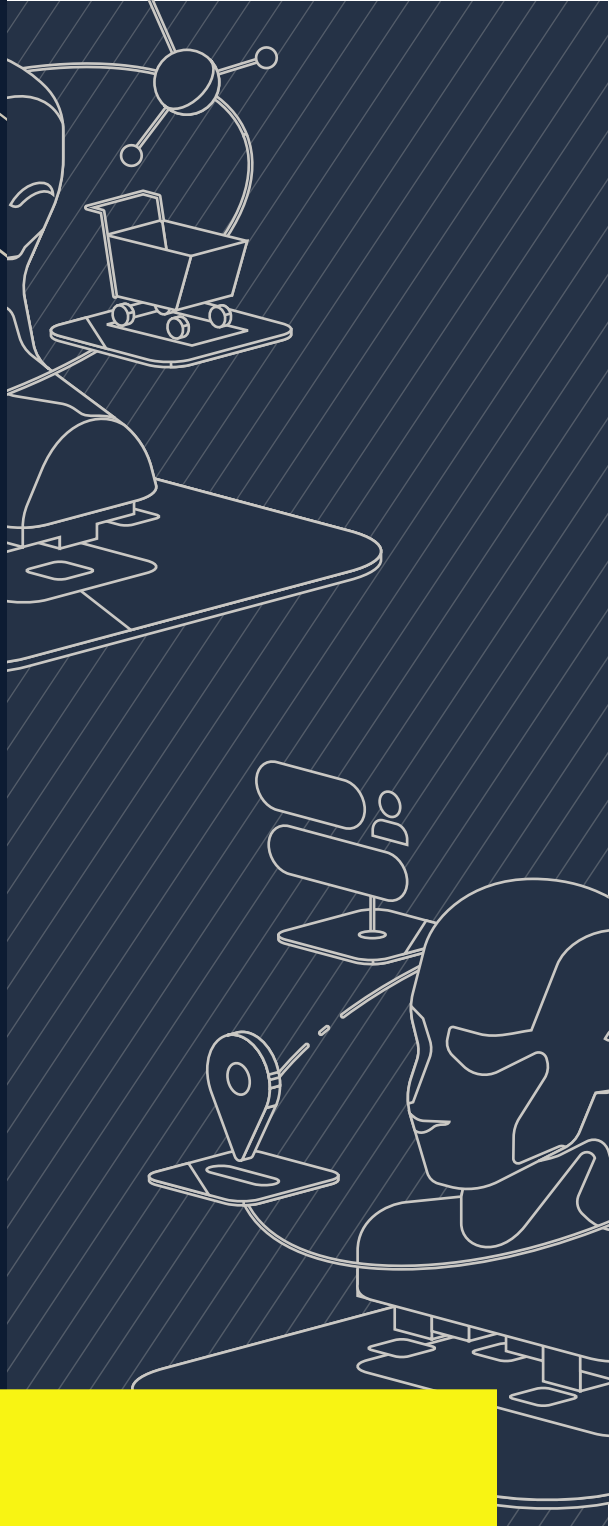
Aerospike



BENCHMARK REPORT

From millions to billions: Benchmarking identity graphs with Aerospike Graph

Contents

Executive summary **3**

Benchmark workload **3**

Data model 4

Query patterns 5

Short read (SR) queries 5

Short write (SW) queries 5

Deployment architecture **5**

Infrastructure setup 6

Key performance metrics **7**

Test methodology **7**

Test environment 7

Bulk data loading process 8

Benchmark execution 8

Results **8**

Latency vs. data volume 8

Read queries 9

Write queries 10

Latency over time 11

Throughput scalability 12

Bulk load cost-performance 13

Infrastructure costs vs. data volume 14

Summary and conclusion **15**

Appendix **16**

Software versions 16

Hardware and sizing 16

General test information 16

Aerospike Database 16

Aerospike Graph Service 16

Load generation server (LGS) aka Tinkerbench 16

Executive summary

In this benchmark, Aerospike Graph demonstrates its ability to power real-time, large-scale graph applications with unparalleled scalability, performance, and cost efficiency.

Conducted across a 100x range of data sizes, our study highlights three key benefits of Aerospike Graph for developers, architects, and graph database operators for identity graph use cases:

1. **Predictable query latencies:** Query latencies remain stable as data grows, with minimal variance between p50 (typical case) and p999 (most stringent), ensuring consistent and predictable performance at scale.
2. **Throughput scalability:** Throughput increases linearly with the number of compute nodes, from 22K QPS on a single node to 600K QPS with 32 nodes, demonstrating linear horizontal scalability.
3. **Cost-effective scaling[1]:** Infrastructure costs decrease as scale increases, dropping from $10/GB at 200GB to $5/GB at 20TB.

These results demonstrate that Aerospike Graph can effectively store, query, and scale to vast graph datasets without performance bottlenecks or prohibitive costs. Its ability to sustain high-throughput workloads while maintaining predictable performance at scale makes it a compelling choice for organizations managing large, growing, and complex graph datasets with demanding performance requirements.

This whitepaper provides detailed insights into Aerospike Graph's benchmark results, testing methodologies, and costs, enabling organizations to make informed decisions about their graph database infrastructure for high-performance, large-scale applications.

Benchmark workload

Identity graphs are critical components in customer data platforms, particularly in AdTech and e-commerce, where they enable identity resolution and cross-identifier targeting. However, the lack of standardized graph benchmarks—by comparison, for NoSQL, there is YCSB, and for relational, there is TPC—hinders the evaluation of graph database performance and scalability in these use cases.

The benchmark workload consists of a graph data model and queries derived from real-world identity graph use cases. It reflects the actual usage pattern of leading AdTech organizations running identity graphs in production.

The workload consists of:

- **A graph data model** representing user identities, households, devices, partners, cookies, and IP addresses.
- **A set of read and write queries** that evaluate the graph database's ability to retrieve, update, and manage identity relationships.

A defining characteristic of this workload is its graph structure and query patterns. The datasets consist of many sparse graphs—a collection of small, independent, or weakly connected subgraphs. As a result, read and write queries primarily interact with localized sections of the graph rather than performing deep traversals across the entire dataset.

[1] Assuming 1 year commit pricing. See section on infrastructure costs vs. data volume for details

Data model

The data model represents an identity graph, incorporating various entities for identity resolution and cross-identifier targeting:

- **GoldenEntity:** Represents a unique user profile
- **Household:** Groups related users or devices
- **Device:** Includes various device types (e.g., mobile, desktop, CTV)
- **Partner:** Represents data providers
- **Cookie:** Stores cookie information
- **IpAddress:** Represents IP addresses associated with users or devices

The image below illustrates this data model.

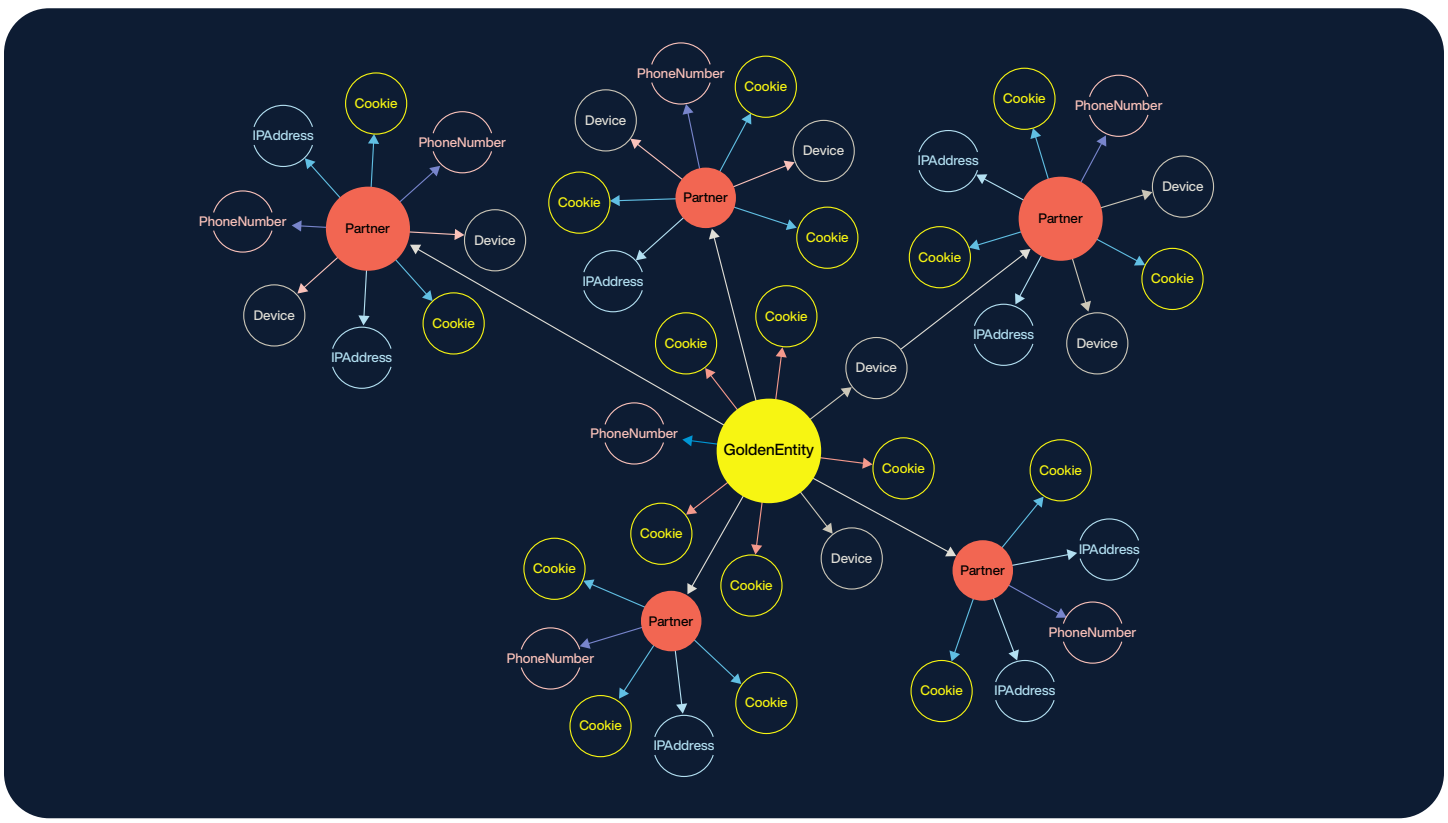


Figure 1: Identity graph benchmark data model.

Query patterns

Below is a brief description of the queries used in this workload.

Short read (SR) queries

The short read queries focus on read operations, querying relationships between users, devices, and tracking identifiers.

- **SR1:** Lookup partner details for a GoldenEntity.
- **SR2:** Start with GoldenEntity and get all signals that a specific partner has provided.
- **SR3:** Fetch all signals (including ones provided by partners) for a GoldenEntity.
- **SR4:** List all tracking identifiers (cookies, device IDs, IP Addresses, etc) associated with a device.
- **SR5:** Multi-hop traversal to find all the connected devices for a GoldenEntity based on a given device ID.

Short write (SW) queries

The short write queries focus on updating the graph with new entities or merging data, such as registering new partners or updating vertex properties.

- **SW1:** Merge two GoldenEntities into a household.
- **SW2:** Register a new partner and associate it with existing entities.
- **SW3:** Update the timestamps for signal vertices connected to a partner.
- **SW4:** Modify the properties of a GoldenEntity.
- **SW5:** Update the score property on edges to partners based on a specific type.

Deployment architecture

Aerospike Graph's architecture is composed of three layers:

- **Application layer:** Aerospike Graph provides developers with the Gremlin graph query language interface. Aerospike Graph supports Gremlin out of the box. (For those in the know, it serves as a first-class citizen in the TinkerPop ecosystem.)
- **Compute layer:** The Aerospike Graph Service (AGS) is responsible for processing graph queries and computations. If users want more performance, they can simply add more AGS nodes/instances independent of the data/storage layer. This is one way Aerospike Graph keeps infrastructure costs down.
- **Storage layer:** Aerospike Database manages and persists graph data. Aerospike employs its proprietary Hybrid Memory Architecture (HMA), where data is persisted on SSDs/NVMes but is accessed at in-memory speeds. This infrastructure efficiency is the second major way Aerospike Graph keeps costs down.

Applications interact with one or more AGS instances, enabling scalable query execution. The diagram below illustrates the overall solution architecture.

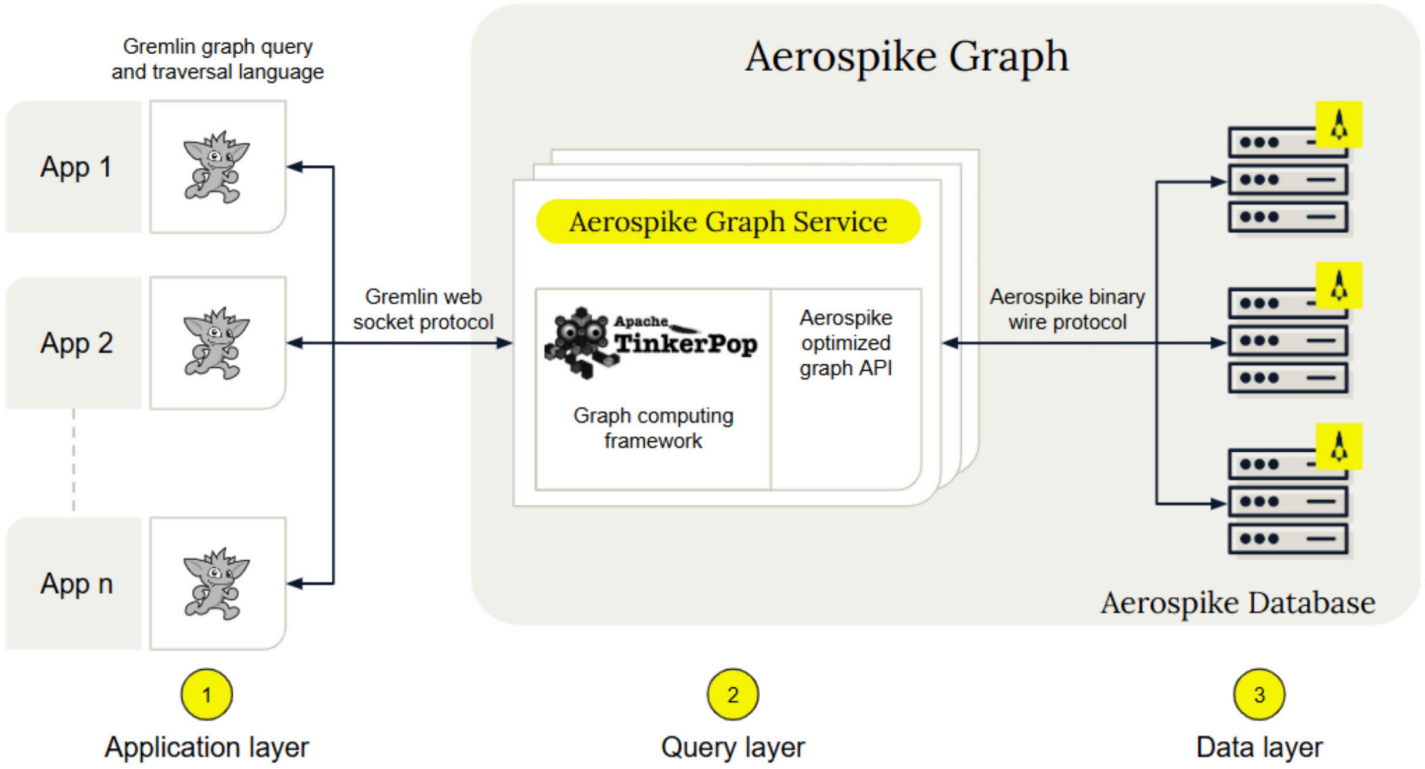


Figure 2: Aerospike Graph architecture scales compute and storage independently for maximum efficiency.

Infrastructure setup

The application layer executed the Tinkerbench application for benchmarking.

Aerospike was deployed with the primary index in RAM, and data was stored on NVMe SSDs using HMA. The replication factor (RF) was set to two to ensure fault tolerance. (Note: RF2 is a standard Aerospike practice.) Clusters were sized such that after the initial data load for each dataset size, 50% of the storage space was left unused[2].

The instance types and hardware specifications for the Aerospike Database, AGS, and Tinkerbench application clients for each of the tests are described in the Appendix.

[2] Capacity planning guide | Fragmentation Considerations

Key performance metrics

To comprehensively evaluate Aerospike Graph's performance, we focused on several key metrics that address the primary concerns of graph database users: performance, scalability, cost-effectiveness, and operational stability. These metrics are crucial because they directly impact the ability of organizations to manage large-scale graph datasets efficiently.

The following metrics were selected for their relevance to real-world use cases and their ability to provide a thorough assessment of Aerospike Graph's capabilities:

1. **Latency vs. data volume:** This metric measures how query response times (p50 to p999) scale with increasing data sizes. It helps assess whether the system maintains low-latency performance as datasets grow.
2. **Throughput scalability:** This metric evaluates the system's ability to sustain high query throughput as workload demands increase with horizontal scaling. Throughput scalability is critical for handling large volumes of queries in real-time applications.
3. **Data ingestion efficiency:** This metric assesses bulk load cost performance across different dataset sizes, providing insights into the efficiency of data ingestion processes.
4. **Cost efficiency vs. data volume:** This metric analyzes how infrastructure costs scale with data volumes, helping organizations plan for future growth.

These metrics were evaluated using a structured methodology outlined in the following sections, providing a comprehensive view of Aerospike Graph's performance capabilities.

Test methodology

Our benchmark utilized three dataset sizes defined by scale factors (SF): 10M, 100M, and 1B. These SFs were chosen to reflect typical growth patterns in identity graph datasets, from small to large-scale deployments. The datasets are characterized by their input CSV file size, edge count, and vertex count, as shown in Table 1.

These SFs allow us to assess how Aerospike Graph performs across a range of data sizes, from small to large datasets.

Scale factor (SF)	CSV file size (GB)	Edge count	Vertex count
10M	35	373M	383M
100M	358	3.73B	3.83B
1B	3,600	37.2B	38.3B

Table 1: Reliability and availability levels commonly required for enterprise applications.

Test environment

The benchmark tests were conducted on the Google Cloud Platform (GCP), utilizing specific instance types for the Aerospike Database, AGS, and Tinkerbench application clients. Detailed hardware specifications are provided in the appendix.

Each dataset was provisioned with an appropriately sized Aerospike cluster based on Aerospike's sizing guidelines (see Appendix for details on cluster sizes for each dataset).

Bulk data loading process

The Aerospike Graph Bulk Loader was used to ingest large-scale datasets. This tool facilitates loading large volumes of graph data from external sources into the Aerospike Database. It is distributed as a JAR file and runs on an Apache Spark cluster, leveraging distributed processing to optimize data ingestion.

The bulk loading workflow followed these steps:

1. **Create the Spark cluster:** Since the benchmark was run on GCP, Dataproc was used as the managed Spark service. The cluster size was adjusted based on the dataset scale (see Appendix for details).
2. **Submit the bulk load job:** The bulk load process was initiated by submitting a Spark job using the Spark submit command. This command processes CSV files and ingests data into the Aerospike Database.

Benchmark execution

To execute queries and collect performance metrics, we used Tinkerbench, a flexible benchmarking harness designed for Tinkerpop-based graph databases. Tinkerbench allows us to simulate varying workload intensities by controlling throughput and concurrency, making it ideal for evaluating real-world performance scenarios.

We executed a predefined set of read (SR) and write (SW) queries, capturing latency (p50 to p999) and throughput (QPS) metrics. For a description of these queries, refer to the Query patterns section.

Results

This section presents the findings from our benchmarking study, focusing on key performance metrics (mentioned in the "Key performance metrics" section above) that highlight Aerospike Graph's capabilities in handling large-scale identity graphs.

Latency vs. data volume

Our tests demonstrated that Aerospike Graph maintains stable query latencies across varying data volumes. The results show that latency remains relatively stable even as data size increases, with minimal variance between typical (p50) and most stringent (p999) scenarios. This can help with predictability for any given dataset for any SLA and across a wide range of dataset sizes.

There are two key takeaways:

1. **First,** for a given data size, latency is stable from the median p50 all the way to p999, which is well bounded, i.e., it does not "hockey stick up." Note: There is variance in the generated data, so some of the higher tail latencies are the result of queries that work with bigger subgraphs.
2. **Second,** as data size increases by one and two orders of magnitude, latencies hardly increase at all (for write queries), or, at most, double (for some read queries). They do not increase proportionally - which is a very good thing.

Read queries

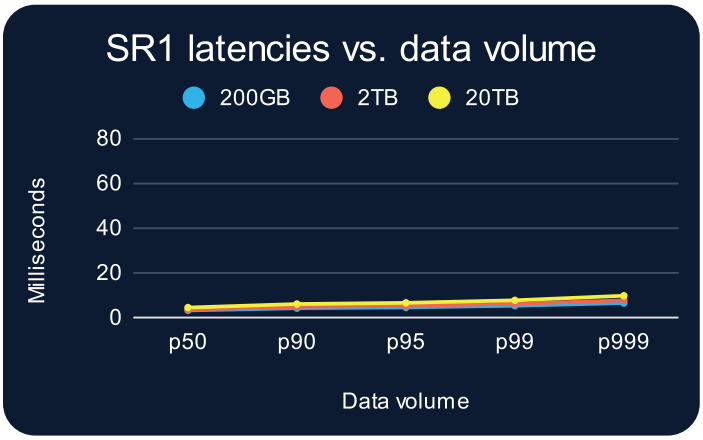


Figure 3 : Find all the partner vertices connected to a GoldenEntity.

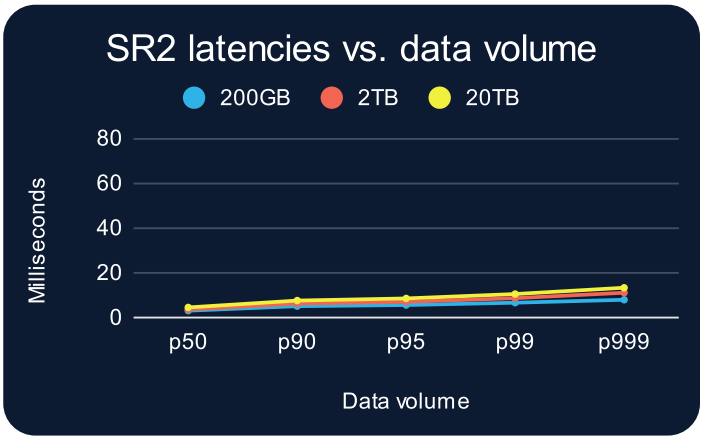


Figure 4: Start with GoldenEntity and get all signals provided by a partner.

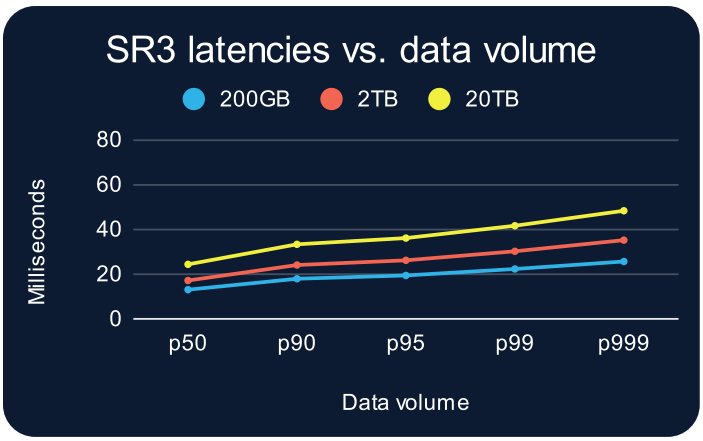


Figure 5: Fetch all signals (including ones provided by partners) for a given GoldenEntity.

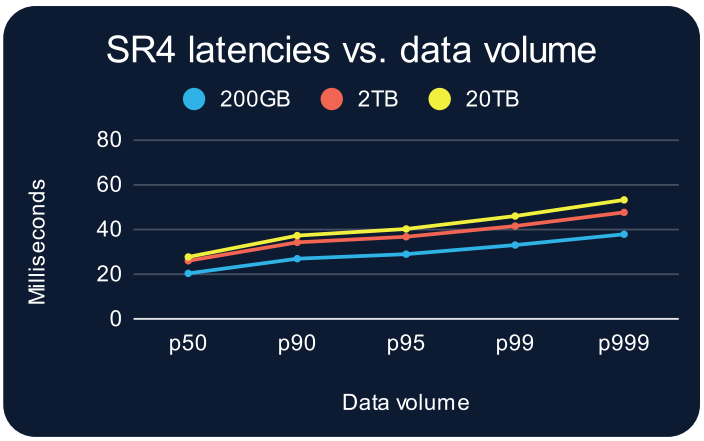


Figure 6: List all tracking identifiers (cookies, device IDs, IP addresses, etc.) associated with a device.

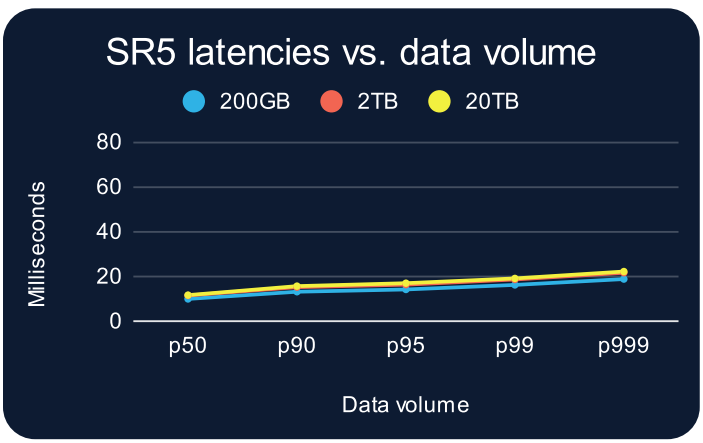


Figure 7: Three-hop traversal to find connected devices for a GoldenEntity.

Write queries

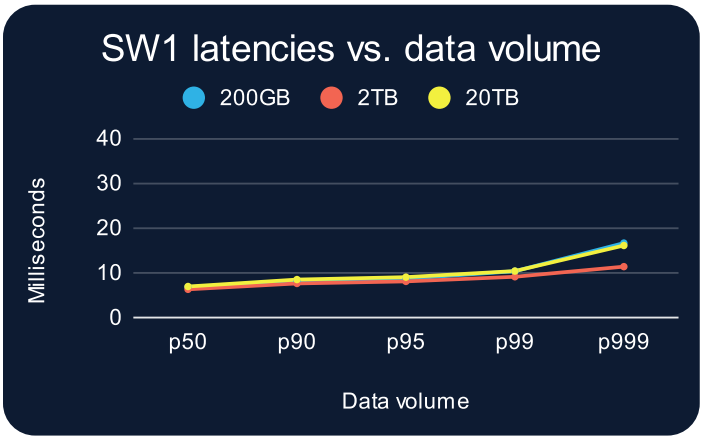


Figure 8: Merge two GoldenEntities into a household.

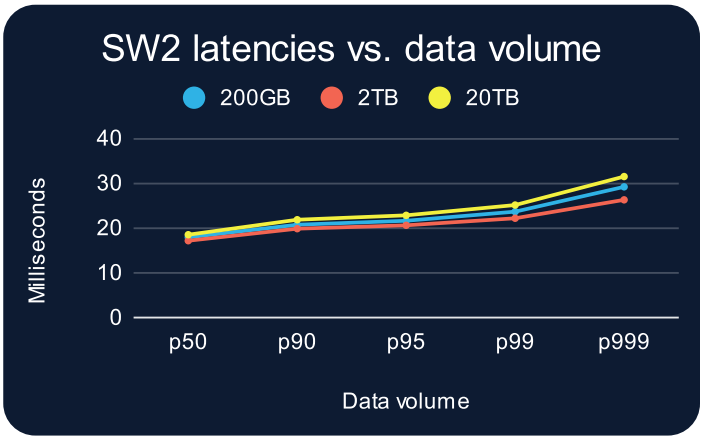


Figure 9: Register a new partner and associate it with existing entities.

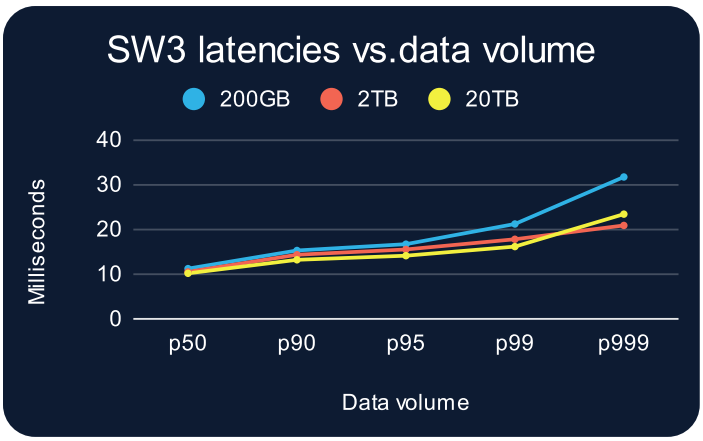


Figure 10: Update the timestamps for signal vertices connected to a partner.

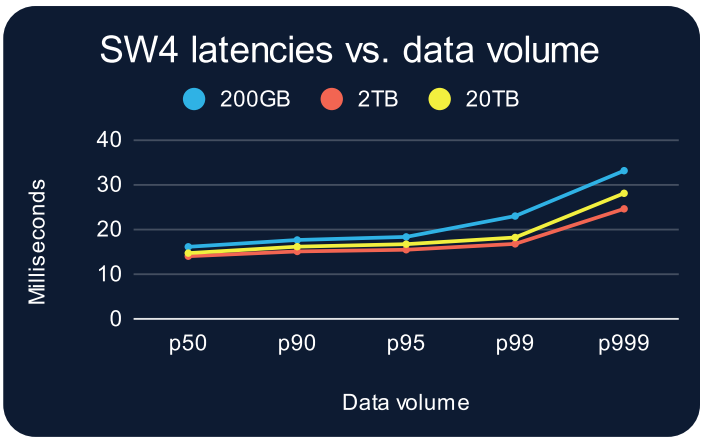


Figure 11: Modify the properties of a GoldenEntity.

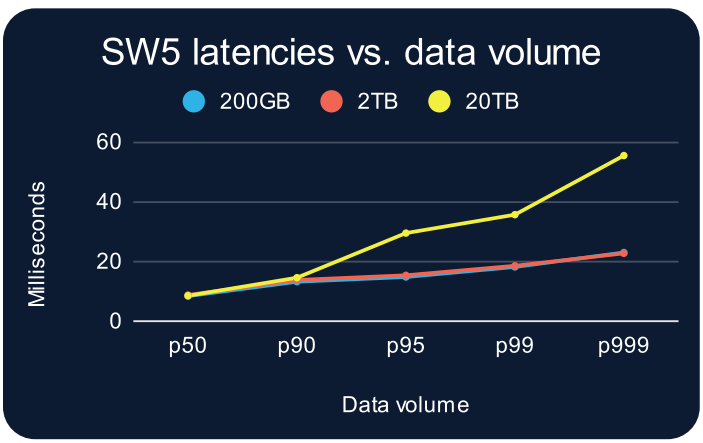
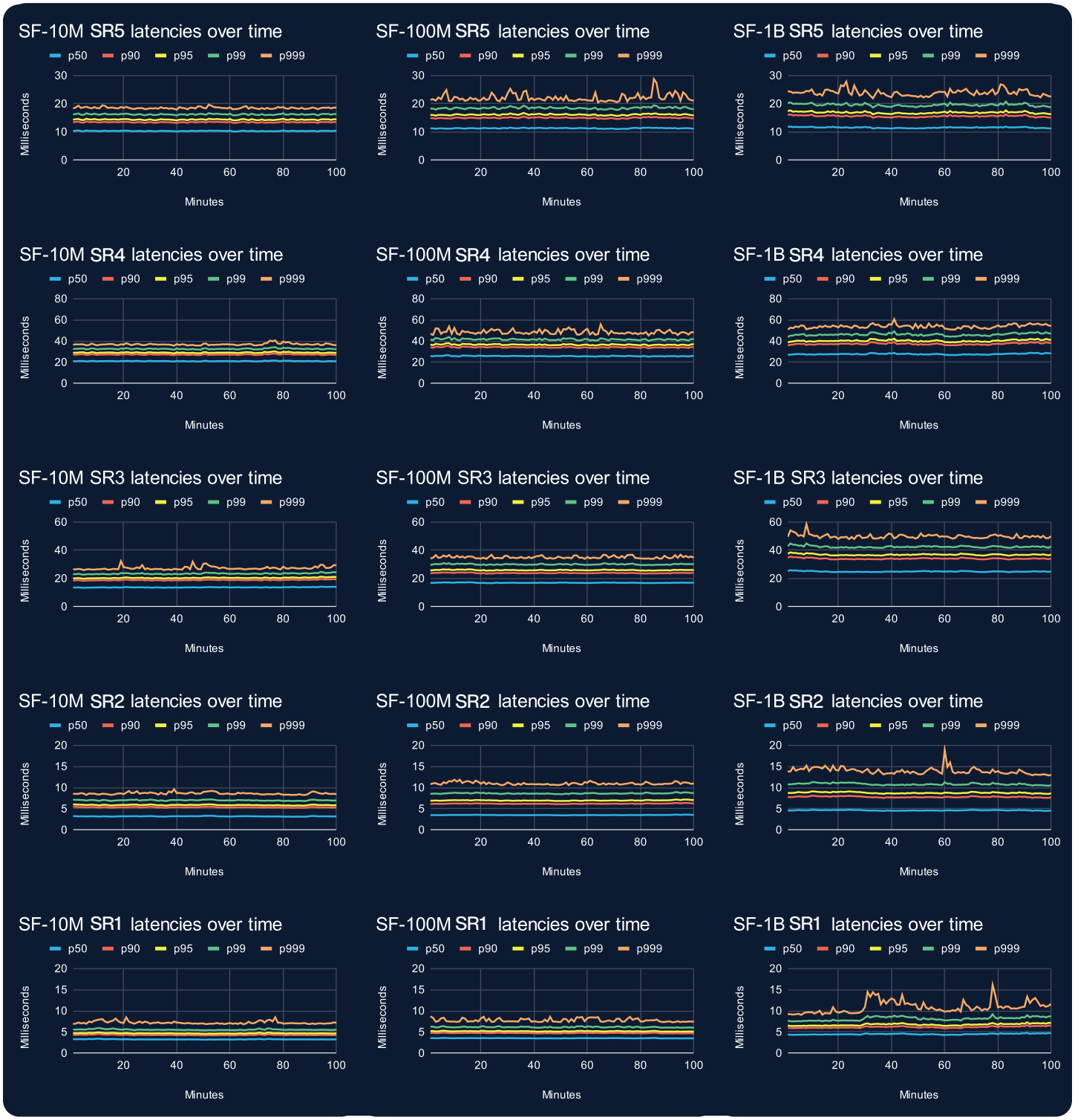


Figure 12: Update the score property on edges to partners based on a specific type.

Latency over time

Short read (SR) latencies over time are remarkably stable across all short read queries, with even p999 latencies avoiding major spikes.



Figures 13.1 - 13.9: AGS short read latencies remaining very stable over time regardless of the data size or query.

Throughput scalability

Aerospike Graph demonstrated linear throughput scalability with the addition of AGS nodes. This means that as workload demands increase, the system can handle higher query volumes without performance degradation.

The SF1B cluster was selected for this test as it provides the highest overall disk throughput capacity, ensuring the disk I/O throughput is not the bottleneck.

To isolate the impact of AGS scaling, the Aerospike cluster size was held constant, while AGS instances were incrementally scaled from 1 to 32.

The chart below shows that throughput increased from 22K QPS with a single AGS instance to over 600K QPS with 32 AGS instances—achieving perfect linear scalability.

Note: This result is expected due to Aerospike Graph's architecture, where each AGS node independently processes queries without requiring inter-node coordination. Because of this shared-nothing design, adding more AGS nodes directly increases throughput without introducing contention.

This perfect linear scaling continues until the Aerospike Database itself reaches saturation. At that point, scaling the Aerospike cluster horizontally further increases throughput, ensuring predictable performance growth even at extreme scales.

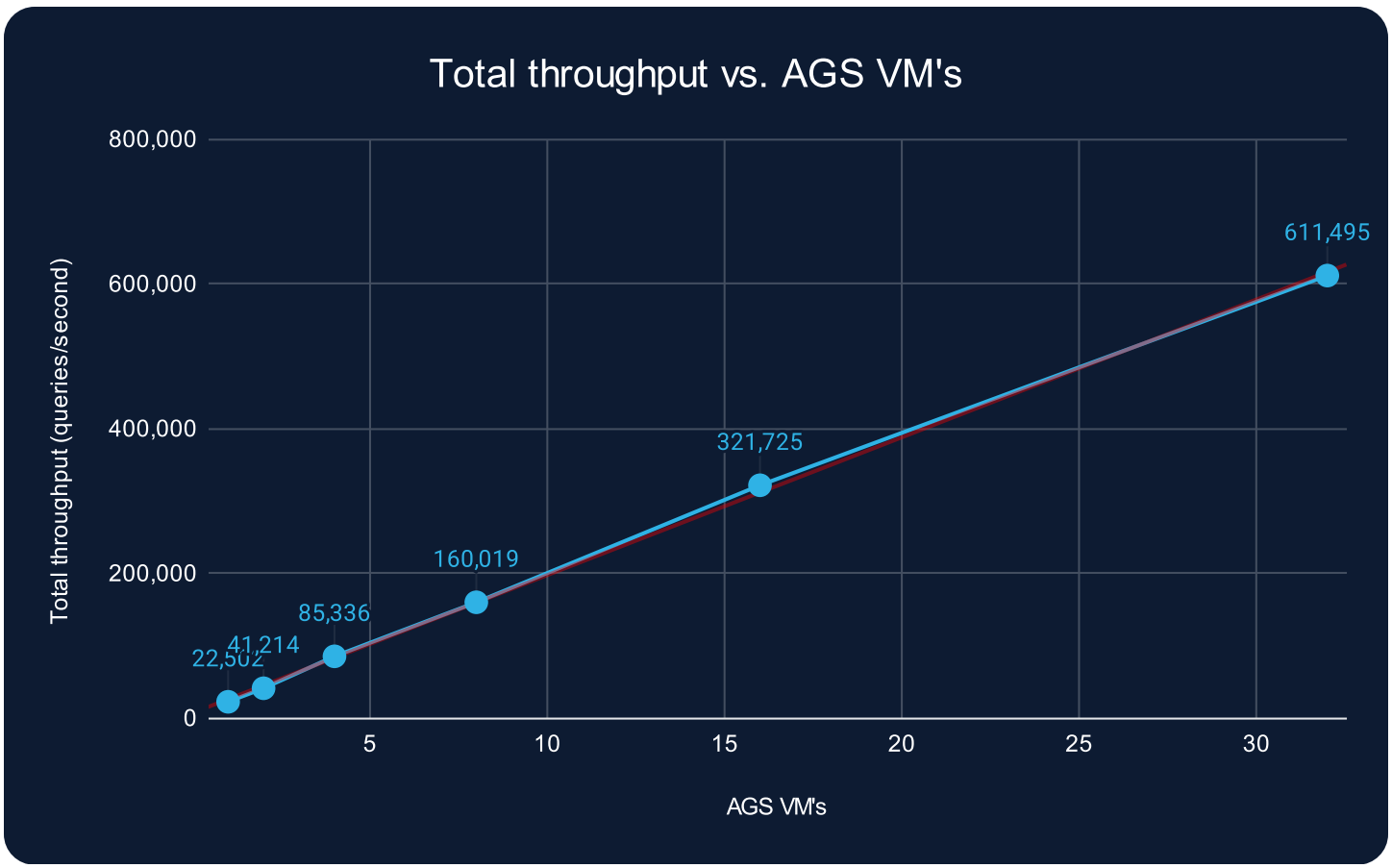


Figure 14: AGS linear throughput scalability as a function of VMs.

Bulk load cost-performance

Our tests evaluated the efficiency of bulk data loading across different dataset sizes. The results highlight the cost-performance trade-offs involved in optimizing bulk load processes.

- **Cost efficiency:** The cost per GB for bulk loading increased by only 2x from SF100M to SF1B despite a 10x increase in dataset size. This demonstrates controlled cost growth and scalability.
- **Bulk load times:** The table below summarizes bulk load times for each dataset size.

Scale factor (SF)	CSV file size (GB)	Dataproc instance type	Spark total memory (TB)	Dataproc workers	Bulk load time (hrs)
10M	35	n2d-highmem-8	1.875	30	3.56
100M	358	n2d-highmem-16	7.5	60	8.00
1B	3,600	n2d-highmem-64	35	70	31.81

Table 2: Bulk load times for each dataset size.

These metrics indicate that while bulk load times increase with dataset size, the cost per GP remains relatively stable (see Figure 15 below). This stability is crucial for managing operational expenses during large-scale data ingestion.

In fact, for bulk loading, Aerospike Graph offers several design trade-offs to optimize speed and cost-efficiency:

1. **Sizing the Aerospike cluster for data capacity[3]:** Ensure the cluster is large enough to accommodate all data by the end of the bulk load process.
2. **Over-provision disks for faster ingestion and IOPS[4]:** Allocating more disk resources increases IOPS (input/output operations per second), allowing faster data ingestion.
3. **Dynamically scale down for efficiency[5]:** Aerospike clusters can be scaled down. One approach is to temporarily scale up the cluster to speed up the initial bulk load, then scale it down afterward to match either
   - Query throughput (QPS) requirements
   - Total data volume
4. **Optimal sizing of the Spark cluster:** Increasing the number of workers and/or memory per worker improves bulk load performance until the Aerospike cluster reaches its I/O or CPU limits. Beyond this saturation point, adding more workers does not improve performance.

By employing the right strategy, bulk load times can be optimized while maintaining cost efficiency and operational flexibility.

In our tests, the cluster was sized for data capacity (option 1 above) and the number of Spark workers was selected based on the disk I/O capacity of the Aerospike cluster, ensuring maximal utilization of available I/O bandwidth.

Multiple factors influence bulk load performance, such as dataset size, Aerospike cluster capacity, and Spark worker configuration. A useful way to compare efficiency is to measure bulk load cost per GB of input data.

[3] Fits on disk in the aerospike cluster, with enough room for your resilience budget.
[4] Bulk loads are bottlenecked on the IOPS available in Aerospike.
[5] In a controlled fashion that allows for the cluster to be fully replicated before the node is removed.

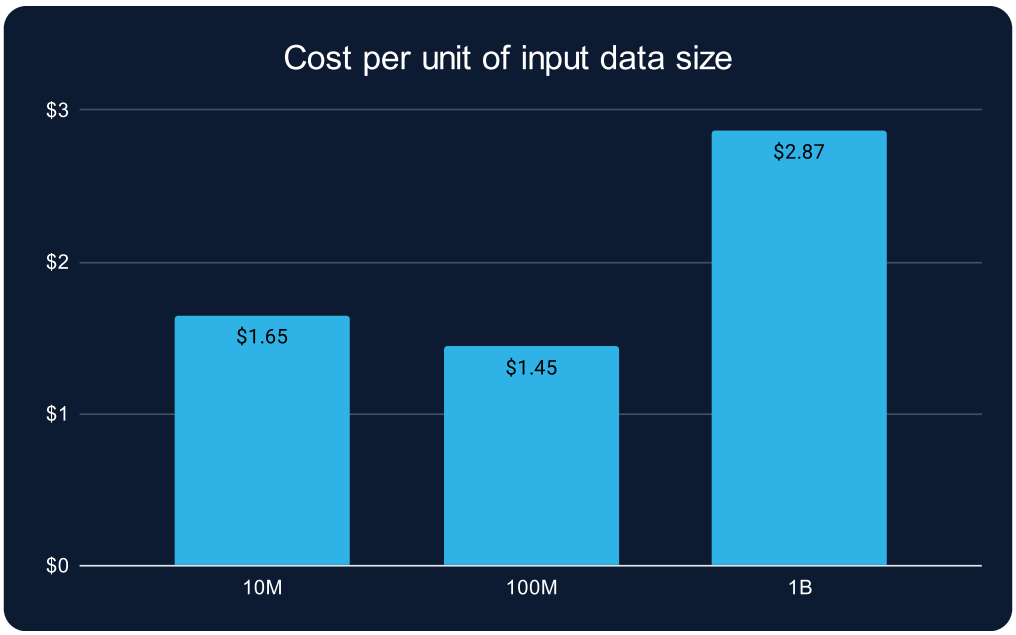


Figure 15: Scalability with controlled growth for cost per GB.

As shown in Figure 15, the cost per GB for SF100M vs. SF1B increases by only 2x, even though the dataset size grows 10x. This demonstrates scalability with controlled cost growth.

It is worth mentioning that the cost per GB for the SF10M and SF100M datasets remains comparable. This similarity is primarily due to the fact that the SF10M dataset was loaded on a cluster that was larger than necessary for its size. As a result, the cost efficiency for the smaller SF10M dataset was impacted, as resources were not fully utilized. In contrast, larger datasets like SF100M and SF1B benefit from more efficient cluster sizing, leading to better cost scaling as dataset sizes increase.

Infrastructure costs vs. data volume[6]

Predictable infrastructure costs are essential for large-scale deployments, enabling organizations to plan for future growth and manage expenses effectively. This section evaluates how infrastructure costs for Aerospike Graph scale with increasing data volumes.

To assess cost scaling, we created Aerospike clusters for each dataset size: SF10M (200GB), SF100M (2TB), and SF1B (20TB). The results show that infrastructure costs scale predictably with data size:

- **Cost efficiency at larger scale:** From 200GB to 2TB, costs increase by 5x despite a 10x increase in dataset size. This indicates improved cost efficiency at larger scales, as costs do not grow proportionally with data size.
- **Linear cost growth:** From 2TB to 20TB, costs grow in a perfectly linear fashion, demonstrating consistent scalability without unexpected overheads.

This linear growth ensures that Aerospike Graph remains cost-efficient as data volumes increase.

[6] GCP costs as published in the Google Cloud pricing calculator as of March 5th, 2025

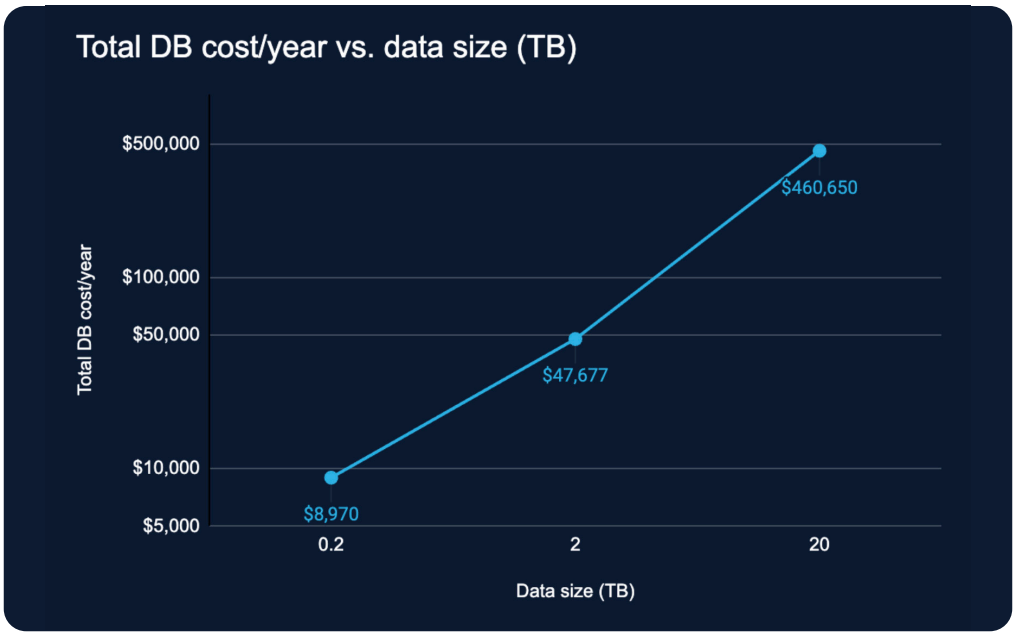


Figure 16: Aerospike Graph remains cost-efficient as data volumes increase.

The predictable cost scaling of Aerospike Graph has significant implications for organizations managing large datasets. It allows them to budget more accurately for infrastructure expenses, ensuring that resources are allocated efficiently. This predictability also supports strategic planning for future growth, as organizations can anticipate and prepare for the costs associated with expanding their datasets.

Summary and conclusion

This benchmark study demonstrates Aerospike Graph's capabilities in handling large-scale identity graphs, focusing on scalability, performance, and cost efficiency.

Key findings include:

- **Predictable low-latency performance:** Aerospike Graph maintains stable query latencies across varying data volumes, with minimal variance between typical (p50) and worst-case (p999) scenarios.
- **Linear throughput scalability:** Throughput scales linearly with the addition of AGS nodes, achieving up to 600K QPS with 32 nodes.
- **Cost-effective scaling:** Infrastructure costs decrease as scale increases, dropping from $10/GB to $5/GB as data grows from 200GB to 20TB.

The benchmark workload simulates real-world identity graph use cases, incorporating a data model with entities like users, households, devices, and partners, and queries that assess read and write operations. The results highlight Aerospike Graph's ability to efficiently store, query, and scale large graph datasets without performance bottlenecks or prohibitive costs.

This study provides a structured methodology for evaluating graph database performance, addressing key concerns for organizations managing complex graph datasets. The findings make Aerospike Graph a compelling choice for high-performance, large-scale applications in AdTech and related industries.

Appendix

Software versions

1. Aerospike version 7.1.0.9
2. AGS version 2.4.2

Hardware and sizing

General test information

Dataset	SF 10M	SF 100M	SF 1B
Input CSV size (GB)	35	358	3,600
User dataset size (TB)	0.219	2.306	23.35
Vertex count	373M	3.73B	37.2B
Edge count	383M	3.83B	38.3B

Aerospike Database

Instance type	n2d-highmem-8	n2d-highmem-16	n2d-highmem-64
Number of nodes	3	8	18
vCPU cores per node	8	16	64
RAM per node (GB)	64	128	512
NVMe per node (each NVMe is 375 GB)	2	4	24
Replication factor	2	2	2

Aerospike Graph Service

Instance type	n2d-standard-8	n2d-standard-8	n2d-standard-8
Number of nodes	1	1	1

Load generation server (LGS) aka Tinkerbench

Instance type	n2d-standard-32	n2d-standard-32	n2d-standard-32

2440 W. El Camino Real, Suite 100, Mountain View, CA 94040 | (408) 462-2376 | aerospike.com