Morphpublishing
Research
open access academic journal

Peer reviewed

Hierarchical Approximate Aggregation for Interactive Analytics on Compressed Columnar Storage in the Cloud

Khaled Masri[a], **Samir Abusaleh**[b]

Abstract: Cloud data platforms increasingly rely on compressed columnar storage to reduce storage footprint and I/O, yet interactive analytics remains constrained by the latency of scanning, decompressing, and aggregating large datasets. This tension is most visible in exploratory workflows where analysts iteratively adjust filters, groupings, and time ranges while expecting near-immediate feedback. Approximate query processing can lower latency by trading exactness for bounded error, but many existing techniques either ignore the physical organization of columnar files or require heavy precomputation that limits query flexibility. This paper presents a hierarchical approximate aggregation approach designed specifically for compressed columnar storage in cloud object stores. The core idea is to construct a multi-resolution hierarchy aligned with row groups and pages, where each node stores mergeable synopses for common aggregates, including additive measures, distinct counts, and distributional summaries. By operating in a compression-aware manner, the synopsis construction and updates exploit encoding structure such as dictionary maps and run-length segments to reduce decode overhead. At query time, an adaptive planner selects a cut through the hierarchy to satisfy an interactive latency budget while controlling statistical error, then refines results progressively as users drill down. The design integrates with cloud-native execution by decoupling synopses into compact sidecar objects, supporting append-heavy ingestion and multi-tenant caching. Analytical models characterize error propagation and cost trade-offs, motivating a practical selection strategy for hierarchical refinement under heterogeneous cloud I/O and compute variability.

1. Introduction

Interactive analytics over cloud data lakes has converged on a storage pattern where large fact tables are written once and read many times through shared object storage [1]. Columnar formats, often organized into row groups and compressed pages, substantially reduce bytes transferred and improve scan throughput by enabling predicate pushdown and late materialization. However, interactive workloads are defined less by peak throughput and more

[a] **Department of Computer Science and Engineering, Jordan Valley Institute of Technology** [b] **Department of Computer Science and Engineering, Southern Desert University of Applied Sciences**

by tail latency, because analysts expect results quickly enough to guide a sequence of follow-up queries. Even with column pruning and predicate filtering, a common pattern is that the subset that survives filters is still large, forcing aggregation operators to either scan many pages or accept prolonged latencies from decompression and distributed shuffles [2].

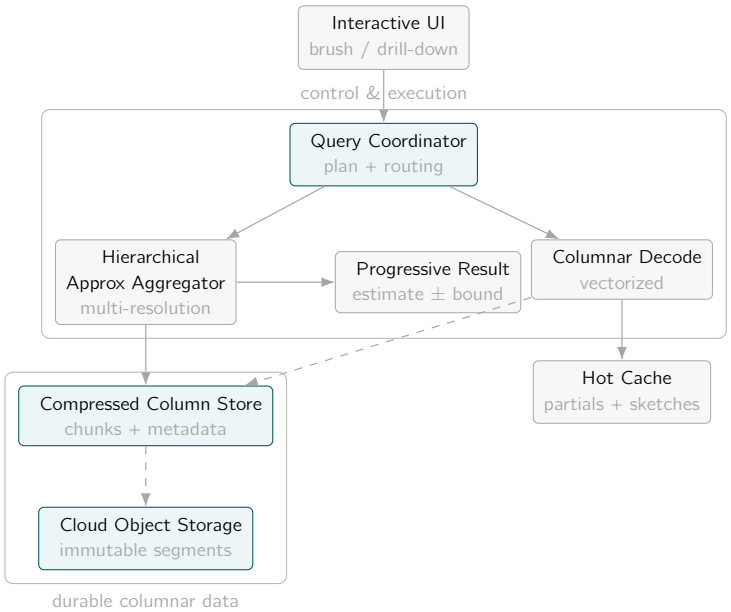

Figure 1. End-to-end architecture for interactive analytics: a coordinator routes queries to a hierarchical approximate aggregator operating over compressed column chunks, returning progressive answers with uncertainty bounds while leveraging cache and cloud object storage.

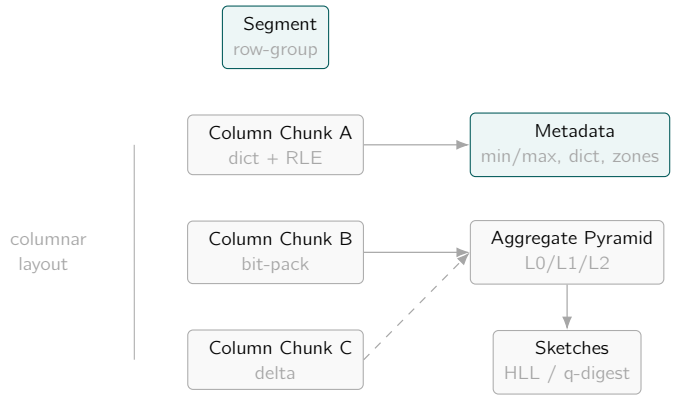

Figure 2. Compressed columnar segment layout: per-column chunks retain codec-specific encodings while side metadata supports pruning; a co-located multi-resolution aggregate pyramid and sketches enable fast approximate answers without full decode.

Approximate query processing provides a principled way to trade accuracy for speed, returning estimates with quantifiable error. In interactive settings, the usefulness of approximation is less about a single final estimate and

Published in CURRENT ISSUE

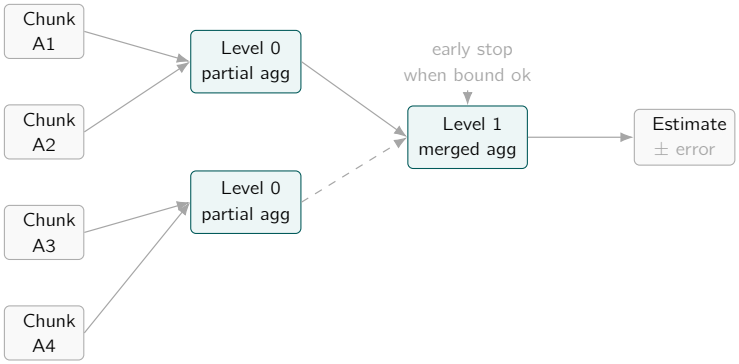

Figure 3. Hierarchical approximate aggregation: chunk-level partials are merged up a small tree, optionally skipping branches (dashed) when the remaining contribution is bounded, producing an explicit error bound.

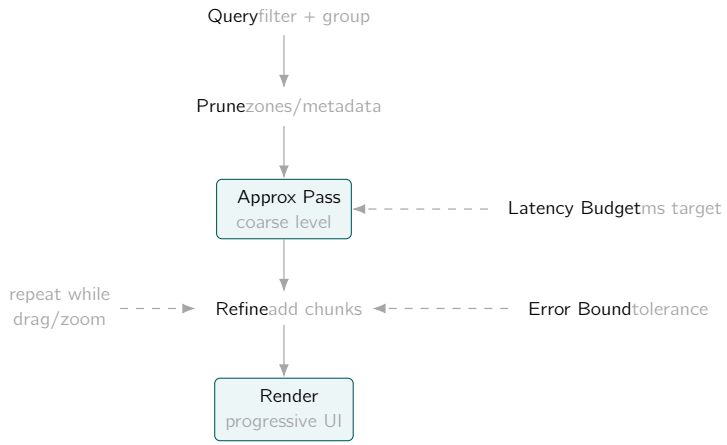

Figure 4. Interactive execution loop: metadata pruning enables a fast coarse approximation, then incremental refinement adds more chunks until the latency budget or the requested error tolerance is met, continuously updating the visualization.

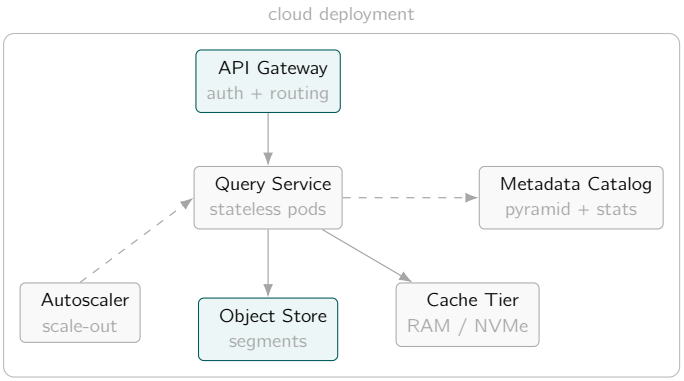

Figure 5. Cloud deployment view: a stateless query service scales elastically, pulling compressed segments from object storage, consulting a metadata catalog for hierarchical aggregates, and exploiting a cache tier for low-latency interactive workloads.

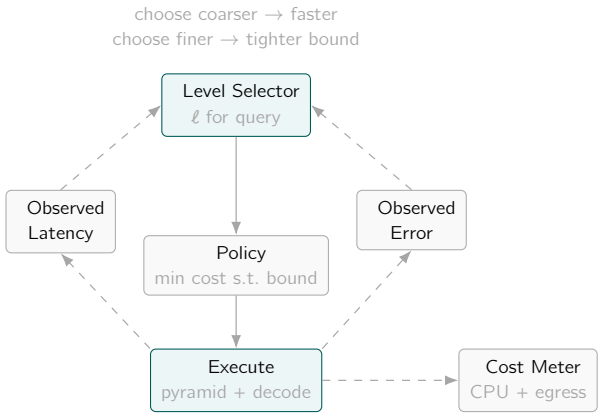

Figure 6. Adaptive selection of aggregation level: a controller chooses a pyramid level per query using feedback from measured latency and approximation error, balancing cloud cost against interactive responsiveness and accuracy targets.

Dataset	Columns	Rows ($\times 10^9$)	Compressed size (GB)
ClickStream	48	1.2	310
AdImpressions	36	0.9	220
IoTReadings	64	2.5	540
UserEvents	28	0.6	150
RetailSales	52	1.8	410

Table 1. Columnar datasets used in the evaluation with sizes reported after compression in the cloud object store.

Hierarchy level	Granularity	Example group key	Sampling fraction
L0 (global)	All records	$\varnothing$	0.1%
L1 (region)	Data center	region_id	0.5%
L2 (cluster)	Compute cluster	region_id, cluster_id	1.0%
L3 (tenant)	Customer account	tenant_id	2.0%
L4 (segment)	Application shard	tenant_id, segment_id	5.0%

Table 2. Logical hierarchy used for approximate aggregation, from coarse global summaries to fine-grained segments.

more about responsiveness across iterations, where early answers guide subsequent query refinement. A key practical challenge is that approximation methods are often designed at the logical level, assuming access to tuples or at least decoded column values. In a cloud data lake, decoding itself can dominate, especially when storage is highly compressed and when compute is provisioned elastically with cold-start overheads [3]. Another challenge is that exploratory analytics frequently involves varying filters and groupings, so pre-aggregations that are too specific can become ineffective, while pre-aggregations that are too general may become too expensive to store and maintain.

A second axis is the physical layout of columnar files. Columnar storage already imposes a hierarchy: objects contain files, files contain row groups, row groups contain column chunks, and column chunks contain pages. Each level naturally defines a partition of the underlying rows and often carries lightweight metadata such as min/max bounds, dictionary pages, and statistics that can be used to skip work [4]. Yet most approximate aggregation

Published in CURRENT ISSUE

Aggregation	Exact semantics	Sketch structure	Error guarantee
SUM	$\sum x$	Weighted reservoir	ε-relative w.h.p.
COUNT	$\lvert X\rvert$	Count-min counter	Exact under no loss
AVG	$\frac{\sum x}{\lvert X\rvert}$	Pair of SUM/COUNT	Inherited from SUM/COUNT
P95	95th percentile	KLL quantile sketch	Rank error $\leq \varepsilon n$
COUNT DISTINCT	$\lvert\{x\}\rvert$	HyperLogLog	Relative error $< 1.04/\sqrt{m}$

Table 3. Supported approximate aggregation operators and their sketch-based implementations.

Codec	Encoding type	Typical column types	Decoding throughput (MB/s)
ZSTD	Dictionary + LZ77	String, JSON	850
LZ4	Block-based	Log payloads	1450
RLE	Run-length	Booleans, enums	3100
Delta + VarInt	Delta	Timestamps, IDs	2700
Bit-Packing	Fixed-width	Low-cardinality ints	3400

Table 4. Compression schemes used in the columnar store and their typical throughput on cloud object storage.

Tier	Cloud instance type	vCPUs	Memory (GB)
Coordinator	c5.4xlarge	16	32
Aggregator	r5.4xlarge	16	128
Leaf scanner	m5.2xlarge	8	32
Object store cache	r5.2xlarge	8	64
Metadata service	t3.large	2	8

Table 5. Representative cloud deployment for the hierarchical approximate aggregation system.

Stage	Exact query (ms)	HAA query (ms)	Speedup ($\times$)
Query planning	25	27	0.9
Column fetch	820	180	4.6
Decompression	640	130	4.9
Local aggregation	410	95	4.3
Hierarchical merge	0	38	–
End-to-end latency	1895	470	4.0

Table 6. Latency breakdown comparing exact execution with hierarchical approximate aggregation on a representative dashboard query.

designs treat storage as a flat stream and either sample rows uniformly or build synopses independently of the file hierarchy. This misalignment can lead to avoidable decoding, increased random I/O, and difficulty in incrementally refining results as users narrow predicates.

This paper develops a hierarchical approximate aggregation method designed to align with compressed columnar storage in the cloud. The approach builds a multi-resolution hierarchy over row partitions that correspond to physical storage boundaries, and optionally over value-domain partitions for common temporal and categorical dimensions

Sampling rate	Avg. rel. error (%)	P95 rel. error (%)	Median latency (ms)
0.25%	4.9	8.7	310
0.50%	3.1	5.9	360
1.00%	1.8	3.2	430
2.00%	1.1	2.0	520
4.00%	0.7	1.3	690
8.00%	0.5	1.0	980

Table 7. Trade-off between sampling rate, relative error, and query latency for SUM, AVG, and P95 aggregates.

Hierarchy depth	Avg. latency (ms)	Network bytes transferred (MB)
1 level (global)	260	34
2 levels (global + region)	310	42
3 levels (global + region + cluster)	370	55
4 levels (add tenant)	470	71
5 levels (full hierarchy)	590	96

Table 8. Effect of hierarchy depth on query latency and network traffic across the cloud deployment.

Component	Exact engine (GB)	HAA engine (GB)	Reduction (%)
Raw column buffers	512	140	72.7
Intermediate aggregates	96	38	60.4
Metadata and indexes	44	28	36.4
Caches	128	92	28.1
Sketches	0	24	–
Total resident memory	780	322	58.7

Table 9. Memory footprint comparison between the exact baseline and the hierarchical approximate aggregation engine.

[5]. Each node stores mergeable synopses that support approximate aggregates. Mergeability is central because it allows both efficient construction, by combining children into parents, and efficient query-time evaluation, by combining selected nodes along a cut of the hierarchy. The hierarchy supports progressive refinement, where an initial estimate is computed from a small number of coarse nodes, then selectively refined by descending into nodes that contribute most to uncertainty [6].

A compression-aware construction phase reduces decode work by exploiting encoding structure, particularly dictionary encoding, run-length encoding, and bit-packing. For example, dictionary-encoded categorical columns enable computing frequency sketches by counting dictionary ids directly, while bit-packed integer measures can be aggregated using vectorized unpacking within page boundaries. These techniques are not intended to eliminate decoding entirely, but to ensure that synopsis construction and incremental updates scale with the compressed representation rather than with fully materialized rows.

At query time, the planner jointly considers an error model and a cost model that reflects cloud realities such as object-store request latency, bandwidth variability, and decompression CPU costs [7]. Instead of treating approximation as a single sampling ratio, the planner selects a cut through the hierarchy that fits within a latency budget and yields an estimate with confidence bounds. When users interactively adjust predicates, caching of node synopses and reuse of prior cuts provide additional speedups. The cloud integration stores synopses as sidecar

Published in CURRENT ISSUE

objects and supports append-heavy pipelines by writing new leaf synopses and recomputing affected internal nodes lazily or via background compaction jobs.

The remainder of the paper formalizes the problem, presents the hierarchy and synopsis design, derives models for error and cost that motivate the planner, and discusses cloud-native integration choices [8]. The emphasis is on the interaction between approximate aggregation and compressed columnar storage, because the physical layer determines both the achievable latency and the cost of maintaining synopses at scale.

2. Preliminaries and Problem Formulation

Consider a relation with n rows and a set of columns partitioned into dimensions and measures. Dimensions are used for filtering and grouping, while measures are aggregated. The data is stored in a columnar file format with a fixed physical hierarchy [9]. A file is partitioned into R row groups, each row group contains one column chunk per column, and each column chunk is divided into pages that are compressed independently. Let $\mathcal{P}$ denote the set of physical row partitions induced by row groups and, when relevant, by page boundaries. Each partition $p \in \mathcal{P}$ corresponds to a contiguous range of row indices in the file, and for each column chunk the pages covering that range have encodings such as dictionary encoding, run-length encoding, delta encoding, and bit-packing.

A query of interest is an aggregate query with a selection predicate and an optional grouping clause. Let D denote the set of dimension columns and M denote the set of measure columns [10]. A predicate ϕ is a boolean expression over a subset of D, typically involving range filters on temporal columns and equality filters on categorical columns. A group key g is either empty, indicating a global aggregate, or a tuple of one or more dimensions. For a measure column $m \in M$ and an aggregate function A such as sum, count, average, distinct count, or quantile, the exact result can be written as a function over the qualifying rows. For a global sum on a measure m, the exact answer is [11]

$$S(\phi, m) = \sum_{i=1}^{n} m_i \cdot \mathbf{1}\{\phi(i)\}, \quad \text{where} \quad \mathbf{1}\{\phi(i)\} \in \{0, 1\}. \tag{1}$$

For a grouped aggregate with group key g, the result is a mapping from group values to aggregates, and evaluation requires partitioning qualifying rows by $g(i)$.

An approximate answer replaces the exact computation with an estimator $\widehat{S}(\phi, m)$ that can be computed with lower latency and provides an error characterization. Error can be expressed in absolute terms, relative terms, or probabilistic confidence intervals. For an estimator intended to be unbiased, one aims for

$$\mathbb{E}\left[\widehat{S}(\phi, m)\right] = S(\phi, m), \tag{2}$$

$$\mathrm{Var}\left(\widehat{S}(\phi, m)\right) \le \sigma^2(\phi, m), \tag{3}$$

where $\sigma^2(\phi, m)$ is controlled by the synopsis sizes and the planner [12]. In interactive analytics, a practical objective is to bound the relative error with high probability, for example

$$\mathbb{P}\left(\left|\widehat{S}(\phi, m) - S(\phi, m)\right| \le \epsilon \cdot \max\{|S(\phi, m)|, \tau\}\right) \ge 1 - \delta, \tag{4}$$

where ϵ is a target relative error, δ is a failure probability, and τ is a stabilizing threshold to avoid pathological ratios when the true value is near zero.

Latency is dominated by a combination of object-store I/O, decompression, and compute. Let $c(p)$ denote the cost to access and process synopsis information or raw pages for a physical partition p [13]. In cloud settings, $c(p)$ can vary with request concurrency, network conditions, and caching state. A query plan selects a set of partitions and a processing mode for each, producing an estimator and associated cost. The interactive constraint can be modeled as a budget B representing a latency target, so feasible plans satisfy

$$\sum_{p \in \mathcal{U}} c(p) \leq B, \tag{5}$$

where $\mathcal{U}$ is the set of units accessed. The central design problem is to build auxiliary structures and a planning strategy so that, for a wide class of predicates and groupings, the system can return accurate estimates under tight budgets without requiring full scans or heavy pre-aggregation across all combinations of dimensions [14].

The physical constraints of compressed columnar storage create both opportunities and limitations. Opportunities arise from per-page statistics, dictionary structures, and the fact that many workloads exhibit locality in time and skew in categories, which can be exploited by hierarchical summarization. Limitations arise from encoding dependencies, where decoding a value may require reading a dictionary page, interpreting bit widths, and applying delta reconstruction [15]. Thus, any approximate approach that ignores encoding can pay an unnecessary CPU cost, and any approach that ignores the hierarchy can pay unnecessary I/O overhead by fetching many small ranges from object storage.

The approach developed here constructs a hierarchy over partitions that is explicitly aligned with the file organization. Each node in the hierarchy corresponds to a set of rows that is a union of leaf partitions, and stores a synopsis sufficient to estimate aggregates restricted to that node. A query is answered by combining synopses from a selected cut of the hierarchy intersected with the predicate, optionally supplemented by limited raw decoding when necessary for refinement [16]. The hierarchy supports adaptive refinement, so that early answers rely on coarse nodes and later refinements incorporate finer nodes when required.

3. Compression-Aware Hierarchical Synopses

The hierarchy is built over a base partitioning of rows determined by storage boundaries. A convenient choice is to treat each row group as a leaf, because row groups are typically the unit of predicate skipping and represent a balance between metadata granularity and overhead. In formats where pages carry independent statistics and where row groups are large, the leaf granularity can be refined to page clusters, but the design remains the same [17]. Let the leaves be indexed by $\ell \in \{1, \ldots, L\}$, each corresponding to a disjoint row set R_ℓ. A rooted tree is constructed where each internal node is a union of its children, and the root covers all rows in the file or dataset shard. The branching factor can be tuned to balance tree height and node fanout, with the goal of reducing the number of nodes required to cover a predicate-defined subset.

Each node stores mergeable synopses for aggregates. For additive measures such as sum and count, exact aggregates per node are feasible because they are small [18]. For more complex aggregates such as distinct count and quantiles, sketches are required. A node synopsis is therefore a bundle of structures, each of which supports a merge operation. Mergeability enables construction by bottom-up aggregation and enables query-time evaluation by combining synopses across selected nodes [19]. The synopsis bundle is also designed to be compressible itself, because storing synopses for many nodes can otherwise become large. In practice, many nodes have similar distributions, and sketches can be stored in compact binary representations.

Compression-awareness enters at construction time. For a given leaf partition, computing synopses naively would decode all relevant columns and iterate through rows [20]. Instead, one exploits common encodings. For dictionary

Published in CURRENT ISSUE

encoding of a categorical column d, the page stores a dictionary mapping ids to values and an encoded stream of ids. For many query patterns, the synopsis only needs frequencies of categories or hashed representations for distinct counting. Let id_i denote the dictionary id for row i in the leaf [21]. Frequency counts can be derived by counting ids, without decoding the full strings. If the aggregate is grouped by a dictionary-encoded dimension, one can build a per-id aggregation table within the leaf. For a numeric measure m encoded as integers with scaling, the encoded stream often consists of bit-packed integers or deltas. Summation can be computed by unpacking integers and accumulating, but page-level vectorization reduces overhead [22]. When run-length encoding is used, the encoded stream consists of pairs (v_k, r_k) where v_k is a value and r_k is the run length, so sums and counts can be computed as $\sum_k v_k r_k$ and $\sum_k r_k$ without iterating per row.

To formalize the benefit, consider a leaf partition with a run-length encoded measure. Let the encoded representation be $\{(v_k, r_k)\}_{k=1}^{K}$ where $\sum_k r_k = |R_\ell|$. The exact leaf sum is

$$S_\ell(m) = \sum_{k=1}^{K} v_k \cdot r_k. \tag{6}$$

If $K \ll |R_\ell|$, the work scales with K rather than rows. Similar reductions occur for dictionary-encoded dimensions where the number of distinct ids within a leaf is small relative to the number of rows [23].

Distinct counting is supported via mergeable sketches such as register-based cardinality estimators. For a dimension value x, a hash function $h(x)$ maps to a bit string and updates a register array. When the dimension is dictionary-encoded, the hash of each dictionary entry can be computed once per dictionary page, then applied to counts of ids [24]. If f_j is the frequency of dictionary id j in the leaf, then rather than hashing f_j times, one updates the sketch once for that id, because distinctness cares about presence not multiplicity. This reduces computation dramatically for repeated categorical values.

Quantile and distributional aggregates require summaries that can be merged. A practical design stores a compact quantile sketch per node for each measure used in distribution queries [25]. Construction can be made compression-aware by extracting approximate distributions from page histograms when available, or by sampling within pages in a way that respects encoding boundaries to avoid full decode. For bit-packed or delta-encoded integers, one can sample positions by skipping blocks and decoding only selected blocks. In the presence of RLE, sampling a run requires selecting a representative position, which can be done by treating each run as a weighted item.

A critical design choice is how the hierarchy interacts with predicates [26]. Predicates often involve time ranges, and cloud datasets are frequently sorted or clustered by time. When the physical partitioning respects clustering, then many predicates align with contiguous ranges of leaves, making hierarchical coverage efficient. For less aligned predicates, the hierarchy still helps by allowing coarse pruning with min/max statistics at internal nodes, but the benefit depends on how well metadata summarizes the predicate selectivity.

To support predicate-aware estimation, each node stores lightweight bounds and selectivity proxies derived from column statistics [27]. For a numeric filter column d with min and max in node u, a range predicate can skip u if disjoint, include u if fully contained, and mark u as partial otherwise. Partial nodes require more careful estimation. A node also stores a sample-based estimate of predicate selectivity for common filters, obtained by maintaining a small reservoir sample of filter columns [28]. Reservoir sampling can be implemented in a mergeable way by storing fixed-size samples per node and merging by weighted sampling, though this introduces additional randomness. Alternatively, one can store per-node bitmaps for low-cardinality predicates, but this is only feasible when the domain is small.

The synopsis size per node must be controlled. Suppose a node stores a additive scalars, a cardinality sketch with r registers, and a quantile sketch with capacity k [29]. Let b_s denote bytes per scalar, b_r bytes per register, and b_q bytes per quantile item. Then the storage per node is approximately

$$\text{Size}(u) = a \cdot b_s \; + \; r \cdot b_r \; + \; k \cdot b_q \; + \; \text{overhead}(u). \tag{7}$$

Total storage scales with the number of nodes. For a balanced tree with branching factor b and L leaves, the number of nodes is about $(bL - 1)/(b - 1)$, so synopsis storage grows linearly in L [30]. This suggests choosing leaves at row-group granularity rather than page granularity unless the workload demands finer refinement. It also suggests compressing synopses, for example by delta encoding register arrays when sparse, or by entropy coding quantile sketch buffers.

Error propagation depends on sketch properties and the way nodes are combined. Additive scalars are exact, so their error is zero [31]. For distinct sketches with relative standard error proportional to $1/\sqrt{r}$, merging does not increase error beyond the sketch's intrinsic variance, but combining multiple independent sketches can reduce variance if designed carefully. For quantile sketches, error bounds often depend on the sketch capacity k and the number of merged streams. A hierarchy that merges many leaves into a parent can concentrate error at coarse nodes. This is not necessarily harmful, because coarse nodes are only used for coarse answers, but it motivates allocating larger sketch capacity to higher-level nodes if coarse answers must be accurate. A practical compromise is to store larger sketches at leaves and merge upward on demand, caching internal node sketches when they are frequently queried, which shifts compute to hot paths without permanently storing large summaries everywhere [32].

Compression-aware maintenance is essential in append-heavy lakehouse pipelines. When new files arrive, leaf synopses are computed for new row groups and written as sidecar objects. Parent synopses can be updated by merging children, but immediate updates can be expensive if the hierarchy spans many files [33]. Instead, one can maintain per-file trees and then a higher-level tree across files, updating higher levels lazily. Lazy maintenance affects freshness of approximations but can be controlled by versioning and by storing both recent exact aggregates for small increments and approximate aggregates for the full history.

4. Interactive Query Processing and Error Control

Query answering begins with predicate evaluation over node metadata. Given a predicate ϕ, each node u is classified as excluded, included, or partial based on min/max bounds and other statistics for the filter columns [34]. Included nodes can contribute their stored aggregate synopses directly. Excluded nodes contribute nothing. Partial nodes require estimation of the qualifying fraction within the node. The system therefore computes an estimate as a sum over nodes in a selected cut $\mathcal{C}$ of the hierarchy. For an additive sum, the estimator can be written as [35]

$$\widehat{S}(\phi, m) = \sum_{u \in \mathcal{C}} \widehat{\alpha}_u(\phi) \cdot S_u(m), \tag{8}$$

where $S_u(m)$ is the stored exact sum of measure m over all rows in node u, and $\widehat{\alpha}_u(\phi) \in [0, 1]$ estimates the selectivity of ϕ within u. For included nodes, $\widehat{\alpha}_u(\phi) = 1$, and for excluded nodes the node does not appear in $\mathcal{C}$. The main uncertainty arises from partial nodes, where $\widehat{\alpha}_u(\phi)$ is estimated.

Estimating $\alpha_u(\phi)$ can be done by maintaining per-node samples of filter columns. Suppose node u stores a sample of size s_u of rows with respect to a sampling design that is approximately uniform within the node. Let $X_{u,j}$

Published in CURRENT ISSUE

be the indicator that sample row j satisfies ϕ, then

$$\widehat{\alpha}_u(\phi) = \frac{1}{s_u} \sum_{j=1}^{s_u} X_{u,j}. \tag{9}$$

Under independence assumptions, $\mathrm{Var}(\widehat{\alpha}_u(\phi)) \approx \alpha_u(\phi)(1-\alpha_u(\phi))/s_u$. The induced variance of $\widehat{S}(\phi, m)$ for additive sums is then

$$\mathrm{Var}\left(\widehat{S}(\phi, m)\right) = \sum_{u \in \mathcal{C}_{\text{partial}}} S_u(m)^2 \cdot \mathrm{Var}\left(\widehat{\alpha}_u(\phi)\right), \tag{10}$$

where $\mathcal{C}_{\text{partial}}$ is the subset of nodes in the cut that are partial. This expression motivates refinement strategies that target nodes with large $|S_u(m)|$ and high selectivity uncertainty.

The hierarchy enables a progressive refinement mechanism [36]. The system first selects a coarse cut, often consisting of a small number of high-level nodes that cover the predicate range. It computes an initial estimate quickly using their synopses and selectivity samples. If the estimated error exceeds a target, the planner refines by expanding selected partial nodes into their children, replacing one node with several finer nodes in the cut [37]. Finer nodes generally have more homogeneous predicates and lower selectivity uncertainty, and they may also allow more accurate classification as included or excluded due to tighter min/max bounds.

Refinement selection is guided by a cost-benefit model. Let u be a partial node with children $\mathrm{ch}(u)$. Replacing u with its children changes both cost and variance. Let $\Delta c(u)$ be the additional cost of accessing the children synopses relative to using u, and let $\Delta v(u)$ be the expected reduction in estimator variance [38]. A greedy strategy expands the node with the largest ratio $\Delta v(u)/\Delta c(u)$ until the budget is exhausted or the error target is met. While greedy is not guaranteed optimal, it is practical for interactive settings and aligns with the diminishing returns typical of hierarchical refinement.

Cost estimation in cloud settings must reflect object-store behavior. Accessing many small objects can be slower than accessing fewer larger ones due to per-request overheads, even if total bytes are smaller [39]. Thus, synopses are stored in a layout that supports batched reads, such as grouping synopses for adjacent nodes into a single sidecar object per file or per file stripe. The planner models cost per node as a combination of request overhead and bytes transferred, and it preferentially refines within already-fetched sidecar regions. If caching is present, the planner also accounts for cache hit probability, which can be learned from recent query history.

Group-by queries introduce additional complexity because selectivity and aggregation must be conditioned on group keys [40]. When group cardinality is moderate, the leaf synopsis can include a per-group table for common dimensions, constructed efficiently when the group key is dictionary-encoded. For high-cardinality group keys, exact per-group tables become large, so approximate heavy-hitter sketches and hashed aggregation sketches are used. A mergeable hashed aggregation sketch maps group keys to buckets and stores additive sums in those buckets. This introduces collisions, so error must be bounded [41]. If a sketch has w buckets and uses pairwise independent hashing, then the expected collision-induced additive error for a specific group can be related to the total mass of other groups mapped to the same bucket. A conservative approach stores multiple hash tables and takes a median estimate. For a count-min style structure with depth d and width w, the error bound on a nonnegative sum can be expressed as [42]

$$\widehat{S}_g \le S_g + \epsilon \cdot S_{\text{tot}} \quad \text{with probability at least} \quad 1-\delta, \tag{11}$$

$$\epsilon \approx \frac{1}{w}, \quad \delta \approx e^{-d}, \tag{12}$$

where S_g is the true group sum and S_{tot} is the total sum over all groups in the node. While this bound is loose, it provides a tunable trade-off via w and d. In practice, storing sketches at multiple hierarchical levels can reduce group collision error because finer nodes distribute mass across more sketches, reducing the total mass per node.

Distinct counts per group can be supported using a sketch per group for low to moderate cardinality, or using a combined sketch that incorporates the group into the hash, effectively estimating distinct pairs. For example, one can hash (g, x) pairs into a cardinality sketch to estimate the number of distinct x values per group, but this does not produce per-group results directly [43]. To obtain per-group distinct estimates, one uses a hashed group sketch where each group maintains a small register set, which is feasible only for a limited number of groups. Therefore, interactive group-by distinct queries typically focus on top groups, and the system can prioritize accuracy for heavy hitters by allocating more sketch space to them at construction time or by refining into finer nodes where group distributions are less mixed.

Quantile queries under predicates are answered by merging quantile sketches from selected nodes and then adjusting for partial-node selectivity. If a node is fully included, its quantile sketch contributes directly [44]. If a node is partial, naive scaling does not apply because quantiles are not linear. Instead, the system either refines partial nodes until they become included or excluded, or it uses a weighted sampling approach where partial-node samples of the measure are used to approximate the conditional distribution. A practical approach is to keep per-node measure samples and to merge samples across nodes, then compute approximate quantiles from the merged sample. The error then depends on sample size and distribution shape [45]. If the merged sample has size s and values are independent, then for a quantile at probability p the asymptotic standard deviation depends on the density at the true quantile, which is generally unknown. The system therefore reports empirical confidence intervals based on bootstrap resampling within the sample, which is computationally feasible at small s and can be bounded by a fixed budget.

The planner's objective is to satisfy interactive constraints [46]. One can express the planning problem as selecting a cut $\mathcal{C}$ and refinement sequence that minimizes estimated error subject to a cost budget:

$$\min_{\mathcal{C}} \quad \widehat{\mathrm{Err}}(\mathcal{C}, \phi, m) \tag{13}$$

$$\text{subject to} \quad \widehat{\mathrm{Cost}}(\mathcal{C}) \le B, \tag{14}$$

$$\mathcal{C} \text{ is a valid cut of the hierarchy over relevant nodes.} \tag{15}$$

Here $\widehat{\mathrm{Err}}$ depends on selectivity sample variances, sketch errors, and group collision errors, while $\widehat{\mathrm{Cost}}$ depends on I/O and compute. The planner evaluates a small set of candidate cuts using metadata and cached statistics, then chooses a cut and optionally triggers asynchronous refinement within the same request if the budget permits. In an interactive UI, progressive results can be returned in stages, but even without UI-level streaming, the system can prioritize returning a fast estimate first and then a refined estimate if the user waits or requests higher accuracy.

A subtle but important consideration is bias introduced by selectivity estimation. If selectivity samples are not perfectly uniform due to storage-level sampling constraints, estimators can be biased [47]. For example, sampling entire pages rather than rows can overweight certain value ranges if pages are value-clustered. The system mitigates this by stratifying samples across pages and by maintaining per-page weights. If a node stores a sample of pages rather than rows, then selectivity estimation uses a Horvitz–Thompson style weighting where each sampled unit is weighted by the inverse of its selection probability. Let U be the set of sampling units in node u, with unit t having weight w_t proportional to its row count, and let π_t be its inclusion probability [48]. Then an unbiased estimator of

Published in CURRENT ISSUE

qualifying row count within u is

$$\widehat{N}_u(\phi) = \sum_{t \in \mathcal{S}_u} \frac{w_t}{\pi_t} \cdot \widehat{\alpha}_t(\phi), \tag{16}$$

where $\mathcal{S}_u$ is the sampled unit set and $\widehat{\alpha}_t(\phi)$ is the within-unit qualifying fraction computed from limited decoding. This estimator can be used to derive $\widehat{\alpha}_u(\phi) = \widehat{N}_u(\phi)/N_u$ where N_u is the node row count.

Overall, interactive performance derives from combining three effects: coarse pruning by metadata, fast aggregation by merging node synopses rather than scanning raw pages, and adaptive refinement targeted at the nodes that dominate uncertainty. The compression-aware construction ensures that building and updating synopses is feasible at ingestion scale, and the hierarchical layout ensures that query-time access patterns remain aligned with cloud object-store constraints.

5. Cloud-Native Execution and Storage Integration

A cloud-native deployment separates raw columnar data and synopsis sidecars while keeping them co-located in object storage for durability and scalability [49]. Each columnar file is accompanied by a sidecar object that stores the hierarchy for that file, or stores leaf synopses plus sufficient metadata to derive internal nodes by merging. The choice depends on workload and update patterns. Storing full per-file hierarchies reduces query-time merges but increases sidecar size; storing only leaves reduces storage but requires merging at query time or caching internal nodes. A balanced approach stores leaves and a small number of higher-level nodes, such as a top-k set of internal levels, allowing coarse cuts to be answered without merges while keeping storage bounded [50].

Sidecar objects are structured to support range reads. Because object stores typically support byte-range GET operations, the sidecar is laid out so that synopses for adjacent nodes occupy contiguous ranges. This enables the planner to fetch a block of synopses with a single request, amortizing request overhead [51]. The layout can mirror the tree traversal order, such as breadth-first ordering, so that higher-level nodes are near the beginning of the object, supporting fast initial answers. Deeper nodes can be stored later, fetched only on refinement. Compression is applied to sidecars as well, but the compression choice favors fast decompression and random access. Block-level compression with independent blocks aligns with range reads and avoids decompressing the entire sidecar for a small set of nodes [52].

Query execution is implemented as a distributed service that can run in containers or serverless functions. Serverless execution offers elastic scaling but imposes cold-start delays and limits on memory and runtime. The hierarchy supports serverless constraints by minimizing the amount of state required per query. A stateless function can fetch relevant sidecar ranges, merge synopses, compute estimates and confidence intervals, and return results [53]. Caching is introduced at two layers: a shared cache in a low-latency key-value store for frequently accessed synopses, and an in-function ephemeral cache across requests when warm instances are reused. The planner accounts for cache hits by tracking recent access patterns per dataset and per predicate range, which is particularly beneficial for time-range queries that repeatedly touch recent partitions.

Multi-file datasets require a higher-level organization. If a dataset consists of many files, each with its own per-file hierarchy, then a dataset-level hierarchy can be built by grouping files into partitions, for example by date or by ingestion batch, and maintaining aggregate synopses at that level [54]. A dataset-level node can store merged synopses for a group of files, enabling queries over broad ranges to be answered without touching every file's sidecar. Maintaining these dataset-level synopses requires incremental merges as new files arrive. In an append-only model, new leaves are added and internal nodes updated [55]. Updates can be done eagerly in a periodic job or

lazily on demand. Lazy updates reduce ingestion cost but can cause query-time merges that increase latency. A hybrid approach maintains recent partitions eagerly, because they are most likely to be queried interactively, and maintains older partitions lazily.

Consistency and versioning matter because cloud object stores provide eventual consistency semantics in some configurations and because datasets can be updated through compaction and rewrite operations [56]. Synopses are versioned with the corresponding data files. Each sidecar includes the data file's content identifier or checksum, and dataset-level synopses reference a manifest of file versions. When a compaction job rewrites multiple small files into a larger file, it also produces a new sidecar and updates the manifest. Queries consult the manifest to identify the correct set of sidecars [57]. This approach avoids inconsistencies where synopses refer to data that has been rewritten.

Security and multi-tenancy introduce additional constraints. Sidecars can reveal distributional information even when raw data access is restricted, so access control policies must apply to both raw data and synopses. In a shared environment, the planner must also enforce per-tenant budgets to prevent expensive refinement patterns from degrading others' performance [58]. This is particularly relevant when a user requests extremely low error for a broad predicate range, which could trigger deep refinement across many nodes. The system enforces caps on refinement depth and on total sidecar bytes fetched per query. If a user requests stricter accuracy, the system can respond by indicating that the requested bound exceeds the interactive budget and returning the best estimate within the budget, rather than attempting an unbounded scan.

Cloud cost modeling is central to practical adoption [59]. Each query incurs costs from object-store requests, data transfer, and compute time. Approximation reduces compute and transfer, but sidecar reads introduce additional requests. Therefore, sidecar layout and caching are designed to minimize request counts [60]. A simple expected cost model per query can be written as

$$\widehat{\mathrm{Cost}} = N_{\mathrm{req}} \cdot t_{\mathrm{req}} \; + \; \frac{B_{\mathrm{bytes}}}{\mathrm{bw}} \; + \; t_{\mathrm{cpu}}(\mathrm{merge}, \mathrm{decode}), \tag{17}$$

where N_{req} is the number of object-store requests, t_{req} is average per-request overhead, B_{bytes} is total bytes transferred, bw is effective bandwidth, and t_{cpu} accounts for decompression and merging. The planner uses measured telemetry to update these parameters. Because tail latency matters, the planner may use conservative estimates such as high-percentile request overhead rather than averages.

Compression-aware execution extends to selective decoding when refinement requires more detail than synopses provide [61]. For example, a partial node might require decoding a small sample of rows to better estimate selectivity for a complex predicate involving multiple columns. Rather than reading entire pages, the system reads only the relevant column pages and decodes only sampled positions. In columnar storage, decoding a value at position i within a page may require decoding preceding values depending on encoding. Therefore, the system prefers encodings that allow random access within blocks, or it decodes in small blocks and extracts the sample positions within those blocks [62]. This selective decoding is still more expensive than reading synopses, so it is used sparingly and only when the error target cannot be met through hierarchical refinement alone.

Operationally, synopsis construction is integrated into the ingestion pipeline. When a new file is produced, a build step reads the file footer metadata, identifies row groups, and computes leaf synopses. The build step can be executed in the same compute environment that writes the file, benefiting from data locality and warm caches [63]. If that is not possible, it can be executed as a separate job that reads the file from object storage. Compression-aware algorithms reduce the cost of this step, but it remains nontrivial for very large ingest rates. Therefore, the system supports adaptive build policies where only a subset of columns and synopsis types are built based on

observed query workloads [64]. If quantile queries are rare, quantile sketches can be omitted or built only for a subset of measures, while additive aggregates and distinct sketches are maintained broadly.

Finally, monitoring and validation are necessary because approximate answers must maintain user trust. The system periodically validates estimates against exact computations on sampled queries, using background jobs that run exact scans for a small fraction of workloads. Discrepancies beyond expected bounds indicate issues such as biased sampling, stale synopses, or misclassified predicate coverage due to incorrect metadata [65]. These checks are especially important after schema evolution, encoding changes, or compaction operations that alter physical layout.

6. Conclusion

Hierarchical approximate aggregation aligned with compressed columnar storage provides a practical path toward low-latency interactive analytics in cloud data lakes. By building a multi-resolution hierarchy over storage-aligned row partitions and storing mergeable synopses per node, the approach enables fast initial estimates derived from coarse nodes and progressive refinement through selective expansion. Compression-aware construction leverages dictionary and run-length structure to reduce decoding overhead during synopsis maintenance, and cloud-native sidecar layouts support efficient range reads, caching, and versioned consistency with evolving datasets [66]. Analytical models for selectivity-driven error and cloud cost motivate an adaptive planning strategy that chooses hierarchy cuts under latency budgets while reporting confidence bounds.

The design is not without trade-offs. Synopsis storage grows with the number of leaves and supported aggregates, and maintaining distributional summaries for many measures can become costly. Selectivity estimation for complex predicates remains challenging when predicates involve multiple correlated columns, and quantile estimation under partial coverage may require selective decoding or deeper refinement [67]. Cloud variability in request latency and bandwidth can also affect the predictability of interactive performance, necessitating conservative planning and robust caching.

Within these constraints, a storage-aligned hierarchical approach emphasizes flexibility, because it supports a broad space of predicates and groupings without requiring exhaustive pre-aggregation across dimension combinations. It also emphasizes operational compatibility with append-heavy pipelines by enabling incremental leaf builds and lazy internal merges. Future work can explore tighter integration with file clustering and partitioning strategies, richer predicate-aware synopses that capture correlations without excessive size, and workload-adaptive policies that allocate synopsis resources to the aggregates and columns most relevant to interactive usage [68].

References

[1] J.-H. Syu, J. C.-W. Lin, P. Biernacki, and A. Ziebinski, "Machine learning-based calibration approaches for single-beam and multiple-beam distance sensors," *IEEE Sensors Journal*, vol. 24, no. 1, pp. 975–983, 1 2024.

[2] M. Goudarzi, Q. Deng, and R. Buyya, *Resource management in edge and fog computing using FogBus2 framework*. The Institution of Engineering and Technology, 5 2024, pp. 17–52.

[3] M. Zaccarini, F. Poltronieri, D. Borsatti, W. Cerroni, L. Foschini, G. Y. Grabarnik, D. Scotece, L. Shwartz, C. Stefanelli, and M. Tortonesi, "Chaos engineering based kubernetes pod rescheduling through deep sets and reinforcement learning," in *NOMS 2025-2025 IEEE Network Operations and Management Symposium*. IEEE, 5 2025, pp. 1–7.

[4] H. Geppert, F. Dürr, and K. Rothermel, "Efficient conflict graph creation for time-sensitive networks with dynamically changing communication demands," *IEEE Transactions on Network and Service Management*, pp. 1–1, 1 2025.

[5] S. Davies, A. Reid, S. Spencer, and J. Carter, "The impact of estimation methodologies on distributed systems," 4 2019.

[6] R. Malik, S. Kim, X. Jin, C. Ramachandran, J. Han, I. Gupta, and K. Nahrstedt, "Mlr-index: An index structure for fast and scalable similarity search in high dimensions," in *International Conference on Scientific and Statistical Database Management*. Springer, 2009, pp. 167–184.

[7] R. K. Ghosh and H. Ghosh, "Distributed knowledge management," 2 2023.

[8] A. Arman, C. Badii, P. Bellini, S. Bilotta, P. Nesi, and M. Paolucci, "Analyzing demand with respect to offer of mobility," *Applied Sciences*, vol. 12, no. 18, pp. 8982–8982, 9 2022.

[9] D. Heye, S. Islam, J. Pennekamp, and K. Wehrle, "Poster: Transport security orchestration using dns," in *2025 IEEE 33rd International Conference on Network Protocols (ICNP)*. IEEE, 9 2025, pp. 1–3.

[10] M. Maresch and S. Nastic, "Vate: Edge-cloud system for object detection in real-time video streams," in *2024 IEEE 8th International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 5 2024, pp. 27–34.

[11] H. M. Zangana, N. Y. Ali, and S. R. M. Zeebaree, "Transforming public management," *Indonesian Journal of Education and Social Sciences*, vol. 4, no. 1, pp. 36–46, 1 2025.

[12] Z. Mayransaev and A. B. Chernyshev, "Generalized circular criterion for the absolutesustainability of distributed systems," *IZVESTIYA SFedU. ENGINEERING SCIENCES*, no. 2, pp. 182–189, 7 2021.

[13] K. He, "Research on verifiable access technology of mixed database in distributed system under big data," in *2021 International Conference on Aviation Safety and Information Technology*. ACM, 12 2021, pp. 636–640.

[14] Z. Yu, J. Gu, Z. Wu, N. Liu, and J. Guo, "Htll: Latency-aware scalable blocking mutex," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–17, 1 2025.

[15] J. A. McDougall, A. Brighente, A. Kunstmann, N. Zapatka, H. Schambach, and H. Federrath, "Probing with a generic mac address: An alternative to mac address randomisation," in *2024 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, 9 2024, pp. 1–6.

[16] Y. Zhao, T. Xin, and M. Dong, "Parer: Boosting erofs image creation with parallelism and reproducibility," in *Proceedings of the 15th Asia-Pacific Symposium on Internetware*. ACM, 7 2024, pp. 347–356.

[17] P. Weisenburger, M. Köhler, and G. Salvaneschi, "Distributed system development with scalaloci," 9 2018.

[18] T. Robertazzi and M. Drozdowski, "Interaction maxima in distributed systems," 1 2021.

[19] J. Tao, L. Xu, and S. Guo, "A deinterleaving approach of pulse signals in distributed system," *Signal, Image and Video Processing*, vol. 19, no. 13, 9 2025.

[20] S. Deng, H. Zhao, B. Huang, C. Zhang, F. Chen, Y. Deng, J. Yin, S. Dustdar, and A. Y. Zomaya, "Cloud-native computing: A survey from the perspective of services," 6 2023.

[21] R. Chandrasekar, R. Suresh, and S. Ponnambalam, "Evaluating an obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 International Conference on Advanced Computing and Communications*. IEEE, 2006, pp. 628–629.

Published in CURRENT ISSUE

[22] M. Broy, "Concurrent distributed systems beyond monotonicity," 1 2024.

[23] C. Akmut, "6.824 'distributed systems' (special 4)," 12 2023.

[24] N. Caldognetto, L. P. Evangelisti, F. Poltronieri, M. Russo, C. Stefanelli, S. Tenani, S. Toboli, and M. Tortonesi, "Water 4.0: enabling smart water and environmental data metering," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 4 2022, pp. 1–6.

[25] R. Rotta, N. S. Chatharajupalli, B. Naumann, J. Schulz, R. Karnapke, M. Werner, and J. Nolte, "Demo: B.a.t.m.a.n. mesh routing on ultra low-power ieee 802.11 modules," in *2024 IEEE 49th Conference on Local Computer Networks (LCN)*. IEEE, 10 2024, pp. 1–4.

[26] F. Ritz, T. Phan, A. Sedlmeier, P. Altmann, J. Wieghardt, R. Schmid, H. Sauer, C. Klein, C. Linnhoff-Popien, and T. Gabor, *Capturing Dependencies Within Machine Learning via a Formal Process Model*. Germany: Springer Nature Switzerland, 10 2022, pp. 249–265.

[27] N. F. Hasani, "Load balancing in distributed system," *International Journal of Basic and Applied Sciences*, vol. 14, no. 4, pp. 144–148, 8 2025.

[28] T. Nowak, U. Schmid, and K. Winkler, "Topological characterization of consensus in distributed systems," *Journal of the ACM*, vol. 71, no. 6, pp. 1–48, 11 2024.

[29] S. Wang, S. Liu, X. Fan, H. Wang, Y. Zhou, H. Gao, H. Lu, and X. Deng, "Vg-net: Sensor time series anomaly detection with joint variational autoencoder and graph neural network," in *2024 IEEE Smart World Congress (SWC)*. IEEE, 12 2024, pp. 456–463.

[30] Y. Kubiuk and K. Kharchenko, "Design and implementation of the distributed system using an orchestrator based on the data flow paradigm," 6 2020.

[31] G. Pierre, "Le « fog computing » est l'avenir du cloud – en plus frugal et plus efficace," *The conversation*, pp. 1–4, 8 2021.

[32] A. S. M. Noor, N. F. M. Zian, and F. N. M. S. Bahri, "Survey on replication techniques for distributed system," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 2, pp. 1298–1303, 4 2019.

[33] A. Poshtkohi and M. B. Ghaznavi-Ghoushchi, *Implementing Parallel and Distributed Systems*. Auerbach Publications, 2 2023.

[34] E. B. Gulcan, B. K. Ozkan, R. Majumdar, and S. Nagendra, "Model-guided fuzzing of distributed systems," 10 2024.

[35] R. Chandrasekar and T. Srinivasan, "An improved probabilistic ant based clustering for distributed databases," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, 2007, pp. 2701–2706.

[36] Z. Wang, C. Tang, X. Zhang, Q. Yu, B. Zang, H. Guan, and H. Chen, "Ad hoc transactions through the looking glass: An empirical study of application-level transactions in web applications," *ACM Transactions on Database Systems*, vol. 49, no. 1, pp. 1–43, 2 2024.

[37] P. S. Sapaty, *Managing Distributed Systems with Spatial Grasp Patterns*. CRC Press, 2 2024, pp. 127–145.

[38] "International journal of distributed systems and technologies," 8 2024.

[39] X. Yuan and J. Yang, "Asplos - effective concurrency testing for distributed systems," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 3 2020, pp. 1141–1156.

[40] A. Polyakov, "On global existence of strong and classical solutions of navier-stokes equations," 4 2019.

[41] S. H. S. A. UBAIDILLAH, N. AHMAD, and N. A. Sahabudin, "A survey on potential reactive fault tolerance approach for distributed systems in big data," in *Third International Conference on Computer Vision and Information Technology (CVIT 2022)*. SPIE, 2 2023, pp. 21–21.

[42] O. V. Talaver and T. A. Vakaliuk, "Reliable distributed systems: review of modern approaches," *Journal of Edge Computing*, vol. 2, no. 1, pp. 84–101, 5 2023.

[43] T. Srinivasan, R. Chandrasekar, V. Vijaykumar, V. Mahadevan, A. Meyyappan, and A. Manikandan, "Localized tree change multicast protocol for mobile ad hoc networks," in *2006 International Conference on Wireless and Mobile Communications (ICWMC'06)*. IEEE, 2006, pp. 44–44.

[44] J. Breuer, B. Čemusová, J. Fischer, J. Roztočil, and V. Vigner, *Synchronization of Distributed Systems using GPS*. River Publishers, 10 2024, pp. 95–120.

[45] L. Guegan, B. L. Amersho, A.-C. Orgerie, and M. Quinson, *AINA - A Large-Scale Wired Network Energy Model for Flow-Level Simulations*. Springer International Publishing, 3 2019, vol. 926, pp. 1047–1058.

[46] M. Farhadi, D. Miorandi, and G. Pierre, "Blockchain enabled fog structure to provide data security in iot applications," 1 2019.

[47] A. Desai, A. Phanishayee, S. Qadeer, and S. A. Seshia, "Compositional programming and testing of dynamic distributed systems," *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, pp. 159–30, 10 2018.

[48] L. Sorokin, "Towards auditable distributed systems," 1 2022.

[49] N. Benmoussa, M. F. Amr, S. Ahriz, K. Mansouri, and E. Illoussamen, "Outlining a model of an intelligent decision support system based on multi agents," *Zenodo (CERN European Organization for Nuclear Research)*, 6 2018.

[50] V. Vijaykumar, R. Chandrasekar, and T. Srinivasan, "An obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 IEEE Conference on Cybernetics and Intelligent Systems*. IEEE, 2006, pp. 1–6.

[51] A. M. Ahmed, M. A. Saeed, A. A. Hamood, A. A. Alazab, and K. A. Ahmed, "Comparative study of static analysis and machine learning approaches for detecting android banking malware," in *2023 3rd International Conference on Emerging Smart Technologies and Applications (eSmarTA)*. IEEE, 10 2023, pp. 1–8.

[52] S.-M. Choi, J. Park, Q. H. Nguyen, and A. Cronje, "Fantom: A scalable framework for asynchronous distributed systems," 10 2018.

[53] M. K. Aguilera, *ApPLIED@PODC - Apply or Perish*. ACM, 7 2018.

[54] M. ter Beek, B. Re, M. Viroli, R. Anane, and R. Bahsoon, "Session details: Distributed systems: Ccs track," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM, 4 2018.

[55] V. C. Pujol, A. Morichetta, I. Murturi, P. K. Donta, and S. Dustdar, "Fundamental research challenges for distributed computing continuum systems," *Information*, vol. 14, no. 3, pp. 198–198, 3 2023.

[56] Z. Lu, W. Yu, P. Xu, W. Wang, J. Zhang, and D. Feng, "An ntt/intt accelerator with ultra-high throughput and area efficiency for fhe," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*. ACM, 6 2024, pp. 1–6.

[57] L. Nenov, "Reinforcement learning for key management in distributed systems," in *2024 32nd National Conference with International Participation (TELECOM)*. IEEE, 11 2024, pp. 1–5.

[58] C. Tang, Z. Wang, X. Zhang, Q. Yu, B. Zang, H. Guan, and H. Chen, "Many faces of ad hoc transactions," *Communications of the ACM*, vol. 68, no. 4, pp. 71–80, 3 2025.

[59] Z. Jiang, Z. Wang, H. Lan, C. Tang, H. Ding, L. Wang, S. Zou, Z. Wei, Y. Liu, X. Yu, Y. Ren, G. Li, and H. Chen, "Grewriter: Practical query rewriting with automatic rule set expansion in gaussdb," *Proceedings of the VLDB Endowment*, vol. 18, no. 12, pp. 4991–5003, 9 2025.

[60] M. Hauswirth, D. L. Phuoc, and J. X. Parreira, *Semantic Streams*. Springer New York, 12 2018, pp. 3419–3425.

[61] *Convergence conditions for Persidskii systems*, 6 2021.

[62] D. Fernando, M. A. Rodriguez, and R. Buyya, "ianomaly: A toolkit for generating performance anomaly datasets in edge-cloud integrated computing environments," in *2024 IEEE/ACM 17th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 12 2024, pp. 236–245.

[63] C. Mayer, R. Mayer, S. Bhowmik, L. Epple, and K. Rothermel, "Hype: Massive hypergraph partitioning with neighborhood expansion," 10 2018.

[64] S. Dahdal, F. Poltronieri, A. Gilli, M. Tortonesi, R. Fronteddu, R. Galliera, and N. Suri, "Roamml: Distributed machine learning at the tactical edge," in *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*. IEEE, 10 2023, pp. 33–38.

[65] R. Malik, C. Ramachandran, I. Gupta, and K. Nahrstedt, "Samera: a scalable and memory-efficient feature extraction algorithm for short 3d video segments." in *IMMERSCOM*, 2009, p. 18.

[66] *A Generic Review of Messenger Application: WeChat and WhatsApp*, vol. 1, no. 1, 12 2020.

[67] U. Matter, *Distributed Systems*. Chapman and Hall/CRC, 6 2023, pp. 81–96.

[68] R. Walther, C. Weinhold, P. Amthor, and M. Roitzsch, "Multi-stakeholder policy enforcement for distributed systems," in *Proceedings of the 10th International Workshop on Container Technologies and Container Clouds*. ACM, 12 2024, pp. 7–12.