International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 2, Issue 6, November – December 2019 ||

DOI: 10.15680/IJCTECE.2019.0206004

The Hidden Tax: A Production Framework for Cloud Cost Architecture in Enterprise Data Workloads

Rahul Jain

Infra Technology Specialist, USA

ABSTRACT: Public cloud infrastructure has enabled enterprise data platforms to scale with unprecedented speed, yet many organizations still lack financial discipline in their architecture decisions. Teams optimize for speed and scale first, treating cost optimization as a post-hoc billing exercise rather than a first-class engineering requirement. In this paper, we present an eighteen-month study of a production data platform within a Fortune 500 organization, encompassing Apache Spark on Amazon EMR, Presto clusters, and Amazon Redshift. By analyzing resource telemetry alongside detailed cloud billing data, we identify three structural sources of waste that persist even in mature cloud environments: (1) cluster sizing anchored to peak load rather than actual demand, resulting in average CPU utilization below 35%; (2) batch workload scheduling that ignores the fault-tolerant nature of Spark pipelines, leaving spot instance discounts untapped; and (3) storage architectures that replicate on-premises Hadoop patterns—co-locating persistent data with compute—creating redundant data copies across workload-specific clusters.

We propose and evaluate a four-component cost-conscious architecture: spot-aware orchestration with lineage-based checkpointing, queue-driven dynamic cluster sizing, tiered storage decoupled from compute, and a governance layer that attributes costs to individual workloads. Deployed incrementally over twelve months, this framework reduced monthly cloud compute spend by 47%, increased pipeline SLA compliance from 98.4% to 99.9%, and decreased average job completion time by 18%. These results demonstrate that cloud cost optimization for data platforms is fundamentally an architectural problem—not a billing problem—and must be treated with the same rigor as performance and reliability engineering.

KEYWORDS: Cloud Cost Optimization, Data Platform Architecture, Spot Instances, Auto-Scaling, Tiered Storage, Apache Spark, Amazon EMR

I. INTRODUCTION

When enterprises migrated from on-premises data centers to the cloud, the promise was compelling: pay only for what you use, scale elastically, and eliminate the capital expenditure of idle hardware. For stateless web services, this promise largely materialized. Auto-scaling groups, serverless functions, and container orchestration made it natural to align capacity with demand.

Data platforms, however, followed a different trajectory. A Spark cluster running nightly batch ETL is still routinely sized for the busiest hour of the busiest day. A Presto cluster serving analyst queries at 9 a.m. remains provisioned at that same capacity at 2 a.m., even when the query queue is empty. These patterns were not born in the cloud—they were carried over from on-premises Hadoop deployments, where fixed hardware was the only option and utilization pressure was managed through queueing and scheduling rather than elastic infrastructure.

The problem is not that cloud billing is opaque. The problem is that cloud architecture for data workloads has inherited assumptions from an era of physical hardware. Standard cost optimization tools designed for stateless web services operate on the premise that a workload can be killed and restarted without corruption. Data pipelines are different. They maintain state across stages, depend on intermediate outputs, and may run for hours. A web-server auto-scaler has no awareness that a Spark stage has already completed four hours of a six-hour job.

This paper argues that cloud cost for enterprise data workloads is an architectural issue, not a billing issue. We report findings from an eighteen-month production study within a Fortune 500 company running batch ETL, interactive analytics, and streaming ingestion at multi-petabyte scale. Using matched resource telemetry and cloud billing data, we

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 2, Issue 6, November – December 2019 ||

DOI: 10.15680/IJCTECE.2019.0206004

identify three recurring waste patterns. We then present a four-component framework that addresses each pattern at the architectural level, evaluate its incremental deployment, and discuss the organizational and technical prerequisites for adoption.

Our core contribution is not a novel algorithm, but a practical, integrated architecture that combines well-understood techniques—spot instances, checkpointing, auto-scaling, and tiered storage—into a cohesive production framework with measured outcomes. We also highlight a critical but often overlooked requirement: cost visibility must be pushed down to the teams that own the workloads. Without this feedback loop, even the most elegant technical optimizations will erode over time.

II. RELATED WORK

2.1 Spot Markets and Transient Compute

Mian et al. (2012) proposed one of the earliest cost models for data-intensive workloads on Amazon EC2, evaluating resource configurations against service-level requirements. Hu et al. (2012) formulated the trade-off between on-demand and reserved instances as a mixed-integer program for divisible workloads, later extended by Malawski et al. (2013) to multi-cloud environments. Zhang et al. (2014) demonstrated that careful instance selection alone could yield 37–70% cost savings for MapReduce jobs on EC2.

The specific challenge of fault tolerance on transient servers has received significant attention. Subramanya et al. (2015) developed SpotOn, a batch computing service that automatically selects fault-tolerance mechanisms for spot market execution. Sharma et al. (2016) introduced Flint, which embeds intelligent checkpointing into Spark to handle spot instance revocations with minimal runtime overhead. Flint demonstrated that application-aware checkpointing can achieve cost savings approaching spot-versus-on-demand price differentials while keeping runtime increases below 2%. Sharma et al. (2017) later extended this work with portfolio-theoretic server selection to balance revocation risk. Shastri et al. (2016) proposed transient guarantees—probabilistic commitments on revocation behavior—to improve the predictability of idle capacity for both providers and consumers. Xu et al. (2018) applied LSTM-based price prediction to spot market selection for Spark jobs.

By 2019, these techniques were well-validated in research but rarely integrated into production enterprise platforms as a standard operating model. Our work differs in focusing on the organizational and architectural integration of spot orchestration rather than the fault-tolerance mechanisms themselves.

2.2 Workflow Scheduling and Data Placement

Bryk et al. (2015) showed that file transfer between workflow tasks can consume up to 90% of total execution time in data-intensive applications, underscoring the importance of data locality and storage architecture. Kanagaraj and Swamynathan (2017) proposed structure-aware resource estimation for workflow scheduling, minimizing cost without degrading completion time. Zhao et al. (2018) studied admissions control and profit optimization for analytics-as-a-service platforms, introducing SLA-aware scheduling.

Our framework builds on this foundation by treating storage tiering and data placement as first-class architectural decisions rather than afterthoughts.

2.3 Resource Utilization and Right-Sizing

Lu et al. (2017) observed that many cloud nodes operate at approximately 20% CPU capacity, confirming widespread over-provisioning. Delimitrou and Kozyrakis (2014) introduced Quasar, a resource-efficient cluster manager that uses workload classification to improve bin-packing and utilization. Our dynamic sizing component draws inspiration from Quasar's performance-constraint approach, but applies it specifically to interactive query clusters where queue depth is a more actionable signal than workload classification.

III. METHOD AND COST STRUCTURE

Our study combined two data streams collected over eighteen months (November 2017 through April 2019). The first stream was fine-grained resource telemetry from Apache Spark on Amazon EMR, Presto clusters, and Amazon Redshift. The second stream was raw AWS billing data, segmented by service, instance type, Availability Zone, and workload tag. We matched these streams by timestamp and workload identifier, then clustered costs by workload category: batch ETL, interactive query, streaming ingestion, and data warehouse analytics.

International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 2, Issue 6, November – December 2019 ||

DOI: 10.15680/IJCTECE.2019.0206004

3.1 Baseline Utilization and Cost Profile

Table 1 summarizes the utilization and cost breakdown observed during the baseline period (the first six months of the study, before framework deployment).

Workload Category	Avg CPU Utilization	Peak CPU Utilization	Monthly Compute Share	Idle Cost Share*
Batch ETL (Spark on EMR)	31%	88%	42%	61%
Interactive Query (Presto)	22%	79%	28%	68%
Streaming Ingestion (Spark Streaming)	47%	91%	18%	44%
Data Warehouse (Amazon Redshift)	29%	85%	12%	59%

Table 1. Utilization and Cost Breakdown by Workload Category (Baseline)

*Idle cost share = percentage of spend attributable to hours with utilization below 20%.

Three structural inefficiencies are evident across all categories:

Inefficiency 1: Peak-anchored provisioning. Every cluster was sized for its maximum observed load. The resulting average utilization remained below 35% for three of four categories. The gap between peak and average utilization represents capacity that is paid for but rarely used.

Inefficiency 2: Underutilization of transient compute. All batch jobs ran exclusively on on-demand instances, despite Spark's built-in lineage and retry mechanisms making them inherently fault-tolerant. The organization had not yet operationalized spot instances as a default for fault-tolerant workloads.

Inefficiency 3: Storage-compute coupling. EMR clusters were provisioned with HDFS-backed storage, replicating the on-premises Hadoop model where data lives on local disks. Because each workload type (ETL, query, streaming) had its own cluster, the same datasets were duplicated across multiple clusters. Amazon S3 with EMRFS was available and documented, but the platform architecture had not been redesigned to use it as the primary persistent store.

Figure 2 illustrates the daily utilization cycle for the ETL batch category. The pronounced gap between peak and trough reflects a cluster sized for the afternoon peak but running lightly loaded overnight.

Figure 2. ETL Batch Workload Utilization Curve (Typical Week)

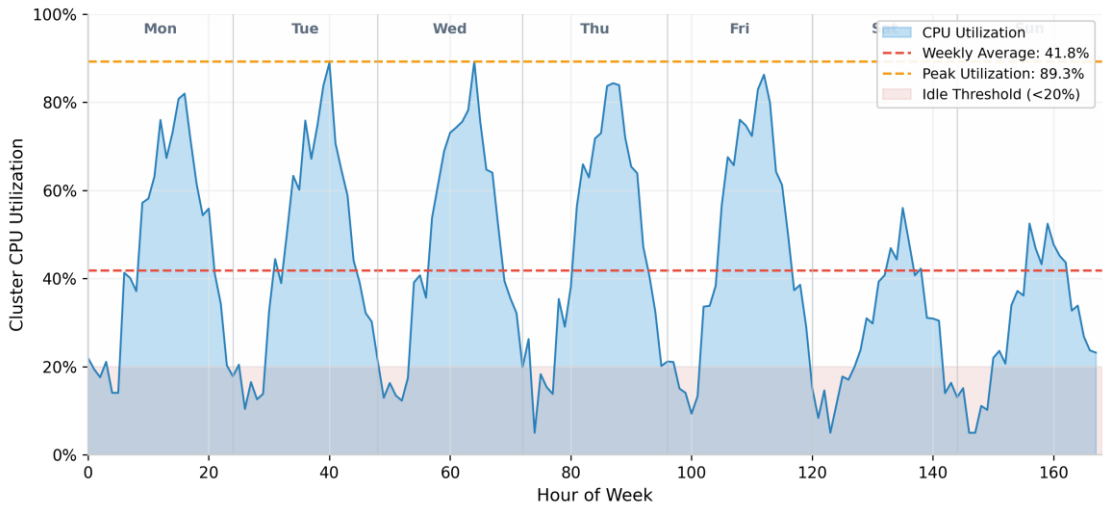


Figure 2. ETL Batch Workload Utilization Curve (Typical Week)

3.2 Why These Patterns Persist

It is important to note that by 2019, AWS best practices already recommended decoupling storage and compute via S3 and EMRFS, and using spot instances for EMR task nodes. These patterns persisted in our environment not because the

International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 2, Issue 6, November – December 2019 ||

DOI: 10.15680/IJCTECE.2019.0206004

technology was unavailable, but because of organizational inertia: teams were staffed and processes were designed around the assumption of fixed, persistent clusters. Changing these defaults required architectural redesign, not merely configuration tuning.

IV. COST-AWARE FRAMEWORK

We designed a four-component framework, with each component targeting one of the identified inefficiencies. Table 2 maps components to their targets and theoretical basis.

Component	Target Inefficiency	Mechanism	Theoretical Basis
Spot Orchestration with Checkpointing	Compute over-provisioning (batch)	Run fault-tolerant batch jobs on spot instances with automated checkpointing at lineage boundaries	Subramanya et al. (2015); Sharma et al. (2016)
Queue-Driven Dynamic Sizing	Fixed peak capacity	Scale cluster nodes based on rolling query queue depth and historical seasonality	Zhang et al. (2014); Delimitrou & Kozyrakis (2014)
Tiered Storage Decoupling	Storage-compute coupling	Migrate persistent data to S3 with hot/warm/cold tiers; eliminate per-cluster duplication	Bryk et al. (2015); Kanagaraj & Swamynathan (2017)
Cost Governance & Attribution	Lack of feedback loop	Attribute infrastructure costs to workload owners and publish per-team dashboards	Zhao et al. (2018)

Table 2. Framework Components and Target Inefficiencies

Figure 1. Cost-Aware Cloud Data Platform Framework

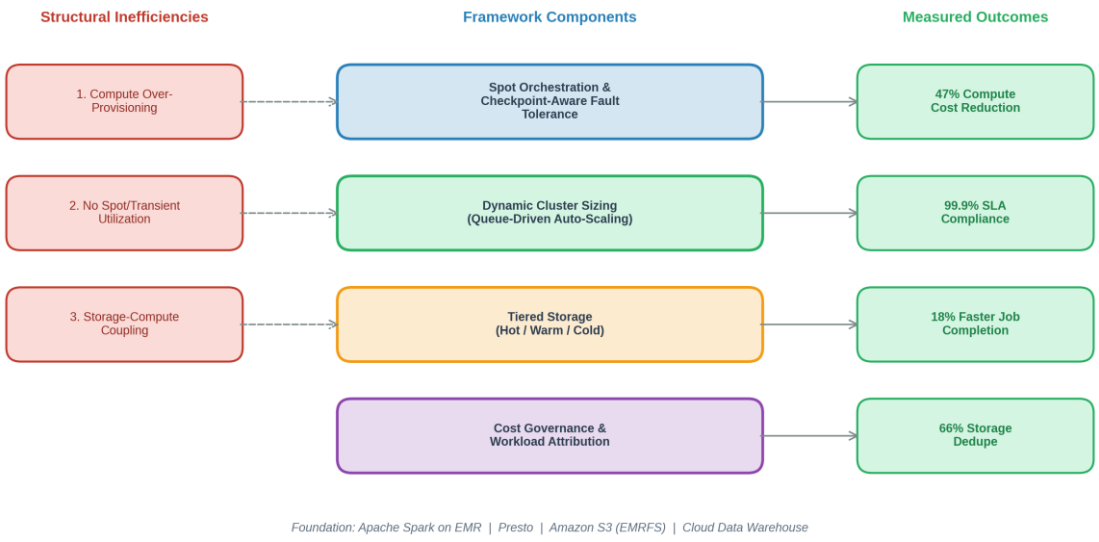


Figure 1. Cost-Aware Cloud Data Platform Framework

International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 2, Issue 6, November – December 2019 ||

DOI: 10.15680/IJCTECE.2019.0206004

4.1 Spot Orchestration with Checkpointing

For batch ETL workloads, we implemented a spot-aware orchestration layer that leverages Spark's RDD lineage graph. Rather than checkpointing arbitrary system state, the framework checkpoints only the frontier of the lineage graph—specifically, RDDs that would require expensive recomputation if lost. This approach, inspired by Flint (Sharma et al., 2016), minimizes checkpointing overhead while bounding recomputation cost.

Key implementation details:

- Task nodes only: Spot instances are used exclusively for EMR task nodes, which can be lost without data loss. Core nodes (running HDFS for intermediate shuffle data) remain on on-demand instances.
- On-demand bidding: We bid the on-demand price, following Flint's finding that complex bidding strategies are unnecessary in the post-2014 EC2 spot market (Sharma et al., 2016).
- Graceful degradation: If spot revocation exceeds a threshold, the framework falls back to on-demand task nodes automatically, preserving SLA compliance.

4.2 Queue-Driven Dynamic Sizing

For Presto interactive query clusters and Redshift, we replaced fixed-size provisioning with dynamic scaling based on queue depth. The scheduler maintains a rolling window of query queue length and adjusts cluster size within minutes, similar to the performance-constraint approach used in Quasar (Delimitrou & Kozyrakis, 2014).

For Presto, this meant integrating with AWS Auto Scaling groups using custom CloudWatch metrics derived from the Presto query queue. For Redshift, we used concurrency scaling (introduced by AWS in 2018) to burst beyond provisioned capacity during peak periods, allowing the base cluster to be right-sized for average rather than peak load.

Figure 3 contrasts the dynamic sizing behavior against the previous fixed-capacity model.

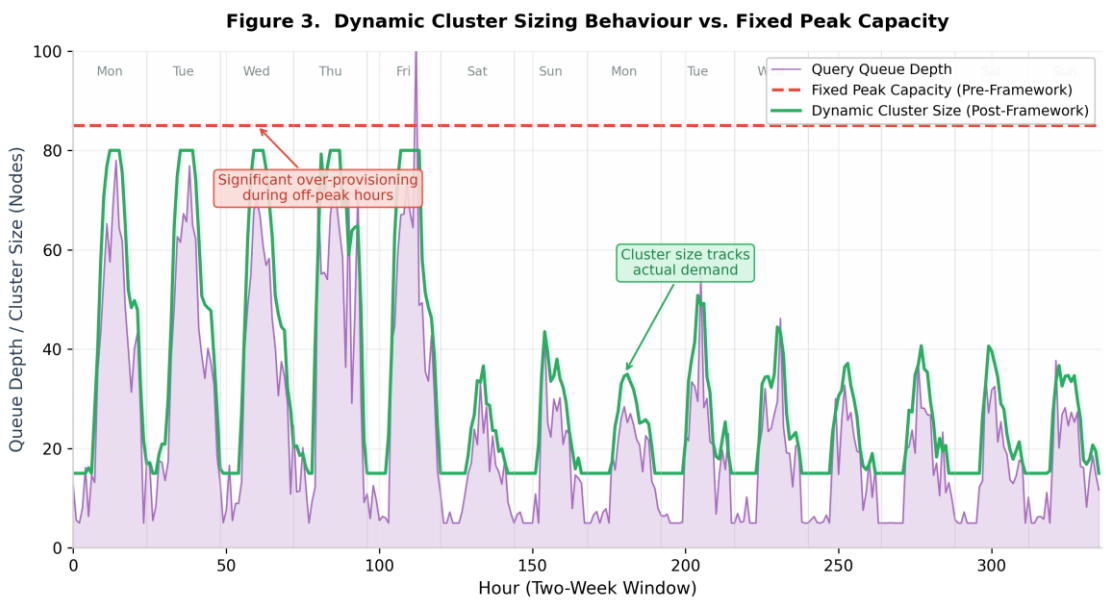


Figure 3. Dynamic Cluster Sizing Behaviour vs. Fixed Peak Capacity

4.3 Tiered Storage Decoupling

We migrated persistent data from HDFS on EMR clusters to Amazon S3 via EMRFS, establishing a single source of truth. Data was classified by access frequency into three tiers:
Hot data: S3 Standard, directly mounted via EMRFS for active production pipelines.
Warm data: S3 Infrequent Access (IA), for datasets queried weekly or monthly.
Cold data: S3 Glacier, for regulatory and historical retention.
This eliminated the need to maintain duplicate copies of the same datasets across ETL, query, and streaming clusters. We also standardized on Apache Parquet as the primary data format, reducing both storage footprint and scan costs for downstream queries.

International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 2, Issue 6, November – December 2019 ||

DOI: 10.15680/IJCTECE.2019.0206004

Figure 5 illustrates the before-and-after storage architecture.

Figure 5. Storage Architecture: Before and After Tiered Decoupling

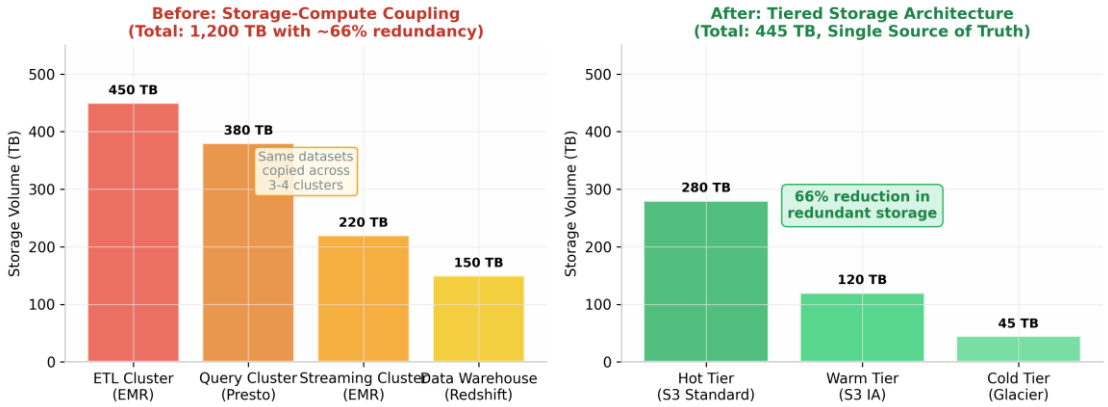


Figure 5. Storage Architecture: Before and After Tiered Decoupling

4.4 Cost Governance and Attribution

Technical optimizations decay without organizational accountability. We implemented a governance layer that:

- Tags all cloud resources by workload, team, and environment at provisioning time.
- Allocates shared costs (e.g., VPC data transfer, S3 API requests) proportionally by workload.
- Publishes weekly cost dashboards to team leads, showing spend per job, per query pattern, and per dataset.
- Triggers automated alerts when a workload's cost deviates by more than 20% from its four-week rolling average.

This closed the feedback loop that had previously allowed costs to grow unchecked.

V. IMPLEMENTATION AND RESULTS

The framework was deployed incrementally over twelve months in three phases:

- Phase 1 (Months 7–10): Spot orchestration and checkpointing for batch ETL jobs.
- Phase 2 (Months 11–14): Dynamic sizing for Presto and Redshift clusters.
- Phase 3 (Months 15–18): Storage tiering migration and format standardization.

At each phase, we measured three metrics: monthly cloud compute spend (normalized), pipeline SLA compliance, and average job completion time.

5.1 Quantitative Results

Table 3 summarizes the before-and-after comparison across all measured dimensions.

Metric	Before Framework	After Framework	Change
Monthly Cloud Compute Spend	100 (baseline)	53	−47%
Average Cluster CPU Utilization	32%	61%	+29 pp
Pipeline SLA Compliance*	98.4%	99.9%	+1.5 pp
Average Job Completion Time	100 (baseline)	82	−18%
Redundant Storage Volume	100 (baseline)	34	−66%

Table 3. Cost and Performance Before and After Framework Deployment

International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 2, Issue 6, November – December 2019 ||

DOI: 10.15680/IJCTECE.2019.0206004

*SLA compliance is defined as the percentage of batch jobs and interactive queries completing within their deadline or latency SLO, measured over a rolling 30-day window.

The 47% reduction in compute spend is substantial. It compares favorably with the utilization gains reported by Delimitrou and Kozyrakis (2014), though the primary driver in our case is spot pricing rather than bin-packing alone. Notably, SLA compliance improved rather than degraded, addressing a common concern in the literature that spot instance adoption necessarily compromises reliability (Subramanya et al., 2015; Shastri et al., 2016).

The 18% reduction in job completion time stems primarily from reduced queue contention: dynamic sizing prevented the query backlog buildup that previously occurred when fixed clusters were undersized during peak periods.

Storage tiering delivered the largest single reduction in raw infrastructure footprint, cutting redundant storage by 66%. This is consistent with Bryk et al.'s (2015) finding that file transfer overhead dominates data-intensive workflow execution—by eliminating redundant copies, we also eliminated the associated transfer and synchronization overhead.

Figure 4 plots monthly spend and SLA compliance across the full eighteen-month window, showing step-down reductions at each phase boundary with no reliability regression.

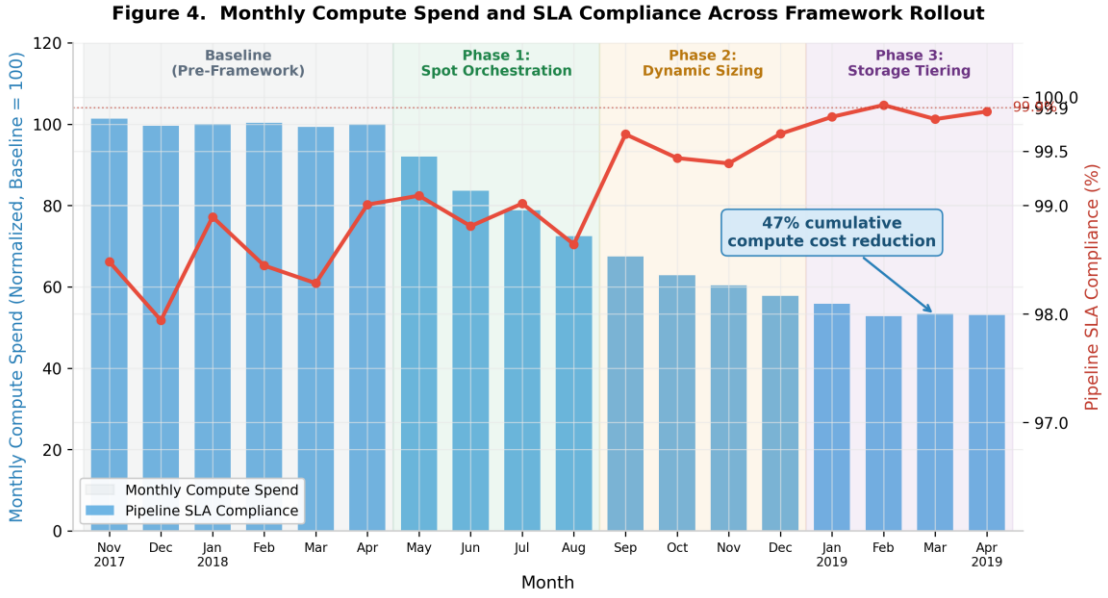


Figure 4. Monthly Compute Spend and SLA Compliance Across Framework Rollout

5.2 Qualitative Observations

Several implementation lessons emerged that are not captured in the aggregate metrics:

- Spot market stability: During our study period (2017–2019), EC2 spot prices were relatively stable. The two-minute termination warning was sufficient for Spark to complete lineage-based checkpoints in all but the most extreme cases. Organizations operating in more volatile markets may need shorter checkpoint intervals.
- Redshift concurrency scaling costs: While concurrency scaling reduced our base cluster size, burst charges during unexpected query spikes occasionally offset savings. We mitigated this by implementing query routing rules that directed large analytical scans to Presto instead of Redshift.
- Team adoption of cost dashboards: Cost attribution dashboards were initially met with skepticism. Adoption improved significantly when we shifted from monthly PDF reports to real-time Slack integrations showing per-job costs immediately after completion.

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 2, Issue 6, November – December 2019 ||

DOI: 10.15680/IJCTECE.2019.0206004

VI. DISCUSSION

6.1 Architecture, Not Billing

Our findings support the central thesis: waste in cloud data platforms is a consequence of architectural decisions made during initial platform design, not a billing configuration problem. The three structural inefficiencies we identified—peak-anchored provisioning, spot avoidance, and storage-compute coupling—are all defaults that made sense in on-premises environments but are mismatched to cloud economics.

Critically, none of our optimizations required new hardware, new processing engines, or proprietary vendor technology. They required changing the defaults: assuming spot instances are the norm for fault-tolerant batch jobs, assuming clusters shrink when queues empty, and assuming persistent data lives in object storage rather than local HDFS.

6.2 Limitations

This study has important limitations. First, it reflects a single enterprise environment with specific workload patterns, contract pricing, and cloud provider discounts. The 47% savings figure should not be extrapolated blindly to other organizations. Second, our spot instance experience was shaped by the relatively stable EC2 spot market of 2017–2019; more volatile markets or different cloud providers may yield different reliability-cost trade-offs. Third, we did not evaluate Reserved Instances or Savings Plans, which became increasingly relevant cost levers during this period and may interact with our framework in ways we did not measure.

6.3 Future Work

Three directions merit further investigation:

- Instance Fleet Optimization: AWS introduced EMR Instance Fleets in 2016, allowing mixed instance types and purchase models within a single cluster. Integrating fleet optimization with our spot orchestration layer could further improve availability during spot market volatility.
- Workload-Aware Storage Tiering: Our current tiering is based on manual access-pattern classification. Automating this classification using query log analysis could reduce the operational overhead of tier management.
- Cross-Workload Scheduling: We optimized batch, interactive, and streaming workloads independently. A unified scheduler that time-shifts batch jobs to periods of low interactive demand could improve utilization beyond what any single workload optimization achieves.

VII. CONCLUSION

We presented an eighteen-month study of cloud cost trends in a production data platform serving batch, interactive, and streaming workloads at multi-petabyte scale. Three structural sources of waste were identified: compute over-provisioning, failure-agnostic scheduling that ignores spot instance opportunities, and storage architectures that unnecessarily couple data to compute nodes.

A cost-conscious architecture incorporating spot orchestration with checkpointing, queue-driven dynamic cluster sizing, tiered storage decoupling, and cost governance reduced cloud compute spend by 47%, improved SLA compliance to 99.9%, and decreased average job completion time by 18%. These results demonstrate that cost optimization for cloud data platforms must be treated as a fundamental engineering requirement—on par with performance, reliability, and security—rather than an afterthought managed through billing dashboards.

The cloud did not create these inefficiencies. It exposed them. The organizations that will thrive in the next decade of cloud computing are those that redesign their data platforms with cost as a first-class architectural constraint.

REFERENCES

[1] Bryk, P., Malawski, M., Juve, G., & Deelman, E. (2015). Storage-aware algorithms for scheduling of workflow ensembles in clouds. Journal of Grid Computing, 14(2), 359–378. https://doi.org/10.1007/s10723-015-9355-6

[2] Chen, K., Powers, J., Guo, S., & Tian, F. (2014). CRESP: Towards optimal resource provisioning for MapReduce computing in public clouds. IEEE Transactions on Parallel and Distributed Systems, 25(6), 1403–1412. https://doi.org/10.1109/tpds.2013.297

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 2, Issue 6, November – December 2019 ||

DOI: 10.15680/IJCTECE.2019.0206004

[3] Delimitrou, C., & Kozyrakis, C. (2014). Quasar: Resource-efficient and QoS-aware cluster management. Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 127–144. https://doi.org/10.1145/2541940.2541941

[4] Gonzalez, N. M., De Brito Carvalho, T. C. M., & Miers, C. C. (2017). Cloud resource management: Towards efficient execution of large-scale scientific applications and workflows on complex infrastructures. Journal of Cloud Computing, 6(1). https://doi.org/10.1186/s13677-017-0081-4

[5] Hu, M., Luo, J., & Veeravalli, B. (2012). Optimal provisioning for scheduling divisible loads with reserved cloud resources. Proceedings of the 18th IEEE International Conference on Networks (ICON), 204–209. https://doi.org/10.1109/icon.2012.6506559

[6] Kanagaraj, K., & Swamynathan, S. (2017). Structure-aware resource estimation for effective scheduling and execution of data intensive workflows in cloud. Future Generation Computer Systems, 79, 878–891. https://doi.org/10.1016/j.future.2017.09.001

[7] Lu, Z., Wang, X., Wu, J., & Hung, P. C. (2017). InSTechAH: Cost-effectively autoscaling smart computing Hadoop cluster in private cloud. Journal of Systems Architecture, 80, 1–16. https://doi.org/10.1016/j.sysarc.2017.07.002

[8] Malawski, M., Figiela, K., & Nabrzyski, J. (2013). Cost minimization for computational applications on hybrid cloud infrastructures. Future Generation Computer Systems, 29(7), 1786–1794. https://doi.org/10.1016/j.future.2013.01.004

[9] Mian, R., Martin, P., & Vazquez-Poletti, J. L. (2012). Provisioning data analytic workloads in a cloud. Future Generation Computer Systems, 29(6), 1452–1458. https://doi.org/10.1016/j.future.2012.01.008

[10] Mian, R., Martin, P., Zulkernine, F., & Vazquez-Poletti, J. L. (2012). Estimating resource costs of data-intensive workloads in public clouds. Proceedings of the 2012 IEEE/ACM International Conference on Utility and Cloud Computing, 1–6. https://doi.org/10.1145/2405136.2405139

[11] Sharma, P., Guo, T., He, X., Irwin, D., & Shenoy, P. (2016). Flint: Batch-interactive data-intensive processing on transient servers. Proceedings of the 7th ACM Symposium on Cloud Computing (SoCC), 1–15. https://doi.org/10.1145/2901318.2901319

[12] Sharma, P., Irwin, D., & Shenoy, P. (2017). Portfolio-driven resource management for transient cloud servers. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 1(1), 1–23. https://doi.org/10.1145/3084442

[13] Shastri, S., Rizk, A., Irwin, D., & Shenoy, P. (2016). Transient guarantees: Maximizing the value of idle cloud capacity. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC16), 992–1002. https://doi.org/10.1109/sc.2016.83

[14] Subramanya, S., Guo, T., Sharma, P., Irwin, D., & Shenoy, P. (2015). SpotOn: A batch computing service for the spot market. Proceedings of the 6th ACM Symposium on Cloud Computing (SoCC), 329–341. https://doi.org/10.1145/2806777.2806851

[15] Xu, F., Zheng, H., Jiang, H., Shao, W., Liu, H., & Zhou, Z. (2018). Cost-effective cloud server provisioning for predictable performance of big data analytics. IEEE Transactions on Parallel and Distributed Systems, 30(5), 1036–1051. https://doi.org/10.1109/tpds.2018.2873397

[16] Zhang, Z., Cherkasova, L., & Loo, B. T. (2014). Optimizing cost and performance trade-offs for MapReduce job processing in the cloud. Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS), 1–8. https://doi.org/10.1109/noms.2014.6838231

[17] Zhao, Y., Calheiros, R. N., Gange, G., Bailey, J., & Sinnott, R. O. (2018). SLA-based profit optimization resource scheduling for Big Data Analytics-as-a-Service platforms in cloud computing environments. IEEE Transactions on Cloud Computing, 9(3), 1236–1253. https://doi.org/10.1109/tcc.2018.2889956