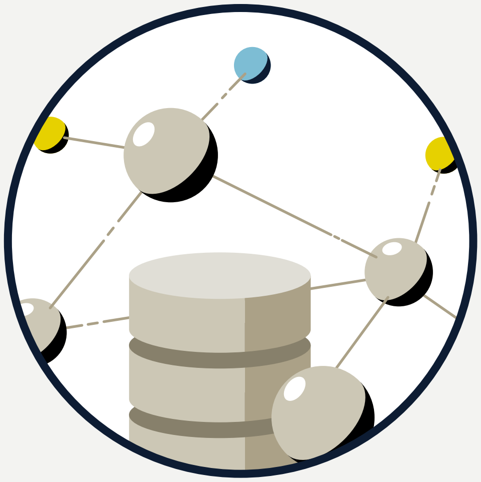
Aerospike



Graph database buyer's guide

Considerations for graph-centric deployments

Contents

Introduction to graph databases **3**

Analytical vs. transactional graph use cases **3**

Graph OLAP 3

Graph OLTP 4

Knowledge graphs, AI/ML, and graph RAG 4

Criteria for choosing a graph database **5**

Performance and scalability 5

Developer friendliness, community, and documentation 5

Total cost of ownership 5

Batch/streaming data import/export 5

Analytics 6

Visual exploration 6

IDE and query language support 6

Operational agility 6

Multi-model vs. native graph databases 6

Checklist for choosing an OLTP graph database 7

Vendor comparison **8**

Disclaimer 8

Neo4j 8

Aerospike 9

TigerGraph 10

Amazon Neptune Database 11

Conclusion 12

Appendix I: Graph terms and concepts **13**

Appendix II: Graph query languages **13**

Introduction to graph databases

Graph databases have emerged as a powerful tool for handling complex, interconnected data for a variety of use cases. The value of graph databases is that the relationships between data points (or edges) are as useful and valuable as the data points (or vertices) themselves. Given their ability to query, traverse, and process "connected data," graph databases are especially well-suited for use cases involving identity resolution, network analysis, recommendation engines, customer 360, fraud detection, and more.

While first-generation graph database implementations were largely about data visualization and discovery, the role of graph databases has expanded to include data modeling and transactional systems. More recently, artificial intelligence (AI) and machine learning (ML) teams and graph vendors are exploring the role graph databases can and will play in AI/ML data pipelines.

There are many factors to consider when choosing a graph database.

Scale: What is the expected scale and throughput of your data? Will you measure it in gigabytes, terabytes, or hundreds of terabytes as well as transactions per second? The graph database you choose must be suited to the scope and performance of your data and ready for unexpected spikes of activity.

Static vs. dynamic data: Are your datasets relatively static over time, or are they dynamic and changeable? Some graph database use cases are analytical in nature while other use cases are more transactional – with frequent reads, writes, updates, and deletes – where transaction throughput, query latencies, uptime, and other operational factors must be considered.

Cost: Are your graph database and data infrastructure costs predictable as data volumes, transaction throughput, and the number of concurrent users rise over time? Graph databases that rely on all data being in memory may not scale linearly – in terms of performance and cost - especially as the size of datasets approaches the amount of available memory.

Analytical vs. transactional graph use cases

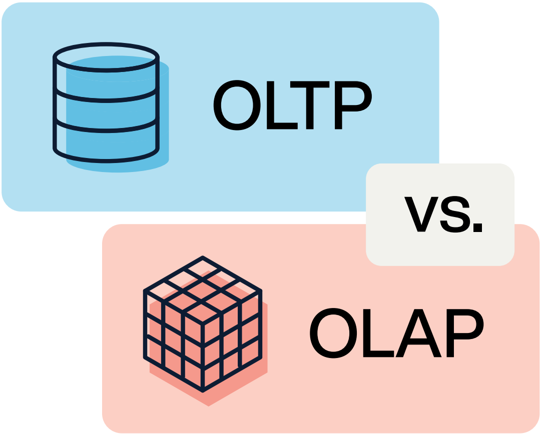
There is an increasing distinction between using graph databases for analytical purposes (graph OLAP) and those for transactional or operational purposes (graph OLTP). Let's examine the differences between these two categories.

Graph OLAP

Online analytical processing (OLAP) is typically used to perform multi-dimensional data analysis, deriving meaning through calculations and visualizations of aggregated historical data. Because of the nature of analytics, OLAP applications may take longer to finish complex queries on highly connected datasets.

Graph databases can deliver acceptable response times by keeping the entire graph dataset in memory. However, this approach can limit the size of the datasets and the scalability of the application.

Graph OLAP use cases often focus on interactive analysis or exploration by analysts where the query latency can be within a few seconds or minutes. There is usually a visualization component to these use cases that can be useful for data science and business intelligence (BI) teams.



Data volumes can range from hundreds of gigabytes to terabytes. Analytical graph datasets often follow the pattern of traditional OLAP in that datasets are fairly static and updated periodically (daily or weekly).

Traditional graph OLAP use cases often rely on the memory space of a single server or workstation to deliver acceptable response times for analysts and, therefore, rely on scale-up strategies (e.g., adding more memory and CPU). However, as data volumes grow or the number of concurrent users expands, customers often find that a distributed, scale-out approach is required.

Here are some examples of analytical (OLAP) graph use cases.

Interactive analysis of buyer behavior: Graph OLAP databases allow real-time exploration of buyer patterns, related book purchases, and author connections. Examples: Analyzing book sales and average prices and identifying key opinion leaders.

Pricing analysis: Graph OLAP databases facilitate pricing analysis by considering relationships between products, competitors, and market trends. Examples: Optimizing pricing strategies, competitive analysis, and dynamic pricing.

PageRank algorithm for key opinion leaders: Graph OLAP databases identify influential nodes (e.g., opinion leaders, influencers) using algorithms like PageRank. Examples: Finding influential users, recommending content, and enhancing marketing strategies.

Graph OLTP

Online transaction processing (OLTP) use cases are often characterized by high volumes of short transactions and fast query processing. OLTP is typically used for data ingest and retrieval in transaction-oriented applications. A real-time operations graph platform serves use cases where the graph query results are needed in under a second or even a few milliseconds. They may serve thousands or even millions of concurrent users or devices. For example, graph OLTP use cases like identity graphs in AdTech and online fraud detection in banking require sub-10 millisecond response times.

Graph OLTP requires both a scale-up (vertical) and scale-out (horizontal) architecture using industry-standard server clusters or clustered cloud instances. Data volumes are easily in the tens to hundreds of terabytes. Throughput requirements - with reads and writes - might be in the tens of thousands to hundreds of thousands of transactions per second.

Here are some examples of transactional/operational (OLTP) graph use cases.

Identity graphs: Identity graphs are used in ad tech in response to the impending loss of third-party browser cookies, which complicates identity resolution. Instead, large amounts of relevant data is used to build and train a machine learning model that is then used to create an identity graph.

Recommendation engines in e-commerce: Graph databases power personalized product recommendations by analyzing user behavior, preferences, and connections. Examples include LinkedIn, Amazon, and Wish.com.

Real-time fraud detection: Graphs reveal hidden relationships and patterns in transaction data, aiding fraud detection. Examples: Identifying suspicious connections, money laundering, and engaging in payment or credit card fraud.

Knowledge graphs, AI/ML, and graph RAG

Knowledge graphs are an increasingly popular means of deploying graph technology in the enterprise and have growing applicability in AI/ML data pipelines. The explosive growth and interest in generative AI (GenAI) and related ML programs have focused some attention on the role of graph databases in AI. Graph databases can serve several functions in an AI/ML data pipeline. Graph databases can help model different entities (users, customers, households), feed data to train ML models, and provide graph algorithms to ML use cases (PageRank, similarity matching, community detection).

Graph databases also have a role to play with large language models (LLMs) and retrieval augmented generation (RAG) which can improve LLM accuracy with more context. This is becoming known as Graph RAG and functions as an extension of knowledge graphs. This dimension of graph databases is growing and evolving rapidly, so a full treatment of this topic is beyond the scope of this paper.

Criteria for choosing a graph database

The functional criteria for evaluating graph databases vary depending on the use case, data volumes, and business needs. However, as the market matures, the top criteria for operationalizing graph databases in enterprise computing environments become clearer.

Performance and scalability

Achieving performance at scale with graph data is a tricky problem that requires strong consideration. With datasets under 500 gigabytes, most graph databases can deliver acceptable performance. Achieving optimum performance becomes an exercise in optimizing queries and using available product features (indexes, caching, tunable parameters, etc.) to improve response time.

Achieving adequate performance for operational use cases such as identity graphs, fraud detection, and mass personalization - where data volumes of multiple terabytes or more are becoming increasingly common - is a more intractable problem.

Look for a graph database designed from the ground up for extreme scale. It should have a distributed architecture, scale-up and scale-out capabilities, high-performance storage options, separate scaling for compute and storage, and advanced index/secondary index features.

Developer friendliness, community, and documentation

The degree to which each solution facilitates application development is a function of its engine and API, data model, and query language. Being easy to work with is hard to quantify, but most of the leading graph databases have powerful graph query languages (Cypher, Gremlin, GSQL, GQL) that give developers the tools they need to build great graph applications.

User communities offer support and access to resources, answer questions, and share knowledge and best practices, while the quality and quantity of documentation are important for onboarding and retaining users.

Total cost of ownership

The total cost of ownership (TCO) of a graph application involves the license and infrastructure cost, which can vary widely between vendors and run-time requirements. Technical debt and the cost of scaling can also be a consideration. If you go down the path with a single vendor, do you have a plan for when your data volumes double or grow exponentially? You must factor into your calculations the possibility of steady increases in data and transaction volumes and the potential for unexpected levels of success and scale.

Batch/streaming data import/export

Graph databases routinely need to import data from other systems in formats that are not graph-native, which means data must be extracted, transformed, and loaded (ETL) and then imported en masse. Even for graph data, importing in batches is essential, as it saves effort and ensures a streamlined and repeatable process. Reliability and options for logging and resuming when errors occur during the process are key.

Analytics

Graph algorithms used for analytics, like the PageRank algorithm, traverse paths in a graph to evaluate node properties and extract metrics. This provides an advantage over graph processing frameworks external to the database. Vendors that offer graph algorithms out of the box (or through github, developer sites, etc.) can help save time and effort.

Visual exploration

Visual interfaces can enable ad hoc discovery by picking a starting point of interest and following connections. They also drive intentional exploration of a specific subset of a graph retrieved via graph queries or analytics. Navigating and exploring large graphs can be daunting, so visual exploration is part and parcel of many graph databases.

IDE and query language support

Integrated development environments (IDEs) are a huge factor in developer productivity. At the very minimum, graph databases should support popular IDEs via connectors, plugins, and libraries. Some graph databases incorporate their own custom IDEs, signifying an investment in onboarding users to produce graph-based applications.

Operational agility

Support for multiple environments, both on-premises and in the cloud, is a requirement for modern databases. This includes support for on-premises deployment on bare metal, virtual machines, or containers. Cloud deployments require capabilities ranging from user-managed container or virtual machine images to turnkey database-as-a-service (DBaaS) solutions that integrate with cloud vendors' core and extended services.

For larger-scale distributed deployments, you must consider how quickly and easily you can scale data, indexes, transactions, concurrent users, and network resources. Can you easily scale up and out? Can you easily add nodes and redistribute data? How readily can you replicate data to different locations and availability zones?

Multi-model vs. native graph databases

Often, the tradeoff between having one specialized database per application and managing those databases will tip the scale in favor of simplicity and reduced TCO. This becomes even more pronounced in times of economic downturn when organizations are looking to do more with less. That's why more organizations are turning to multi-model, multi-capable databases that can effectively and efficiently deal with different scenarios.

The native advantage

Native graph databases are explicitly designed to handle graph data structures efficiently. They excel in traversing relationships, making them the preferred choice for applications where relationship-centric queries are the primary focus. An example of this might be a social graph where you are connecting different users together and need to follow relationships like location, buying preferences, and demographic information. With optimized storage and querying mechanisms, native graph databases offer blazingly fast performance for specific use cases.

One of the key features contributing to this speed is index-free adjacency. In native graph databases, relationships between vertices are stored directly as pointers, allowing rapid traversal. When you need to move from one node to another, the database engine can do so with minimal disk I/O because it knows the exact location of related nodes. This index-free adjacency ensures near-instantaneous access to connected nodes and is a cornerstone of native graph database performance. This advantage can be negated as datasets grow beyond the capacity of a single machine, or when running in distributed environments. Some vendors provide different techniques to better scale graph databases across multiple machines.

The versatility of multi-model

Multi-model databases are chosen for their versatility. These databases support multiple data models within a single platform, accommodating a wide range of data types, including graph, document, key-value, vector, and more.

Multi-model databases are designed to handle the high demands of operational workloads, providing the necessary real-time transactional capabilities for applications with thousands or even millions of concurrent users, far surpassing traditional graph databases focused on static data sets.

One common assumption is that native graph databases inherently provide superior performance. While this can be true for certain query patterns, multi-model databases have made significant strides in improving their graph capabilities. Modern multi-model databases often incorporate efficient graph processing engines, making them competitive in terms of performance.

Checklist for choosing an OLTP graph database

✔ When evaluating graph databases for transactional use cases, make sure you consider the following criteria to determine if your graph database vendor solution can withstand the rigors of highly dynamic, mission-critical environments:

✔ Does the solution provide customizable high-performance query APIs?

✔ Can it deliver sub-second query latencies and optimizations to ensure those latencies for both single-hop and multi-hop queries?

✔ Can it deliver second-level data freshness (i.e., near real-time update)?

✔ Does your platform provide horizontal scaling with fault tolerance on terabytes of data?

✔ Can it scale storage and/or compute resources independently?

✔ Can your platform deliver sufficient throughput headroom (e.g., 1 million QPS [query per second])?

✔ Does it provide flexible data backfill from offline to online?

✔ Is it compliant with enterprise data security and privacy requirements?

Vendor comparison

Disclaimer

The following comparison is based on a high-level analysis of each of the leading vendors. This comparison uses input from publicly available information from the vendors themselves, feedback from customers and partners, and our experts. We have endeavored to be objective and fair in these appraisals. Each vendor mentioned offers strong, successful graph databases. Readers are advised to do their own research in their search for the ideal graph database for their needs. If you wish for additional information or clarification on any of the information within this buyer's guide, please contact us at [info@aerospike.com].

Neo4j

Neo4j is a pioneer of graph databases and graph use cases. Many organizations have adopted it to support increasing interest in exploring the power of connected data. With an early focus on graph analytics and visualization, Neo4j is among the strongest vendors for graph OLAP use cases. Neo4j was designed as a single-instance, in-memory database, which presents some challenges with scalability as data volumes grow or in transactional environments.

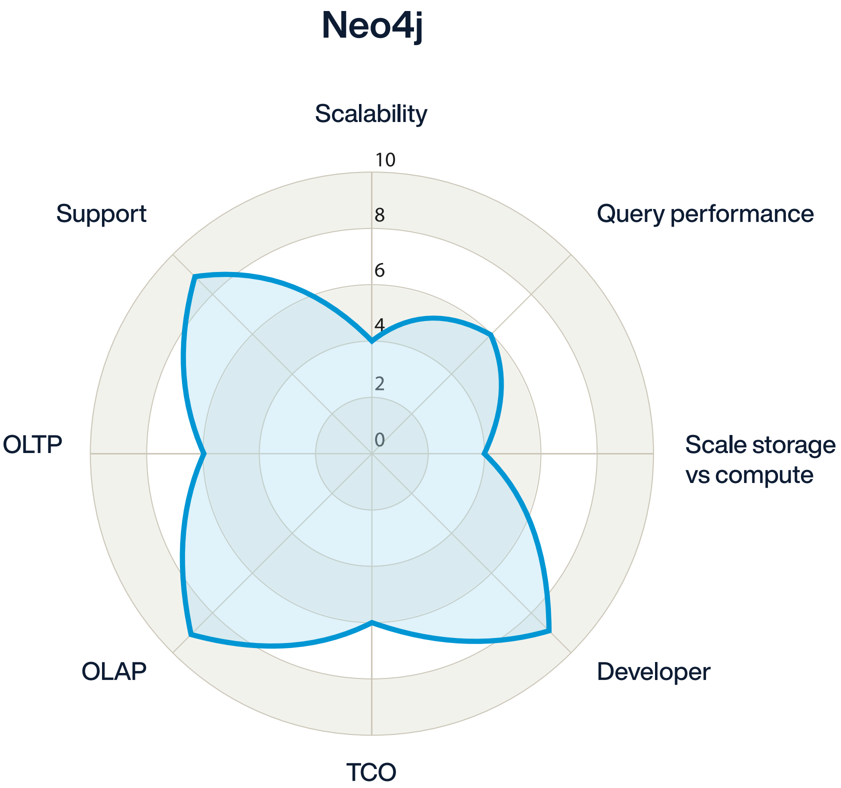
Deployed: On-prem, cloud (Neo4j Aura)

Query language: Cypher

Data model: Graph only

Use case focus: OLAP/analytics, visualization

Support: Neo4j support



Aerospike

Aerospike is a multi-model real-time database supporting key-value, document, vector, and spatial and graph data models. Aerospike delivers high transaction volumes at extremely low latencies on datasets ranging from gigabytes to petabytes. Aerospike Graph was introduced in 2023 to target the gap left by other graph vendors in the graph OLTP space. Aerospike can deliver transactions in a few milliseconds, even in multi-hop graph queries, and is unique in that it scales computing (graph queries and processing) with storage (real-time database).

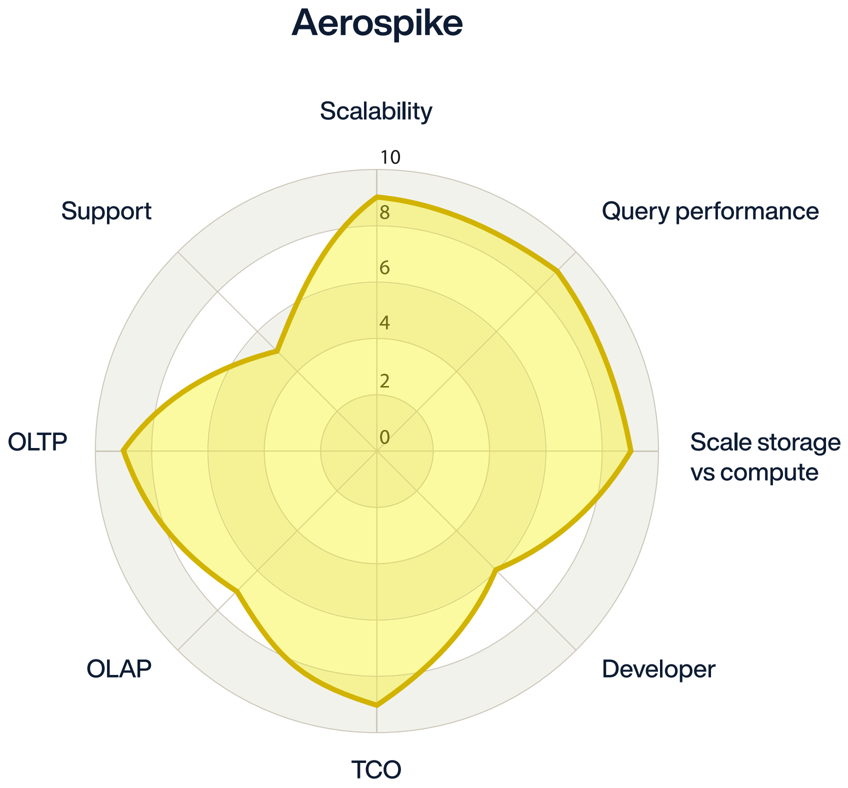
Deployed: On-prem, cloud

Query language: Gremlin

Data model: Key-value, document, graph, vector

Use case focus: OLTP (OLAP Q4 2024)

Support: Aerospike support



TigerGraph

TigerGraph was designed as a distributed multi-purpose graph database. It can be used in both OLTP and OLAP use cases. TigerGraph's architecture is horizontally scalable, allowing it to handle massive graphs with hundreds of billions of entities and up to one trillion relationships. Its parallel query execution ensures efficient exploration of stored data. The system integrates a graph storage engine (GSE) for physical storage and compression, along with a graph processing engine (GPE) for parallel processing and scalability.

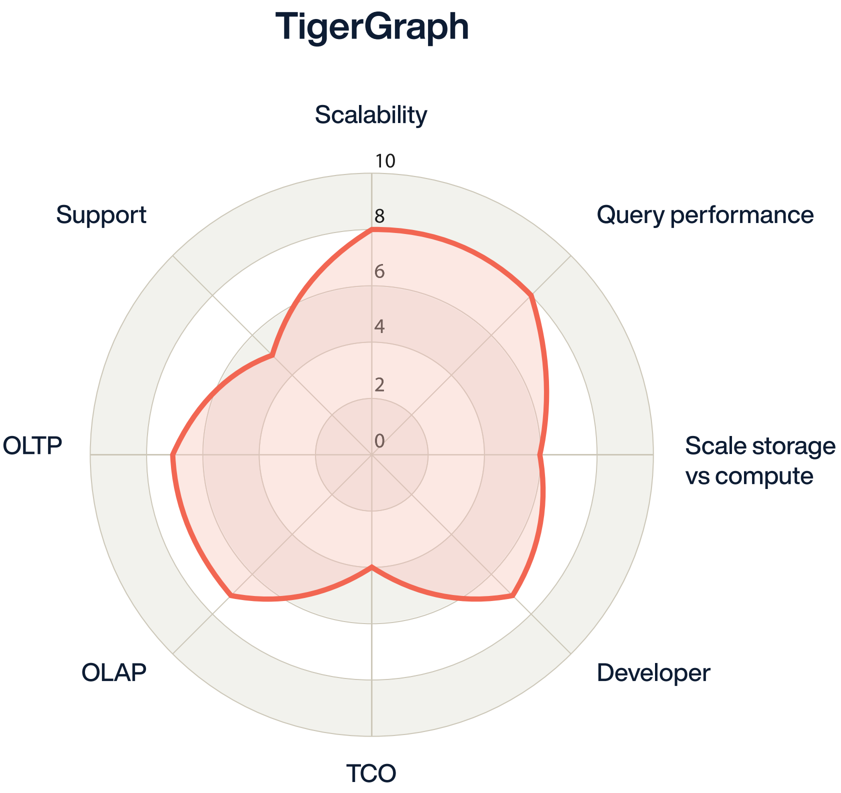
Deployed: On-prem, cloud (TigerGraph Cloud), hybrid cloud

Query language: GSQL

Data model: Graph only

Use case focus: OLTP and OLAP

Support: TigerGraph support



Amazon Neptune

Amazon Neptune Database is a fully managed graph database service provided by AWS. It supports open graph standards (Gremlin and SPARQL) and integrates tightly with other AWS offerings to handle patching, backups, and high availability automatically. Neptune was designed and built for graph OLAP use cases, prioritizing relationship-heavy queries. Neptune Database's elasticity is constrained by AWS's managed scaling policies, and it can be less predictable under high-concurrency transactional loads that require low latency.

Performance characteristics:

- Aerospike Graph is capable of 4x the throughput (OPS) of Amazon Neptune Database.
- Aerospike Graph delivers up to 8x lower read and write latency than Amazon Neptune Database.
- Aerospike Graph bulk loads data 6x faster than Amazon Neptune Database.

Deployed: Cloud, AWS only

Query language:

1. Gremlin for property graph data models
2. SPARQL for RDF data models

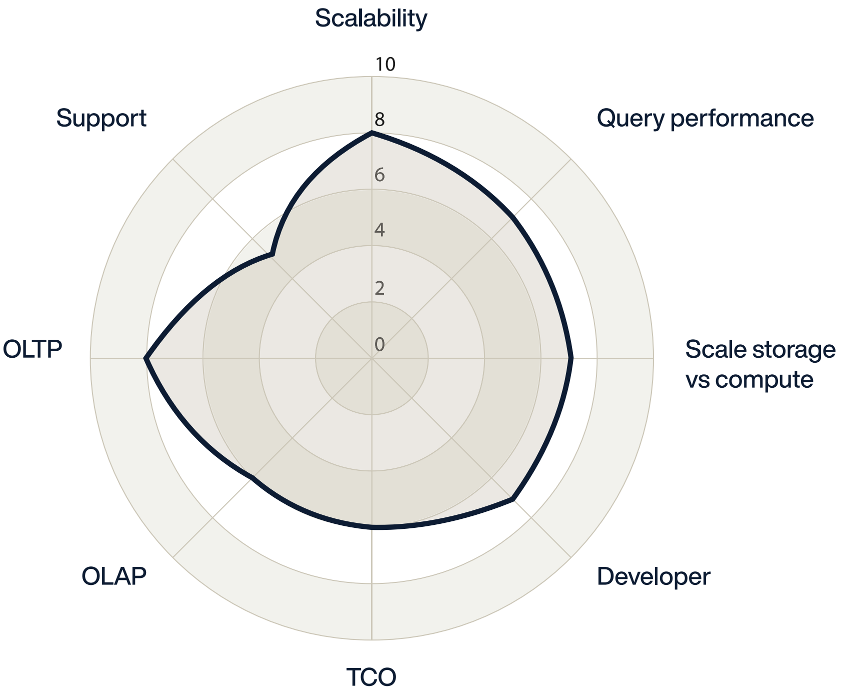
Data model:

1. Property graph model
2. Resource Description Framework (RDF) model

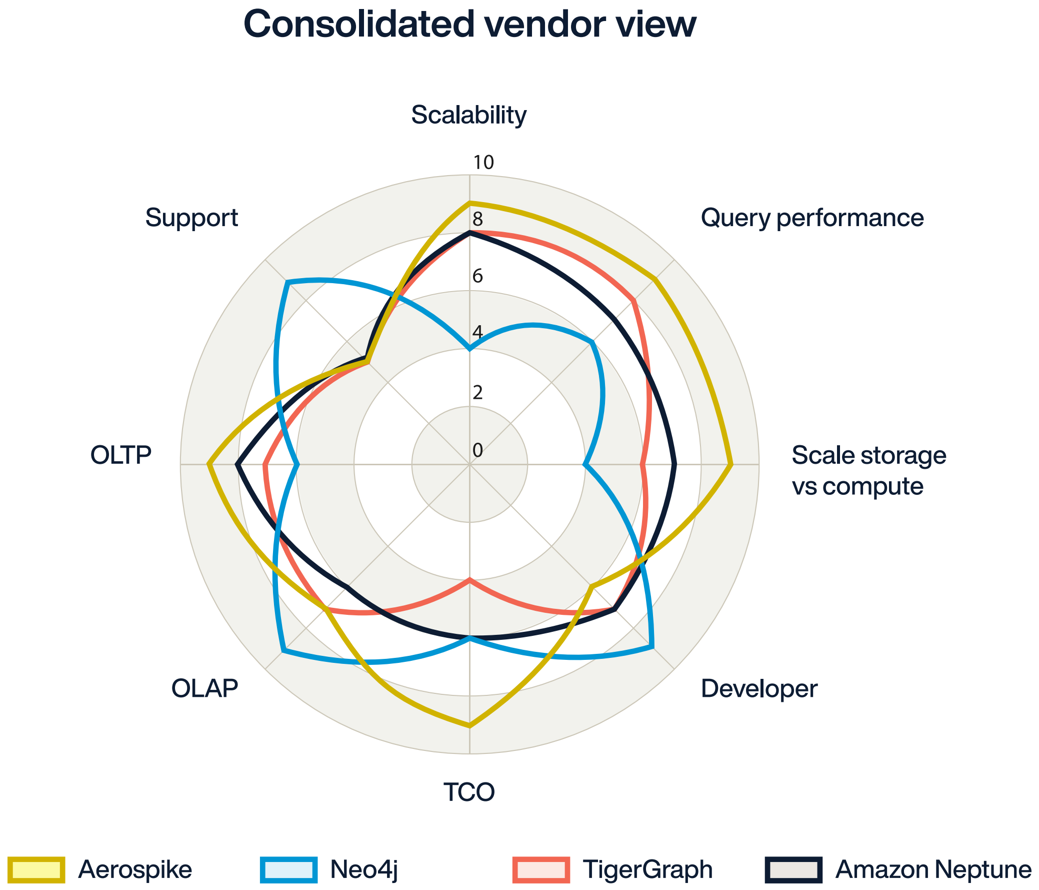
Use case focus: OLAP

Support: AWS support, community support

Amazon Neptune



Consolidated vendor comparison



Conclusion

Graph databases have been in active use for over a decade, focusing on analytical use cases, data visualization, and data discovery functions. With the continuing growth of cloud computing, distributed databases, and now AI/ML, the role of graph databases has evolved to include more operational workloads. These workloads introduce greater requirements on graph databases, including scaling to terabytes, low-latency transactions, high data throughput, enterprise-grade availability, and disaster recovery.

With the buying guidelines we have laid out in this document, you should be able to make a better-informed decision about which approach to graph databases best fits your particular requirements.

Appendix I: Graph terms and concepts

Since graph data is its own data model, it has its own set of terms and concepts. Here are the main ones.

Vertex/vertices/nodes: An element in a graph representing an entity or a data point. Vertices can be anything—people, accounts, locations, etc. Each vertex has a unique identifier.

Edge/relationship: A connection between vertices representing a relationship or connection. Edges define how nodes are related to each other.

Label: An identifier for a vertex type that distinguishes it from other vertex types in the graph. For example, "person," "household," "device," etc.

Property: Describes attributes of nodes and edges. Properties provide additional information about the data stored in the graph.

Traversal: The process of navigating the graph to retrieve or modify data.

Graph query language: A language used to specify queries and commands for interacting with a graph database

Graph OLAP: A category of analytical graph use cases that reflect traditional Online Analytical Processing (OLAP), which is characteristic of traditional business intelligence. In graph OLAP, data sets are relatively static.

Graph OLTP: A category of transactional graph use cases that reflect traditional Online Transaction Processing (OLTP) in their need for frequent updates to datasets and transactional integrity. These use cases are often operational in nature and interact with other enterprise systems (fraud detection, identity resolution, personalization).

Appendix II: Graph query languages

A graph query language is a specialized programming language designed to work with graph databases. Graph query languages allow users to specify how they want to interact with graph data. They define the structure of the graph they want to query and the rules for traversing the graph and retrieving relevant information. When you use a graph query language, you can express queries that retrieve specific patterns from the graph.

Gremlin

Gremlin is the graph traversal language of Apache TinkerPop. Gremlin is a functional, data-flow language that enables users to succinctly express complex traversals on their application's graph. This is the Gremlin way of expressing graph queries and implementing graph algorithms. Every Gremlin traversal is composed of a sequence of (potentially nested) steps.

Gremlin is supported by Aerospike, JanusGraph, Datastax, and Amazon Neptune.

Cypher and openCypher

Cypher is a declarative graph query language designed for expressive and efficient data querying in a property graph. Initially intended for use with the graph database Neo4j, it was later opened up through the openCypher project. Cypher is graph-optimized and understands data connections. It follows connections in any direction to reveal previously unknown relationships and clusters.

Cypher is supported by Neo4j/AruaDB, Amazon Neptune, and AnzoGraph.

GraphQL

GraphQL is a developer-friendly query language for APIs that has gained widespread adoption. GraphQL excels at fetching data from various sources, bundling it together, and returning it to clients. Developers like GraphQL (short for Graph Query Language) because it simplifies making simultaneous requests to multiple data sources.

GraphQL is supported by MongoDB, Yugabyte, ArangoDB, OrientDB, and relational databases through connectors.

GSQL

GSQL is a proprietary graph query language designed for expressing graph operations and analytics. Developed by TigerGraph, GSQL provides a high-level syntax similar to SQL while supporting a MapReduce interpretation. It is schema-based, optimized for storage efficiency and query speed, and offers built-in parallelism for efficient algorithm execution.

GSQL is unique to TigerGraph.

About Aerospike

Aerospike is the real-time database built for infinite scale, speed, and savings. Our customers are ready for what's next with the lowest latency and the highest throughput data platform. Cloud and AI-forward, we empower leading organizations like Adobe, Airtel, Criteo, DBS Bank, Experian, PayPal, Snap, and Sony Interactive Entertainment. Headquartered in Mountain View, California, our offices include London, Bangalore, and Tel Aviv.

For more information, please visit https://www.aerospike.com.