost Optimization
5. **Case Study:** Global Technology Brand Name Realizes up to 25% Cost Savings on Amazon EMR with Pepperdata
6. **Case Study:** Autodesk Reduces Amazon EMR Costs by 50% and Boosts Performance with Pepperdata
7. **Case Study:** Within Five Days, Pepperdata Reduced Extole's Amazon EC2 Compute Cost by an Additional 30 Percent
8. **Case Study:** Pepperdata Reduced the Cost of Amazon EMR on EKS by 42.5%
9. **Case Study:** Pepperdata Saves Large Consumer Internet Company 33% for Apache Spark on Amazon EKS
10. **Data Sheet:** Pepperdata Capacity Optimizer
11. **Solution Brief:** Pepperdata for Amazon EMR
12. **Solution Brief:** Pepperdata for Amazon EKS

About us

Founded in 2012 in Silicon Valley, Pepperdata has saved enterprises over $200 by optimizing performance of their large-scale data-analytics investments both on premises and in the cloud. Pepperdata Capacity Optimizer is the only real-time cost optimization solution that delivers up to 47 percent greater cost savings—continuously and autonomously—with no application changes, no recommendations, or manual tuning required. Our customers include the largest, most complex, and highly-scaled clusters in the world, at top enterprises such as Citibank, Autodesk, IQVIA, Royal Bank of Canada, and those in the Fortune 5. For more information, visit pepperdata.com.