

Cost-Optimized Data Lake Design Patterns for Enterprise-Scale Analytics

Gaurav Kumar Sinha

Abstract: Data lakes promised enterprises a single, inexpensive place to keep all of their data in its original form and to analyze it with whatever tools suit the task. In practice, many lakes have become expensive. Storage grows without bound, small files overwhelm metadata and query engines, clusters sit idle between jobs, the same data is copied into warehouses and marts, queries scan far more bytes than they need, and nobody can say which team is responsible for which part of the bill. This paper presents a catalog of eight design patterns for building cost-optimized data lakes at enterprise scale. The patterns cover storage tiering and lifecycle management, columnar formats with partitioning and compaction, elastic and interruptible compute, query-in-place with scan reduction, zoned layers with retention, ingestion mode selection, selective materialization of marts, and governance with cost accountability. We position the patterns against the classical warehouse and lake architectures and against a cloud reference architecture, and we evaluate them in a simulation of an enterprise with 1.2 petabytes of data, 8 terabytes of daily ingestion and 120,000 queries per day. Together, the patterns reduced the monthly cost index of a naive lake from 100 to 41, cut median bytes scanned per query by 91%, improved median query time from 18 seconds to 4.2 seconds, reduced data duplication from 3.1 copies to 1.4, and raised the share of cost that could be attributed to an owner from 66% to 94%. The study also quantifies the trade-offs of each pattern, including retrieval charges, interruption overhead and data freshness, and offers a decision guide for practitioners.

Introduction

Organizations have more data than ever and more ways to use it. Transaction systems, application logs, sensors, partners and public sources continuously produce structured tables, semi-structured events, documents and images, and the people who wish to analyze them range from business analysts building dashboards to data scientists training models. The data warehouse, the traditional home of analytical data, was designed for a world of carefully modeled, structured data and predictable reporting. It requires data to be cleaned and shaped before it is loaded, and its storage and compute have historically been expensive. The data lake emerged as a complement and, for some organizations, an alternative. It stores data cheaply in its raw form, defers modeling until the data are read and supports many kinds of analysis on the same stored data [3], [24].

The economic argument for the lake was strong. Cloud object storage costs a fraction of warehouse storage, scales without planning and keeps data of any type. Yet experience has shown that low unit prices do not guarantee low total cost. A lake that ingests everything and deletes nothing grows indefinitely. A lake of millions of tiny files makes every query slow and every metadata operation costly.

7319 Longspur Hollow Lane Katy, TX 77493-4704 United States

Keywords: Data Lake, Cost Optimization, Cloud Analytics, Storage Tiering, Columnar Formats, Lakehouse, Data Warehouse, FinOps

A lake in which each team makes its own copy of the data multiplies storage and processing. A lake served by clusters that run around the clock pays for capacity that is used only a fraction of the time. A lake in which queries read whole files, because the layout does not let engines skip irrelevant data, pays for every byte scanned. And a lake without clear ownership cannot be managed, because no one knows who benefits from the spending or who can approve its reduction.

These problems are not inevitable. They follow from design choices, and better design choices exist. The practices of cloud financial management, which bring engineering, finance and business together to manage cloud spending, have shown that cost is a property of architecture and operations and not only of price lists [20]. For data lakes, there is a body of practical knowledge about how to arrange storage, formats, compute and governance so that cost follows value. That knowledge is scattered across vendor documentation, engineering blogs and experience, and it is seldom presented as a coherent set of patterns with their trade-offs and with quantitative evidence.

This paper provides such a presentation. Its contributions are as follows: (i) a catalog of eight cost-optimization design patterns for enterprise data lakes, described by their mechanism, cost lever and trade-offs; (ii) a positioning of these patterns against the classical warehouse and lake architectures and a cloud reference architecture; (iii) a simulation study that quantifies the effect of each pattern and of their combination on cost, performance, duplication and accountability; (iv) an analysis of the trade-offs, including retrieval charges, interruptions and freshness; and (v) a decision guide. The remainder of the paper is organized as follows. Section II reviews background, Section III discusses related work, Section IV presents the methodology and the pattern catalog, Section V describes the experimental setup, Section VI reports results, Section VII discusses implications and limitations, and Section VIII concludes.

Literature Review

A. Data Warehouses and Data Lakes

The data warehouse is the classical architecture for analytics. It collects data from operational systems and external sources, integrates it into an enterprise model and then serves it to business intelligence tools through subject-oriented data marts [1], [2]. The left side of Fig. 1 shows this arrangement: operational systems and external sources feed an enterprise data warehouse, from which data marts are derived, and business intelligence tools and applications use the marts. Its strengths are consistency, quality and performance for known questions. Its weaknesses are the up-front effort of modeling, the cost of storage and compute for large and varied data, and rigidity when needs change.

The data lake takes a different approach, shown on the right side of Fig. 1. External and internal data sources feed a data lake directly, where the data are kept in many forms, such as tables, documents and other structures, and analytical tools and applications read them as needed. The idea is to store data first and decide on its structure later, which gives flexibility and low cost of storage [3]. Without discipline, however, the lake can degrade into a swamp, in which data are undocumented, duplicated and unusable. Research on data lake architectures and metadata management emphasizes the role of catalogs, metadata and governance in keeping lakes useful [7], [8], and systems such as Constance explored intelligent metadata extraction and management for lakes [6].

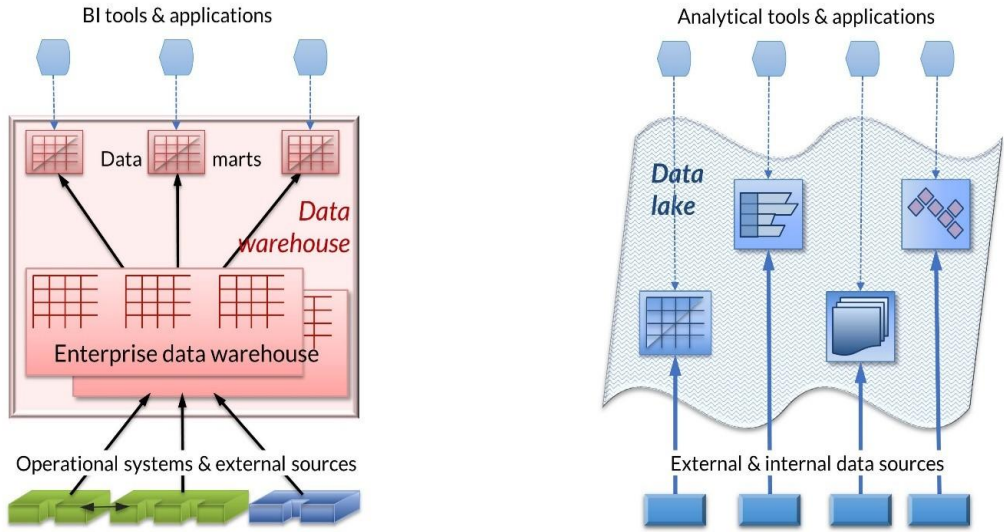


Fig. 1. Data warehouse architecture (left) with an enterprise data warehouse and data marts for BI tools, and data lake architecture (right) with direct access for analytical tools and applications.

B. Warehouse Data Flows

Fig. 2 illustrates how data move through a warehouse system in batch mode. Source systems, here a central bank providing exchange rates, two stores providing sales and a supplier providing a product list, deliver data to a staging area built on polyglot storage. In staging, the extracts are first overwritten with each delivery, and then joined, combined and shaped into sales and product tables, a stage that corresponds to the transformation step of extract, transform and load processing [19]. The staged tables are appended to the core data warehouse, which keeps history in relational storage. From the core, data marts are derived by appending and joining, one in a multidimensional store for sales analysis and others in relational storage holding aggregates of items, and these marts serve online analytical processing and reporting. The figure marks the data flows as batch flows and distinguishes data tables, multidimensional models and underlying storage systems.

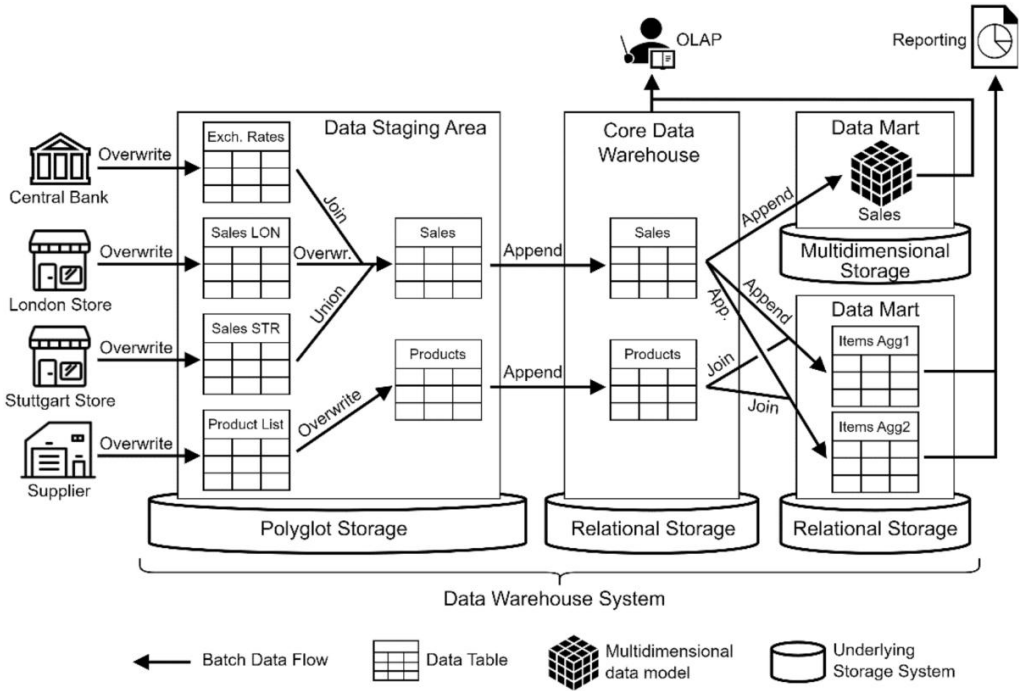


Fig. 2. Data warehouse system with batch data flows: staging area, core data warehouse and data marts, fed by source systems and serving OLAP and reporting.

The figure also illustrates the cost structure of layered architectures. Each layer holds its own copy of the data, in its own storage system, with its own processing. The staging area overwrites, the core appends and the marts derive further copies, so that the same business facts exist in several places. Layering provides clarity and performance, but unmanaged it multiplies storage and compute, and the same concern applies when a lake and a warehouse coexist and data are copied between them. A central question of this paper is how to retain the benefits of layering while limiting its cost.

C. Lakehouse and Open Table Formats

The lakehouse architecture attempts to unify the two worlds by adding warehouse-like management, such as transactions, schema enforcement and efficient queries, on top of open formats stored in cheap object storage [4]. Table formats that provide transactional metadata layers over files in object stores make it possible to update tables atomically, to evolve schemas and to time travel over previous versions [5]. Such formats reduce the need to copy data into a separate warehouse, which is a major source of duplication, and they provide the metadata on which optimizations such as compaction and data skipping depend. Query engines for lakehouses have been engineered to run fast on open formats [25].

D. Storage Formats, Engines and Elasticity

Columnar storage organizes data by column rather than by row, so that queries read only the columns they need, and similar values stored together compress well [10], [22], [23]. Nested columnar formats and the engines that exploit them made

interactive analysis over very large datasets practical [9]. The separation of storage from compute, in which data reside in shared object storage and compute resources are provisioned independently, underlies modern elastic warehouses and query engines [11], [12]. Managed warehouses were designed to simplify operation and reduce cost [13], and earlier frameworks for large-scale processing, from MapReduce to Spark, established the model of running many parallel tasks over distributed data [14]-[16]. The foundations of cloud computing, in which resources are rented on demand and elasticity converts capital expense into operating expense, explain why compute can be treated as ephemeral [18]. Distributed file systems laid the groundwork for storing large data reliably on commodity hardware [17]. Textbook treatments of data-intensive systems describe these building blocks in depth [21].

E. Governance and Financial Accountability

Governance is the set of practices by which an organization knows what data it holds, who owns it, who may use it and for how long it should be kept. A data catalog records datasets, schemas and lineage. Access control determines who may read and write. Retention rules determine when data must be deleted, which is both a cost measure and, for personal data, a legal requirement under regulations such as the European General Data Protection Regulation [26]. Financial accountability extends governance to cost. It requires that spending be attributed to teams and products through tagging and allocation, that budgets and alerts exist and that owners have the information and the incentive to optimize [20]. Without it, the benefits of technical optimization cannot be sustained.

TABLE I. Comparison of Warehouse, Lake and Lakehouse Architectures

Aspect	Data Warehouse	Data Lake	Lakehouse
Data types	Structured, modeled	Any, raw and processed	Any, with table abstractions
Schema	On write	On read	Enforced and evolvable
Storage cost	Higher	Low, object storage	Low, object storage
Typical weakness	Rigid and costly at scale	Swamp, small files, weak management	Maturity of tooling and discipline needed
Main cost risk	Capacity and copies	Unbounded growth and inefficient scans	Maintenance operations and metadata

Related work

Related work can be grouped into lake architecture and management, storage and query engine design, and financial management of cloud analytics. Work on lake architecture proposes reference architectures with zones for raw, cleansed and curated data, and emphasizes metadata management, data discovery and governance as essential to keep lakes from becoming swamps [6]-[8]. These studies concentrate on usability and manageability, and cost is treated implicitly, as a consequence of cheap storage, rather than as an explicit design objective.

Work on storage and query engines supplies the mechanisms on which cost optimization depends. Columnar formats, compression, partitioning and data

skipping reduce the amount of data read [9], [22], [23]. Transactional table layers enable compaction and efficient updates over object storage [5]. Disaggregated storage and compute make it possible to scale each independently and to use compute only when needed [11], [12]. Performance studies of such systems typically report speed and throughput, and less often the monetary cost of achieving them, or they report cost for a single technique in isolation.

Work on the financial management of cloud analytics is largely practitioner oriented. It describes organizational practices of allocation, budgeting, commitment management and optimization, and it stresses collaboration between engineering and finance [20]. Cost optimization is presented in generic terms, such as rightsizing and scheduling, and is less often tied to the specific structures of data lakes, such as zones, table formats and ingestion modes.

Cloud providers publish reference architectures for analytics on their platforms, which combine managed services for ingestion, storage, cataloging, processing and consumption. Such architectures illustrate how the components of a lake fit together, but they are vendor-specific and do not by themselves explain how to arrange the components to minimize cost. We use one such reference architecture, described in Section IV, as a concrete anchor for the patterns, while keeping the patterns themselves independent of any provider.

Three open issues remain. First, there is no widely used catalog of cost-oriented design patterns for data lakes with quantitative evidence of their effect. Second, the interactions among techniques, in which, for example, file layout affects both query cost and compute cost, are rarely analyzed. Third, the organizational side of cost, including accountability and guardrails, is seldom studied alongside the technical side. This paper addresses all three in a simulation setting.

Methodology

A. A Cloud Reference Architecture

Fig. 3 shows a reference architecture for analytics on a public cloud. At the left are data sources, such as application servers, containers and on-premises systems, and the object store itself can also act as a primary data store. Data move through four kinds of workloads. Streaming workloads include data streamers, implemented by managed streaming services and managed Kafka, and data aggregators, implemented by stream processing and serverless functions. Batch workloads include data collectors, which land data in object storage by means of managed delivery, storage gateways and transfer services, and data transformers, which process data in batches on managed clusters, ETL services and batch compute services. Both kinds of workloads feed a governance layer, which contains a data catalog and the data lake itself, built on object storage with a service that manages access permissions. Three groups of consumers read from the lake: data scientists, who use managed cluster frameworks and a machine learning platform; data analysts, who use a serverless query service and a cloud data warehouse; and visualization users, who use a business intelligence service and third-party tools.

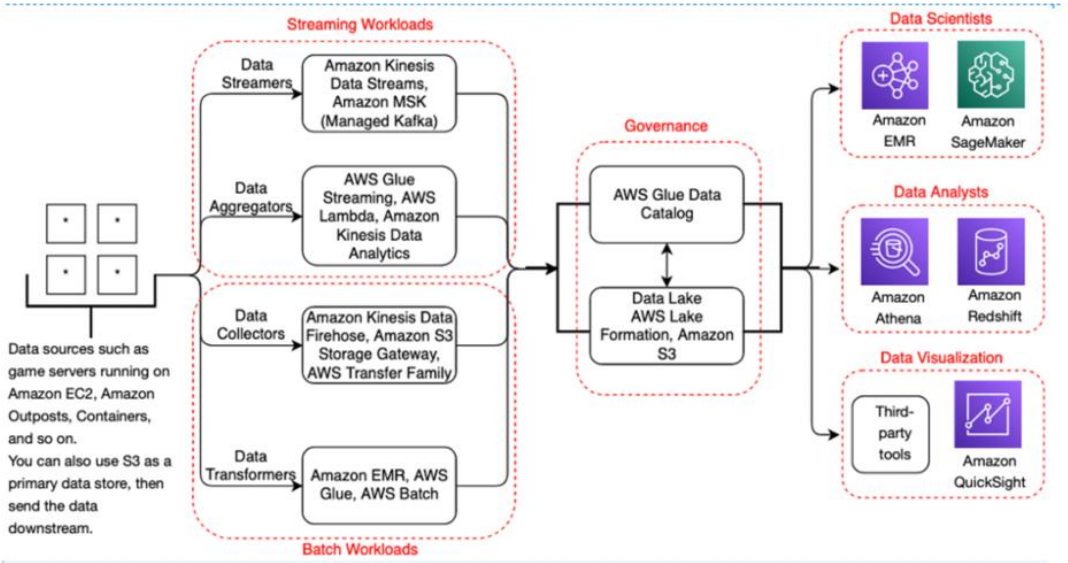


Fig. 3. Reference architecture for cloud analytics: streaming and batch workloads feed a governed data lake, which serves data scientists, data analysts and visualization users.

The architecture exposes the places in which cost arises. Ingestion has a cost that depends on whether data are streamed record by record or delivered in batches. The object store has a storage cost that depends on volume, tier and number of objects. Transformation and analysis have compute costs that depend on cluster size and run time. The query service charges according to the bytes scanned. The warehouse and the visualization tools may hold additional copies. The catalog and the access service provide the governance on which attribution and retention depend. Each pattern in the catalog targets one or more of these places.

B. The Pattern Catalog

Table II lists the eight patterns. P1 places data in storage tiers according to access patterns and moves or deletes it automatically as it ages. P2 stores data in columnar formats, partitions it according to common filters and compacts small files into larger ones. P3 separates compute from storage, scales compute elastically and runs fault-tolerant work on interruptible capacity. P4 queries data in place with serverless or elastic engines and reduces the bytes scanned through pruning and column selection. P5 organizes the lake into zones, such as raw, cleansed and curated, and sets retention for each, so that intermediate copies do not live forever. P6 chooses the ingestion mode, streaming, micro-batch or batch, according to the freshness that consumers actually need. P7 materializes data marts or warehouse copies only for workloads that justify them, relying on open table formats to serve the rest directly from the lake. P8 provides governance with cost accountability, including a catalog, ownership, tagging, quotas, alerts and guardrails.

TABLE II. Catalog of Cost-Optimization Patterns for Enterprise Data Lakes

Pattern	Mechanism	Main Cost Lever	Main Trade-Off
P1 Storage tiering and lifecycle	Move aged data to cheaper tiers, expire what is not needed	Storage	Retrieval charges and slower access
P2 Columnar formats, partitioning and compaction	Compress, skip data and merge small files	Storage, scan, compute, requests	Compaction work and write latency
P3 Elastic and interruptible compute	Autoscale, separate from storage, use spare capacity	Compute	Interruptions and rerun overhead
P4 Query in place with scan reduction	Prune partitions and columns, cache results	Query scan	Needs good layout and statistics
P5 Zoned layers with retention	Raw, cleansed and curated zones with expiry	Storage, duplication	Reprocessing if raw data expire too early
P6 Ingestion mode selection	Match streaming, micro-batch or batch to need	Ingestion and compute	Data freshness
P7 Selective materialization	Build marts only for hot, stable workloads	Storage, compute, duplication	Possible staleness and dual maintenance
P8 Governance and cost accountability	Catalog, owners, tags, quotas, alerts	All, through behavior	Tooling effort and process friction

C. Storage Tiering and Lifecycle Management (P1)

Data have a temperature. Recently written data are read often, and the share of reads falls sharply with age. Object storage offers tiers with different trade-offs, from a hot tier with low access cost and high storage price to infrequent-access, archive and deep archive tiers whose storage price is much lower but whose retrieval incurs fees, minimum storage durations and, for the deepest tiers, delays of hours. A lifecycle policy moves objects between tiers by age or by observed access and deletes objects when retention ends. The pattern needs an understanding of access patterns, and many platforms can also move objects automatically according to observed access. Policies should consider the minimum storage duration of a tier and the size of objects, since moving many small objects to a cold tier can cost more than it saves.

D. Formats, Partitioning and Compaction (P2) and Query in Place (P4)

Raw text formats such as comma-separated values and JSON are easy to produce but inefficient to query, because engines must read and parse entire records. Columnar formats store each column together, compress it effectively and keep statistics about ranges of values, so that engines read only the necessary columns and skip blocks that cannot match a filter [9], [23]. Partitioning arranges files in

directories by the values of commonly filtered columns, such as date, so that a query for one week touches only that week's files. Sorting or clustering within partitions improves skipping on further columns. Streaming and frequent small batches produce very many small files, which slow queries because each file has overhead and increase request charges. Compaction merges small files into larger ones, ideally of a few hundred megabytes, and open table formats make it possible to do this safely while readers continue to work [5]. These measures are the foundation of query in place. When the layout is good, a serverless query service that charges by bytes scanned will read little, and an elastic engine will finish quickly and release its resources.

E. Elastic Compute (P3), Zones (P5), Ingestion (P6) and Materialization (P7)

Because storage and compute are separated, clusters need not be permanent. Batch transformation and training jobs can run on clusters that are started for the job, sized to it and stopped afterwards. Jobs that tolerate restarts, such as most batch transformations built on frameworks that recompute lost tasks [14], [15], can run largely on interruptible spare capacity, which costs much less than on-demand capacity but can be reclaimed by the provider. Interactive workloads use autoscaling with floors and ceilings. The zoned layout of the lake echoes the layered flow of Fig. 2: a raw zone holds data as received, a cleansed zone holds validated and standardized data, and a curated zone holds data shaped for consumption. Each zone has a retention rule. Raw data are often kept for a period long enough to reprocess, intermediate data for a short time, and curated data as long as they are used. The ingestion mode determines freshness and cost. Record-by-record streaming delivers data in seconds but is the most expensive per terabyte, micro-batching every few minutes costs considerably less, and hourly or daily batches cost least. Materialization copies data into marts or warehouse tables. With open table formats, many workloads can be served directly from the lake, and marts are built only for workloads whose volume, concurrency and stability justify the copy.

F. Governance and Cost Accountability (P8)

The catalog of Fig. 3 is the foundation of governance. Every dataset is registered with an owner, a purpose, a retention class, a sensitivity level and a cost center, and these attributes are propagated to storage and compute resources as tags. Cost is allocated by tag so that every team sees its share, and the share that cannot be attributed is tracked as a metric to be driven down. Budgets and alerts warn owners of unusual growth, quotas limit the data that ad hoc queries may scan before approval is required, and reports list datasets that have not been accessed for long periods so that they can be archived or deleted. Retention classes link cost control to legal requirements, since data that must be deleted should not remain in cheap archives. This pattern changes behavior and thereby multiplies the effect of the technical patterns [20].

G. Cost Model and Metrics

We use a cost model with six categories: storage, compute, query scan charges, data transfer, request and metadata operations, and operations and governance, which includes tooling and the effort of administration. All costs are expressed as an index, in which the monthly cost of the naive baseline is 100, so that the findings do not depend on the price list of a single provider. We also report the median bytes

scanned per query, query latency percentiles, the number of object requests per day, the number of copies of data per dataset, the share of cost attributable to an owner, and the share of stored bytes that are unused and unowned.

Experimental Setup

A. Simulated Enterprise

The evaluation used a simulation of a mid-sized enterprise. The lake held 1.2 petabytes at the start and ingested 8 terabytes per day over a horizon of fourteen months, with the stored volume growing accordingly. Of the ingested volume, about 60% came from application logs and event streams, about 25% from extracts of operational databases and about 15% from files delivered by partners. The lake contained about 3,000 tables and served 450 analysts and data scientists who issued about 120,000 queries per day. Workloads were divided into dashboards, which accounted for about 60% of queries and were repetitive and filtered by date, ad hoc analyses, about 30%, and feature extraction for machine learning, about 10%, which scanned large ranges. About 8% of stored bytes had been accessed in the last 30 days, 12% between 31 and 90 days, 24% between 91 and 365 days and 56% had not been accessed for more than a year, and recent data accounted for about 71% of reads.

B. Baseline

The baseline is a naive lake of the kind that often results from unplanned growth. All data were stored in the hot tier as comma-separated and JSON files, many of them small, without partitioning. Clusters ran around the clock, sized for the peak. Data were copied into a warehouse and several marts for every consuming team, and ingestion was streamed record by record for all sources. No retention rules existed, and no cost allocation was applied beyond the total bill. The optimized lake applied all eight patterns. For the analysis of individual patterns, each was applied alone to the baseline, and for the analysis of combinations, patterns were applied cumulatively.

C. Parameters of Each Pattern

Storage tiers were modeled with relative storage prices of 100 for the hot tier, 50 for infrequent access, 15 for archive and 5 for deep archive, with retrieval fees and minimum durations on the cold tiers. Tiering policies moved data to infrequent access after 90 days, to archive after one year and to deep archive after two years, unless the retention class required otherwise. Files in the optimized layout were stored in a columnar format with compression, partitioned by date and sorted by a common key, with compaction to target files of several hundred megabytes. Batch compute used autoscaling with up to seventy percent interruptible capacity, with an assumed rerun overhead for interrupted tasks. Dashboards and ad hoc queries ran on a serverless query engine that charges by bytes scanned, with result caching for repeated queries. Raw data were retained for 365 days, cleansed data for 180 days and curated data as long as they were accessed. Ingestion used micro-batches of five minutes by default, with streaming reserved for datasets whose owners documented a freshness requirement under a minute, which was about 12% of datasets. Marts were built only for workloads with sustained high concurrency.

D. Repetitions and Interpretation

Each configuration was simulated with ten random seeds, which varied query

arrival, file sizes and access patterns, and results are means. Differences smaller than the variation between seeds, about three percent, are not interpreted. The simulation is a model of costs and performance based on typical behavior, and the costs are relative indices. The results show the direction and approximate size of the effects and should not be read as predictions for a specific platform or price list.

TABLE III. Experimental Configuration

Component	Setting
Initial volume and growth	1.2 PB, 8 TB per day, 14 months
Tables and users	About 3,000 tables, 450 users, 120,000 queries per day
Query mix	60% dashboards, 30% ad hoc, 10% machine learning feature extraction
Ingestion mix	60% logs and events, 25% database extracts, 15% partner files
Relative storage prices	Hot 100, infrequent access 50, archive 15, deep archive 5
Tiering thresholds	90 days, 1 year and 2 years
Retention	Raw 365 days, cleansed 180 days, curated while accessed
Ingestion default	Five-minute micro-batches, streaming for about 12% of datasets
Repetitions	Ten seeds, means reported

Results

A. Overall Cost

Table IV compares the cost of the baseline and the optimized lake by category. In the baseline, compute was the largest component, at 41, followed by storage at 38. Query scan charges were 9, data transfer 4, request and metadata operations 3, and operations and governance 5. The optimized lake reduced total cost to 41, a reduction of 59%. Storage fell to 12 and compute to 17, which together account for most of the saving. Query scan charges fell to 1, request and metadata costs to 1 and data transfer to 3, while operations and governance rose to 7, because the additional tooling and administration of the optimized design has a cost of its own. That cost is small relative to the savings, but it is not zero, and it should be planned for.

TABLE IV. Monthly Cost Index by Category (Baseline Total = 100)

Cost Category	Naive Baseline	Optimized Lake (All Eight Patterns)
Storage	38	12
Compute	41	17
Query scan charges	9	1
Data transfer	4	3
Request and metadata operations	3	1
Operations and governance	5	7
Total	100	41

B. Contribution of Individual Patterns

Table V shows the saving obtained when each pattern is applied alone to the baseline, together with its main side effect. The largest savings came from elastic compute (P3), at 24 index points, and from columnar formats with partitioning and compaction (P2), at 22. Storage tiering (P1) saved 16, zoned layers with retention (P5) saved 10, selective materialization (P7) saved 8, query in place (P4) saved 7, ingestion mode selection (P6) saved 6 and governance (P8) saved 5 directly. The savings overlap and do not add up. Their sum exceeds the combined saving of 59, since, for example, compacting and compressing files reduces the cost of both storage and compute, which are also reduced by tiering and elastic clusters. The side effects are as important as the savings. Tiering introduced retrieval charges and delays, compute savings brought interruptions that required about three percent of work to be rerun, and ingestion in micro-batches reduced the freshness of data from seconds to minutes. Governance has an indirect effect, because it enables the others to be sustained.

TABLE V. Saving of Each Pattern Applied Alone and Main Side Effect

Pattern	Saving (Index Points)	Main Side Effect Observed
P1 Storage tiering and lifecycle	16	Retrieval charges, slow access to deepest tiers
P2 Columnar formats, partitioning, compaction	22	Compaction compute, small delay in writes
P3 Elastic and interruptible compute	24	About 3% of work rerun after interruptions
P4 Query in place with scan reduction	7	Gains depend on layout from P2
P5 Zoned layers with retention	10	Reprocessing needed if raw data expire early
P6 Ingestion mode selection	6	Data freshness of minutes instead of seconds
P7 Selective materialization	8	Possible staleness of marts
P8 Governance and cost accountability	5	Tooling effort and process friction

C. Storage Tiering

Table VI shows the distribution of bytes across tiers before and after the lifecycle policy. In the baseline, all data resided in the hot tier. After the policy, 30% of bytes remained hot, which included recent data and datasets pinned hot because of frequent access, 25% were in infrequent access, 25% in archive and 20% in deep archive. Weighting by relative storage price, this distribution cost about 47% of the all-hot price, and after adding retrieval fees and request charges on cold tiers, the storage component of the cost index fell from 38 to 21 with tiering alone. Reads were served overwhelmingly from the hot and infrequent tiers, so that about 97% of queries were unaffected, while the remaining 3% that touched archived data had to

wait for retrieval or to pay a fee. This confirms that the pattern is effective when access follows age, and it also shows that retrieval from the deepest tiers should be planned and not treated as routine.

TABLE VI. Distribution of Stored Bytes Across Storage Tiers

Tier	Relative Storage Price	Share of Bytes, Baseline (%)	Share of Bytes, After Policy (%)	Access Characteristics
Hot	100	100	30	Immediate, no retrieval fee
Infrequent access	50	0	25	Immediate, retrieval fee
Archive	15	0	25	Minutes to hours, retrieval fee
Deep archive	5	0	20	Hours, retrieval fee, long minimum duration

D. Formats, Layout and Query Performance

Table VII shows how layout affected queries. With small unpartitioned text files, the median query scanned 420 gigabytes, had a median run time of 18 seconds and a 95th percentile of 140 seconds, and the lake received about 1.9 million object requests per day. Converting to a partitioned columnar format reduced the scan to 61 gigabytes and the median run time to 6.5 seconds, and requests to 1.1 million. Compaction and sorting reduced the scan further to 38 gigabytes, the median run time to 4.2 seconds, the 95th percentile to 26 seconds and the requests to about 0.12 million per day. Overall, the median scan fell by 91%, the median run time by 77% and the request count by 94%. The result is that cost and performance improve together, since the same measures that reduce what is read also reduce the time taken to read it. This alignment of cost and speed is the main reason that layout is the most important lever for query-based workloads.

TABLE VII. Effect of File Format, Partitioning and Compaction on Queries

Layout	Median Bytes Scanned (GB)	Median Run Time (s)	p95 Run Time (s)	Object Requests per Day (millions)
Small text files, unpartitioned	420	18.0	140	1.9
Partitioned columnar files	61	6.5	52	1.1
Partitioned columnar, compacted and sorted	38	4.2	26	0.12

E. Compute and Ingestion

Table VIII compares compute strategies, with the always-on cluster sized for the peak set to an index of 100. Autoscaled on-demand clusters, started per job and stopped afterwards, reduced the compute index to 62. Adding interruptible capacity for fault-tolerant batch work, at about seventy percent of the batch capacity, reduced it to 41, after allowing for about 3% of rerun work. Serverless query for ad hoc workloads gave an index of 46 for that workload, with the best economics for sporadic use and less advantage for sustained heavy use. Table IX compares

ingestion modes. Record-by-record streaming, set to 100 per terabyte, delivered data in seconds. Micro-batches every five minutes cost 38 and delivered in minutes, hourly batches cost 21 and daily bulk loads cost 14. Because only about 12% of datasets needed freshness under a minute, defaulting to micro-batches and reserving streaming for the justified cases cut ingestion costs substantially with little loss of value.

TABLE VIII. Compute Cost Index by Strategy (Always-On Cluster = 100)

Compute Strategy	Cost Index	Notes
Always-on cluster sized for peak	100	Baseline, idle capacity most of the day
Autoscaled on-demand clusters per job	62	Separated storage and compute
Autoscaled with interruptible capacity for batch	41	About 3% rerun overhead
Serverless query for ad hoc workloads	46	Best for sporadic use, charges by bytes scanned

TABLE IX. Ingestion Modes: Freshness and Relative Cost per Terabyte

Ingestion Mode	Data Freshness	Relative Cost per TB
Record-by-record streaming	Seconds	100
Micro-batch every five minutes	Minutes	38
Hourly batch	Up to one hour	21
Daily bulk load	Up to one day	14

F. Duplication, Zones and Governance

Layered designs, such as the staging, core and mart structure of Fig. 2, multiply copies of data. In the baseline, each dataset existed on average in 3.1 copies, counting raw files, staged versions, warehouse tables and marts built by individual teams. Zoned layers with retention, together with selective materialization and open table formats, reduced this to 1.4 copies, because intermediate zones expired and marts were created only where justified. Governance changed the visibility of cost. In the baseline, only 66% of the cost could be attributed to an owner, so that 34% was unallocated, and 11% of stored bytes had not been accessed for two years and had no owner. With catalog-based ownership and tagging, the unallocated share fell to 6% and the stale and unowned bytes to 2%. Guardrails that required approval for queries scanning more than one terabyte reduced the share of such queries from 4.2% to 0.3%, and they alone prevented a series of expensive accidental scans. Table X summarizes these measures.

TABLE X. Duplication and Governance Measures

Measure	Naive Baseline	Optimized Lake
Copies of data per dataset	3.1	1.4
Share of cost attributable to an owner (%)	66	94
Stale and unowned stored bytes (%)	11	2
Queries scanning more than 1 TB (%)	4.2	0.3
Median bytes scanned per query (GB)	420	38
Median query run time (s)	18.0	4.2

G. Cumulative Application

When the patterns were applied cumulatively in a recommended order, the cost fell in stages. Introducing governance and cost allocation first made the other savings visible and had a small direct effect. Columnar formats and compaction came next, and by themselves reduced the total index to about 78. Adding elastic and interruptible compute lowered it to about 58, tiering and retention to about 47 and selective materialization together with ingestion mode selection brought it to the final 41. This order has a rationale. Layout first is helpful since it improves both storage and compute, and compute elasticity is most valuable once jobs are shorter. Tiering and retention follow because they depend on the understanding of access patterns that governance provides, and materialization choices are best made when the lake can serve most workloads directly.

Discussion

The results suggest that the cost of a data lake is driven less by the price of storage than by decisions about layout, compute, copies and accountability. Storage remained a large component of the baseline bill, but compute, scanning and duplication together exceeded it, and these are controlled by design. The combination of patterns cut the cost by more than half while improving query performance, which indicates that, for analytics, cost and performance are often aligned. A well-organized lake reads less data, finishes sooner and releases its resources earlier.

The architectures of Fig. 1 and Fig. 2 help explain why. The classical warehouse achieved performance through careful modeling and layered copies, at a high cost of storage and effort. The lake reduced storage cost and removed the modeling bottleneck but, left unmanaged, lost the performance and organization that the warehouse enforced. The patterns in this paper recover the discipline of the warehouse within the lake. Zones and retention echo the layers of the warehouse flow while expiring the copies that are no longer needed. Columnar layouts and compaction reinstate the physical organization that makes queries efficient. Selective materialization keeps marts where they are justified. Open table formats provide the transactional management that the warehouse offered, in cheap object storage [4], [5]. In this sense, the cost-optimized lake is a convergence of the two traditions.

The reference architecture of Fig. 3 shows how many parts must be coordinated and that cost arises at every junction. Managed services reduce operational effort but also create charges that depend on usage patterns that engineers may not see. For that reason, the governance pattern is not an administrative afterthought. The share of cost that is attributable to an owner, the number of stale datasets and the number of expensive queries are measures of whether the organization is able to

manage the lake. Technical patterns without such accountability tend to erode as new datasets and users arrive.

Several limitations should be noted. The study is based on a simulation with assumed parameters, and real platforms differ in pricing, tier characteristics, minimum durations, request charges and the behavior of engines. The absolute numbers, therefore, should not be transferred directly, and the relative ordering of the patterns may change with pricing. The workload model is simplified, and real access patterns can be less predictable, which would reduce the savings from tiering and caching. Engineering effort for migration, such as converting formats, restructuring pipelines and retraining users, was not included in the cost, and it can be substantial and may take months to recover. Performance and cost during compaction and other maintenance operations were modeled in a simple way. The effects of data quality, security and privacy requirements on design were treated only through retention classes. Finally, the study considers a single enterprise, and organizations with very different sizes or regulatory environments may find different patterns more valuable.

Practical guidance follows. Begin with governance, so that cost can be seen and attributed. Convert frequently queried data to partitioned, compacted columnar tables, which gives the largest and most immediate effect on query-heavy workloads. Make compute elastic and place fault-tolerant batch work on interruptible capacity. Introduce lifecycle policies after measuring access patterns, and be careful with small objects and retrieval fees. Choose the ingestion mode according to documented freshness needs, with micro-batches as a default. Keep the number of copies low, create marts only where justified and set retention for each zone. Monitor continuously, since the patterns lose effect if layout, access patterns and ownership are allowed to drift.

Conclusion

This paper presented a catalog of eight design patterns for cost-optimized enterprise data lakes, covering storage tiering and lifecycle management, columnar formats with partitioning and compaction, elastic and interruptible compute, query in place with scan reduction, zoned layers with retention, ingestion mode selection, selective materialization and governance with cost accountability. We related the patterns to the classical warehouse and lake architectures and to a cloud reference architecture, and we evaluated them in a simulated enterprise with 1.2 petabytes of data, 8 terabytes of daily ingestion and 120,000 daily queries. Together the patterns reduced the monthly cost index from 100 to 41, cut median bytes scanned per query by 91%, improved median query time from 18 seconds to 4.2 seconds, lowered duplication from 3.1 to 1.4 copies and increased the share of cost attributable to an owner from 66% to 94%. Compute elasticity and columnar layout provided the largest individual savings, tiering and retention provided substantial storage savings, and governance enabled the others to be sustained.

Future work will proceed in four directions. The first is validation on real platforms with production workloads and actual pricing, including the cost of migration. The second is the development of adaptive, learning-based policies that choose tiers, compaction schedules and materializations from observed access patterns. The third is to extend the catalog to cover data quality, security and privacy requirements, and the cost of compliance. The fourth is to study the interaction of data lakes with machine learning platforms, including feature stores and the storage of model artifacts and training data.

References

- [1] W. H. Inmon, Building the Data Warehouse, 4th ed. Indianapolis, IN, USA: Wiley, 2005.
- [2] R. Kimball and M. Ross, The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, 3rd ed. Indianapolis, IN, USA: Wiley, 2013.
- [3] J. Dixon, "Pentaho, Hadoop, and data lakes," James Dixon's Blog, 2010.
- [4] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia, "Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics," in Proc. Conference on Innovative Data Systems Research (CIDR), 2021.
- [5] M. Armbrust et al., "Delta Lake: High-performance ACID table storage over cloud object stores," Proceedings of the VLDB Endowment, vol. 13, no. 12, pp. 3411-3424, 2020.
- [6] R. Hai, S. Geisler, and C. Quix, "Constance: An intelligent data lake system," in Proc. ACM SIGMOD International Conference on Management of Data, 2016, pp. 2097-2100.
- [7] P. Sawadogo and J. Darmont, "On data lake architectures and metadata management," Journal of Intelligent Information Systems, vol. 56, no. 1, pp. 97-120, 2021.
- [8] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena, "Data lake management: Challenges and opportunities," Proceedings of the VLDB Endowment, vol. 12, no. 12, pp. 1986-1989, 2019.
- [9] S. Melnik et al., "Dremel: Interactive analysis of web-scale datasets," Proceedings of the VLDB Endowment, vol. 3, no. 1-2, pp. 330-339, 2010.
- [10] D. J. Abadi, S. R. Madden, and N. Hachem, "Column-stores vs. row-stores: How different are they really?" in Proc. ACM SIGMOD, 2008, pp. 967-980.
- [11] B. Dageville et al., "The Snowflake elastic data warehouse," in Proc. ACM SIGMOD, 2016, pp. 215-226.
- [12] M. Vuppapalapati, J. Miron, R. Agarwal, D. Truong, A. Motivala, and T. Cruanes, "Building an elastic query engine on disaggregated storage," in Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2020, pp. 449-462.
- [13] A. Gupta et al., "Amazon Redshift and the case for simpler data warehouses," in Proc. ACM SIGMOD, 2015, pp. 1917-1923.
- [14] A. Thusoo et al., "Hive: A warehousing solution over a map-reduce framework," Proceedings of the VLDB Endowment, vol. 2, no. 2, pp. 1626-1629, 2009.
- [15] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in Proc. USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), 2010.
- [16] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2004, pp. 137-150.

- [17] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in Proc. ACM Symposium on Operating Systems Principles (SOSP), 2003, pp. 29-43.
- [18] M. Armbrust et al., "A view of cloud computing," Communications of the ACM, vol. 53, no. 4, pp. 50-58, 2010.
- [19] P. Vassiliadis, "A survey of extract-transform-load technology," International Journal of Data Warehousing and Mining, vol. 5, no. 3, pp. 1-27, 2009.
- [20] J. R. Storment and M. Fuller, Cloud FinOps: Collaborative, Real-Time Cloud Financial Management, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2023.
- [21] M. Kleppmann, Designing Data-Intensive Applications. Sebastopol, CA, USA: O'Reilly Media, 2017.
- [22] M. Stonebraker et al., "C-Store: A column-oriented DBMS," in Proc. International Conference on Very Large Data Bases (VLDB), 2005, pp. 553-564.
- [23] D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos, and S. Madden, "The design and implementation of modern column-oriented database systems," Foundations and Trends in Databases, vol. 5, no. 3, pp. 197-280, 2013.
- [24] I. Khine and Z. Wang, "Data lake: A new ideology in big data era," ITM Web of Conferences, vol. 17, article 03025, 2018.
- [25] A. Behm et al., "Photon: A fast query engine for lakehouse systems," in Proc. ACM SIGMOD, 2022, pp. 2326-2339.
- [26] European Parliament and Council of the European Union, "Regulation (EU) 2016/679 on the protection of natural persons with regard to the processing of personal data (General Data Protection Regulation)," Official Journal of the European Union, 2016.