pepperdata

BENCHMARK

Pepperdata Reduces the Cost of Running Spark Workloads on Amazon EMR 7.0 by 51%

Benchmark Results Using a TPC-DS Model on Amazon EMR 7.0 at Scale

Introduction

Given that 30 percent (or more) of cloud computing services regularly goes to waste, and as minimizing that waste has become the top priority for FinOps practitioners, 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 has improved resource utilization and optimized price/performance in some of the most complex and highly-scaled enterprises around the globe over the last decade.

Because each enterprise environment is different, Pepperdata embarked upon third-party benchmarking to provide an objective standard against which to measure the true power of Capacity Optimizer in reducing waste and unnecessary spend on Amazon EMR 7.0 in an AWS Graviton-based cluster. The Graviton instance type was chosen after testing several instance types to determine which delivered the best price/performance for this benchmarking effort.

Standardized Benchmark Methodology

Pepperdata selected TPC-DS, the decision support benchmark from the Transaction Processing Performance Council, 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.)

Benchmark Environment and Execution

Pepperdata set up the Spark benchmarking test environment on Amazon EMR 7.0 using Graviton instance types to test Pepperdata Capacity Optimizer according to the following parameters:

- **Workload:** TPC-DS with Spark SQL (104 queries)
- **Amazon EMR version:** 7.0.0
- **Graviton instance types:**
  - Master node instance type: m7g.xlarge
  - Core node instance type (min: 1, max: 1): c7g.8xlarge
  - Task node instance type (min: 1, max: 50): c7g.8xlarge
- **Apache Spark configuration:**
  - spark.driver.cores: 4
  - spark.driver.memory: 6g
  - spark.executor.cores: 4
  - spark.executor.memory: 6g
  - spark.executor.memoryOverhead: 2g
  - spark.dynamicAllocation.enabled: TRUE
  - spark.dynamicAllocation.maxExecutors: 200

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

Key Performance Metrics

Pepperdata used the following metrics to assess the performance and effectiveness of Capacity Optimizer:

- Total instance hours consumed
- Average concurrent container count
- Memory Utilization
- CPU Utilization

All of these metrics were obtained through the Pepperdata dashboard. The Pepperdata dashboard gathers and aggregates hundreds of cluster metrics in near real time to provide visibility into the performance of all applications running in a cloud environment via a single pane of glass.

Summary of Key Findings

Using the standardized workload and benchmark environment described above, Pepperdata measured the following:

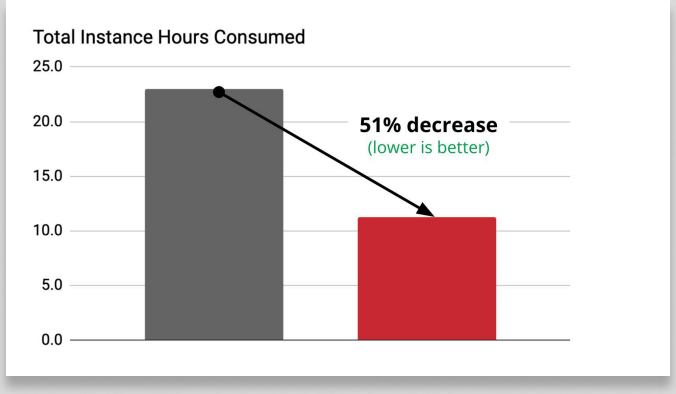


Figure 1: **Total instance hours consumed decreased by 51%.**
The 51 percent reduction that Capacity Optimizer achieved in this environment translates to reduced cost, since cloud pricing is directly correlated with instance hour utilization.

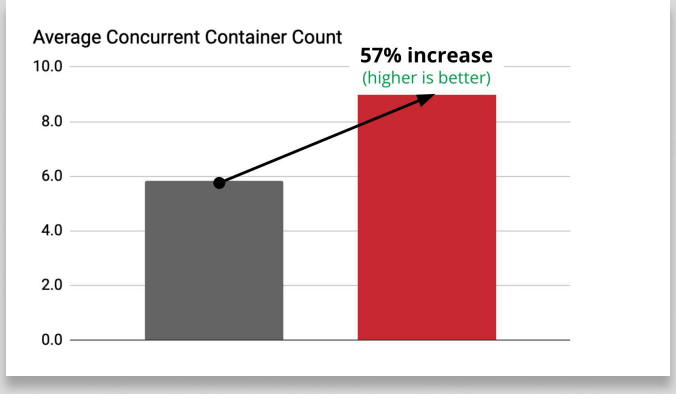


Figure 2: **Average concurrent container count increased by 57%.**
By increasing the average concurrent container count by 57 percent, Capacity Optimizer increased the ability of Amazon EMR 7.0 to process applications, thereby improving overall throughput.

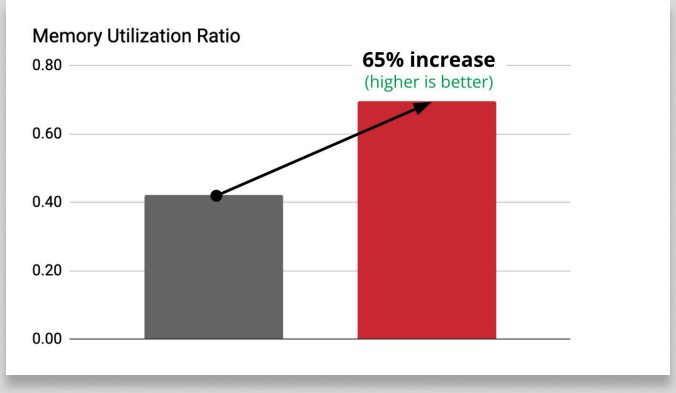


Figure 3: **Memory utilization increased by 65%.**
Increased memory utilization means that Capacity Optimizer enabled Amazon EMR 7.0 to run workloads more efficiently, leading to improved performance.

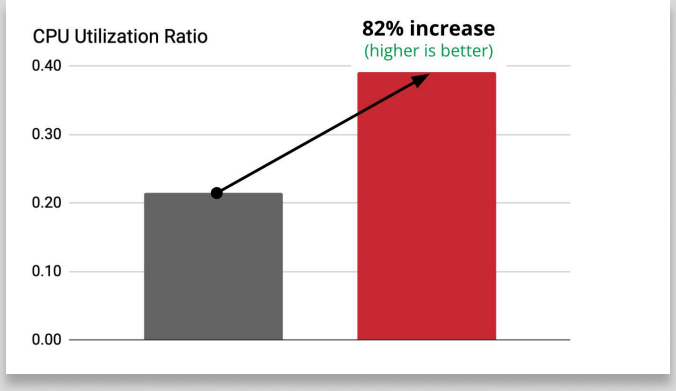


Figure 4: **CPU utilization increased by 82%.**
Increased CPU utilization indicates that Capacity Optimizer enabled Amazon EMR 7.0 to process more tasks and more efficiently utilize its processing power.

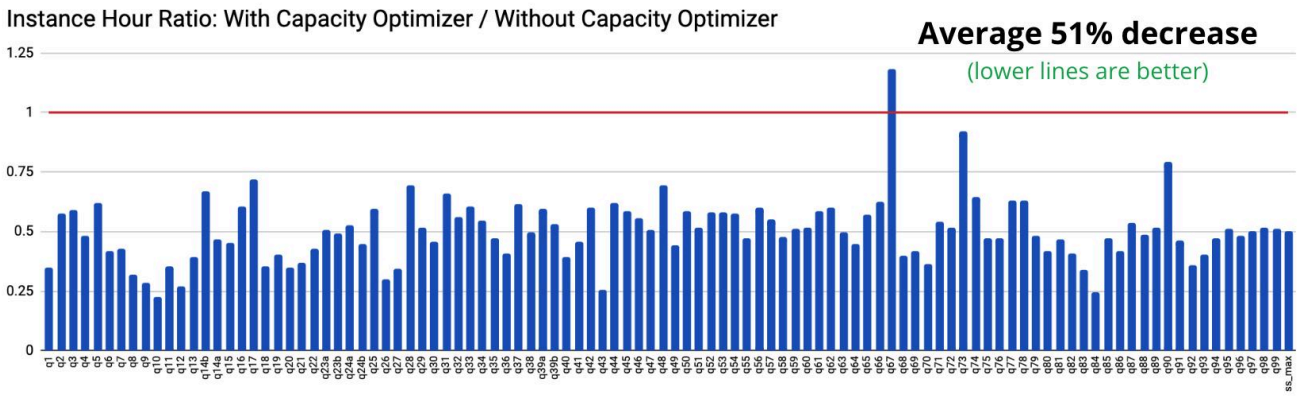
Cost Ratio Findings

In addition, Pepperdata compared the instance hours required to run all 104 queries both with and without Capacity Optimizer. Instance hours utilized by each query correlates directly to cost in the cloud environment. The more instance hours used, the higher the cloud bill at the end of the month. Pepperdata calculated and charted a ratio of two numbers for each of the 104 TPC-DS queries:

$$\frac{\text{instance hours required WITH Capacity Optimizer}}{\text{instance hours required WITHOUT Capacity Optimizer}}$$

- If Capacity Optimizer had no impact on the number of instance hours required to run a query, this ratio would be **one**.
- If Capacity Optimizer decreased the number of instance hours required to run a query, this number would be **less than one**. **This is a desirable outcome because fewer instance hours means lower cost.**
- If Capacity Optimizer increased the number of instance hours required to run a query, this number would be **greater than one**.

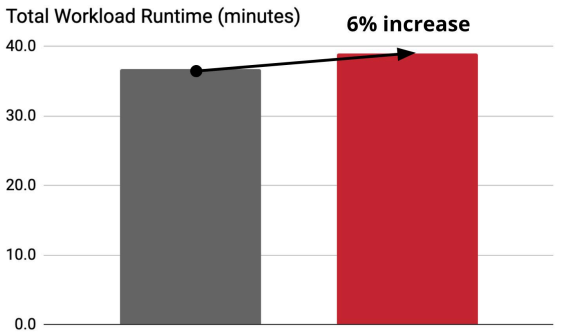
The following chart displays the query-by-query results that Pepperdata calculated. All the queries except one (Query 67) resulted in significant instance hour savings versus running the query without Capacity Optimizer enabled.



The power of Capacity Optimizer is in safely reducing the amount of resources required to run applications. As one the longest-running and CPU-intensive queries in the dataset, Query 67 benefited from excess pre-configured capacity to schedule more Spark-level tasks, allowing it to take advantage of the additional parallelism. In this way, the original provisioned capacity was ideal for Query 67 but wasted for all other queries. The Capacity Optimizer parameters for this benchmark were chosen to optimize for cost, meaning that fewer resources were available in general, which led to the longer Query 67 runtime to accommodate its peak requirements. A less aggressive set of parameters more attuned to SLA requirements would have provided Query 67 more resources to complete faster.

Overall Duration Results

Although overall duration was not a primary metric of interest in evaluating the effectiveness of Capacity Optimizer, Pepperdata did observe an approximate 6% increase in the overall runtime of the entire suite of queries.



Based on this criteria it is important to note that the **Capacity Optimizer parameters selected for this benchmarking effort were chosen to be relatively aggressive** since this benchmarking workload had a primary function of exploring instance hour count and container, CPU, and memory utilization related to costs. In a real-world customer environment, these parameters could be set less aggressively for more SLA-sensitive workloads, which would reduce the overall run-time duration.

Less aggressive settings would also result in a cost optimization below the 51 percent that was achieved in this benchmarking work. Importantly, less aggressive settings could also enable a **potentially equal or even reduced workload runtime to meet or exceed SLA goals**. Pepperdata provides this level of flexibility in its Capacity Optimizer settings to enable customers to tailor the desired level of optimization to a particular cluster's business and performance requirements.

Conclusion

Pepperdata's benchmarking work demonstrates that running a simulated TPC-DS industry-standard Apache Spark workload on Amazon EMR 7.0 with Pepperdata Capacity Optimizer demonstrated significant cost savings and performance improvements. **Capacity Optimizer reduced the total instance hours and related costs by an average 51 percent** (from 22.9 instance hours to 11.3 instance hours) and enabled workloads to utilize 65 percent more memory and 82 percent more CPU. In addition, Capacity Optimizer increased the average concurrent container count by 57 percent, meaning that more applications were able to run in a given time period, increasing the overall throughput of the environment.

As more companies migrate workloads to the cloud, these findings have important implications for cost and resource management. Capacity Optimizer's ability to autonomously optimize workloads such as Apache Spark on Amazon EMR 7.0 helps ensure that cloud resources are used efficiently and cost-effectively, making the cloud an even more attractive and viable option for Spark and other high-performance workloads.

Learn more about Capacity Optimizer here or explore a free Proof of Value in your environment. **It takes only six hours for Pepperdata to show you how much you can save.** Please contact us directly at info@pepperdata.com for more information.

Pepperdata installs in under 30 minutes in most enterprise environments. We guarantee a minimum of 100% ROI, with a typical ROI between 100% and 660%.

About Pepperdata

Pepperdata is the only cost optimization solution that delivers up to 47% greater cost savings—continuously and in real-time—on Amazon EMR and EKS with no application changes or manual tuning. Our customers include the largest, most complex, and highly-scaled clusters in the world, at top enterprises such as Citibank, Autodesk, Royal Bank of Canada, and those in the Fortune 5.
For more information, visit pepperdata.com.

Pepperdata, Inc.
530 Lakeside Drive
Suite 170
Sunnyvale, CA 94085

Start a Free PoV
www.pepperdata.com

Send an Email
eval@pepperdata.com