AEROSPIKE

ACID Support in Aerospike

June 13, 2014

Table of Contents

Introduction

The CAP Theorem postulates that only two of the three properties of ***consistency***, ***availability***, and ***partition tolerance*** can be guaranteed in a distributed system at a specific time. Since availability is paramount in most deployments, the system can either provide consistency or partition tolerance. Note, however, that these three properties are more continuous rather than binary. Aerospike is by and large an AP system that provides high consistency by using the following techniques:

- Trade off availability and consistency at a finer granularity in each subsystem
- Restrict communication latencies between nodes to be sub-millisecond
- Leverage the high vertical scale of Aerospike (1 million TPS and multiple terabyte capacity per node) to ensure that cluster sizes stay small (between 1 and 100 nodes)
- Virtually eliminate partition formation as proven by years of deployments in data center and cloud environments
- Ensure extremely high consistency and availability during node failures and rolling upgrades (in the absence of partitions that are rare anyway)
- Provide automatic conflict resolution to ensure newer data overrides older data during cluster formation

More details of how Aerospike provides ACID consistency in the presence of node failures are described below.

Atomicity

For read/write operations on a single record, Aerospike makes strict guarantees about the atomicity of these operations:

- Each operation on a record is applied atomically and completely. For example, a read or a write operation on multiple bins in a record is guaranteed a consistent view of the record. The system performs all the checks upfront on the master that could lead to potential failures and creates an in-memory copy of the final record. If errors are encountered, no new record is written to storage and the old record will be kept intact. The client will be notified about the reason for the failure. Once the system is past the checks, there should not be any reason why the write should fail.

- After a write is completely applied and the client is notified of success, all subsequent read requests are guaranteed to find the newly written data: there is no possibility of reading stale data. Therefore, Aerospike transactions provide immediate consistency.

In addition to single record read/write operations, Aerospike supports distributed multi-key read transactions (batch/scan/query) using a simple and fast iteration interface where a client can simply request all or part of the data in a particular set. This mechanism is

currently used for database backup and basic analytics on the data and delivers extremely high throughput.

Consistency

In the context of RDBMS, consistency implies that the data must honor all the rules specified by correctness rules like check-constraints, referential integrity constraints (primary - foreign key), etc., at the end of every transaction. These are not currently applicable to Aerospike, as the database does not support such constraints yet.

However, in case of distributed systems, consistency can have a different meaning. Consistency as defined in the CAP theorem requires that all copies of a data item in the cluster are in sync. For operations on single keys with replication and secondary indexes, Aerospike provides immediate consistency using synchronous writes to replicas within the cluster, i.e., the client will be intimated about the successful write only if the replica is also updated. No other write is allowed on the record while the update of its replica is pending. This enables Aerospike to achieve a high level of consistency even in the presence of node failures and rolling upgrades provided the cluster exhibits no network level partitioning.

Multi-key read transactions are implemented as a sequence of single key operations and do not hold record locks except for the time required to read a clean copy. Thus the multi-key read transaction provides a consistent snapshot of the data in the database (i.e., no “dirty reads” are done).

In Aerospike, it is possible to relax immediate consistency if performance requirements outweigh the benefits of high consistency. For example, this may be necessary when write levels of the data are much higher than the read rates. Aerospike’s support for relaxing consistency models gives operators the ability to maintain high performance during the cluster recovery phase after node failure. For instance, read and write transactions to records that have unmerged duplicates can bypass the duplicate merge phase.

Isolation

All record writes in Aerospike are directed to the master nodes that atomically co-ordinate the writes across replica nodes. Aerospike provides **read-committed** isolation level using record locks to ensure isolation between multiple transactions. Therefore, when a bunch of read and a write operations for a record are pending simultaneously in parallel, they will be internally serialized before completion though their precise ordering is not guaranteed.

Aerospike also supports an optimistic concurrency control scheme using Check and Set (CAS) from the application layer. This allows multiple read-modify-write cycles from the client to not overwrite each other. User defined functions (UDF) running inside the database cluster nodes avoid the need to perform a read-modify-write cycle from the application layer as the UDF operations on a record are also serialized using record locks. So, when an operation within a UDF reads a copy of the record, it is guaranteed that no other concurrent transaction can change the value of this record until the execution of this UDF code is completed.

For multi-key read operations, the Aerospike client anchors the iteration operation and requests data from all the cluster nodes in parallel. Snapshots of the index are taken at various points to allow minimal lock hold times with reference counting so that the tree locks are held for the minimum possible time. As data is retrieved in parallel from the working nodes, it is forwarded to the client, with client flow control exerting backpressure on the distributed transaction.

Durability

Aerospike provides durability using multiple techniques:

- Persistence by storing data in flash/SSD on every node and performing direct reads from flash
- Replication within a cluster by storing copies of the data on multiple nodes
- Replication across geographically separated clusters

Durability is achieved by SSD based storage and replication. The replication to replica (or prole) nodes is done synchronously. I.e., the client application will be intimated only after the record is successfully written on the replica nodes. So, even if one node failed for any reason, we will have one more copy in another node thereby ensuring durability of the committed records. This is similar to the data + log file approach of traditional RDBMS where, If a data file is lost, the data can be recovered by replaying the log file. However, in case of Aerospike when one copy of the data is lost on a node, the data need not be recovered. Instead, the latest copy of the lost data is instantly available in one or more replica nodes in the same cluster as well as in nodes residing in remote clusters.

Aerospike also supports rack aware replication. Generally, in datacenters, multiple machines are mounted on several racks where each rack may share common infrastructure like power sources, network switches, etc. There is a slightly higher probability that all the machines on a rack will fail together (compared to the probability of all nodes across multiple racks failing at the same time). To mitigate this real-life situation, Aerospike lets nodes of the same cluster to be equally distributed across multiple racks. If such rack-aware configuration is enabled, Aerospike will avoid putting both master and replica copies of a partition within machines that are in the same rack. Therefore, even in the situation where an entire rack of machines fail together, there will

be a copy of all of the data in the failed rack elsewhere in the cluster, thereby ensuring high data durability.

On top of this, Aerospike's cross data center (XDR) support can be used to asynchronously replicate data to a geographically separated cluster providing an additional layer of durability. This will ensure sure that all of the data in the Aerospike database survives on a remote cluster even if an entire cluster fails and data is unrecoverable.

Resilience to simultaneous hardware failures

In the presence of node failures, applications written using Aerospike clients are able to seamlessly retrieve one of the copies of the data from the cluster with no special effort. This is because, in an Aerospike cluster, the virtual partitioning and distribution of data within the cluster is completely invisible to the application. Therefore, when application libraries make calls using the simple Client API to the Aerospike cluster, any node can take requests for any piece of data.

If a cluster node receives a request for a piece of data it does not have locally, it satisfies the request by generating a proxy request fetch the data from the real owner using the internal cluster interconnect and subsequently replying to the client directly. The Aerospike client-server protocol also implements caching of latest known locations of client requested data in the client library itself thus minimizing the number of network hops required to respond to a client request.

During the period immediately after a cluster node has been added or removed, the Aerospike cluster automatically transfers data between the nodes to rebalance and achieve data availability. During this time, Aerospike's internal "proxy" transaction tracking allows high-consistency to be achieved by applying reads and writes to the cluster nodes which have the data, even if the data is in motion.

Offsite data storage and cross data center portability

Aerospike provides online backup and restore, which, as the name indicates, can be applied while the cluster is in operation. Even though data replication will solve most real-world data center availability issues, an essential tool of any database administrator is the ability to run backup and restore. An Aerospike cluster has the ability to iterate all data within a namespace (similar to a map/reduce). The backup and restore tools are typically run on a maintenance machine with a large amount of inexpensive, standard rotational disk.

Aerospike backup and restore tools are made available with full source. The file format in use is optimized for high speed but uses an ASCII format, allowing an operator to validate the data inserted into the cluster, and use standard scripts to move data from one data store to another. The backup tool splits the backup into multiple files. This allows restores to occur in parallel from multiple machines in the case of needing a very rapid response to a catastrophic failure event.

Deployment modes – AP versus CP

In the presence of failures, the cluster can run in one of two modes – AP (available and partition tolerant) or CP (consistent and partition tolerant). The AP mode that Aerospike supports today prioritizes availability and therefore can be consistent only when partitions do not occur. In the future, Aerospike will add support for CP mode as an additional deployment option.

High consistency on AP mode

A key design point of Aerospike is to setup cluster nodes that are tightly coupled so that partitions are virtually impossible to create. This means that a replicated Aerospike cluster provides high consistency and high availability during node failures and restarts so long as the cluster does not split into separate partitions.

Here are the key techniques used in Aerospike that minimize and virtually eliminate network based partitioning:

- **Fast and robust heartbeats**: Aerospike heartbeats are sent at a regular pace. Note that the cluster nodes are expected to be close to each other thus requiring less than millisecond latency for node-to-node heartbeat messages. Heartbeats can be sent on UDP (in multicast mode) or on TCP (mesh mode), which is more reliable. On top of this Aerospike has a secondary heartbeat mechanism where the data transfer will augment the primary heartbeats. So, even if the primary heartbeat fails, if there are continuous read/write operations in the database, the cluster will be held together.

- **Consistent Paxos based cluster formation**: Aerospike uses a fast Paxos based algorithm to coalesce the cluster. The short heartbeat interval is critical since it enables the Paxos based algorithm to discover node arrivals and node departures extremely quickly and then re-coalesce the new cluster within tens of milliseconds. This is a case where Aerospike **prioritizes consistency over availability** for fleetingly small moments (low milliseconds) and the system quickly recovers availability as soon as the cluster coalesces. In practice, this short-term unavailability during cluster formation preserves consistency and is barely registered as a glitch on a system that routinely handles hundreds of thousands of transactions per second. Cluster node additions or removals are quite rare and these glitches are barely noticeable to the application.

- **High performance results in smaller clusters**: Aerospike can perform at extremely high scale, typically an order of magnitude better than comparable systems. Thanks to efficient hash-based algorithms, which avoid any hotspots, Aerospike can handle huge amounts of read/write transactional throughput without compromising on the performance or latency on a single node. By using high capacity SSDs, each node of the cluster can hold and serve a huge amount of data thus keeping the size of the cluster relatively small. Basically, an Aerospike cluster is virtually ten times smaller than comparable clusters of other databases that can handle the same load. Smaller cluster sizes means that in most cases, all Aerospike cluster nodes can be connected using the same switch with adequate fail-safe backup. This is recommended but not mandatory.

In addition to avoiding network partitioning, Aerospike uses additional techniques that ensure consistency during node failures and rolling upgrades:

- **Single node failures**: When using replication factor >= 2, if a single node fails then the remaining nodes will have all the data of the cluster. We do automatic re-balancing (migration) of the data between the surviving nodes. While the migration is going on Aerospike allows writes to continue. To make sure that no writes are lost in a race condition between the act of re-balancing and accepting writes, Aerospike maintains a journal of changes that will be reapplied, after proper checks, at the end of a partition's migration.

- **Rolling upgrades**: Upgrading the software is a common requirement. The software may be upgraded in more number of cases (thereby necessitating a node down) than the case of unplanned failure of the nodes. This is a non-issue with Aerospike because of how Aerospike handles single node failure cases very gracefully without any data loss, as explained above.

- **Transaction repeatable read setting**: When multiple nodes have merged into the cluster in a short amount of time, there may be many copies of the record created in the cluster. But only one version of the record is the correct one. To overcome such a scenario, Aerospike provides a configuration option to enable repeatable read. For every read operation, when repeatable read is enabled, Aerospike will consult all nodes in the cluster that claim to have data belonging to the partition, pick the most recent record (correct record), and return it to the client. The system will continue to use this merge algorithm during read operations until all the copies of a partition are merged as part of the migration process. Therefore, there cannot be a situation where different read requests will return different versions of the data (repeatable read requires this). Note that this option comes at the cost of higher latency caused by checking all copies of a record in the cluster before returning to the client. In this case, therefore, the application achieves higher consistency while getting lower availability that is manifested as higher latency for the read operation.

Aerospike continues to improve reporting of consistency errors by performing all the preliminary checks up front before writing to the disk on the master. This will ensure that writing on the master node is fail-safe. However, the system could fail in writing to one or more replica nodes.

- If the replica node communicates back an error, the write is retried as the prole can fail only due to temporary reasons like timeout or running out of disk space (which is continuously garbage collected).

- The tougher case is the one in which a prole did not communicate back anything to the master and the network connection is lost. The master will not know if the record was successfully written or not. In this case, the client should receive an "Unknown" transaction state back.

Handling conflicts

Finally, we describe how Aerospike handles the case if a partitioning of the cluster were to happen. Note that, in AP mode, when a cluster splits into multiple partitions or factions, each faction of the cluster continues to operate. One faction – or the other – may not have all of the data, so an application reading data may have successful transactions stating that data is not found in the cluster. Each faction will begin the process of obeying the replication factor rules (replicating data) and may accept writes from clients. Application servers that read from the other faction will not see the applied writes, and may write to the same primary keys. At a later point, if the factions rejoin, data that has been written in

both factions will be detected as inconsistent. Two policies may be followed. Either Aerospike will auto-merge the two data items (default behavior today) or keep both copies for application to merge later (future).

Auto merge works as follows:

- TTL (time-to-live) based: The record with the highest TTL wins
- Generation based: The record with the highest generation wins

Application merge works as follows:

- When two versions of the same data item are available in the cluster, a read of this value will return both versions, allowing the application to resolve the inconsistency.
- The client application – the only entity with knowledge of how to resolve these differences – must then re-write the data in a consistent fashion.

Availability in CP Mode

In CP mode, when the cluster splits into two or more active clusters, availability needs to be sacrificed. For example, the minority quorum(s) could be made to halt. This action prevents any client from receiving inconsistent data, but will reduce availability. Aerospike smart clients can also be made to detect cluster-partitioning occurrences and act appropriately to restrict access to exactly one of the partitions. In this scenario, per the CAP theorem, the system will be prioritizing consistency over availability in order to allow for partition-tolerance.

There will be use cases where availability can be sacrificed and a CP system is needed. To enable Aerospike to be used in more domains, we plan to add a configuration for operating the cluster in CP mode in addition to the AP mode that is supported now. The actual mode of an individual cluster will be configurable by the operator based on their needs.

Aerospike's first step towards a CP system would be to support a static cluster size. The static cluster concept works as follows:

- The idea of static cluster is to predefine the nodes of the cluster. The system administrator is allowed to specify the exact set of cluster nodes and engage a **"static-cluster"** switch so that the composition of the cluster is fixed to include exactly all of the current nodes of the cluster.
- While the static-cluster switch is engaged, any cluster state change will not automatically result in changes to the partition map. The partition map determines how master and replicas of partitions are stored in nodes. I.e., the partition map is fixed and no migration of data is allowed whenever the static-cluster switch is engaged.

- The operator needs to disengage the static-cluster switch before the cluster configuration can be changed in a controlled manner to add or remove nodes. The operator then needs to wait for all migrations to be completed before re-engaging the static-cluster switch.

- If the cluster is split into one or more islands, and if the client sends a request to one of the nodes in an island, the following happens:

    - If the request is a read request, and one of the nodes in the island has a copy of the data, the read request will be serviced.

    - If the request is a write request, and the node is the master (or the master is in the island), it will perform the write only if all the replicas are in the same island. Our experience tells us that when there is network based cluster partitioning, it is normally one or a very few nodes that get separated. In the fairly common scenario where only one node gets separated from its cluster, only $1/n^{th}$ of the data is unavailable for writes (where n is the number of nodes in the cluster). This may be quite acceptable in order to obtain full consistency in the presence of partitions.

- Bigger sized islands may form if the nodes are connected through a hierarchy of switches and the intermediate (non-leaf) switch fails. So, we intend to continue our current recommendation that all the nodes of Aerospike should be connected to the same switch, if possible. However, it is not mandatory.

Summary

This document has provided a brief technical overview of ACID support in Aerospike. So far, Aerospike has focused on continuous availability and provides the best possible consistency in an AP system by using techniques to avoid partitions. Going forward, the system will be enhanced to alternatively support a CP configuration that will allow complete consistency in the presence of network partitioning by reducing availability.