International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V73I6P110

Volume 73 Issue 6, 82-88, June 2025

Original Article

Optimizing Cost and Performance in Cloud Data Lakes

Dharanidhar Vuppu[1], Mounica Achanta[2]

[1]*Sr Data Engineer, SurveyMonkey, Texas, United States of America.*
[2]*Independent Research at IEE, Texas, United States of America.*

[1]*Corresponding Author : dharanidhar@ieee.org*

Received: 30 April 2025 Revised: 01 June 2025 Accepted: 19 June 2025 Published: 30 June 2025

Abstract - *As organizations increasingly shift towards cloud-native data platforms, balancing cost efficiency and query performance has become a central challenge for data engineering teams. Cloud data lakes, especially those built using Amazon S3 for storage and Snowflake for computing, offer immense scalability and flexibility—but at scale, they also expose inefficiencies that can silently drive up operational costs and hinder performance.*

This paper presents lessons learned from designing, maintaining, and optimizing large-scale data pipelines that process billions of records across Amazon S3 and Snowflake. Drawing from real-world implementation experience, we explore common pitfalls such as suboptimal file sizing, inefficient warehouse usage, and schema design flaws that directly impact cost and performance. We detail practical strategies to address these challenges, including S3 lifecycle management, Snowflake clustering, workload-aware warehouse sizing, and cost-conscious modeling in dbt. Beyond optimization techniques, this article emphasizes the role of data engineers in making architectural decisions that balance performance with budget constraints. The goal is to make pipelines faster or cheaper in isolation and to create sustainable, scalable data systems that deliver value to technical and business stakeholders. Through this exploration, we contribute actionable insights to the data engineering community navigating the evolving landscape of cloud data lakes.

Keywords - *Cloud Data, Data Lakes, Query Performance, Data Pipelines, Resilience.*

1. Introduction

Cloud data lakes are transforming how we manage and organize information. They offer much flexibility and can scale effortlessly as the business grows, making it easy to handle different types and volumes of data. That is why many companies choose tools like Amazon S3 for low-cost storage and Snowflake for powerful, cloud-based computing. It is a game-changer for managing data efficiently.

At a high level, the appeal is simple:

- Store raw and historical data cheaply in S3.
- Use Snowflake to run fast, on-demand queries without managing infrastructure.
- Scale effortlessly as data grows.

However, this setup has real-world challenges in practice, especially when operating at scale. As data pipelines grow to process billions of records daily, even small inefficiencies can lead to runaway costs or unpredictable performance. A few common issues include:

- S3 accumulates unnecessary object versions and tiny files that inflate storage bills.
- Poorly tuned Snowflake warehouses consume more credits than necessary.
- Queries become slower over time due to a lack of pruning or model sprawl.

This paper reflects lessons learned from managing such systems, not in theory, but in production. Based on hands-on experience across marketing analytics, behavioral data, and large-scale reporting use cases, the goal is to:

- Highlight common pain points faced by data engineers in cloud data lake environments.
- Share practical strategies that improved either cost or performance, sometimes both.
- Offer actionable insights to help others make smarter architecture and design decisions.

Ultimately, this is not just a “how-to” for S3 and Snowflake. It is about contributing to the data engineering community with patterns that scale, avoid waste, and help teams build reliable, efficient data platforms in the cloud. (Kim, 2009)

2. System Architecture: S3 + Snowflake in a Modern Data Lake

In most modern data lake implementations, separating storage and computing is a guiding principle. In our setup, we use Amazon S3 as a flexible data storage space, while Snowflake is a powerful tool that helps us analyze that data. This section will explain how these two parts work together and the additional tools that keep everything running smoothly and effectively. (Gray & Shenoy, 2000)

At a high level, the architecture follows a multi-zone design pattern:

- Zone 1: Raw Layer (S3)
  All raw data, including event logs, API extracts, and third-party ingestions, lands in S3. This zone is append-only and often includes nested folders by date or source system. Compression formats like Parquet or gzip are used to reduce the storage footprint. This layer can act as a historical storage of data.
- Zone 2: Structured Layer (Snowflake)
  Using external tables or batch loads, data from S3 is ingested into Snowflake. Here, transformations are applied using dbt (data build tool); at this point, we can build dimensions and fact tables. The structured layer includes staging tables, deduplication logic, and semantic standardization.
- Zone 3: Reporting & Aggregations (Snowflake)
  This is the final consumer-facing layer. It includes fact and dimension models, aggregated metrics, and pre-joined tables that power dashboards and self-serve queries. The zone is optimized for downstream tools like Tableau or Power BI.

Supporting tools and orchestration:

- Airflow handles the orchestration of ingestion and transformation tasks. Each step in the pipeline is scheduled, monitored, and retried if needed.
- dbt is used extensively for managing transformation logic, version control, testing, and documentation of models.
- Fivetran or other ingestion tools may be used for quick integration with platforms like Salesforce, Google Ads, or GA4.

A few architectural decisions that helped this setup scale:

- Avoid direct queries on external tables and instead load them into Snowflake-managed storage.
- Using incremental dbt models to minimize compute during daily refreshes.
- Structuring S3 folders by partition keys (like date or source) to enable efficient data ingestion and lifecycle management.

This system has been designed with both agility and sustainability in mind. It allows for rapid iteration while keeping cost and performance considerations front and center. Over time, this architecture has evolved to support new use cases, higher data volumes, and stricter SLA expectations—all while avoiding major rework or downtime. (Borra, 2022)

3. Cost and Performance Challenges at Scale

Cloud data lakes promise that they scale effortlessly. Moreover, for the most part, they do—until they do not. As data pipelines mature and volumes grow, subtle inefficiencies begin to compound. What starts as a few dollars here and a few seconds there can evolve into thousands in unnecessary spending and long-running queries that miss SLAs.

Here are some of the most common cost and performance challenges observed when operating S3 and Snowflake at scale:

3.1. Storage Cost Creep in S3

While S3 is relatively cheap on a per-GB basis, costs can add up quietly:

- Frequent object versioning without lifecycle policies results in unnecessary storage.
- Small files—especially from streaming or micro-batch jobs—lead to inefficient reads and higher overhead.
- Ingesting compressed JSON or CSV instead of columnar formats like Parquet increases storage and query costs downstream.

3.2. Warehouse Over-Provisioning in Snowflake

In early stages, teams tend to overestimate computing needs:

- Using large or XL warehouses for small transformations leads to wasted credits.
- Developers often forget to set auto-suspend, causing warehouses to stay idle but billed.
- Long running or poorly optimized queries block other processes and trigger warehouse scaling without real need.

3.3. Lack of Pruning or Partitioning

As tables grow into hundreds of millions or billions of rows, full table scans become expensive:

- Without clustering keys, Snowflake struggles to skip irrelevant partitions.
- Queries on wide tables with many columns increase I/O and degrade performance.
- Filters applied in downstream models cannot use pruning if the upstream logic is not aligned.

3.4. Inefficient Transformation Logic

Data models that are simple to write may not scale well:

- Overuse of full refresh dbt models can spike compute usage unnecessarily.
- JOINs without proper keys or filters lead to a data explosion and memory issues.
- Caching assumptions often break down as more users or dashboards hit the system simultaneously.

3.5. Operational Blind Spots

Without observability, it is hard to trace where the money is going:

- Teams lack visibility into which queries or users consume the most resources.
- Query retries, failures, or suboptimal logic are discovered too late, sometimes after billing cycles.

Flaws in Snowflake or S3 themselves do not cause these challenges. Instead, they stem from how the tools are used—and often, from applying local, small-scale practices to global, large-scale systems. Recognizing and proactively addressing these issues becomes critical as data platforms grow. The following sections will discuss the practical techniques and design changes that helped tackle these challenges head-on.

4. Performance Bottlenecks

As data grows and processes become more complex, performance issues often creep in subtly. What runs smoothly with a few million records can quickly break down at scale. Here are some of the most common bottlenecks we have run into:

4.1. Full Table Scans Due to a Lack of Clustering

- Without clustering keys, Snowflake struggles to prune large datasets efficiently.
- Even filtered queries scan entire tables, increasing latency and warehouse load.

4.2. Bloated Models and Wide Tables

- Models that include too many columns—especially unused ones—slow down queries.
- Wide joins and SELECT * patterns increase memory consumption and reduce cache efficiency.

4.3. Overuse of Full-Refresh Models

- Rebuilding large tables daily, instead of using incremental logic, wastes compute and slows pipeline execution.
- This also increases the load on downstream models that depend on fresh upstream data.

4.4. Small File Inefficiencies in S3

- Ingesting thousands of tiny files leads to more metadata reads and fragmented performance.
- Batch loading suffers when files are not properly consolidated or compressed.

4.5. Unoptimized Joins and Filters

- JOINs without indexed keys or filter pushdown slow down query execution.
- Nested or multi-step CTEs can become black boxes if not periodically reviewed.

4.6. Concurrent user Load on Shared Warehouses

- Dashboards and ETL jobs running on the same warehouse compete for resources.
- Without isolation, heavy workloads can throttle or delay each other.

4.7. Lack of Monitoring and Feedback Loops

- Without visibility into query plans, slowdowns often go unnoticed until users complain.
- Missed opportunities to cache results, rewrite queries, or optimize model structure.

Addressing these bottlenecks early helps prevent long-term scalability issues. The following sections will explore how many challenges can be addressed through architectural refinements, adherence to modeling best practices, and enhanced resource allocation strategies.

5. Strategies for Cost Optimization

One key takeaway from working with cloud data platforms is that costs are not just about how much storage or compute you use—they heavily depend on your choices when designing your architecture. How you structure your data, how often you refresh models, and how efficiently you use computing resources can all greatly impact your overall spending.

Here are several cost-saving strategies that proved effective when working with Amazon S3 and Snowflake at scale:

5.1. Enforce Auto-Suspend and Right-Size Warehouses

It is tempting to default to medium or large Snowflake warehouses "just to be safe." However, that often leads to wasted credits. Instead:

- Use auto-suspend aggressively (e.g., 60 seconds).
- Assign different warehouse sizes for different workloads. Not everything needs a large warehouse—ELT jobs, audits, and test runs can often run on an x-small or small warehouse.
- Run performance benchmarking periodically to validate sizing. (Lee, 2020)

5.2. Adopt Incremental Models in dbt

Running a complete refresh daily might work when the data is small but it is unnecessary and expensive at scale.

- Leverage is_incremental() logic in dbt to process only new or updated records.
- Use merge or insert overwrite patterns to avoid redundant computation.
- Track refresh cost trends using dbt artifacts and query history.

5.3. Manage S3 Lifecycle Policies

Storage costs in S3 may look low, but they balloon over time without governance.

- Apply lifecycle rules to transition older data to lower-cost storage classes like Glacier or S3 Infrequent Access.
- Enable version expiration for buckets that do not need to retain all object versions.
- Consolidate small files into larger batches where possible to reduce metadata overhead.

5.4. Limit Materializations to what is Needed

Not every dbt model needs to be a table. Consider:

- Using ephemeral models for lightweight logic that can be compiled into downstream models.
- Switching from tables to views for low-frequency lookups or dimension models.
- Avoiding excessive layers of intermediate materialized tables—each adds storage and refresh cost.

5.5. Apply Cost Tagging and Monitoring

Snowflake and AWS both support resource tagging.

- Use tags to track cost by project, team, or environment (e.g., dev, staging, prod).
- Monitor usage in Snowflake's ACCOUNT_USAGE views to find the top queries and users consuming resources.
- Establish dashboards that show warehouse usage trends, credit burn rate, and unused models.

5.6. Archive and Separate Cold Data

Not all data needs to be in your main fact tables.

- Move older, rarely accessed data to an archive table or keep it in S3 with external table access.
- Split high-volume tables by period (e.g., current quarter vs. history) to reduce scan size.

These strategies helped reduce cloud spend while keeping the platform flexible and responsive. The key is not just reducing costs blindly but doing so in a way that supports performance and usability for the data teams and analysts downstream.

Next, we will explore tackling performance tuning without compromising cost efficiency.

6. Strategies for Performance Optimization

While cost optimization is important, performance tuning is equally critical, especially when working with large datasets and tight SLAs. Long-running queries, delayed dashboards, or sluggish data loads can frustrate teams and block decision-making. Fortunately, many performance issues are solvable with better modeling, smarter design choices, and a few tweaks in using Snowflake and S3. (Chang, 2015)

Here are the strategies that helped improve performance across various workloads:

6.1. Use Clustering Keys Wisely

Snowflake does not use partitions like traditional databases, but clustering keys can guide how data is organized on disk.

- Apply clustering on high-cardinality columns frequently used in filters, like event_date, user_id, or account_id.
- Monitor clustering depth and re-cluster only when the benefits outweigh the cost.
- Do not over-cluster—too many keys or aggressive re-clustering can hurt more than help.

6.2. Optimize File Formats and Sizes in S3

When ingesting data from S3, file characteristics matter:

- Prefer columnar formats like Parquet or ORC over row-based formats like CSV or JSON.
- Avoid large numbers of small files, especially for batch loads—group them into larger chunks to reduce metadata operations.
- Ensure files are compressed appropriately (e.g., Snappy for Parquet) to speed up reads.

6.3. Minimize Unnecessary Columns and Joins

Wide tables and heavy joins are common performance killers:

- Trim unused columns in staging and final models. Even a few extra columns can significantly increase query time.
- Push filtering logic as early as possible in the transformation pipeline.
- Where possible, denormalize data models used for reporting to avoid runtime joins.

6.4. Leverage Result Caching and Task Chaining

Snowflake supports automatic result caching for repeat queries:

- Schedule pre-aggregation models ahead of business hours so analysts hit cached results.
- Use task chaining in Snowflake to sequence transformations efficiently and predictably.

6.5. Build Dashboard-Specific Tables

Generic, one-size-fits-all models tend to underperform when used in multiple reporting tools.

- Create purpose-built tables tailored to the needs of each dashboard or reporting layer.
- Aggregate metrics ahead of time where real-time granularity is not required.

6.6. Use Query History and EXPLAIN Plans

Snowflake's query history is a goldmine for debugging slow queries:

- Use EXPLAIN plans to identify full scans, excessive I/O, or unnecessary sorting.
- Check if queries use filters as expected or scan entire tables due to missed optimizations.
- Regularly review the slowest queries and batch jobs to identify recurring patterns.

6.7. Isolate Workloads with Separate Warehouses

Performance suffers when competing jobs run on the same compute resources.

- Allocate separate warehouses for ETL pipelines, ad-hoc analysis, and dashboard refreshes.
- This avoids contention and ensures critical processes do not get throttled.

These techniques do not require massive re-engineering—they are often small changes that compound into meaningful gains. Shaving even a few seconds off queries or reducing warehouse load can make a big difference in user experience and team productivity in fast-moving data environments.

With cost and performance under control, the next section will explore broader takeaways from running this architecture at scale.

7. Lessons from Operating at Scale

Building a cloud data lake is one thing—scaling it to serve a growing organization with increasing data demands is another. Over time, how we build, monitor, and think about data pipelines must evolve.

Many lessons are only learned the hard way—through performance incidents, unexpected cost spikes, or inconsistent data quality surfacing in stakeholder meetings.

Here are some of the most important lessons that emerged from managing S3 and Snowflake at scale:

7.1. Simplicity Scales Better than Cleverness

It is easy to over-engineer early on, adding too many layers, abstractions, or custom logic. However, at scale, simplicity wins.

- A well-documented incremental model is better than a "smart" one that's hard to debug.
- Easy-to-understand pipelines are easier to maintain, optimize, and onboard others into.

7.2. The Most Expensive Queries often do not Look Dangerous at First

Some of the worst credit-burning queries were ad-hoc dashboards running wide queries on massive datasets with no filters.

- It is not always the big pipelines—sometimes a slow, repeated report runs every hour.
- Educating business users on query design and working with analysts on data access patterns made a surprising difference.

7.3. Monitoring Needs to Evolve with Growth

On a small scale, you can get away with checking job status manually. At scale, that is no longer feasible.

- Invest early in automated alerts, lineage tracking, and usage dashboards.
- Set up cost monitors—not just for billing, but to catch usage spikes before they become issues.

7.4. Not All Data Needs to be Immediately Available

There is pressure to make everything fresh and real-time, but that is not always necessary.

- Breaking out "cold" vs. "hot" data or summarizing older records kept query performance high and costs low.
- Understanding what business users *need*—versus what they ask for—helped avoid over-engineering.

7.5. Collaboration with Stakeholders is a Form of Optimization

Some performance issues were not technical—they came from unclear expectations.

- Frequent check-ins with analytics teams and dashboard users helped prioritize optimizations that mattered.
- Collaborating on SLAs and refresh timings helped align data freshness with actual business needs.

7.6. Build for Flexibility, not just Speed

In fast-paced teams, requirements change quickly. The architecture needs to support agility.

- Keeping raw data in S3 allowed us to reprocess historical data when upstream logic changed.
- Avoiding tight coupling between stages (like dbt models chained too tightly) allowed us to independently rerun or tweak pipeline parts.

7.7. Always Validate Assumptions at Higher Volumes

A query that works fine on 10 million rows may not behave the same on 1 billion.

- Test with production-like volumes before rolling out optimizations.

- Small design decisions—like sort order, filter pushdown, or column pruning—behave differently under load.

Scaling is not just about handling more data—it is about building smarter systems that adapt to change, manage complexity, and support users without constant firefighting. These lessons shaped how we think about long-term sustainability in our data lake and continue influencing how new pipelines are designed.

Next, we will look at emerging trends that may further reshape how data teams think about cloud data lake optimization. (Ravi & Musunuri, 2020)

8. Future Trends and Opportunities in Cloud Data Lake Optimization

The cloud data lake landscape is evolving quickly. What was considered advanced a few years ago, separating storage and compute, leveraging auto-scaling warehouses, or adopting columnar formats, is now table stakes. As teams aim to handle even more data with tighter budgets and faster turnaround, new tools and approaches are emerging to help data engineers stay ahead.

Here are some trends and opportunities likely to shape the future of cost and performance optimization in data lakes:

8.1. Open Table Formats (Iceberg, Delta Lake, Hudi) are Gaining Traction

The rise of open table formats brings powerful features like time travel, schema evolution, and ACID transactions to data lakes.

- These formats allow for better partition management and faster queries directly on S3, reducing the need to ingest everything into Snowflake.
- As Snowflake expands support for Iceberg tables, we will see tighter integration between raw data and compute layers, potentially reducing duplication and load times.

8.2. Increased Focus on FinOps and Chargeback Models

Cost transparency is becoming a priority, especially in larger organizations where multiple teams share the same Snowflake or S3 environment.

- FinOps practices push for granular cost attribution, resource tagging, and usage dashboards.
- Data engineers may partner more closely with finance teams to build usage-aware architectures and align data freshness with business value.

8.3. AI-Assisted Optimization and Observability

AI is making its way into query tuning and monitoring:

- Some platforms now offer AI-driven warehouse autoscaling, anomaly detection, and query rewrite suggestions.
- This opens up opportunities to catch inefficient patterns earlier and reduce the manual overhead of tuning and diagnostics.

8.4. Unified Data Pipelines Across Batch and Streaming

Traditionally, teams separated batch and real-time workloads. However, the tools are starting to converge.

- Frameworks that support both (e.g., Apache Iceberg + Snowpipe Streaming) allow for more flexible data ingestion and lower latency.
- This can reduce pipeline complexity and duplication while keeping costs in check by choosing the right compute mode at the right time.

8.5. Serverless and Pay-Per-Query Compute Models

For infrequent or unpredictable workloads, serverless query engines (like AWS Athena or Snowflake's automatic scaling options) are becoming more attractive.

- These models eliminate the need to pre-provision warehouses and can be cost-effective for teams with irregular usage patterns.
- As more teams adopt these options, architectural patterns may shift away from long-running, always-on compute setups.

8.6. Better Metadata and Data Quality Tooling

As optimization grows more complex, metadata becomes critical.

- Expect stronger adoption of data catalogues, column-level lineage, and automated documentation tools.
- Metadata can power more intelligent decisions, like skipping unneeded data, pruning historical segments, or flagging redundant models.

8.7. Shift from Raw Performance to Sustainable Performance

Instead of just trying to make things fast, teams are starting to ask: "What is fast *enough* for our users, and how do we maintain it over time?"

- This mindset pushes toward performance baselines, refreshes SLAs, and monitoring dashboards as part of the development lifecycle.
- It is a shift from firefighting to engineering for resilience and predictability.

These trends suggest a maturing of the data engineering space. It is no longer just about building pipelines—it is about building smart, transparent, and adaptable systems that can grow with the business.

As new tools and paradigms emerge, the role of the data engineer will continue to evolve—from just writing SQL to making architecture and cost-impact decisions that directly affect organizational agility.

Next, we conclude with key insights and reflections from our experience scaling cloud data lakes. (Kriushanth, Arockiam, & Mirobi, 2013)

9. Conclusion

Running data lakes at scale requires solid architecture, consistent discipline, and iterative refinement. The combination of Amazon S3 and Snowflake offers incredible flexibility, but that flexibility comes with responsibility, especially when controlling costs and maintaining performance. (Plale & Kouper, 2017)

Throughout this article, we have explored practical challenges and real-world strategies that emerged from building and operating large-scale pipelines. A few key takeaways that stand out:

- Cost and performance problems usually do not come from one bad query—they build up from many small, often overlooked design choices made over time.
- Optimization is not just about tweaking code. It means understanding how people use the data, staying aligned with business goals, and helping teams see how their work affects the bigger picture.
- Tools like dbt, Airflow, and Snowflake's native features offer a strong foundation, but how you use them makes all the difference. Incremental models, warehouse tuning, and smart S3 policies often outperform fancy architectural overhauls.
    - Monitoring and feedback loops are essential. As scale increases, problems become harder to detect and more expensive to fix if left untracked.

Ultimately, cost and performance optimization is not a one-time project—it is a mindset. It means designing pipelines for what works today and what will hold up under heavier load, stricter SLAs, or tighter budgets tomorrow.

As the cloud ecosystem continues to evolve, data engineers have a growing opportunity to shape data systems and the efficiency and sustainability of entire analytics platforms. By sharing lessons, patterns, and missteps, we can help each other navigate the complexity and build smarter, more resilient data lakes for the future. (Hlupić, Oreščanin, Ružak, & Baranović, 2022).

References

[1] Tomislav Hlupić et al., "An Overview of Current Data Lake Architecture Models," *Jubilee International Convention on Information, Communication and Electronic Technology*, Opatija, Croatia, pp. 1082-1087, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[2] Beth Plale, and Inna Kouper, "The Centrality of Data: Data Lifecycle and Data Pipelines," *Data Analytics for Intelligent Transportation Systems*, pp. 91-111, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[3] Victor Chang, "Towards a Big Data System Disaster Recovery in a Private Cloud," *Ad Hoc Networks*, vol. 35, pp. 65-82, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[4] J. Gray, and P. Shenoy, "Rules of Thumb in Data Engineering," *Proceedings of 16th International Conference on Data Engineering*, San Diego, CA, USA, pp. 3-10, 2000. [CrossRef] [Google Scholar] [Publisher Link]

[5] Dong Kyu Lee, "Data Transformation: A Focus on the Interpretation," *Korean Journal*, vol. 73, no. 6, pp. 503-508, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[6] Won Kim, "Cloud Computing: Today and Tomorrow," *Journal of Object Technology*, vol. 8, no. 1, pp. 65-72, 2009. [Google Scholar] [Publisher Link]

[7] Praveen Borra, "Snowflake: A Comprehensive Review of a Modern Data Warehousing Platform," *International Journal of Computer Science and Information Technology Research*, vol. 3, no. 1, pp. 11-16, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[8] M. Kriushanth, L. Arockiam, and G. Justy Mirobi, "Auto Scaling in Cloud Computing: An Overview," *International Journal of Advanced Research in Computer and Communication Engineering*, pp. 2278-1021, 2013. [Google Scholar] [Publisher Link]

[9] Vamsee Krishna Ravi, and Aravindsundeep Musunuri, Cloud Cost Optimization Techniques in Data Engineering, *SSRN*, vol. 7, no. 2, pp. 861-874, 2020. [CrossRef] [Google Scholar] [Publisher Link]