LTIMindtree

WHITEPAPER

Understanding the Cost Optimization & Sustainability Opportunities in Snowflake

LTIMindtree

Table of Contents

1. Abstract

This paper emphasizes the cost optimization and sustainability opportunities that can be realized with Snowflake by optimizing costs through design, process, and sustainable development practices. It also provides best practices for meeting cost and sustainability goals simultaneously.

2. Introduction

When purchasing a mobile data plan or using cloud resources, it is crucial to consider usage and pricing. Cloud usage can be particularly challenging to forecast, and as organizations strive to be data-centric, they require more cloud resources for enhanced business insights and better decision-making. Snowflake offers a transparent pricing model, but the real concern is when data grows over time, making compute costs unpredictable, especially with complex real-time analytics on large datasets. Therefore, cost optimization becomes paramount for IT and business teams to use resources wisely and adjust to new realities. With most organizations moving to the cloud, sustainability is now a major driver for the industry.

While Snowflake is committed to adopting sustainable practices of the cloud, customers must embrace sustainability-friendly solutions in the cloud, be conscious of costs, eliminate wastage, and reduce environmental footprints. Organizations need to consider their environmental impact when storing and managing data in the cloud. This includes finding ways to organize, process, interact, and transfer data cost-effectively and sustainably. By being mindful of these aspects, Organizations can enhance their decision-making capabilities while also contributing positively to the environment.

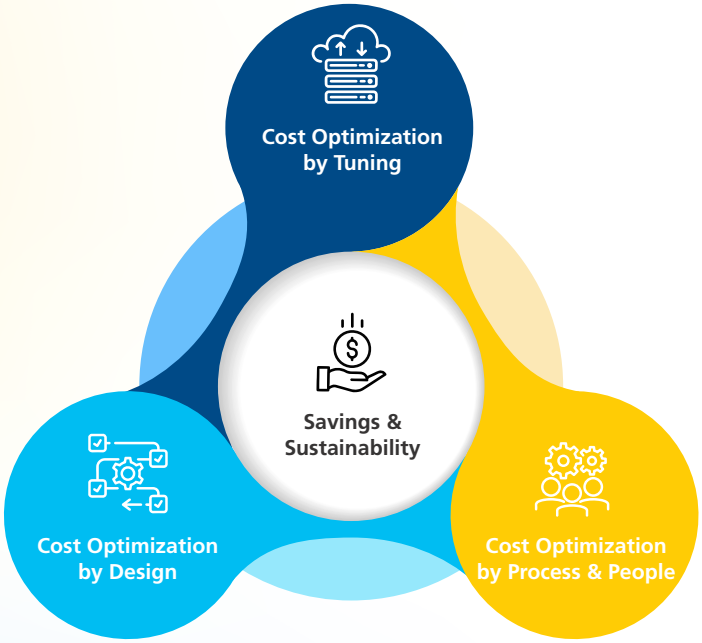

Figure 1: Illustration of three ways of continuous cost optimization for $ savings and sustainability

3. Optimizing cost by design

Unlike on-premise solutions, you also "**design for cost**" on the cloud. Any learning from the poor design of the cloud comes after significant spending and unproductive utilization of cloud resources. As more use cases are onboarded to Snowflake, it becomes more challenging to optimize and almost **impossible to re-design**. Hence, the cost should never be treated as a separate entity, but a **mandatory non-functional attribute** that should play an integral part in the design. The **cost-conscious design**, which simultaneously brings implicit sustainable practices, should be applied to the end-to-end process to onboard a use case. There is no automatic saving just for building solutions on Snowflake. However, a proper cost strategy offers **significant savings compared** to on-premises. The design decisions should be made in conjunction with all the stakeholders with the design standards outlining the desired Service Level Agreement (SLA) for cost, performance, reliability, sustainability, etc. While designing the solutions, bear in mind the following:

3.1. Define the object naming convention

The cost-conscious design starts with naming the objects in Snowflake. A naming convention framework is essential for improving findability and discoverability and helps trace costs. Without any framework in the naming, the metadata becomes **unfriendly and chaotic** for any cost analysis. Hence, the naming conventions cannot be left to the discretion of project teams but enforced via a framework that should be **descriptive and consistent** across the organization. The key design considerations for naming in the context of cost:

- Better identification of applications, subject area, use case, etc.
- Identification of environment types in the object names as applicable
- Enable easy identification of source systems data
- Identify computational resources by applications/business units, process types such as ETL/Reporting/Ad-hoc, etc., and sub-process types such as batch load, replication load, etc.

3.2. Design the tags for governance

Object and query tagging features are often considered good-to-have features and overlooked by many architects. Tagging falls under "**Know your data: what it is and where it is**." It is a powerful feature that is mandatory for any data governance. In the context of cost, it is an important design feature to be leveraged to classify the objects/logs for onboarding any use cases. Tags should not only be leveraged for Snowflake DB objects but should also be enabled in Extract, Load, Transform (ELT)/ETL processing to alter sessions for classifying the query logs, and it also helps **arrive at the cost per query**. Additionally, the tags help to:

- Slice and dice the cost of usage
- Identify the performance bottlenecks
- Increases the velocity of Root Cause Analysis
- Enable traceability of objects in the Snowflake query history
- Enable **charge-back model**

Below are the common consideration for tags

- Application ID
- Project ID
- Cost center
- Work Breakdown Structure (WBS) element
- **Sustainability reference flag**

If tags, by design, are not part of the solution, then it becomes complex and challenging to enable later as it requires global changes to all objects. The query logs already generated cannot be altered with tags, and the cost visibility is lost on such logs. Hence, the tags should be a **mandatory feature in the design checklist.**

3.3. Optimizing the compute costs by design

Compute warehouse is used to execute queries and load/unload the data etc. The credits are metered only for the time that compute resources are running and their size. It has the biggest impact on the overall cost. The utilization of compute resources plays a key role in sustainable development as there is no limit on its usage. As the compute is user managed, bear in mind the following factors.

- **Understanding the workload:** The workloads can be CPU intensive such as computing complex queries, I/O intensive such as read/write database operations, and GPU intensive workloads for data science requirements; it can also be heterogenous at times. There are specific SLA requirements for each type of workload to meet the business requirements, such as near real-time data, month/quarter-end cut off, etc. The workloads include ETL, BI reporting, data science, or ad-hoc querying requirements. After understanding the workload dynamics, the compute resource should be configured accordingly to meet the SLAs.
- **Allocate the right resources:** The workloads can be static, time-bound, unpredictable, and ad-hoc based. Assigning the right size to the workload is a key design decision. Over-provisioning the compute resource can be expensive and may not serve the purpose. It is recommended to start with a small size and scale over time as needed. **Being conservative** on the compute while comprehending the workload behavior helps with the cost and drives sustainability goals.
- **Segregate the workload:** Single virtual warehouse for all the workload results in resource contention. It is recommended to assign dedicated computing prowess based on the nature of workloads. Unlike On-premises systems, it eliminates the need for prioritizing resources during peak times. It also helps in better cost management and monitoring. Assigning dedicated compute for each use case helps in customizing the controls that can potentially yield better.
- **Leverage the elastic architecture:** Horizontal scaling feature dynamically allows to increase or decrease the clusters depending on the load. This way, the compute utilization can be optimized as the credits are consumed only for the resources used. The sudden concurrency requirements during the period end are automatically addressed with this feature optimally. However, the elasticity limit should be set to cater to the workload productively.

- **Embrace lean coding:** Modularize the codes by moving the repeatable and the reusable aspects, such as environmental variables, master data lookup, etc., into templates. It simplifies the code and improves the readability and maintainability. E.g., the lean coding principle can be adopted for **Infrastructure-as-a-Service** with pre-defined and approved guardrails at the Snowflake account level.
- **Enforce the native configurations:** There are various configurations in Snowflake platform to control the costs. E.g., statement time out in seconds, credit quota, min/max cluster configuration, etc. Enabling these configurations at the platform level prevents wasting credits without manual intervention. Incorporating these controls by a pre-defined design framework minimizes cost during use case enablement.
- **Leverage the native optimization features:** Materialized views are pre-computed by Snowflake. It incrementally refreshes the data for underlying data change and makes it readily available for consumption. It helps in optimizing the costs of frequently processing complex queries. **Search optimization** is another feature available to improve during consumption. **Micro partition clustering** on large tables helps organize the data in the storage layer to improve the query response time and save cost eventually. **Query acceleration service** automatically scales up the compute size to help improve the query elapsed time. However, there is a **trade-off** in using these features as a cost is associated with using them versus the anticipated savings. Having guardrails at the platform level on using these features is required as an incorrect application can lead to more cost.
- **Design to leverage cache:** Snowflake consumes credits to process a query for the first run and store the query result set in the cache. The **identical queries** on the subsequent run don't do the heavy lifting but are rather pulled from the query result cache at no cost. The applications that fire identical queries from multiple threads/sessions should leverage Snowflake cache. Hence, the application-generated queries should not have any unnecessary dynamic variables such as current date, last day of the month, etc. Adhering to Snowflake cache requirements while consuming data from applications helps save many credits. E.g., Business Object (BO) reports that consume data from Snowflake should utilize the cache. The first BO execution consumes the credits, followed by "No cost" consumption from the cache with much better performance. Once the BOBJ reports are scheduled to build the cache immediately after ETL jobs process the underlying data, the ad-hoc user executions will route directly to the query result cache to render the data.

- **Avoid anti-patterns:** **Cursors/loops** are examples of anti-patterns in Snowflake that consume a significant number of credits. Moving the procedural logic to SNOWPARK helps save the credits as the loops are executed in memory using the data frames. **Frequent merge statements** on a large dataset are another pattern that consumes significant credits. Instead of merging, **delete and bulk** insert ingestion patterns can be leveraged as applicable to save credits. There should be a standard data ingestion framework to acquire data from source systems, rules on when and when not to use, in line with the Snowflake principles, and cost-conscious design.
- **Optimal data ingestion patterns:** ETL/ELT solution design plays a major role in Snowflake credit consumption. Performing full load for every run is a costly process that doesn't add value to the business or to IT teams. It is important to design incremental loads for large data sets for better cost savings. Sorting the data while **naturally clustering the data** in Snowflake improves the performance during consumption. A reusable framework for ETL patterns for each source system type should be designed upfront and approved by the architecture team for the project teams to adopt. It enforces the standards, saves time and cost, and works towards all stakeholders' targets.
- **File size optimization:** It is no secret that **ingesting files to Snowflake using COPYINTO is faster.** COPY INTO is a powerful API that helps load the data files more optimally than relational INSERT INTO using ODBC connectors. However, loading one large file to Snowflake can be expensive as it keeps the cluster active for longer. And it also demands the need for large clusters to process the huge files. Splitting the files into smaller chunks and compressing them relieves the computing pressure and loads them in parallel with a relatively lesser computing size.

3.4. Optimizing the storage cost by design

Storage cost in Snowflake is much cheaper than compute cost. But cheaper doesn't mean optimized. The storage metrics analysis provides insights into the underlying design problems. Also, keeping the redundant data in the same region with uncontrolled data retention increase the carbon footprint and accounts for indirect emissions. Based on the standard US electricity rates, one terabyte of cloud data storage can create a carbon footprint equivalent to ~2 tons annually. Hence, it is important to keep the storage at an acceptable level.

- **Copying data across environments:** In Snowflake, the data can be cloned from one environment to another, provided they are logically separated by database naming within the same account. As clone is a metadata operation, the data doesn't get copied or duplicated, and no computation is needed for cloning the data. Copying large datasets in TBs can be faster during cloning at no extra cost. Design rules should properly govern and address such requirements to leverage cloning more than CTAS.

- **Leverage the native features for storage optimization:**

Time travel is a native Snowflake feature that helps to look at the data at any given point in the past subjected to Snowflake edition. Snowflake maintains the metadata to restore the table data that are deleted/updated. In case of full table drop or truncation, the entire copy of the table is maintained. Performing full load on tables for every ETL will result in an exponential increase in the time travel storage as Snowflake requires maintaining the copy of the entire data set for each run. The time travel parameter should be configured in such scenarios and cannot be left to a default value. Data retention time in days allows us to define the DB, schema, or object level setting. The parameter should be properly defined for each use case during the design phase to optimize the implicit storage cost.

Temporary and transient tables: Transient table is a native Snowflake feature that helps optimize storage cost. It should be used in storing the intermediate data during the processing. As there is limited or no implicit storage available for such tables in Snowflake, the transient tables should be considered in ETL worktables. There should be a design framework that enforces the usage of temporary/transient tables programmatically for the application teams /project teams. Having such a framework helps optimize implicit storage. Uncontrolled configuration of time travel or using permanent tables for intermediate data storage during ETL processing exponentially spikes the implicit storage size.

External tables: Using this feature helps both storage and compute costs in certain use cases, such as rendering data from Simple Storage Service (S3)/Azure Data Lake Service (ADLS), the external stage data is not used frequently, etc. If the data is granular and is already offloaded to the external stage location, the external table can be built on top. As a result, the storage cost and the compute cost to ingest from the external stage location to Snowflake are saved provided there is a trade-off with performance.

3.5. Optimizing the cloud services, serverless, and data transfer cost

Besides storage and computing, Snowflake charges for cloud services for metadata management, infrastructure management, APIs, security access control, etc. Snowflake only bills for resources that exceed 10% of the daily credit usage as cloud services cost. Otherwise, there are no cloud service costs. It is essential to assess the patterns to control the usage. e.g.

- Frequent cloning of database objects
- Frequent execution of DDL commands
- Stored procedures with procedural logic, such as cursors/loops
- High frequent information schema queries

Snowflake manages serverless features such as Snowpipe. There are several factors, such as **file size, file type, query complexity**, etc., that cause credit consumption. It is not possible to estimate the serverless compute cost in advance. However, there are some design standards that need to be adhered for cost optimization.

- Staging small files results in queuing, and the costs are calculated based on file queue count.
- Staging large files will always result in longer processing time and higher costs.
- An optimal file size of ~**100 MB compressed** is recommended to continuously load data using pipes.

Using Snowflake replication services incurs a cost. As part of **business continuity and Disaster Recovery (DR)**, there is a need to replicate the Snowflake account to other accounts subjected to compliance needs. These are **unavoidable costs** incurred as it is a mandatory requirement for organizations. However, keeping multiple copies of data across regions creates emissions and contradicts the sustainability practice. A few factors need to be considered for these services.

- Define the replication scope that is subjected to DR
- Eliminate the objects that are not required for replication
- Decide the replication frequency, such as daily or weekly
- Ingress and egress costs vary by region and cloud provider

4. Optimizing cost by tuning

It is completely wrong to state that all the cost optimization needs are addressed in the design. Yet, there can be significant bottlenecks at the component level, such as lack of throughput and poor query elapsed time that degrades the performance and consumes unproductive credits. The bottlenecks are typically caused by **over-utilization of Snowflake** resources at the platform level, bad connectors, poorly written SQLs, full table scans, cursors, recursive SQLs, etc. Once the bottlenecks are identified, the appropriate tuning is required for **elimination.**

Performance tuning is a **highly iterative process** that takes time to identify patterns and develop skills to accurately narrow the critical bottlenecks. Some bottleneck elimination inevitably does not impact the cost; sometimes, it can lead to another bottleneck. Determining **the performance goals with exit criteria** makes the process transparent and simple for all stakeholders. Unlike traditional systems, the performance problem in Snowflake is not just a system problem but also a cost problem. Hence, the measure of success for performance is the **end-user experience and consumption experience** while interacting with the data in Snowflake within the estimated cost.

Proactive monitoring and setting up guardrails at the platform level can be considered the first cost optimization level. The tuning involves two sequences of steps:

4.1. Bottleneck identification

Snowflake logs all the usage metrics of the account in the query history. Gaining visibility into the query logs with all the necessary statistics to analyze, such as

- Compute resource that caused the spike in credit usage in a given period
- Elapsed time includes the compilation, queuing, execution and provisioning time
- Data spillage metrics highlight the compute resource misfit to the query
- Query type highlights the nature of the SQL statement, such as merge, delete, CTAS, etc.
- Session ID to identify the process such as ETL, BI, or ad-hoc query, etc.

Depending on the workload and its landscape, there are many ways to explore, analyze and optimize. Figure 2 illustrates the credit distribution % of the ETL workloads mapped against the query types. The spike in credit consumption is narrowed down to the type of computed queries. The long-running expensive query IDs can be further drilled down with the logs.

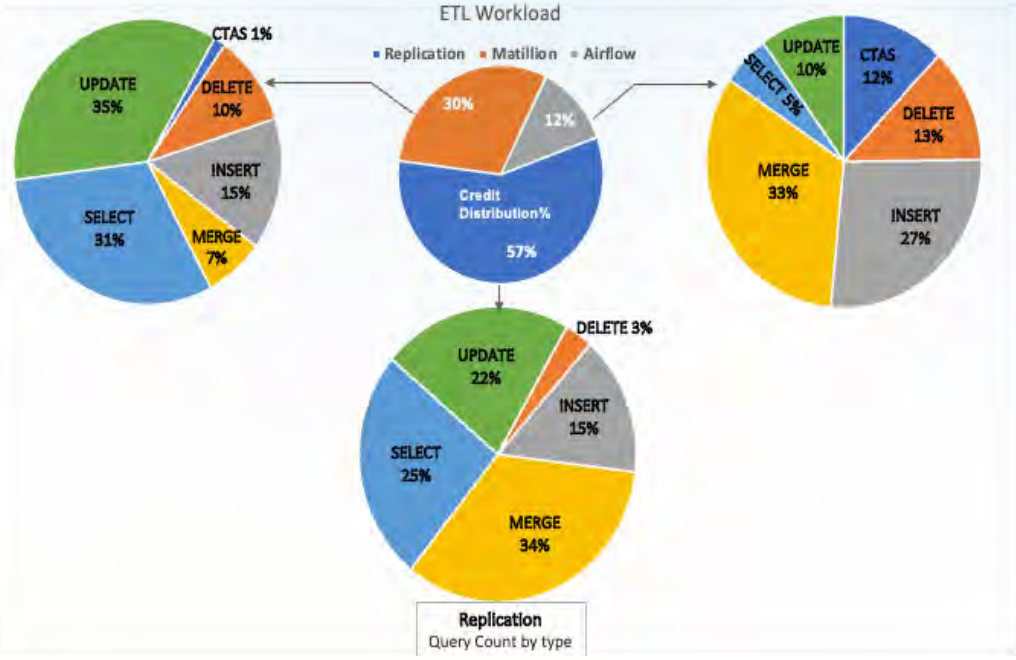

Figure 2: Illustration of credit distribution % of ETL workloads

Below is the illustration of storage metrics distribution. The implicit storage size is exponentially higher than the active storage. It is not a cost problem but rather :

- An indication of the improper implicit configuration
- Potential usage of a permanent table for storing intermediate results in ETL processing
- Frequent changes to micro-partition and **potential micro-partition churning**
- Truncate and load data patterns on large data sets

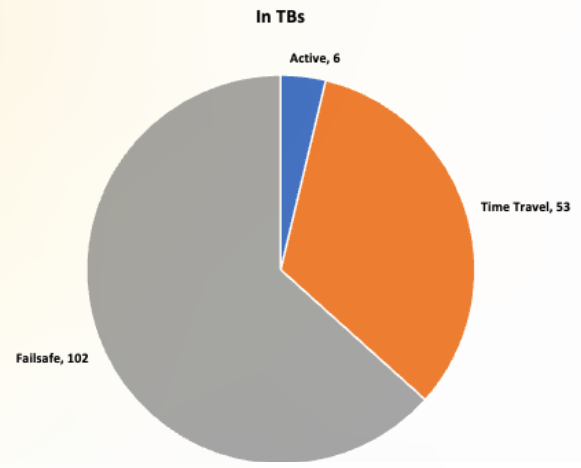

Figure 3: Illustration of storage metrics distribution in TBs

4.2. Bottleneck elimination

The main purpose of bottleneck elimination is to improve the effective use of a particular Snowflake resource and reduce the credit consumption for the same output. There are different forms of bottlenecks that can be eliminated by making changes. E.g.:

- **Change the Snowflake platform resources**
  - Overprovisioning of computing resource
  - Underutilization of compute
  - Revisit the compute configurations
  - Cluster the large tables
  - Optimal mix/max cluster configurations
- **Change in the way that applications interface with Snowflake**
  - Sub-optimal file size and compression during ETL runs
  - Non-cloud native connectors (standard ODBC)
  - Inefficient SQL statements by BI tools
- **SQL tuning: Significant amount of credits can be saved by identifying the bad SQLs.**
  - Data spillage: Sort operations in the window analytics function are performed in memory for better performance. However, large data sets result in data spilling to local storage (SSD) and a remote location on Snowflake-managed S3. It increases credit consumption exponentially as the compute resource is active throughout the process. Fetching the required data that makes to the final result set is recommended.
  - Predicate push down: Query pruning relies on the predicates to fetch the data from the storage layer. If a table is large and fragmented, then fetching the result set takes more time in scanning and filtering during the query execution. Besides, the data types play a key role in pruning. The date datatype stored as varchar can impact performance significantly. Sequencing the WHERE clause filters based on selectivity helps the pruning effect and improves the query elapsed time.
  - Using common table expressions: CTE brings modularity to the query and makes it more comprehensible. More repeated subqueries in a data model are the candidates for CTE, and can be referenced. It helps the compilation time and reduces the overall query elapsed time.
  - Avoid selecting all the columns (Select *): The data is stored in a columnar format in Snowflake, and the metadata of the columns is managed by cloud services layer. Hence, selecting the required columns renders data faster from the storage layer as the optimizer has the necessary metadata information for the columns. It also reduces the data volume transferred in memory and constructs the cache with relevant information.

5. Optimizing cost with process and people

Many design standards, tuning approaches, tools, and technologies are available to provide 360-degree cost visibility. But they can take you to a certain extent in optimization. More than Snowflake's capabilities, the non-compliance of standards and the process by the people lead to potential cost issues and others. The standards are only as good as how people follow them. Non-compliance to design standards leads to gaps in functionalities and eventually results in a deviation from actual Snowflake recommendations. For instance, the production and non-production are logically separate in Snowflake, but credit metering for cost remains the same. Hence, understanding the Snowflake pricing model and adherence to key design principles are fundamental for cost optimization and sustainability.

While onboarding a new use case, all the stakeholders need to design a set of reliable, reusable, and sustainable standards to meet the performance/cost SLA. Then these standards should be **programmatically enforced** by a reusable framework as applicable. Many automation tools, such as Terraform, are available to develop and deploy such frameworks. Sample cost guard rails that can be automated using the framework:

- Spinning up a new compute warehouse with a pre-defined configuration
- Enforcing compute size for prod/non-prod environments
- Compute configurations such as session timeout, auto suspend configuration, etc.
- Storage configurations such as time travel parameters
- Database/schema creation using naming convention framework
- Cloning data across environments (lower to higher)
- Metadata-driven orchestration and schedules

While such frameworks funnel and enforce everyone to standards, observability is a critical success factor for the key stakeholders to review the effectiveness of the solutions. The observability tools can unearth many red flags for cost that would require changes at the component or platform levels. Such red flags should be prioritized and addressed based on the cost savings and the effort required. It is a continuous improvement process to optimize the spending.

6. Sustainability opportunities in Snowflake

Sustainability of Snowflake is the product vendor's commitment to the environment. From the Snowflake customer perspective, **sustainability** is about understanding the impact of the resources used, effectively allocating cost, eliminating wastage, applying cost-conscious design and best practices for effectively utilizing the resources. There is not much difference between the principles of cost optimization and sustainability practice, but they complement each other. Both should be an integral part of the end-end process of onboarding a use case.

Sample eco-friendly considerations for building a sustainable solution, implicit from a Snowflake customer perspective, are as follows:

- **Reducing the active idle time:** The idle and active compute consumes credits unproductively. Necessary parameters should be configured to pause or suspend the resources to remove the idle time.
- **Need vs. ability:** As Snowflake customers can afford near-unlimited capability, it is essential to utilize the resources judiciously. Be conservative upfront in provisioning the resources and do course corrections over the period.
- **Over-sizing for speed:** It is a common perspective that performance issues can be solved by scaling up the compute. Throwing the infrastructure at an unknown problem is not a solution, but identifying the bottleneck and applying necessary fixes is.
- **Green coding:** More lines of code to compute, the higher the emission. Hence, it is entailed having environmental intentions in the coding while building data applications. Being a Snowflake customer, it is not about quantifying the emissions level rather, embracing the principles that counter them, such as lean coding, templating the reusable blocks of code, modularization, CI/CD etc. Green coding aims to produce the same result with less compute.
- **Cache:** Caching in Snowflake reduces the need for heavy lifting. It requires lesser resources required to produce the same output.
- **Data replication:** Rationalizing the replication inventory and replicating only the productive data can be considered a sustainable practice.

7. Balancing the cost and performance

Regarding cloud management, matching the workload to the right resource levels is key to **getting the right price-performance levels**. The mismatch results in cost concerns, performance bottlenecks, or both, which are never a Snowflake product problem, but a lack of understanding of the features, deviation from standards, operational muddle, and poor platform hygiene.

- Overprovisioning resources with anticipation of speed and better performance can be expensive
- Throwing the infrastructure at a performance problem will not only spike the credit but a contribution to the environmental footprint
- Exploiting the shared resources, such as one compute warehouse for ETL and reporting, to save cost

As the demand changes over time and the workloads can also be heterogenous, the design decisions should be flexible and scalable to match the dynamics and get the right price-performance levels. Thus, it is essential to adhere to cost optimization and sustainability best practices to extract the best out of Snowflake for the right performance to the right workloads at the right time. Getting the best out of Snowflake can also turn out into **cost minimization and performance maximization**, which are two conflicting objectives. Hence, the trade-off between cost and performance should be determined by the efficiency of the workloads meeting the business demands. Even the **smallest improvements magnify into a positive impact** on cost and can be accounted towards sustainability goals.

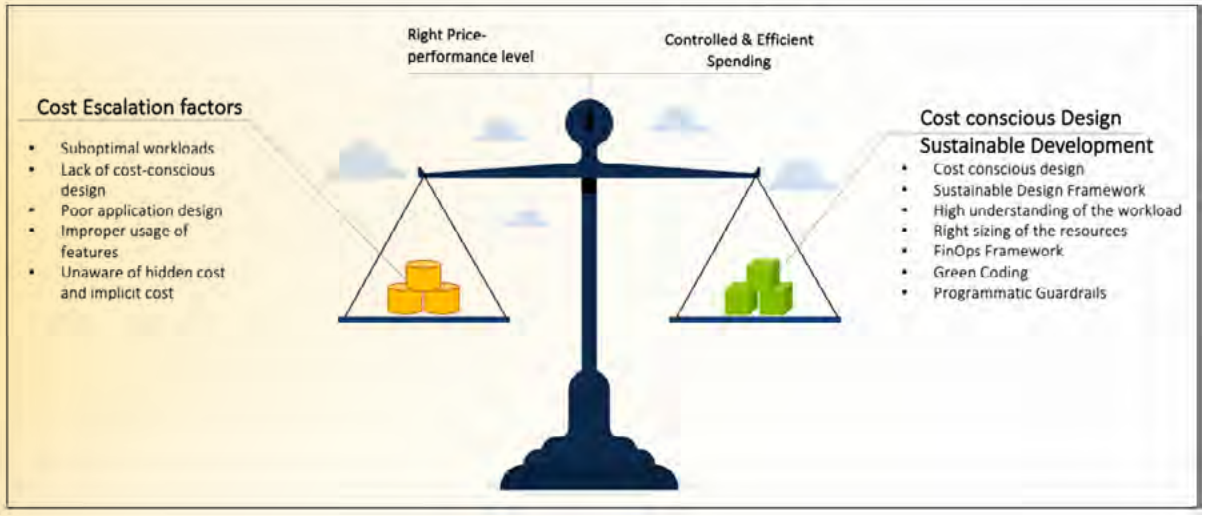

Figure 4: Illustration of balancing the cost escalations by design

8. Conclusion

Snowflake's cost breakdown offers insights into workloads' performance and resource utilization. Ineffective resource utilization due to design issues, architecture deviations, and non-compliance to standards can affect cost and sustainability goals. Therefore, cost optimization and sustainability should be part of the life cycle from analysis to maintenance. Both aspects should be integral to design thinking and decision-making.

Post-implementation tuning is also crucial, starting with setting up standards and guardrails at the platform level to tune proactively followed by identifying and fixing bottlenecks at the component level. The goal is to improve resource utilization in Snowflake to simultaneously meet cost and sustainability goals.

A well-defined FinOps framework is essential for streamlining cost and performance. It brings better visibility to cost, resource wastage, and unproductive utilization. Transforming dollars spent into complex business problem-solving, decision-making, unlocking new capabilities & use cases, meeting objectives, and cost-effective utilization of Snowflake resources is crucial for cost optimization and sustainability.

9. References

- https://docs.snowflake.com/en/guides-overview-cost
- https://www.snowflake.com/en/data-cloud/pricing/cost-and-performance-optimization/
- https://www.snowflake.com/blog/commitment-to-improving-economics-for-customers/
- https://docs.snowflake.com/en/guides-overview-performance
- https://www.snowflake.com/en/company/overview/sustainability-at-snowflake/

About the Author

Muthusivan Vellapandian

Associate Principal - Data Engineering LTIMindtree

Muthusivan V is working as a Solution Architect in the Snowflake Tech-Consulting group at LTIMindtree. He has around 16+ years of experience in data practice in solutioning and technology consulting for customers across the globe. He specializes in strategizing and migrating SAP HANA and Teradata workloads to Snowflake. He is passionate about Snowflake, solving complex migration problems in large transformational programs and business problems through data modernization. He likes to learn and practice modern data stacks (such as dbt, Matillion, etc.). Along with a passion for data and cloud, he likes astronomy and sports and is an ardent Manchester United and CSK supporter.

LTIMindtree is a global technology consulting and digital solutions company that enables enterprises across industries to reimagine business models, accelerate innovation, and maximize growth by harnessing digital technologies. As a digital transformation partner to more than 700 clients, LTIMindtree brings extensive domain and technology expertise to help drive superior competitive differentiation, customer experiences, and business outcomes in a converging world. Powered by 82,000+ talented and entrepreneurial professionals across more than 30 countries, LTIMindtree — a Larsen & Toubro Group company — combines the industry-acclaimed strengths of erstwhile Larsen and Toubro Infotech and Mindtree in solving the most complex business challenges and delivering transformation at scale. For more information, please visit **www.ltimindtree.com**.