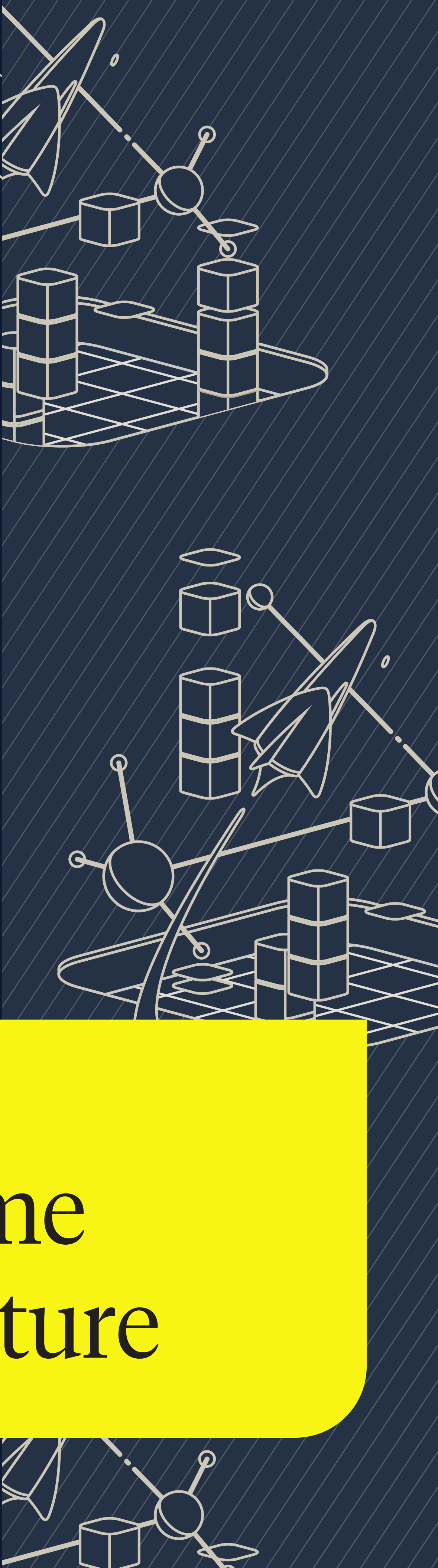
Aerospike



WHITE PAPER

Aerospike real-time database architecture

Contents

Executive overview

The rapidly evolving technology landscape and an increasingly globalized economy are forcing firms in banking, retail, telecommunications, financial markets, and other industries to deliver highly personalized products and services in real time around the clock, placing incredible demands on data management infrastructures.

These real-time services require extremely high performance and availability. Companies have used a variety of solutions over time, including mainframes, clustered relational databases, in-memory databases, and, most recently, NoSQL and NewSQL databases.

Real-time applications create enormous strain on these systems due to:

- An overwhelming consumer demand from mobile devices that produce enormous real-time load on their systems, with the DBMS quickly becoming the bottleneck.
- The requirement for a richer set of application features combined with a great real-time consumer experience.
- Round-the-clock availability makes any breach of a service-level agreement (SLA) via increased latencies, downtime, and maintenance windows unacceptable.

That's why so many firms have turned to Aerospike's NoSQL operational database platform over the past decade. Aerospike leverages the latest hardware innovations to deliver high-scale transaction processing cost-effectively, enabling users to shrink their server footprints by up to 90% and cut operational costs while fulfilling aggressive SLAs.

This paper describes the Aerospike Database architecture, explaining how it provides self-healing clusters that support zero-downtime maintenance while delivering strongly consistent transactions with extremely low latency at scale. This lowers operational costs and helps your business grow. Let's explore the technology that makes this possible.

Design goals

Aerospike built a distributed NoSQL platform to do what other systems couldn't: scale easily, deliver exceptional — and predictable -- performance, provide nearly 100% uptime, guarantee strong data consistency, integrate into existing IT infrastructures, and offer unmatched total cost of ownership (TCO). Deployed on-premises or in the cloud, Aerospike's unique design exploits the latest processor, storage, and networking technologies to satisfy demanding business needs.

Applications that benefit from Aerospike's unique capabilities share some or all of these needs:

- SLAs that require sub-millisecond database response times
- Support for multiple data models with consistent performance, scale, and accuracy
- High throughput for mixed workloads (millions of operations per second and beyond)
- Support for managing hundreds of billions of records in databases of hundreds of terabytes to petabytes
- High availability and fault tolerance for mission-critical applications
- High scalability for handling unpredictable increases in data volumes and transactions
- Adaptable infrastructure for managing varying types of data with minimal effort
- Strong, immediate data consistency
- Support for global transactions that span multiple data centers or cloud regions.
- Low TCO

Today, firms around the world use Aerospike as a system of engagement or a system of record; in some companies, Aerospike serves as both. Aerospike's ability to be easily deployed at the edge or at the core further distinguishes it from other NoSQL systems. Active/active configurations that span multiple data centers or cloud regions support global processing needs. Fig. 1 illustrates several ways in which Aerospike can be deployed to support real-time operational and analytical applications, integrating into enterprise IT infrastructures through Apache Spark, Apache Kafka, and other connectors.

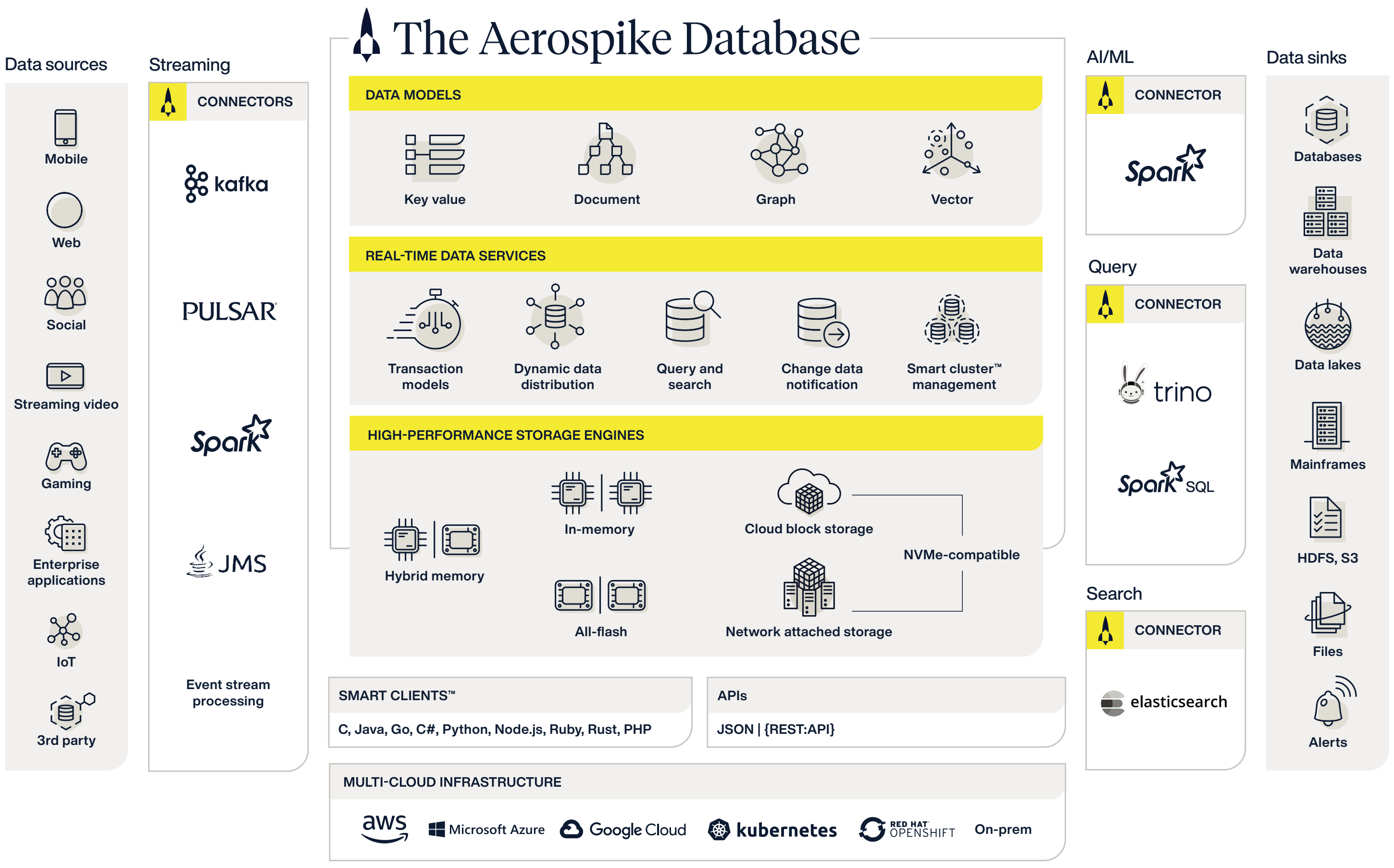


Figure 1: Sample Aerospike usage patterns.

Architectural overview

Aerospike is a shared-nothing, distributed DBMS platform optimized for high-scale operational workloads with a mix of read/write operations. Its schema-free data model provides considerable flexibility, especially compared with relational DBMSs, which require up-front schema definitions. Written in C, Aerospike avoids Java runtime inefficiencies and exploits modern hardware so firms don't need to over-provision servers to accommodate workload growth or spikes. Aerospike Community Edition is free; its architecture is the same as the Enterprise Edition, which offers unlimited scalability, cross-site operations, additional security options, additional storage options, and more. This paper focuses on Enterprise Edition.

Client and server layers

Aerospike's Smart Client layer supports popular programming languages (C/C#, Go, Java, PHP, Python, Node.js, Ruby, and several others) and a query interface. Its server layer, shown in Fig. 2, includes a real-time transaction engine, a data distribution component, a smart cluster management subsystem, dynamic data rebalancing, and a high-performance storage engine. These components form a single Aerospike server that automatically distributes and replicates data across multiple computing nodes for availability and fault tolerance.

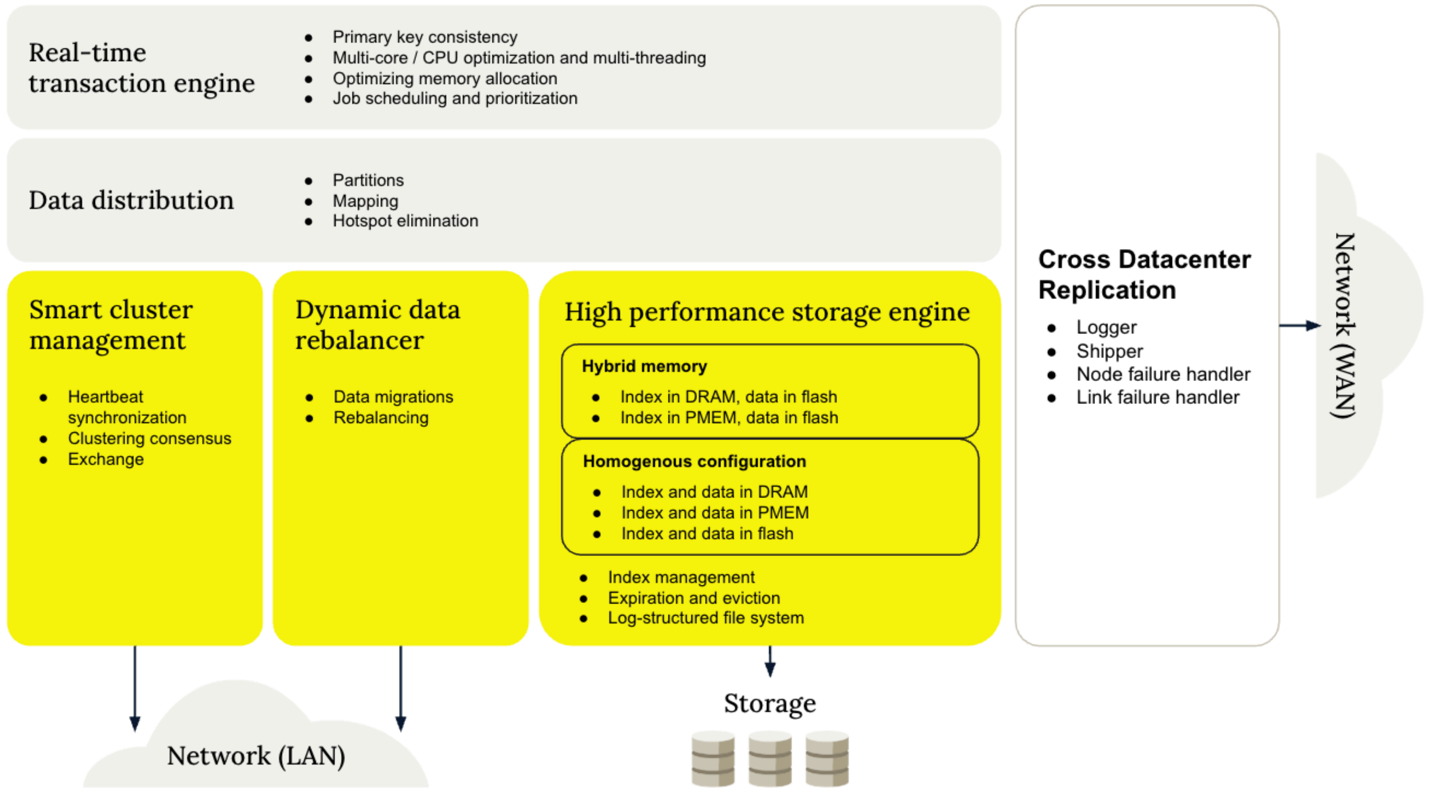
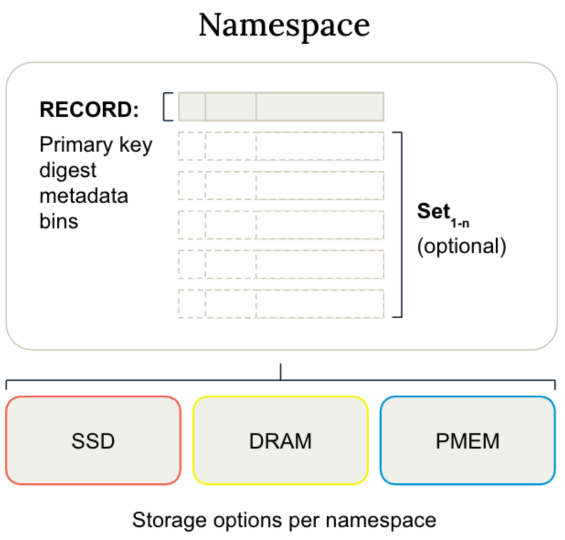


Figure 2: Architecture of Aerospike's server layer.

Although many customers deploy Aerospike in a single data center or cloud region, firms can "stretch" a server across multiple locations for global transaction processing. We'll discuss multi-site clusters later, but essentially, they allow active/active operations across multiple geographies with synchronous data replication and strong, immediate data consistency. Finally, returning to Fig. 2, Cross Datacenter Replication (shown at right) allows firms to asynchronously replicate data between multiple Aerospike systems.

Data models

The Aerospike database does not require a traditional RDBMS schema. Rather, the data model is determined by how you use the system. For example, if you wanted to add a new data type to a record, you would write that data type into the record without first updating any schema.



Conceptual components of the Aerospike data model.

Key-value store model

Records are uniquely identified by a key composed of the name of the Set and the _user key_. The user key is the primary identifier for a record from the app you are building on top of Aerospike. Although a value in a K-V store model is traditionally one opaque and schemaless blob of data, Aerospike records consist of one or more Bins of typed or untyped blob data.

Document store model

A variant of the key-value store model, where a single record is a single document of data encoded in XML, YAML, JSON, and BSON. Aerospike customers commonly store a document of data in a map data type. An Aerospike Map is a superset of JSON that can contain different data types, including nested Map and List structures.

Maps transparently use MessagePack compression for efficient storage. Whereas a traditional document store model stores one document of data per record, an Aerospike record can contain multiple bins, multiple scalar, and collection data types per record.

Java developers may choose to employ the Spring framework to process and manage document data. Spring Data is part of the Spring framework, which makes it easy to map data from a Java application onto the Aerospike Database and read it back. Learn more about Spring Data and the Spring framework.

The Spring Data Aerospike implementation supports both synchronous and reactive programming paradigms.

Graph data model

Aerospike has introduced the graph data model to support our Aerospike Graph database solution. For more information, check out the Aerospike Graph page and product documentation.

Data types

Aerospike users model data in sets of records contained in namespaces. By comparison, relational DBMS users model data in tables contained in databases. Each record stored in Aerospike has a key and named fields (“bins”). Values in bins are strongly typed, but the bins themselves are not, so a single bin across multiple records can contain different types of data. Furthermore, records in the same set can have different bins, allowing considerable schema flexibility. No bin is maintained if a record lacks a particular field, providing efficient storage for sparsely populated data sets.

Aerospike supports popular scalar, complex, and specialized data types: integers, doubles, strings, byte arrays, BLOBs, geospatial data (GeoJSON), and probabilistic data (HyperLogLog). In addition to primary key lookups, Aerospike supports a variety of data retrieval and manipulation operations, including parallel scans, secondary indexing, geospatial indexing and filtering, and user-defined functions.

Users who migrate to Aerospike from relational DBMSs typically denormalize their data to avoid costly join operations and maximize fast key lookups. Significant performance improvements, greater storage efficiency, greater data modeling flexibility, and comparably strong data consistency are among the benefits users report after migrating to Aerospike.

Collection data types

Aerospike also features collection data types that contain an arbitrary number of scalar data type elements, as well as nesting other CDT (List, Map) elements. Collection data types (CDTs) are flexible, schema-free containers that can hold scalar data or nest other collections within them. The elements in a CDT can be of mixed types.

CDTs are a superset of JSON, supporting more data types as elements of the collection, such as integers for Map keys and binary data as List values. Collections come with extensive APIs for performing multiple operations in a single record transaction, with policies to control whether an error in any stage of the transaction causes a rollback, skips to the next operation, or partially succeeds before moving to the next operation.

Indexes

Indexes are powerful tools used in the background of a database to speed up querying. They power queries by providing a method to quickly look up the requested data without scanning an entire dataset. Aerospike provides several indexing options for developers to obtain the best possible performance for database operations.

Primary index

Aerospike's primary key index is a blend of distributed hash table technology with a distributed tree structure in each server. The entire keyspace in the namespace is separated using a robust hash function into partitions. A total of 4096 partitions are equally distributed across cluster nodes. See data distribution for details on hashing and partitioning.

Aerospike uses a red-black tree structure called a sprig. You can configure the number of sprigs for each partition. Configuring the right number of sprigs is a trade-off between extra space overhead and optimized parallel access.

Secondary index

A secondary index (SI) is a data structure that locates all records in a namespace or a set within it based on a bin value in the record. When a value is updated in the indexed record, the secondary index automatically updates. Starting with Database 6.0, SI query results are unaffected by the state of the cluster. Querying by partition returns correct results when the cluster is stable and during data migration after a cluster size change. In Database 6.0, you need compatible clients, such as Java client 6.0, to support rebalance-tolerant queries.

Applications that benefit from secondary indexes include rich, interactive business applications and user-facing analytic applications. Secondary indexes also enable Business Intelligence tools and ad-hoc queries on operational datasets.

Set index

Starting with Aerospike Database version 5.6, you can index the membership of records to their set by optionally creating a set index. Using the set index, the performance of set-level operations, such as scanning all the records in the set, can be improved.

Geospatial Index

Aerospike supports the GeoJSON geospatial data type. All geospatial functionality (indexing and querying) only executes on GeoJSON data types. In addition to integers and strings, Aerospike supports Geo2DSphere data types for indexes.

Let's explore Aerospike's architecture in further detail, beginning with one of its most distinguishing features: how it manages memory and storage.

Storage management

To achieve high transaction throughput and low data access latencies at scale, Aerospike exploits the latest advances in storage technologies, including non-volatile memory extended (NVMe) Flash or solid-state drives (SSDs) and persistent memory (PMEM). In addition, Aerospike makes efficient use of dynamic RAM (DRAM) and even supports traditional rotational devices. Fig. 3 illustrates a few sample configurations; others are possible, including hybrid architectures that combine DRAM or PMEM with Flash.

We'll briefly explore how Aerospike capitalizes on the Flash, PMEM, and other technologies. For more details, see Aerospike's documentation.

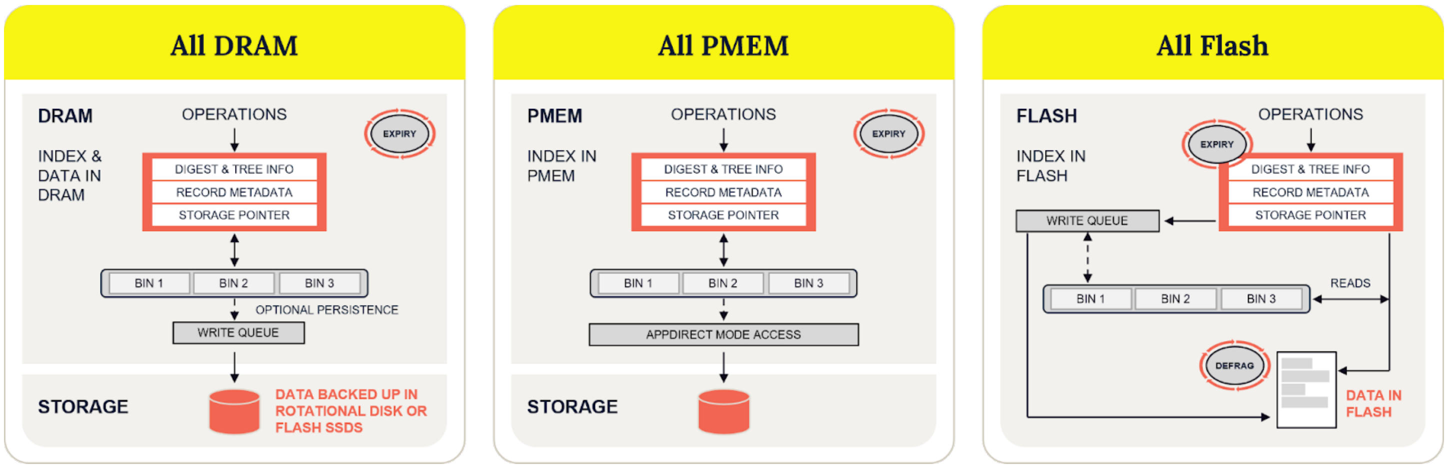


Figure 3: Sample homogeneous storage configurations.

Aerospike has always taken advantage of the unique characteristics of the underlying storage to gain maximum performance with the least amount of hardware. Each storage engine within Aerospike offers the optimal mix of performance and price to meet the unique demands of each workload, with enterprise stability, maximum performance, and unlimited scale.

Aerospike 7 introduced a unified storage format across our three storage engines. This enables new capabilities for handling in-memory workloads, high performance, and efficiency.

Homogeneous configurations

Aerospike can be configured as a pure in-memory system (with no persistence) or as an in-memory system that maintains all indexes and user data in DRAM, with data persisted to disk only as a backup. Even with the latter configuration, all reads are from DRAM. Writes are applied to an in-memory buffer and flushed in batches of 128K or 1MB blocks to Flash or rotational disks. Optionally, applications can commit every write directly to persistent storage.

Aerospike also enables users to maintain all index and user data in PMEM. Indeed, Aerospike was the first commercial open DBMS to support Intel® Optane™ DC PMEM, providing native index and data support modeled on its earlier optimizations for NAND. Aerospike outperforms in-memory DBMSs that simply map PMEM to existing data structures. An all-PMEM Aerospike configuration offers scalability, persistence, high performance (comparable to DRAM for reads), and fast restart times. It is ideal for firms looking to accelerate machine learning and other demanding projects over large data volumes that fit in PMEM.

Finally, users with ultra-large data sets can store all index and user data on Flash. Such a configuration is particularly useful for firms with database sizes so large that they can't be accommodated with PMEM by current processors. Storing indexes in Flash rather than DRAM provides considerable cost savings while still delivering strong performance.

Hybrid memory configurations

Many firms configure Aerospike to persist data to Flash and keep indexes in DRAM or PMEM. This hybrid approach offers significant price/performance benefits over other NoSQL solutions.

Reduced I/O is a key advantage of Aerospike's hybrid architecture. For index access, disk I/O is completely eliminated. For data, if the target record is in the local cache, it's returned without any I/O. Otherwise, Aerospike executes a single I/O. The read latency of I/O in NAND Flash has little penalty for random access so that Aerospike can deliver fast, predictable performance. For further efficiency, Aerospike treats Flash storage as a block device and uses a custom data layout. Access to Flash storage is done in a massively parallel manner, and specialized hashing techniques evenly map the keys to storage devices within nodes. You'll learn more about Aerospike's data distribution shortly.

When DRAM is used for indexes, Aerospike avoids index rebuilds whenever possible, as it stores primary indexes in a shared memory space separate from the service process's memory space. For routine maintenance, Aerospike can avoid data scans to rebuild primary indexes, as an Aerospike process can simply attach to the current copy of the index in shared memory and service transactions.

A hybrid PMEM / Flash configuration offers considerable scalability advantages, allowing Aerospike nodes to surpass 100TB each, as indexes are no longer limited by common DRAM sizes. Keeping indexes in PMEM speeds cold restart times by avoiding the need to rebuild indexes during machine reboot.

Aerospike as an in-memory cache

Aerospike is commonly used as an in-memory cache for use cases such as content caching, session state caching, and speeding up access to data stored in mainframes and data warehouses such as Teradata or Snowflake. Prior to version 7.1, the Aerospike Database had one time-to-live (TTL) based eviction behavior, which decided the order by which records are removed once a namespace eviction threshold is breached. When evictions kick in, the records closest to their expiration are deleted first.

Aerospike 7.1 added LRU eviction behavior, which is controlled by the default-read-touch-ttl-pct configuration for namespace and set levels or optionally assigned by client read operations. With this mechanism, the first read that occurs within a defined percent from the record's expiration will also touch the record (in the master and replica partitions), extending its expiration by the previous TTL interval (calculated as the delta between the record's void time and last update time).

This approach efficiently persists record read activity, resulting in higher-precision LRU evictions than a sampling-based implementation used by other in-memory caches. As a result, a namespace configured to allow record TTLs will efficiently retain the freshest records without requiring any application-side workarounds.

Defragmentation

Aerospike implements a custom file system with a copy-on-write mechanism and periodically runs a background process to reclaim space. Each device maintains a map of blocks and tracks the fill factor consumed by valid records. When the fill factor of a block falls below a configurable threshold, Aerospike moves valid records to a buffer that's flushed to disk when full. To avoid mixing new and old writes, Aerospike uses two write buffer queues: one for client writes and one for defragmenting.

Data distribution

Aerospike uses a sophisticated hashing and partitioning scheme to distribute data across nodes. Specifically, it hashes a record's primary key into a 160-bit digest using the RIPEMD algorithm, which is extremely robust against collisions. This digest space is further divided into 4096 non-overlapping partitions, which Aerospike automatically distributes across the cluster. A partition is the smallest unit of data ownership in Aerospike.

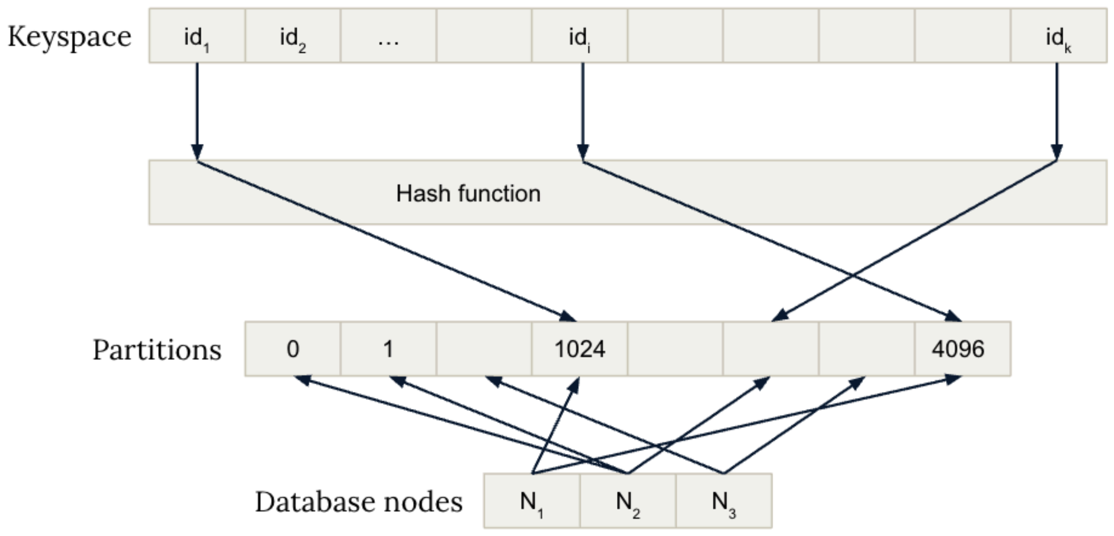


Figure 4: Partitioning scheme.

Aerospike assigns records to partitions based on the primary key digest, as shown in Fig. 4. Even if the distribution of keys in the key space is skewed, the distribution of keys in the digest space and, therefore, in the partition space is uniform. This unique data partitioning scheme helps Aerospike avoid hotspots, thereby promoting scalability and fault tolerance.

Other aspects of Aerospike's data distribution scheme are worth noting. To avoid cross-node traffic for reads and further promote uniform data distribution, Aerospike partitions the indexes in the same way as the data itself to ensure co-location. Writes don't require inter-node communication for index updates; they only cause inter-node activity for data replication.

Aerospike's approach delivers several benefits: uniform workloads across nodes, predictable read/write performance, cluster elasticity, and efficient, non-disruptive data rebalancing. In addition, because Aerospike's Smart Client understands the data partitioning scheme, it sends data access requests to the appropriate node, minimizing costly network "hops."

Cluster management

The cluster management subsystem handles node membership and ensures that all nodes concur on current membership, which can change due to system upgrades, node failures, network disruptions, expansion (or removal) of nodes, etc. Because cluster changes can significantly impact data access latencies, Aerospike quickly takes appropriate action.

Heartbeats and node health

Aerospike detects node arrivals or departures via heartbeat messages exchanged between nodes, with each node maintaining a list of adjacent nodes that have recently sent heartbeats. Nodes departing the cluster are detected by the absence of heartbeats for a configurable timeout interval and removed from the adjacency list. Since a choked network can cause arbitrary packet loss, Aerospike allows other regularly exchanged messages (such as replica writes) to serve as surrogate heartbeats. This minimizes the impact an unstable network will have on the cluster view and prevents unnecessary work.

In addition, every node scores the health of its neighboring nodes by periodically computing the weighted moving average of the expected number of messages received per node versus the actual number received. A node is deemed "unhealthy" when its average message loss exceeds twice the standard deviation across all nodes. If such a node belongs to the cluster, it is removed. If it isn't yet a member, it's not allowed to join until its average message loss falls within acceptable limits.

Data migration and rebalancing

A change in cluster membership implies some need to migrate (move) data between nodes to rebalance the cluster and ensure that the latest version of data is available at each master and replica copy in the current cluster. Once consensus is reached on a new cluster view, all nodes run a deterministic algorithm that assigns the master and replica nodes to each data partition.

To minimize the impact of data migrations on the cluster, Aerospike versions partitions and only migrates data that differs between the versions. Furthermore, if one partition is a subset of a copy on another node, Aerospike avoids migration altogether. Nodes with the fewest records in their partition versions begin migration first, allowing them to complete the process quickly.

To avoid reacting too quickly to node arrivals and departures, Aerospike batches adjacent node events together, acting on these once every quantum interval (typically set to a slightly higher value than the heartbeat timeout interval). In this way, Aerospike manages multiple node additions or removals simultaneously - a clear advantage for scaling out a cluster. Aerospike also offers several configuration options that administrators can adjust in real time to speed or delay migrations depending on the cluster's current workload.

Aerospike implements a uniform balancing algorithm that combines random assignment of keys to partitions and partitions to nodes, ensuring that rebalancing isn't needed until cluster membership changes. Key-range-based partitioning schemes cannot enforce uniform data balancing across nodes without constant rebalancing, as different key ranges may have a non-uniform distribution of records that keeps changing as records are added to the database. This results in frequent reassignment of key ranges to nodes to enforce uniform data distribution across nodes.

Finally, Aerospike's rebalancing mechanism is transparent and doesn't impact cluster behavior, simplifying operations and contributing to predictable performance. The rebalancing mechanism is robust even if a node fails during the rebalancing operation. By not requiring operator intervention, clusters will self-heal even at demanding times.

Transaction engine

Aerospike's real-time transaction engine reads and writes data on request, providing data consistency and isolation guarantees. To deliver high throughput with low latency, Aerospike scales up and out. It takes full advantage of available hardware to maximize efficiency, minimize operational complexity, and lower TCO while delivering exceptional performance and availability. As you'll see, Aerospike is optimized for multi-core CPUs, integrates with the latest networking technologies, and addresses memory fragmentation in unique ways.

Multi-core system

Commodity processors have a multi-core, multi-socket architecture with up to 64 cores. Their caches have Non-Uniform Memory Access (NUMA), allowing system memory bandwidth to scale with the number of physical processors. Latency and throughput behavior is asymmetric based on access to data in local memory in the same socket (rather than remotely from another socket). Applications sensitive to latency need local memory traffic and must use a threading model with a locality of access per socket to scale with the number of physical processors.

Aerospike groups multiple threads per socket instead of per core, thus aligning with a NUMA node. These transaction threads are also associated with specific I/O devices. The interrupt processing for the client-side network communication and disk-side I/O is bound to the core where these threads run. This helps reduce the amount of shared data accessed across multiple NUMA regions and lowers latencies.

Support for ARM processors

Starting with release 6.2, Aerospike Database supports 64-bit ARM processors compatible with the ARMv8.2-A instruction set (Neoverse N1 microarchitecture), such as AWS Graviton2 processors. This augments support for 64-bit Intel/AMD processors compatible with x86-64-v2 microarchitectures.

Networking optimizations

Aerospike is able to run some operations in the network listener thread to avoid context switches further. To fully exploit parallelism, the system creates as many network listener threads as cores. The client request is received, processed, and fulfilled without relinquishing the CPU; this is done in a manner that is non-blocking, short, and predictable.

Memory defragmentation

Aerospike natively allocates all memory used for the resident index in slabs for maximum control and efficiency. This is important because increases in data volumes, transaction rates, and DRAM capacities have made memory fragmentation a major challenge. To avoid fragmentation, Aerospike uses the `jemalloc` memory allocator and custom extensions, including special-purpose slab allocators (arenas), to handle different object types within the server process. Grouping data objects by namespace into the same arena optimizes object creation, access, modification, and deletion.

Data structure design

Aerospike uses partitioned, single-threaded data structures with fine-grained individual locks. This reduces memory contention across partitions and promotes the high performance of atomic operations on modern CPUs. By contrast, nested structures (such as B+ trees) require locks at each level, which can restrict concurrent data access and inhibit throughput. Aerospike allows safe and concurrent read, write, and delete access without multiple locks. Aerospike's structures are designed so frequently accessed data has locality and falls within a single cache line to reduce cache misses and data stalls.

Scheduling and prioritization

In addition to key-value operations, Aerospike supports scans, batch queries, and secondary index queries. To manage varied workloads efficiently, Aerospike uses:

1. **Type-based job partitioning.** Each job type is allocated its own thread pool and is prioritized across pools. Each job is further prioritized within its own pool.
2. **Effort-based unit of work.** The basic unit of work is the effort needed to process a single record, including lookup, I/O, and validation. Each job is composed of multiple units of work, which defines its effort.
3. **Controlled load generation.** The thread pool has a load generator, which controls the rate of generation of work. It is the threads in the pool that perform the work.

Cooperative scheduling ensures that worker threads yield CPU so that other workers can finish their jobs after X units of work. These workers have CPU core and partition affinity to avoid data contention when parallel workers access certain data.

Concurrent workloads of a certain job type run on a first-come, first-served basis to minimize latencies. Longer-running tasks, such as scans and queries, are scheduled in a round-robin fashion, allowing many to be run in parallel for a period of time while others are paused and rescheduled dynamically. In this way, all long-running jobs make progress, while short-running jobs complete quickly. Users can tailor Aerospike's scheduling configuration as needed.

Data consistency

Aerospike administrators often configure namespaces for strong consistency (SC) to prevent data loss and stale or dirty reads, just as traditional enterprise systems do. Alternatively, namespaces can operate in availability mode, which offers fewer guarantees, but makes as much data as possible available during failures. In both modes, Aerospike's guarantees are immediate, and its transaction scope is a single record. Performance is comparable, so the trade-off is primarily consistency vs. data availability during failures.

Availability mode

Very briefly, availability mode allows reads and writes to occur concurrently in multiple sub-clusters when a network or other failure leaves some nodes unable to communicate with others (sometimes called a "split brain" scenario). Because each surviving sub-cluster can process reads and writes, reads on one sub-cluster will not see data updated on another sub-cluster during the split. Similarly, concurrent writes to copies of the same record on different sub-clusters will create conflicts at cluster recovery time, potentially resulting in data loss.

Strong consistency (SC) mode: Overview

By contrast, SC mode prevents conflicting writes. It ensures that reads see the most recently committed data by prohibiting multiple master copies of the same data from being active during failures. Enforcing this requires knowledge of the cluster's state, which Aerospike achieves by monitoring node heartbeats and consulting an internal roster. The roster lists the nodes in a healthy cluster and identifies the master and replica copies of data partitions owned by each. Every node in the cluster stores a copy of this roster.

Unlike most systems, Aerospike needs only two stored copies of user data for strong consistency. Using an adaptive scheme that adds more write copies on the fly when necessary, Aerospike optimizes performance in most cases while incurring a modest overhead for those that rarely occur. Aerospike's approach also reduces network traffic, CPU overhead, and storage costs. TCO savings can be substantial, particularly for high data volumes.

To prevent data loss, Aerospike commits writes to all replicas before returning success to the client. If a write to a replica fails, the master will ensure the write is completed to the appropriate number of replicas. Aerospike processes all writes for a given record sequentially so that writes won't be re-ordered or skipped. Independent tests of the Jepsen workload revealed no errors when operating with Aerospike's recommended configuration settings. To preserve the appropriate ordering of events across the cluster, Aerospike uses a customized Lamport clock that combines timestamps with additional information, including a counter that increments when certain events occur. This approach is superior to a simple timestamp-based architecture, which can suffer from clock synchronization problems between nodes that lead to inadvertent data inconsistencies.

SC mode: Failure handling

Aerospike follows several rules to recover from failures and form new sub-clusters in SC mode. If the roster-master copy of a data partition is unavailable, Aerospike designates a new master from the available replicas and creates new replicas to enforce the replication factor setting. This occurs in the background and does not impact availability. During a cluster split, only one sub-cluster can accept requests for a given partition according to these rules:

1. If a sub-cluster has both the master and all replicas for a partition, the partition is available for both reads and writes in that sub-cluster.

2. If a sub-cluster has a strict majority of nodes and has either the master or a replica for the partition, the partition is available for both reads and writes in that sub-cluster.
3. If a sub-cluster has exactly half of the nodes and has the master, the partition is available for both reads and writes in that sub-cluster.

SC mode: Operational examples

With that backdrop, let's consider some examples. Fig. 5 depicts four operational states of a 5-node system with a replication factor of two. Node 5 maintains the roster master copy of a given data partition, while Node 4 stores the replica. The top row shows a healthy cluster with all nodes operational and all data available.

The second row shows Nodes 1-3 split from Nodes 4-5. Read/write requests for the target data are permitted only on Nodes 4-5, which contain both the master and replica copies as designated in the roster (see Rule 1). In the third row, Nodes 1-4 are split from Node 5. Because Node 4 contains the roster-replica of the data and is part of a majority sub-cluster, it will assume the master role for this data and process read/write requests (see Rule 2). Furthermore, Aerospike will automatically replicate this data to another available node (Node 3, in this example) to comply with the replication factor setting. Note that Node 5 will not accept read/write requests because it is part of a minority sub-cluster (in this case, a sub-cluster of one); this prevents data loss and stale reads. Finally, the last row depicts what would happen if Node 4 subsequently split from Nodes 1-3. Aerospike would not process read/write requests for any record in this partition because Nodes 4 and 5 – designated in the roster as the replica and master owners of the data – don't form their own sub-cluster and aren't part of a majority sub-cluster of nodes listed in the roster.

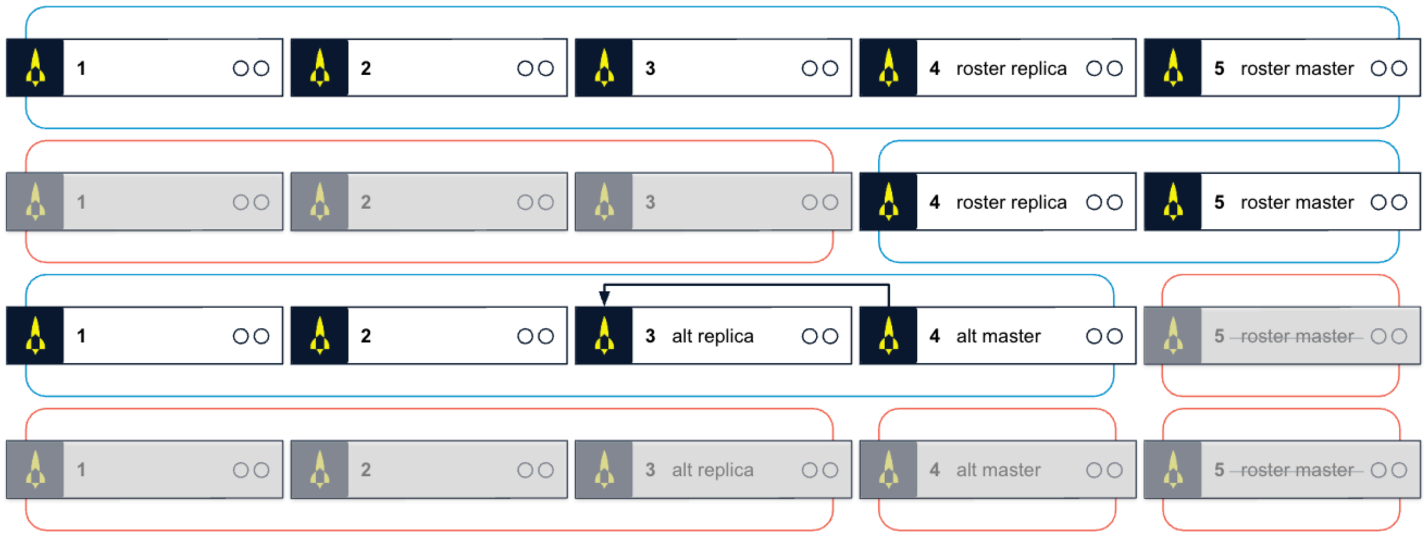


Figure 5: Enforcing strong consistency in different cluster states.

Aerospike's strong consistency model addresses subtle situations to ensure appropriate behavior. For example, consider a failure scenario in which the roster-master node for a record hasn't yet detected a network problem impacting the cluster's state and, therefore, hasn't relinquished master ownership of its data even though the new master (in a separate sub-cluster) has detected the cluster's changed state and taken over. During this transitional period (sometimes called the "master overhang" period), clients might be writing to both the old and new masters, which could introduce data inconsistencies if only timestamps were used to order events. To prevent such inconsistencies, Aerospike associates "regime" information with each partition.

This information, coupled with the last update time and record generation data, comprises Aerospike's customized Lamport clock, which enables Aerospike to properly order events across the cluster and avoid data loss.

During rolling upgrades, Aerospike provides full data availability. If a sub-cluster has fewer than the replication factor number of nodes missing, it serves as a super-majority sub-cluster, and all partitions are active for reads/writes. In addition, data is fully available if the system splits into exactly two sub-clusters; all partitions are active for reads/writes in one or the other sub-cluster.

SC mode: Read consistency options

Applications can choose between linearized access or session consistency for each read operation. The former is most restrictive, imposing a linear view of the data for all clients that access it. Essentially, when one application completes its write operation, all later reads by any application will see the value of that write or a later write. Regime information for each data partition enables Aerospike to determine if all copies are in sync at the time of the read request. If so, the read can proceed; otherwise, a cluster change may be in process, and the client will need to retry the operation. Session consistency is less restrictive, guaranteeing only that the same application will read its own writes. If occasional stale reads across separate application instances are acceptable, the much higher performance associated with availability mode can also be achieved with session consistency.

For more details on Aerospike's approach to data consistency, see Aerospike's documentation.

Security

Aerospike secures users' operational data through various authentication and authorization mechanisms, data and transport-layer encryption, and audit trails. Together, these capabilities provide enterprise users with the security needed to manage their mission-critical data.

Users authenticate to Aerospike using internally managed passwords or, if desired, through external services such as Lightweight Directory Access Protocol (LDAP) or Kerberos. Each user can be authorized to assume one or more roles that convey certain privileges. To simplify operations, Aerospike provides several predefined roles, and administrators can create new roles if desired. Predefined roles include:

- Read from a data set.
- Write to a data set (but not read from it). Write-only permission is particularly useful for broadly capturing highly sensitive information but limiting reporting on it.
- Read/write to/from a data set.
- Read/write/execute UDFs over a data set.
- Data administration, such as creating/dropping objects and aborting scans or queries.
- System administration, which subsumes data administration privileges and adds system-level operations, such as dynamically reconfiguring the server and enabling/disabling auditing.
- User administration, such as creating/dropping users and assigning/revoking roles.

Aerospike also supports whitelisting to restrict data access at the domain and user level.

Encryption is supported for data-in-motion (TLS 1.2) and data-at-rest (AES-128 and AES-256). To minimize overhead, Aerospike encrypts data per record rather than disk sector. Internal tests revealed no measurable performance loss with record sizes of 1 KB or greater. Aerospike allows the following security items to be managed and stored in a HashiCorp vault: LDAP user credentials & TLS certificates, XDR remote destination passwords, encryption-at-rest key, and network TLS certificates and keys.

Smart Client layer and APIs

Aerospike's client layer is an open source library that supports more than ten languages and maintains awareness of the cluster's current state. Its API supports creating, reading, updating, and deleting (CRUD) data and running queries and user-defined functions.

To maximize performance, Aerospike's client caches information in memory about how data is distributed in the cluster (a map of data partitions to nodes). Retrieving a record typically involves only one network "hop": the Aerospike client requests the data from the node where it resides, and the server returns it. Scans or queries are processed in parallel on multiple nodes.

Without correct transaction routing from the client, a request would require more network overhead. Either a proxy would be placed in the middle of the transaction, increasing latency and decreasing throughput, or transactions would flow over the cluster interconnect, placing the extra network hop between servers. Aerospike's design avoids such overhead.

Global transaction processing

The global economy has changed the game for many transactional applications, including trade settlements, supply chain management, currency exchanges, parcel tracking, and others. Client demands for quick turnarounds clash with business processes that take hours or days to complete. This has forced many firms to redesign their underlying transaction processing infrastructures to include a highly resilient, geographically distributed database platform with strong data consistency and performance.

Active/active database infrastructures often emerge as mandatory in these digital transformation projects. Such platforms span two or more regions and service application requests at all locations. Aerospike offers two options: Cross Datacenter Replication (XDR) and multi-site clustering. Let's explore each in turn.

XDR

XDR transparently and asynchronously replicates data between Aerospike clusters. Firms often use XDR to replicate data from Aerospike-based edge systems to a centralized Aerospike system. XDR also enables firms to support continuous operations during a crisis (such as a natural disaster) that takes down an entire cluster.

XDR replication is log-based. Applications write to their "local" cluster, and XDR logs minimal information about the change (not the full record). For efficiency, XDR batches changed data and ships only the latest version of a record. Thus, multiple local writes for one record generate only one remote write – an important feature for "hot" data. XDR also keeps a pool of open connections for each remote cluster, pipelining replication requests in a round-robin fashion.

XDR avoids any single point of failure. Indeed, Aerospike continues to ship changes as long as at least one node survives in the source and destination clusters. It also adjusts easily to new node additions in any cluster and can utilize all available resources equally.

Because applications can write data at any location in an active/active configuration, conflicting writes can occur at the record or bin level if the same data is updated concurrently at multiple sites, resulting in different values at different locations or loss of some data. Some applications can tolerate this relaxed form of data consistency. For those that cannot, firms may choose to restrict write access to certain data to a given location, thereby avoiding the potential for conflicting concurrent updates. Note that the asynchronous nature of the shipping algorithms means that when a site failure occurs, the most recent updates from the failed site will not be available at the other sites. The speed and throughput of XDR is best suited to use cases where low latency is a must. The technology trades off strong consistency between sites for low latency reads and writes. More details on XDR are in this Aerospike Database 5-related blog, as well as this blog on active/active capabilities.

Multi-site clustering

As the name implies, Aerospike's multi-clustering support enables a single cluster to span multiple geographies, as shown in Fig. 6. This allows users to deploy shared databases across distant data centers and cloud regions with no risk of data loss. Automated failovers, high resiliency, and strong performance are hallmarks of Aerospike's implementation. Two features underpin Aerospike multi-site clustering: rack awareness and strong consistency.

Rack awareness allows replicas of data partitions to be stored on different hardware failure groups (different racks). Through replication factor settings, administrators can configure each rack to store a full copy of all data, maximizing data availability and local read performance. Aerospike evenly distributes data among all nodes within each rack to prevent hotspots.

Only one node maintains a master copy of a given data partition at any time. Other nodes (typically on other racks) store replicas, which Aerospike automatically synchronizes with the master. As noted earlier, the roster tracks the locations of masters and replicas; it also understands the racks and nodes of a healthy cluster. Returning to Fig. 6, note that each data center has one rack with three nodes, and each node has a copy of the roster (shown in blue). Given a replication factor of three, this example shows the roster-master copy of a yellow data partition on Node 3 of Rack 2; replicas exist on Node 1 of Rack 1 and Node 2 of Rack 3.

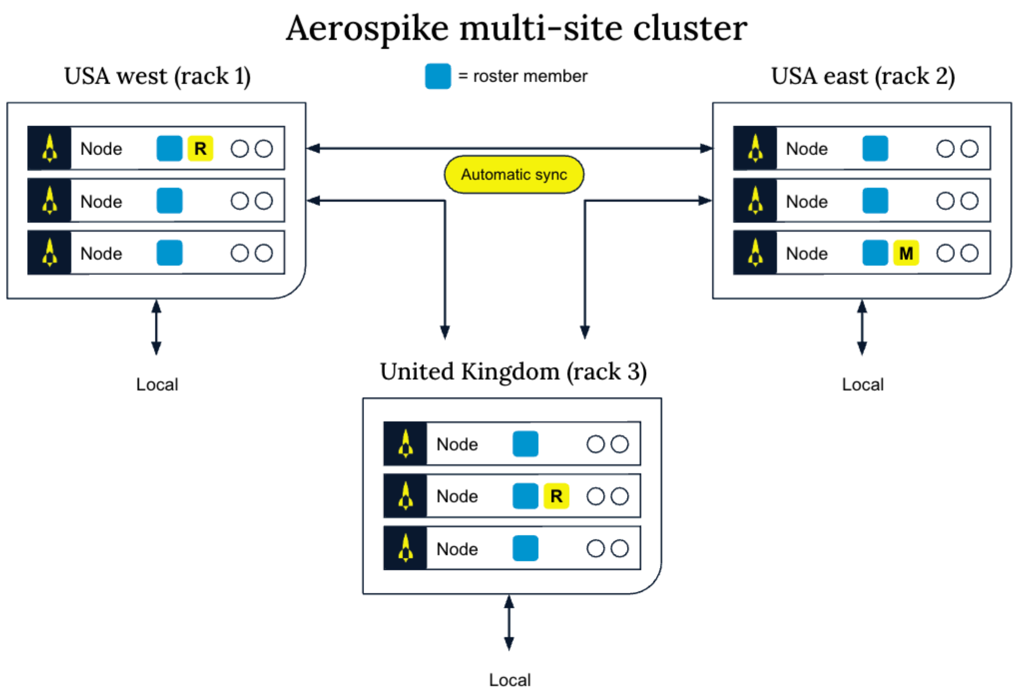


Figure 6: Sample Aerospike system spanning three sites with a replication factor of three (3).

Aerospike routes an application's request to read a data record to the appropriate rack/node in its local data center. Referring to Fig. 6, an application in USA East seeking to read data in the yellow partition will access the master copy in Rack 2 Node 3 with one network hop. Similarly, a UK application seeking the same record will also enjoy "one-hop" access, as Aerospike will retrieve the data from the replica in Rack 3 Node 2. By intelligently processing read requests, Aerospike can deliver sub-millisecond read latencies in multi-site clusters.

Writes are processed differently. For consistency across the cluster, Aerospike routes each write to the rack/node with the current master of the data. The master node ensures that the write is applied to its copy and all replicas before committing the operation. In Fig. 6, a write to the yellow partition would be routed to Rack 2 Node 3 regardless of the request's origin. Routing writes, and synchronizing replicas introduces overhead, so writes aren't as fast as reads. However, most firms using Aerospike's multi-site clusters experience write latencies of a few hundred milliseconds or less, which is well within their target SLAs.

Resiliency against failures is critical for global transaction processing. An Aerospike multi-site cluster follows the same overall rules for enforcing strong data consistency as a single-site cluster, automatically taking corrective actions for most common scenarios. For example, if the roster-master becomes unavailable due to a node or network failure, Aerospike designates a new master from the available replicas and creates new replicas as needed to satisfy the replication factor. In a multi-site cluster, the new master will typically be on another rack.

Consider a scenario where one data center becomes unavailable, perhaps due to a network or power failure. In Fig. 7, the UK site (Rack 3) is unreachable by the rest of the cluster. Aerospike will automatically form a new sub-cluster consisting of USA West (Rack 1) and USA East (Rack 2) to continue to service reads and writes without any operator intervention.

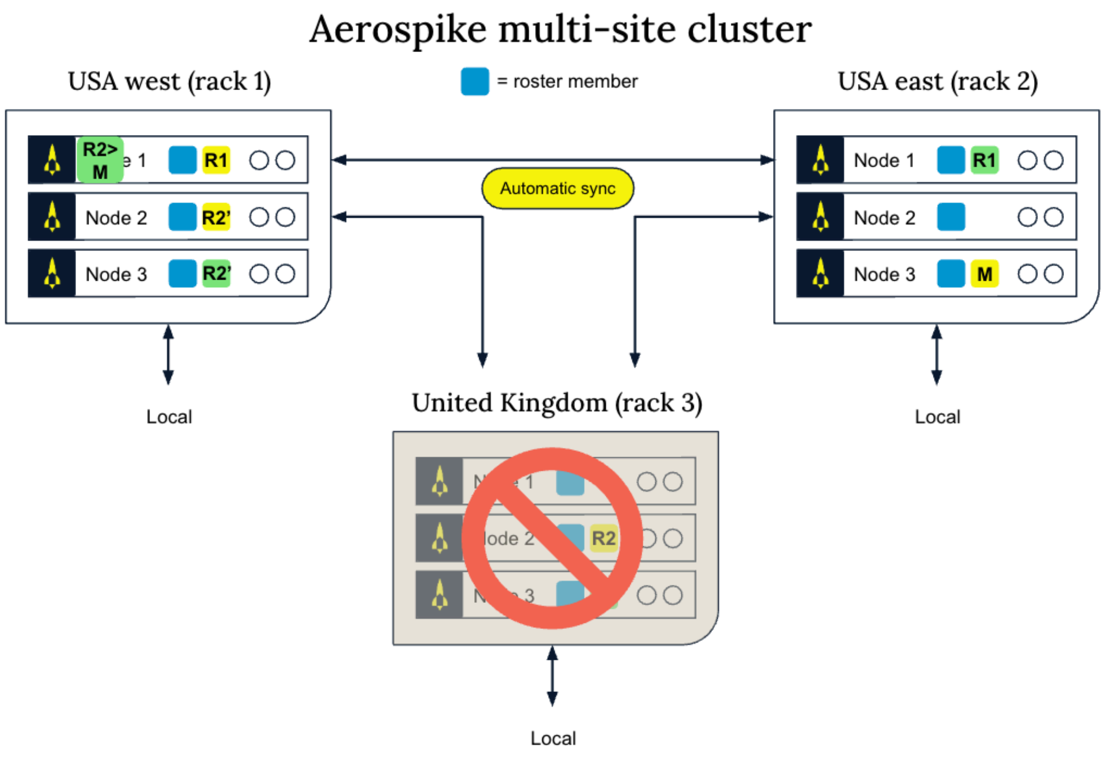


Figure 7: Failure scenario when one data center becomes unreachable.

Let's first consider access to the yellow data. USA East will handle all writes, as it contains the master, while reads will be processed at either location in the sub-cluster. To maintain a replication factor of 3, Aerospike will automatically create a new yellow replica (R2' on Rack 1 Node 2). Now consider the green data. The rack containing its master is not in the active sub-cluster, so Aerospike will promote a replica to the master. In this example, R2 in USA West is promoted to master, and all write operations involving that data will be directed there. Reads will be serviced at either active location in the sub-cluster. To maintain the replication factor of 3, Aerospike will automatically create a new green replica (R2' on Rack 1 Node 3).

When the UK data center recovers, it will synchronize its data with the active sub-cluster. Upon rejoining the cluster, the UK will again host the master copy of the green data and a replica of the yellow data. Aerospike will remove temporary replicas on other nodes and revert any temporary masters to replicas.

A separate white paper describes multi-site clustering in greater detail.

Integration options

To help firms integrate Aerospike into their IT infrastructures, Aerospike offers connectors for Spark, Kafka, and Java Message Service (JMS). In addition, more than a dozen community-developed connectors are available for Apache Hadoop, the Spring framework, ASP.NET, and more. While it's beyond the scope of this paper to describe the various Aerospike connectors, we'll briefly summarize Aerospike Connect for Spark.

Aerospike's Spark connector enables Spark developers to expand the scope of their real-time analytics by readily combining Aerospike data with data streamed through Spark. In particular, Spark developers can load Aerospike data into Datasets and DataFrames for analysis with Spark SQL, Spark ML, and other Spark libraries. Furthermore, data sourced from Spark can be persisted in Aerospike for subsequent use by non-Spark applications, expanding the breadth of its usage. Aerospike's efficient use of volatile, non-volatile, and persistent memory affords Spark users an inexpensive backing store for large volumes of real-time data.

Aerospike Connect for Spark features several performance optimizations. For example, the connector allows users to parallelize work on a massive scale by leveraging up to 32,768 Spark partitions to read data from an Aerospike namespace, storing up to 32 billion records across 4,096 partitions. Aerospike will scan its partitions independently and in parallel; such support, when combined with Spark workers, enables Spark users to rapidly process large data volumes of Aerospike data (as of this writing, 100TB within a couple of hours).

In addition, Spark programmers can use query predicates to limit Aerospike processing to relevant partitions. This avoids costly full data set scans, speeds data access, and avoids unnecessary data movement between Aerospike and Spark. Filtering large data sets on the Aerospike server also minimizes memory needs on the Spark side, avoiding potential runtime errors or high operational costs. Aerospike administrators can control the number of parallel scans via a configuration setting, tailoring the number of scan threads to the needs of Spark applications. In addition, Connect for Spark uses worker thread pooling to minimize overhead when writing Spark data to Aerospike. Further optimizations include batch reads for queries with primary keys and partition filtering for secondary keys.

Combining Aerospike and Spark makes it possible to address high-scale application challenges that would be impractical otherwise. Examples include AI/ML training, 360-degree profiles of millions of customers, transformations of massive public and private data sets, and more.

Performance benchmarks and TCO comparisons

Any system architecture is compelling only if it delivers tangible benefits. As noted earlier, many firms moved to Aerospike after finding other solutions couldn't meet their performance, availability, scalability, or budgetary needs. Here's a short summary of some of the compelling benefits cited by just a few customers and partners:

- Signal, makers of an identity resolution platform, reduced its server count from 450 to 60 and improved its performance at the 99th percentile by 100x after migrating to Aerospike. Business processes now execute in a fraction of the time (10% or less), and the firm has freed up resources to focus on new strategic projects.
- Playtika, an online gaming firm, cut its server footprint by a factor of 6 and boosted performance by 300% with Aerospike. The firm expects to save up to $4.2 million in TCO over 3 years.
- Hewlett Packard Enterprise's AI-Driven Big Data Solutions team benchmarked several PMEM-aware NoSQL systems to meet anticipated needs for ultra-fast performance at extreme scale. Aerospike emerged as the leader. One system couldn't even fully load the data within the target timeframe. In addition, Aerospike's per-node performance was clocked at 1667% faster than another system for the target workload.
- A major European bank relies on Aerospike for payment processing of 2000 transactions per second and up to 43 million transactions per day for a per transaction cost of €0.0020 or less. Other solutions didn't meet the bank's needs for resiliency (100% uptime), consistency (no data loss and no dirty or stale reads), and low transaction costs.
- A leading brokerage firm uses Aerospike to offload work from its mainframe solution for financial trades. Initially, it used a caching layer over its mainframe DBMS to improve response time, but the firm turned to Aerospike when it realized that the cache would need to grow from 150 to 1000 nodes to meet future needs. The firm increased processing speeds 5x while tripling the database size. Aerospike enabled the firm to accomplish this with 90% fewer servers deployed, saving an estimated $10,000 per trading day. Furthermore, Aerospike's schema flexibility and extreme uptime enabled the firm to launch more than a dozen applications on Aerospike after its deployment; these applications delivered new industry-leading capabilities for credit risk monitoring, enhanced customer service, and more.

For more, see these comparative benchmark and TCO reports.

Conclusion

As the pace of business accelerates, firms are pushing their data management infrastructures to manage more data and more transactions in real time at increasingly rapid speeds. Downtime simply isn't acceptable, and neither are high operational costs.

Aerospike designed its architecture with these aggressive demands in mind. Using techniques described earlier in this paper, Aerospike harnesses the high performance of contemporary processors, storage devices, and network hardware to deliver a high-throughput, low-latency database platform for real-time applications. With Aerospike, large volumes of real-time data can be used "as is" in various analytical pipelines (such as Spark, Kafka, JMS, and others) with ease and efficiency.

Unlike other platforms, Aerospike scales up and out, providing predictable, ultra-fast performance and predictable, low TCO. That's why firms in banking, telecommunications, retail, financial markets, technology, and other industries have deployed Aerospike at the edge as a system of engagement and, at the core, as a system of record to digitally transform their infrastructures for today's global economy.

Want to learn more about Aerospike's architecture? Contact Aerospike to arrange a briefing or discuss potential pilot projects.

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.