AEROSPIKE

WHITE PAPER

Five Signs You Have Outgrown Cassandra (and What to Do About It)

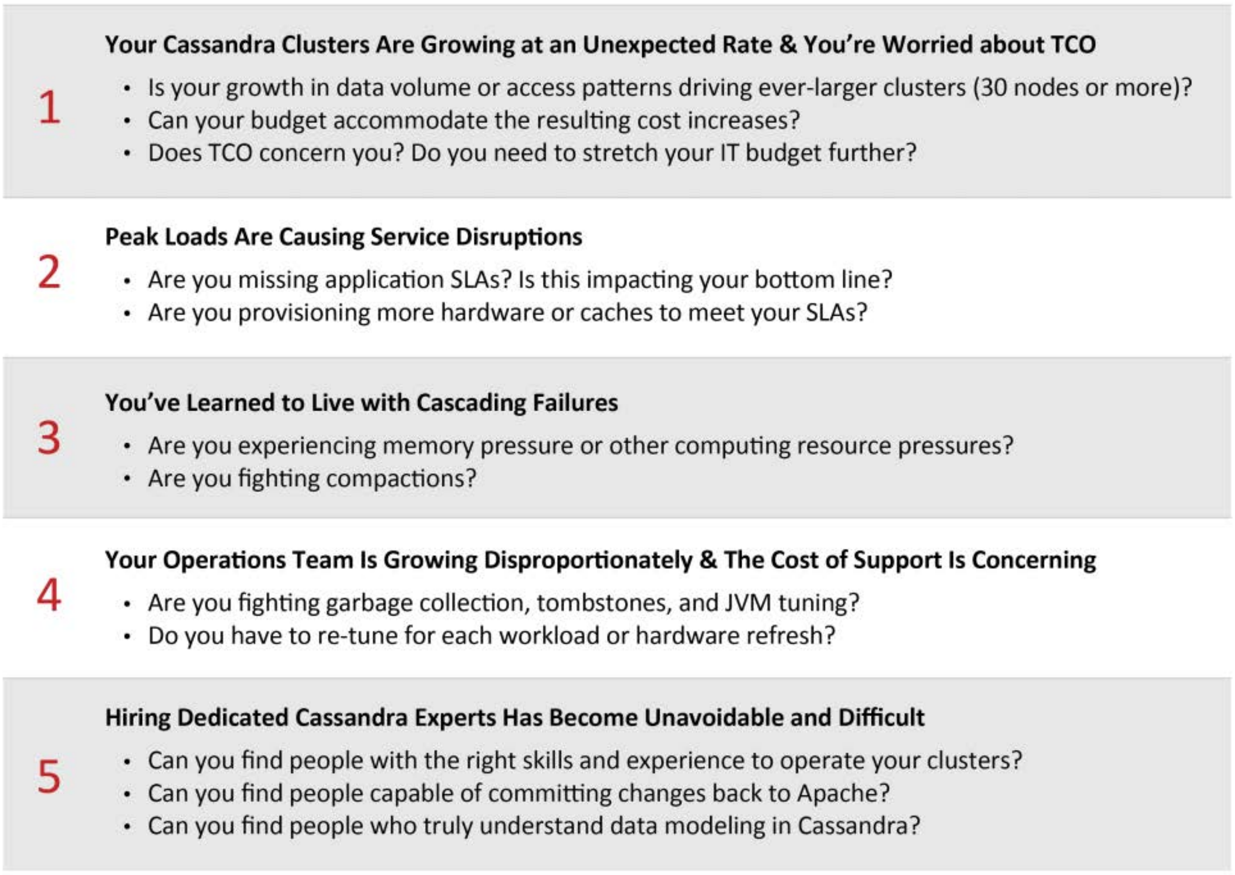
Executive Summary

Cassandra is a well-known NoSQL database, maintained under the Apache Foundation and commercialized by a number of companies. While it's easy for organizations such as yours to start with Cassandra, you find yourself (or soon will be) facing increasingly large costs and complexity of day-to-day operations as your application load grows.

This impacts not only your Line of Business (LOB) budget, but also your operational stability, and further, your customer experience. Your Cassandra infrastructure hampers your organization's ability to be agile, to compete, and to bring new products and services to market. Aerospike, the leading enterprise-grade NoSQL database, can save you 5x or more in Total Cost of Ownership (TCO) while providing proven, unparalleled uptime and availability. Aerospike is used in production and trusted by industry-leading organizations for their mission-critical applications.

Five Signs

What are the five signs that your company may have outgrown Cassandra?



1 **Your Cassandra Clusters Are Growing at an Unexpected Rate & You're Worried about TCO**

- Is your growth in data volume or access patterns driving ever-larger clusters (30 nodes or more)?
- Can your budget accommodate the resulting cost increases?
- Does TCO concern you? Do you need to stretch your IT budget further?

2 **Peak Loads Are Causing Service Disruptions**

- Are you missing application SLAs? Is this impacting your bottom line?
- Are you provisioning more hardware or caches to meet your SLAs?

3 **You've Learned to Live with Cascading Failures**

- Are you experiencing memory pressure or other computing resource pressures?
- Are you fighting compactions?

4 **Your Operations Team Is Growing Disproportionately & The Cost of Support Is Concerning**

- Are you fighting garbage collection, tombstones, and JVM tuning?
- Do you have to re-tune for each workload or hardware refresh?

5 **Hiring Dedicated Cassandra Experts Has Become Unavoidable and Difficult**

- Can you find people with the right skills and experience to operate your clusters?
- Can you find people capable of committing changes back to Apache?
- Can you find people who truly understand data modeling in Cassandra?

About Aerospike

Founded in 2009, Aerospike has diligently focused on building a mission-critical, highly available, distributed, and record-oriented key-value NoSQL database. Aerospike powers the AdTech industry; its customers include AdForm, Applovin, AppNexus, BlueKai, InMobi, Rubicon Project, Trade Desk, and many others. Aerospike also drives innovation in a number of other sectors, including Telecommunications (with Nokia, HP Enterprise, Airtel, NTT, and Viettel), Financial Services, Gaming (with King, DraftKings, and Curse), and eCommerce (with Williams-Sonoma and Kayak).

Designed and built to exploit the characteristics of Flash/SSD and poised to take advantage of storage class memory, Aerospike provides unprecedented value to its customers. Our technology is driving fundamental changes in how people think about, store, and access their data; it's the key ingredient for building rich, engaging applications and services. Aerospike is driving digital transformation across many industries by enabling our customers to build relevant systems of engagement; this includes better recommendation engines in retail and marketing, fraud prevention in payment processing and cybercrime detection, and billing and service enablement in telecommunications. Aerospike's combination of extraordinary uptime, high availability, and consistent performance all but eliminates service disruptions for your customers.

Five Signs You Have Outgrown Cassandra

There are many business and technical demands driving your organization: delivering new applications faster, reducing costs, providing a reliable and engaging experience, maintaining your Net Promoter Score, driving digital transformations, and more. How do these map to the signs that you have outgrown your Cassandra cluster?

Sign #1: Your Cassandra Clusters Are Growing at an Unexpected Rate & You're Worried about TCO

It's a dirty little secret: the NoSQL community and vendors have encouraged you to build big - really big - database clusters. It became a matter of honor to be running thousands of nodes (and in Apple's case, 75,000 nodes [1]). But what are the consequences of such expansive clusters? In a large cluster, the actual probability and incidence of having a node fail on you goes from theory to a daily - if not hourly - occurrence. It's dozens of hard drives or SSDs per server, over hundreds or thousands of servers. Vendors directly benefit from your big clusters. Ever notice how they price by the node? There's no incentive for them to reduce their number.

[1] http://cassandra.apache.org/

To illustrate our point, let's use the data presented on drive failures by Google at Fast16. This research showed a failure rate of 1-2% for SSDs and 2-20% for HDDs. What does this mean in practical terms? Let's start with one server with one drive and work our way up to 75,000 servers each with 4 drives, as you would see in very large deployments. The failure rates in increasingly large server deployments are illustrated in Table 1 below:

Number of Servers	**Number of Drives Per Server**	**Failure Rate SSD 2% per year**	**Failure Rate HDD 10% per year**
1	1	1 every 50 years	1 every 10 years
100	1	2 per year	10 per year
100	2	4 per year	20 per year
1,000	2	40 per year 0.8 per week	200 per year 3.8 per week
1,000	4	80 per year 1.5 per week	400 per year 7.6 per week
75,000	4	6,000 per year 16 per day Every 90 minutes	30,000 per year 82 per day Every 17 minutes

Table 1. Observable failure rates in server deployments

The larger the cluster, the more components you have; hence, hardware failure goes from a mere possibility to a practice that occurs daily, if not hourly. Server sprawl creates more hardware failures that your operations teams need to deal with. By contrast, smaller clusters mean fewer components, which reduces the number of actual failures with which you must deal.

Cassandra does a great job of horizontal scaling: you simply add more nodes. The more important question is, are you able to fully utilize each node before you need to buy, provision and manage another… and another... and another? You know the answer: you have found that Cassandra cannot fully utilize a database node. Thus, when you hit a resource limit - either CPU, storage IOPs, or DRAM for the JVM heap - your only alternative is to scale out. Yet each node you add creates more complexity. Each node you add also results in significantly greater cost, because Cassandra vendors like to charge by the node. Further, reliability suffers: the law of large numbers means that you will see actual failures, and see them more frequently than you can imagine.

As Cassandra renders you unable to utilize the performance of your servers in its entirety, you are thus forced to perform some unnatural acts best described in the following way by the Cassandra community and its committers:

> *"Instead of scaling the compute side over the metal, we do silly things like run multiple instances per box…"* [2]

Indeed, running multiple Cassandra instances per box is so common that DataStax, one of the Cassandra vendors, created a Multi-Instance feature as part of their Enterprise version to automate this deployment topology. However, running multiple instances per server just adds further operational complexity and compounding failure modes when a server goes down.

Why is Cassandra so inefficient with compute resources? The use of Java - with multiple JVM providers, and with a number of garbage collection (GC) strategies (e.g., HotSpot's CMS, G1, etc.) - create many variables that developers and ops people can try to optimize and tune [3] [4]. To get the most out of a node, you need to carefully read the logs, adjust JVM parameters, debug [5], look at thread dumps [6], etc. Naturally, you need to do so for each different workload and cluster configuration, especially if the hardware is different. And when you upgrade the hardware on your existing cluster? Yes, you need to retune all over again. What if you add a new workload to existing data? You've guessed it - you need to retune.

This tuning is difficult, and changes are prone to error [7]. Most ops teams find it easier to expand the cluster using the same configs and hardware profile. It's a practical - though costly - approach for the Line of Business owner, IT budget owner, or whoever has to pay the bill.

Think back on how you initially sized your cluster. How did you accomplish this task? You followed the best practices from numerous community blogs, the Apache wiki, or documentation from one or more of the vendors. Hopefully, you took into account the additional storage space needed depending on your choice of compaction strategy (STCS vs. LCS). You may have taken into account space for snapshots. Were you then surprised when the application was deployed and used a huge amount of additional storage space? This is where you needed to know with precision which features the application team used when the application was constructed, as noted by one community user:

> *"...we attempted to use a CQL Map to store analytics data, we saw 30X data size overhead vs. using a simpler storage format and Cassandra's old storage format, now called COMPACT STORAGE. Ah, that's where the name comes from: COMPACT, as in small, lightweight. Put another way, Cassandra and CQL's new default storage format is NOT COMPACT, that is, large and heavyweight."* [8]

[2] https://issues.apache.org/jira/browse/CASSANDRA-7486
[3] https://issues.apache.org/jira/browse/CASSANDRA-8150
[4] https://issues.apache.org/jira/browse/CASSANDRA-7486
[5] https://alexzeng.wordpress.com/2013/05/25/debug-cassandrar-jvm-thread-100-cpu-usage-issue/
[6] https://support.datastax.com/hc/en-us/articles/204226009-Taking-Thread-dumps-to-Troubleshoot-High-CPU-Utilization
[7] https://tobert.github.io/pages/als-cassandra-21-tuning-guide.html
[8] http://blog.parsely.com/post/1928/cass/

As we will describe later, how you model data - and which features you use in Cassandra - dramatically affects your utilization, reliability and response times.

Finally, your compaction and backup strategy can also have a huge impact on your CAPEX. Because you are reliant on compaction to reduce storage requirements, you may end up backing up older generations of the data again and again until the compactions can catch up. This may require significant additional storage capacity, as was noted by Rohit Shekhar of Datos.io in his team's experiments:

> *"Case in point: [...] secondary storage was as high as 12 times the primary storage for level compaction."* [9]

Sign #2: Peak Loads Are Causing Service Disruptions

Ingesting huge amounts of data - either periodically or as part of the regular usage of the application- can be critical in many applications. However, as the data is written, the application will need to read and modify the same data; such mixed workloads constitute the normal pattern for applications like activity streams, profile stores, trade stores, etc. Write-once workloads, where the data is never modified, like log streams, are not the norm.

If mixed reads and writes are such a common use case, why is this pattern such a problem for Cassandra? Quite simply, this is due to an architectural choice made by the designers of Cassandra: namely, its log-structured file system and the eventual consistency of data.

Kyle Kingsbury's (a.k.a. @aphyr's) post about Cassandra [10] states that without vector clocks, Cassandra has to rely on a very precise usage model from its users. Without adhering to these models, Cassandra will lose acknowledged writes, meaning that there are few guarantees to read the correct information. As a developer, you can try to code around the problem with various consistency models [11], so you can at least get a quorum across the copies held across the nodes of the clusters for reads and writes. This adds unpredictable latency to any operation, as the operation can only be as fast as the slowest node. The most common solution is to cache more of the data to avoid disk reads; this leads to larger clusters and more DRAM, violating many of the tuning guides regarding the size of the JVM heap.

That's not the only challenge for mixed workloads, as was noted by the venerable Oracle corporation:

> *"Cassandra uses consistent hashing over a peer-to-peer architecture where every node in the system can handle any read-write request, so arbitrary nodes become coordinators of requests when they do not actually hold the data involved in the request operation. That means both an extra network hop (minimum) for each call and it means the failure of a single node can have*

[9] https://datos.io/backup-challenges-cassandra-compaction/
[10] https://aphyr.com/posts/294-jepsen-cassandra
[11] https://docs.datastax.com/en/cassandra/2.0/cassandra/dml/dml_config_consistency_c.html

> *system wide performance impacts as other arbitrary nodes change their behavior in response to the failed node."* [12]

One Cassandra vendor also acknowledged this behavior in their documentation:

> *"Client read or write requests can be sent to any node in the cluster. When a client connects to a node with a request, that node serves as the coordinator for that particular client operation. The coordinator acts as a proxy between the client application and the nodes that own the data being requested. The coordinator determines which nodes in the ring should get the request based on how the cluster is configured."* [13]

Thus, even in healthy clusters, there are inevitable network hops to service the simplest of requests. These compounding factors lead to a wide variance in read latencies. Choosing a sensible partition key is only a partial solution: it simply limits the number of nodes that must be checked rather than eliminating the need in the first place.

During any situation where nodes become unavailable, further memory pressure [14] is applied to the coordinator node, since it needs to keep track of any hinted handoff for writes that will need to be re-applied later. This memory pressure can lead to instability throughout the cluster, as we will see in the next sign.

Compactions add another layer of complexity. In any log-structured merge file system, you need to periodically prune and compress the trees, removing older and redundant version of the data and cleaning tombstones (deleted records). Cassandra has both major and minor compactions, which the community spends a lot of time figuring out how to tune for the given workload and hardware [15]. This is a significant problem: during the time compactions run, they adversely affect the read and write latency and throughput of operations, and impact your SLAs (again). You know this when your log files start to get sprinkled with the following types of error messages:

```
Scanned over 100000 tombstones; query aborted
org.apache.cassandra.db.filter.TombstoneOverwhelmingException
```

How can you better deal with peak load, then? You will want to expand your Cassandra cluster as part of your regular capacity planning, or to deal with seasonal events like holiday sales. But don't wait until the last moment to expand your cluster, or you risk being too late. Some guides, such as the Threat Stack Blog, state:

> *"Budget days to bring a node into the cluster. If you've vertically scaled [with fewer large nodes], then it will take over a week."* [16]

[12] http://www.oracle.com/technetwork/database/nosqldb/overview/ondb-cassandra-hbase-2014-2344569.pdf
[13] http://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archIntro.html
[14] http://www.datastax.com/dev/blog/modern-hinted-handoff
[15] https://medium.com/@foundev/how-i-tune-cassandra-compaction-7c16fb0b1d99#.78lo047w7
[16] http://blog.threatstack.com/scaling-cassandra-lessons-learned

And as you expand your Cassandra cluster, expect that this will have operational impact and that your application will have missed SLAs. As one community user remarked:

> *"The bottom line, is that your queries do have a higher chance of failing before the new node is fully-streamed."* [17]

Sign #3: You've Learned to Live With Cascading Failures

For mission-critical systems, availability is the most crucial aspect of a data layer. After all, isn't availability why you picked AP from the CAP theorem and selected Apache Cassandra in the first place? You've chosen a system that has distribution and replication of data, so when a node becomes unavailable - momentarily or permanently - the data lives elsewhere. Right?

Wrong, actually. Data distribution sounds well and good (and is the correct solution), but if a single node outage causes a cascading failure across your cluster, every node becomes the single point of failure for the cluster. And cascading failures are common [18] [19], especially when CPU pressure causes nodes to stop self-reporting [20] and inflicting cluster rebalances, causing further CPU and I/O pressure on the surviving nodes. As one Cassandra user noted, "Cassandra seems to have two modes: fine and catastrophic". [21]

The failure of one node has often been observed to cause cascading failures [22] [23] across the whole cluster. Research papers have shown these problems to be systemic with Cassandra [24].

What, then, are the typical sources of cascading failures? They include:

- Memory pressure caused by hinted handoff during failover
- Compactions trashing the row cache
- I/O and memory pressure from memtable flushes during high load
- Compactions causing I/O, and thus, CPU pressure
- Compactions not occurring fast enough, causing memory pressure
- Memory use causing frequent garbage collection, and thus, CPU pressure
- A large number of tables, causing memory pressure

Let us tackle each of these in turn.

Memory Pressure caused by hinted handoff during failover - As noted by one Cassandra vendor in several pages of their documentation, the cause of this is clear. If you rely on Cassandra's ability to store writes on a coordinator node to replay later when the designated node returns to the cluster, this

[17] http://stackoverflow.com/questions/37283424/best-way-to-add-multiple-nodes-to-existing-cassandra-cluster
[18] http://danluu.com/postmortem-lessons/
[19] https://www.usenix.org/conference/osdi14/technical-sessions/presentation/yuan
[20] https://moz.com/devblog/cassandra-in-production-things-we-learned/
[21] http://www.slideshare.net/planetcassandra/pd-melting-cass/12?src=clipshare
[22] http://mail-archives.apache.org/mod_mbox/cassandra-user/201106.mbox/%3CBANLkTinqdatJs2u-khFngboSP6h-621=Pw@mail.gmail.com%3E
[23] http://www.stackdriver.com/post-mortem-october-23-stackdriver-outage/
[24] http://ucare.cs.uchicago.edu/pdf/socc14-cbs.pdf

functionality consumes large quantities of memory; thus, another node’s outage causes significant memory pressure on all the other nodes:

> *"If this happens on many nodes at once this could become [sic] substantial memory pressure on the coordinator. So the coordinator tracks how many hints it is currently writing, and if this number gets too high it will temporarily refuse writes (with* `UnavailableException` *) whose replicas include the misbehaving nodes."* [25]

This is not just a theoretical problem; it's a very real one that’s a function of your data usage and data design with hinted handoff, as was noted by one Cassandra user:

> *“Serializing the big rows causes high memory pressure…”* [26]

Users of Cassandra often recommend disabling hinted handoffs - and thus reducing availability - to avoid cascading failures:

> *“Don’t use hinted handoffs (ANY or LOCAL_ANY quorum). In fact, just disable them in the configuration. It’s too easy to lose data during a prolonged outage or load spike, and if a node went down because of the load spike you’re just going to pass the problem around the ring, eventually taking multiple or all nodes down”* [27]

Compactions trashing the row cache - As noted by another Cassandra vendor, compactions, which increase during a single node outage as greater load is applied on surviving nodes, also increase pressure on the I/O, memory, and CPU of those nodes:

> *“Cassandra compaction thrashes the [O/S] page cache, because it reads and writes everything, and after compaction the most frequently used data is likely to no longer be in the cache.”* [28]

A Cassandra cluster is often sized using assumptions about the effectiveness of the row cache; an ineffective row cache leads to a greater number of connections and transactions in flight. This causes difficulty (at the very least, performance issues) for surviving nodes. However, the effects of the cache are negated when the blocks being cached are paged out by another database feature.

I/O and memory pressure from memtable flushes during high load - Flushing memtables is critical because writes are blocked until the flush succeeds [29]. But there is a cascading effect if flushing is not tuned correctly:

> *“... proper tuning of these thresholds is important in making the most of available system memory, without bringing the node down for lack of memory.”* [30]

Indeed, during a node outage, your carefully selected tuning becomes invalid.

[25] http://www.datastax.com/dev/blog/modern-hinted-handoff
[26] http://java.cz/dwn/1003/72451_CassandraCZJUG_horky.pdf
[27] http://blog.threatstack.com/scaling-cassandra-lessons-learned
[28] http://www.scylladb.com/technology/memory/
[29] https://docs.datastax.com/en/cassandra/2.1/cassandra/dml/dml_write_path_c.html
[30] https://wiki.apache.org/cassandra/MemtableThresholds

Compactions causing I/O, and thus, CPU pressure - Choosing Leveled vs. Size-Tiered compactions will dramatically change I/O and CPU pressure. This is a function of the reads and writes at this moment in time: a predominantly read-heavy application will get the adverse effects of a data load job, causing pressure on both I/O and CPU.

As was noted by one vendor:

> *"Since SSTables are immutable, this process puts a lot of pressure on disk io as SSTables are read from disk, combined and written back to disk."* [31]

Compactions not occurring fast enough, causing memory pressure - The immutable nature of Cassandra's log structure means that the process of compactions is not only inevitable; depending on your data access patterns, it may downright imprison you. Indeed, when "the compaction is not able to complete" [32] , this causes unavailability. As was also expressed on Target's tech blog:

> *"The nodes would OOM frequently when compacting a specific column family… What I discovered is that Cassandra was reading a lot of tombstones each time, and this was putting lots of extra data on the heap. This would just snowball when the cluster was under load, and blow the heap."* [33]

Memory use causing frequent garbage collection, and thus, CPU pressure - With a Java-based code base, garbage collection is inevitable and uncontrollable. Tuning is possible, but often described as a "dark art". Unfortunately, the side effects of garbage collection are real, as users report:

> *"...garbage collection was happening 20+ times a second, even when Cassandra was under tiny load."* [34]

> *"In both cases the C* nodes end up doing garbage collection for ~90 secs per sweep"* [35]

This impacts the latency of responses and throughput of the system, as vital system resources are used to manage memory.

A large number of tables, causing memory pressure - The way in which you constructed the data schema can also impact the memory pressure, and changes to application design and use can radically change hardware requirements. As noted by Ryan Svihla, a Solution Architect at DataStax:

> *"There is in general a cluster max effectively [sic] limit on table counts. Anything over 300 starts to create significant heap pressure."* [36]

[31] http://www.planetcassandra.org/blog/impact-of-shared-storage-on-cassandra/
[32] http://stackoverflow.com/questions/29273276/cassandra-node-heap-pressure-during-compaction-after-bulk-load
[33] http://target.github.io/infrastructure/tuning-cassandra
[34] http://target.github.io/infrastructure/tuning-cassandra
[35] http://stackoverflow.com/questions/29273276/cassandra-node-heap-pressure-during-compaction-after-bulk-load
[36] https://medium.com/@foundev/domain-modeling-around-deletes-1cc9b6da0d24#.goi7cxibs

Sign #4: Your Operations Team Is Growing Disproportionately & The Cost of Support Is Concerning

The number of aspects that an operational team must consider at cluster provisioning time is large. This leads to a time-consuming process for provisioning each cluster. Worse, the operations team must continually monitor Cassandra clusters for changes in application patterns, and re-tune the clusters on a frequent basis. Failure to retune leads not only to poor performance, but also (eventually) to CPU pressure, which limits garbage collection capabilities; this causes memory pressure, and in turn, outages.

The community and committers are well aware that Cassandra cannot utilize compute resources effectively, as can be seen on Cassandra's own issue tracking system:

> *"Instead of scaling the compute side over the metal, we do silly things like run multiple instances per box. It's not really silly if it gets results, but it is an example of where we do something tactically, get so used to it as a necessary complexity, and then just keep taking for granted that this is how we do it."* [37]

In order to increase utilization, clusters are forced to get wider, and multiple Cassandra nodes are commonly required on the same compute node. This pattern greatly increases operational complexity, necessitating not just more time from your operations staff to plan and deploy, but also more complex and compounding failure modes to diagnose and fix.

The setup of the JVM and other tuning parameters for the specific workload and hardware means there is no "out of the box" setting that will consistently work. Tuning is inevitable, as the committers of Cassandra themselves have noted:

> *"... there's a bunch of different workloads and a bunch of different hardware that C* runs on, and the idea of having a default that's optimal for everyone is unrealistic. It may very well be that G1 is a better "good enough" default for most distributions, large heap or no, and that's the conversation on IRC… "* [38]

The fact that tuning is a necessity can have a significant impact on day-to-day operations, as Daniel Parker, an Engineer at Target, observed:

> *"We were having to restart nodes frequently to clear the heap. This was not an option to continue doing, especially pre-peak when we expect nearly 10x the traffic volume."* [39]

[37] https://issues.apache.org/jira/browse/CASSANDRA-7486
[38] https://issues.apache.org/jira/browse/CASSANDRA-10403
[39] http://target.github.io/infrastructure/tuning-cassandra

Sign #5: Hiring Dedicated Cassandra Experts Has Become Unavoidable and Difficult

Technical resources are always hard to find, employ and retain. Finding staff with deep expertise - even Apache code committers - is a step beyond most organizations' capabilities; yet, this is often recommended to effectively operate Cassandra clusters. Is it truly a badge of honor to have a team of committers like Netflix or Apple [40], or even employing and motivating a team of "Cassandra whisperers" [41]? Does your business cost structure allow several hundreds of thousands of dollars per year in head-count to maintain free software? Can you justify buying a company [42] just for its Cassandra expertise? As Ted Wallace, VP of Data Delivery at BlueKai, noted:

> *"We ultimately found out that to do Cassandra you need people who are focused on keeping Cassandra alive and running. We didn't want to invest in creating a team of 'Cassandra whisperers'. We didn't want to be experts at managing Cassandra"* [43]

Your operations staff must also understand the differences between the multiple versions of Cassandra, and be able to tune effectively for different versions. Will your staff choose the generic Apache Cassandra distribution, or a vendored version created by your Cassandra distributor? If your team selects a Cassandra distributor, will they provide current releases? Will they keep up with the community for features, bug fixes, vulnerabilities and currency, or will your operations staff need to patch a vendor distribution with fixes from the open source version? Even a core committer like DataStax will defer the migration from Apache Cassandra 2.1 to 3.0 in their enterprise releases; is your organization willing to wait? As narrated by DataStax themselves in 2016:

> *"Today, it typically takes DataStax four to six months to certify a new, major version of open source Cassandra and ensure it is ready for enterprise deployments. This time may shorten as the tick-tock process drives down defect rates."* [44]

[40] http://www.planetcassandra.org/mvps/
[41] http://www.aerospike.com/blog/bluekai-nosql-speed-scale-simplicity/
[42] http://appleinsider.com/articles/15/03/25/apple-acquires-big-data-analytics-firm-acunu
[43] http://www.aerospike.com/blog/bluekai-nosql-speed-scale-simplicity/
[44] http://web.archive.org/web/20160322221453/http://www.datastax.com/2016/01/comparing-open-source-apache-cassandra-and-datastax-enterprise-release-models

The reality is that it often takes a minimum of eight months to vendor in Apache code. Here is an example of the release timeline for the DataStax Enterprise Edition:

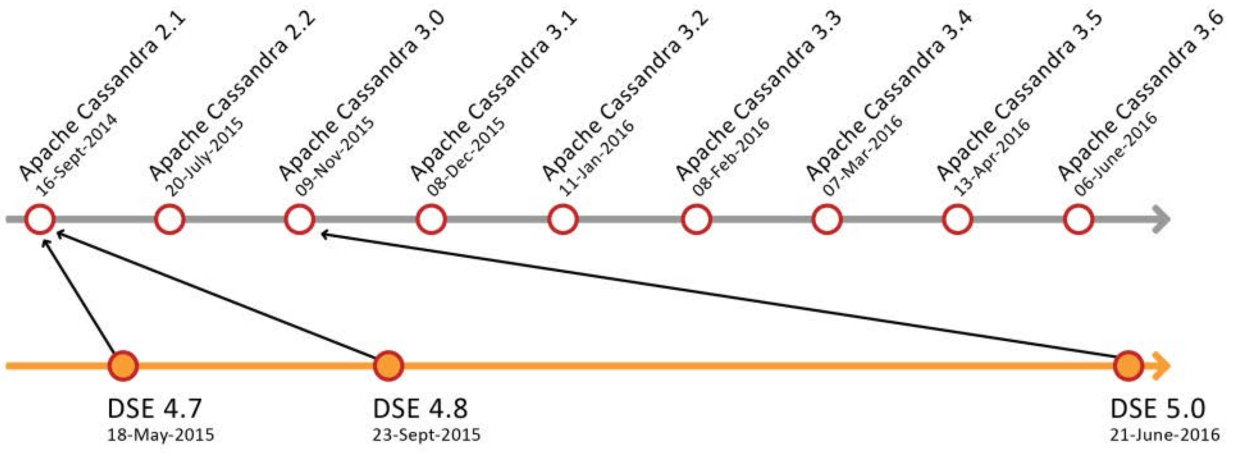


Figure 1. Vendoring in Cassandra releases from Apache

You will be left with a choice between the following options: wait until a patch is vendored in, uptake a newer Apache release yourself (and abandon your support subscription), or fix the code base you have.

Committing changes between distributions is just one more complex task for your operations staff to undertake. Your staff's harder challenge occurred much earlier in the process - when the application developers designed the schema, primary and partition keys, and decided on which features to use. As Juan Valencia, Principal Engineer at ShareThis, offered:

> *"We've made a lot of mistakes in data modeling over the course of development. Setting up our data models correctly was tricky."* [45]

Distributed counters seem like a reasonable feature for a distributed database, especially when performing real-time analytics, and likely, the application team chose them. But as noted by Andrew Montalenti, CTO of Parse.ly:

> *"When I'm in a good mood, I sometimes ask questions about Counters in the Cassandra IRC channel, and if I'm lucky, long-time developers don't laugh me out of the room. Sometimes, they just call me a "brave soul"... All of this is to say: Cassandra Counters — it's a trap! Run!"* [46]

Your operations staff can't rest by just creating a checklist of features used; they must know how the data is used, and what its lifecycle is. Consider something simple like a queue, where you need to maintain order and also expunge data as soon as it's processed. That simple data model design leads to

[45] http://www.informationweek.com/strategic-cio/why-we-picked-cassandra-for-big-data/a/d-id/1318250
[46] http://blog.parsely.com/post/1928/cass/

operational problems down the road, as Cassandra then needs to manage large numbers of tombstones (deleted records). As Ryan Svihla, a Solution Architect at DataStax, remarked:

> *"You realize that based on your queue workflow instead of 5 records you'll end up with millions and millions per day for your short lived queue, your query times end up missing SLA and you realize this won't work for your tiny cluster."* [47]

The use of TTL (Time-To-Live) as a mechanism to remove data after a specified time period is another potential trap. As one Cassandra user observed in their own attempts to use TTLs:

> "... [TTL auto-expire] was able to effect a denial of service for all logins through creating a large amount of garbage [tombstone] records. Once the records for these failed logins had expired, all queries to this table started timing out." [48]

Finally, your operations staff must absorb the consequences of your development staff picking the wrong primary or partition key. A poor choice inevitably ends up with hotspot nodes, which can cause one or more of the following: high memory, CPU pressure, or I/O pressure. This leads to the kind of cascading failures described above. One story from the community demonstrates this chain reaction:

> *"The failing node, in fact, was a hotspot! Because of an error in a primary key of one of a high loaded table! Table's data was not properly distributed across all cluster nodes. And the large portion of data was concentrated on that node! This led to two problems:*
> *1 - significant amount of queries (read/write) were addressed to that node;*
> *2 - huge keys - about ~5 megs per key;*
> *These two problems led to node load and, due to huge keys, instability (high pressure on GC)."* [49]

[47] https://medium.com/@foundev/domain-modeling-around-deletes-1cc9b6da0d24#.goi7cxibs
[48] https://www.tildedave.com/2014/03/01/application-failure-scenarios-with-cassandra.html
[49] https://www.reddit.com/r/cassandra/comments/3uzlnp/cassandra_high_gc_pressure/

Aerospike: The Enterprise-Grade NoSQL

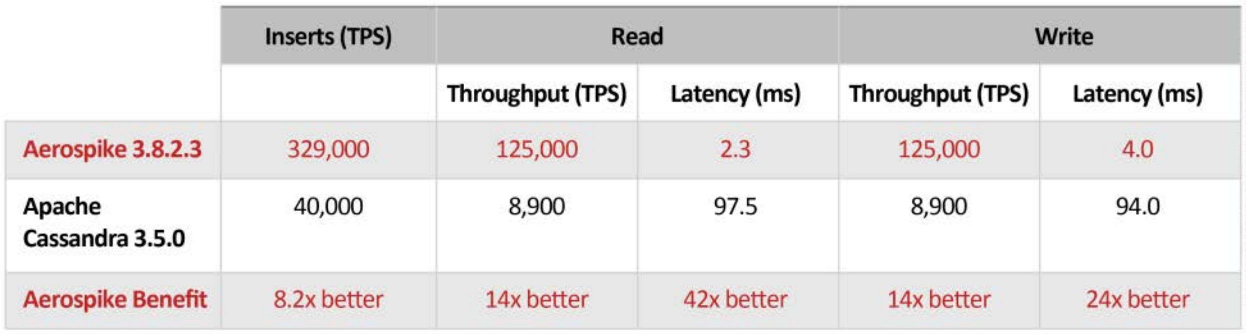
As noted earlier, Aerospike was designed and built from the ground up to take advantage of modern computing architectures. Aerospike undertook an alternative approach, building from a clean sheet a technology stack and fundamental IP that has enabled its customers to obtain and enjoy an unprecedented low TCO (Total Cost of Ownership), unparalleled uptime, high availability, and reduced database infrastructure complexity. Let's explore how Aerospike has achieved this.

Reduced TCO

Aerospike is written in C by a development team with deep expertise in networking, storage and databases. By removing the layers of file system, block, and page caches, and instead building a proprietary log-structured file system designed for the way flash devices work, Aerospike can deliver unprecedented resource utilization. The bottom line is that Aerospike clusters are sized to have (on average) at least 5 times fewer servers than the equivalent Cassandra cluster. For instance, AdForm was able to decrease its number of nodes from 32 with Cassandra to 3 with Aerospike, and achieve a fourfold expansion of data [50]. Aerospike enabled the company to sustain the identical throughput as with their old Cassandra cluster, but with lower - and consistent - latency, and unmatched availability. As Jakob Bak, AdForm's CTO, notes:

> *"With Aerospike, we have been able to drastically cut down on the number of Cassandra servers, which provided a great cost reduction."* [51]

Lower cost. Higher availability. Predictable performance. You get to pick all three with Aerospike. The recently published YCSB benchmark comparing Aerospike and Cassandra shows in great detail how to prove to yourself this reduction in costs. Using the standard YCSB benchmark, we observed the following gains:



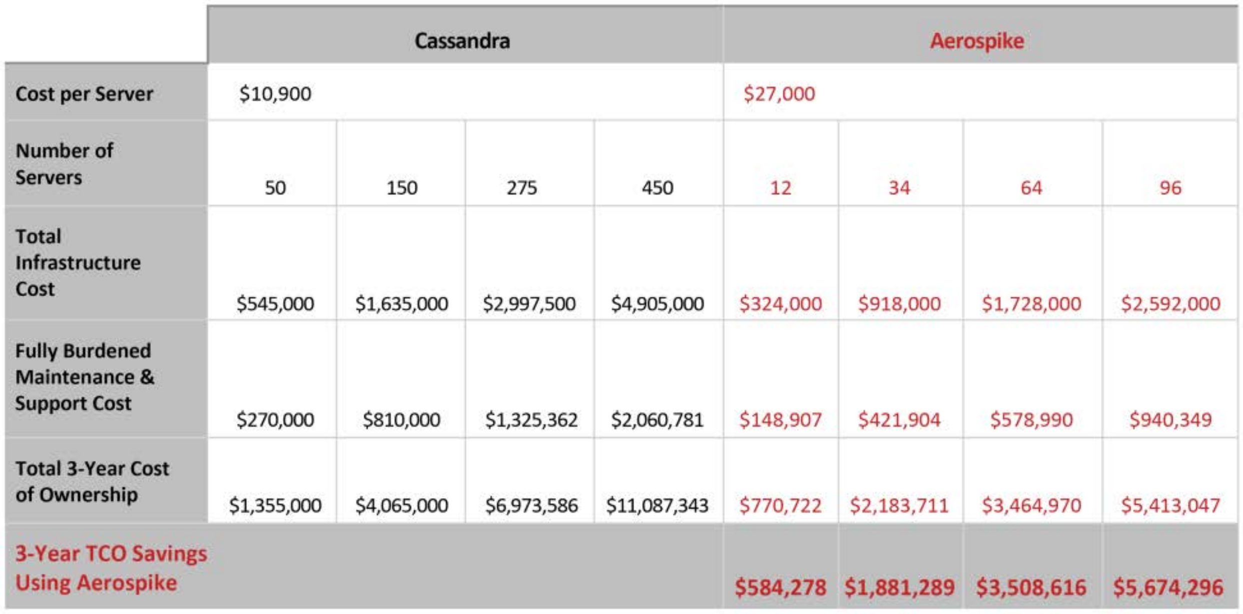
	Inserts (TPS)	Read		Write	
		Throughput (TPS)	Latency (ms)	Throughput (TPS)	Latency (ms)
Aerospike 3.8.2.3	329,000	125,000	2.3	125,000	4.0
Apache Cassandra 3.5.0	40,000	8,900	97.5	8,900	94.0
Aerospike Benefit	8.2x better	14x better	42x better	14x better	24x better

Table 2. Summary of YCSB Benchmark, Aerospike vs. Cassandra

[50] https://vimeo.com/101290545 and http://www.aerospike.com/adform-divorces-cassandra-scales-4x-with-2x-reduced-servers/

[51] http://www.aerospike.com/industry/adtech/adform-divorces-cassandra-scales-performance-by-4x-with-2x-fewer-servers/

Table 2 shows Aerospike's ability to fully utilize the hardware. But it's not just speeds and feeds that prove Aerospike's superiority. Accordingly, let's illustrate a simple TCO calculation - based on actual Aerospike customer data - with the savings our customers were able to achieve (Table 3):



	Cassandra				Aerospike			
Cost per Server	$10,900				$27,000			
Number of Servers	50	150	275	450	12	34	64	96
Total Infrastructure Cost	$545,000	$1,635,000	$2,997,500	$4,905,000	$324,000	$918,000	$1,728,000	$2,592,000
Fully Burdened Maintenance & Support Cost	$270,000	$810,000	$1,325,362	$2,060,781	$148,907	$421,904	$578,990	$940,349
Total 3-Year Cost of Ownership	$1,355,000	$4,065,000	$6,973,586	$11,087,343	$770,722	$2,183,711	$3,464,970	$5,413,047
3-Year TCO Savings Using Aerospike					**$584,278**	**$1,881,289**	**$3,508,616**	**$5,674,296**

Table 3. Cost Comparison, Aerospike vs. Cassandra

Table 3 represents a composite of multiple Cassandra replacements derived from actual customer implementations. The table depicts a 3-year TCO comparison - for exactly the same problem set - using a Cassandra solution vs. an Aerospike solution. To generate this table, we first estimated the size of the Cassandra cluster required; we then estimated the size of the required Aerospike cluster under the same assumptions. The existing Cassandra clusters used HDDs, while the Aerospike cluster was sized to use SSDs. Despite the cost difference between both drive types (SSDs cost more), the cumulative 3-year TCO savings obtained by using an Aerospike solution are very clear and real.

Even if your Cassandra costs are sunk costs in the short term (for example, because you have pre-purchased a year's worth of instance hours with a cloud provider), switching to Aerospike right now starts the process of saving money. Using the data above, this would still result in a savings of nearly $6M by Year 3.

As Aerospike is a native C implementation, there are no inefficiencies from a Java runtime. The primary key index is stored as a parentless red-black tree, enabling ultra-fast key lookups in DRAM; the data is then retrieved from the proprietary log-structured file system. This file system allows parallelization across all the devices on the chassis; a typical Aerospike node will have 8-12 SSDs - sometimes as many as 16. By reading and writing in parallel to all devices, and removing the block and page caches, Aerospike can fully utilize all the IOPs and disk slots available before running out of CPU.

Aerospike's implementation is not faster simply because it uses C. It uses performance-tuned libraries, such as re-implementations of msgpack, and specific tested versions of JEMalloc. The code is a highly optimized, reference-counted multithreaded implementation, requiring certain developer skills in which Aerospike specializes.

Predictable Performance

Since Aerospike has master-based replication, operations are forwarded to the primary node for the record. This is equivalent to having a specific coordinator node for each portion of the data. The Primary Key is generated by a cryptographic algorithm, RIPEMD-160, used by the Bitcoin algorithm, which has had zero detected hash collisions in any use. The first twelve bits of this generated hash identify the partition where the record resides.

Aerospike supports all the popular languages (C/C++, C#, Java, Python, Go, Node.js, PHP, Ruby, Perl, and Erlang) with a vendor-supported language client. These high-performance native clients provide translation of native datatypes to and from Aerospike, as well as interoperability between languages, greatly improving developer productivity.

Unlike the proxy model used in Cassandra, where requests are rerouted by the Coordinator node, Aerospike clients maintain a dynamic partition map. This identifies the master node for each partition, which enables the client to route the read or write request directly to the correct node without any additional network hops. Unlike Cassandra's Coordinator, this removes an unnecessary network hop. Because the data is written synchronously to all copies of the data, there is no need to do any form of quorum read across the cluster to get a consistent version of the data. As AdForm's CTO, Jakob Bak, opines:

> *"Even more important is the super fast key-value store and extraordinary predictability we get with Aerospike, providing the responsiveness our clients require to compete in the crowded Internet and mobile markets."* [52]

[52] http://www.aerospike.com/industry/adtech/adform-divorces-cassandra-scales-performance-by-4x-with-2x-fewer-servers/

This approach enables Aerospike to excel at mixed read/write workloads, without the unnecessary complications and impact to latency of eventually consistent systems. To this point, Valery Vybornov, Head of R&D at IMHO Vi, notes:

> *"It met our demands for random access response time (mixed reads/writes) under our typical load of several hundred writes/some thousand reads per second."* [53]

Yet Aerospike is about more than maintaining predictable performance for a short burst of time, or shining in 5-minute benchmarks. Aerospike enables predictable performance over days and months - the type of performance that allows your business and applications to grow seamlessly as you broaden your presence in local and international markets. As Ko Baryimes, Kayak's SVP of Technology, observes:

> *"As we continue to rapidly expand into international markets, we needed a solution that was reliable and could scale to serve offers across our network. Aerospike enabled us to achieve multi-key gets in less than 3 milliseconds, deploy with ease and scale with very low jitter."* [54]

Predictable scaling was key to the success of Aerospike at marketing analytics firm Ix+1I, as their CTO, Patrick DeAngelis, expresses:

> *"We've seen Aerospike scale to a few billion key values with no compromise to performance, and we've even seen response times under 1 ms, which is phenomenal."* [55]

[53] http://www.aerospike.com/blog/scaling-to-meet-russias-rapid-internet-ad-trajectory/
[54] http://www.aerospike.com/press-releases/kayak-selects-aerospike-delivers-personalized-offers-at-scale/
[55] http://s3-us-west-1.amazonaws.com/aerospike-fd/wp-content/uploads/2015/02/x-1casestudy_2012.pdf

As we saw above, the recent Cassandra benchmark results showed the overall TCO savings that are possible with Aerospike. But this is just part of the story. A comparison of the variance of Aerospike vs. Cassandra - that is, the range of response times and throughput, rather than a simple average - tells a very interesting tale:

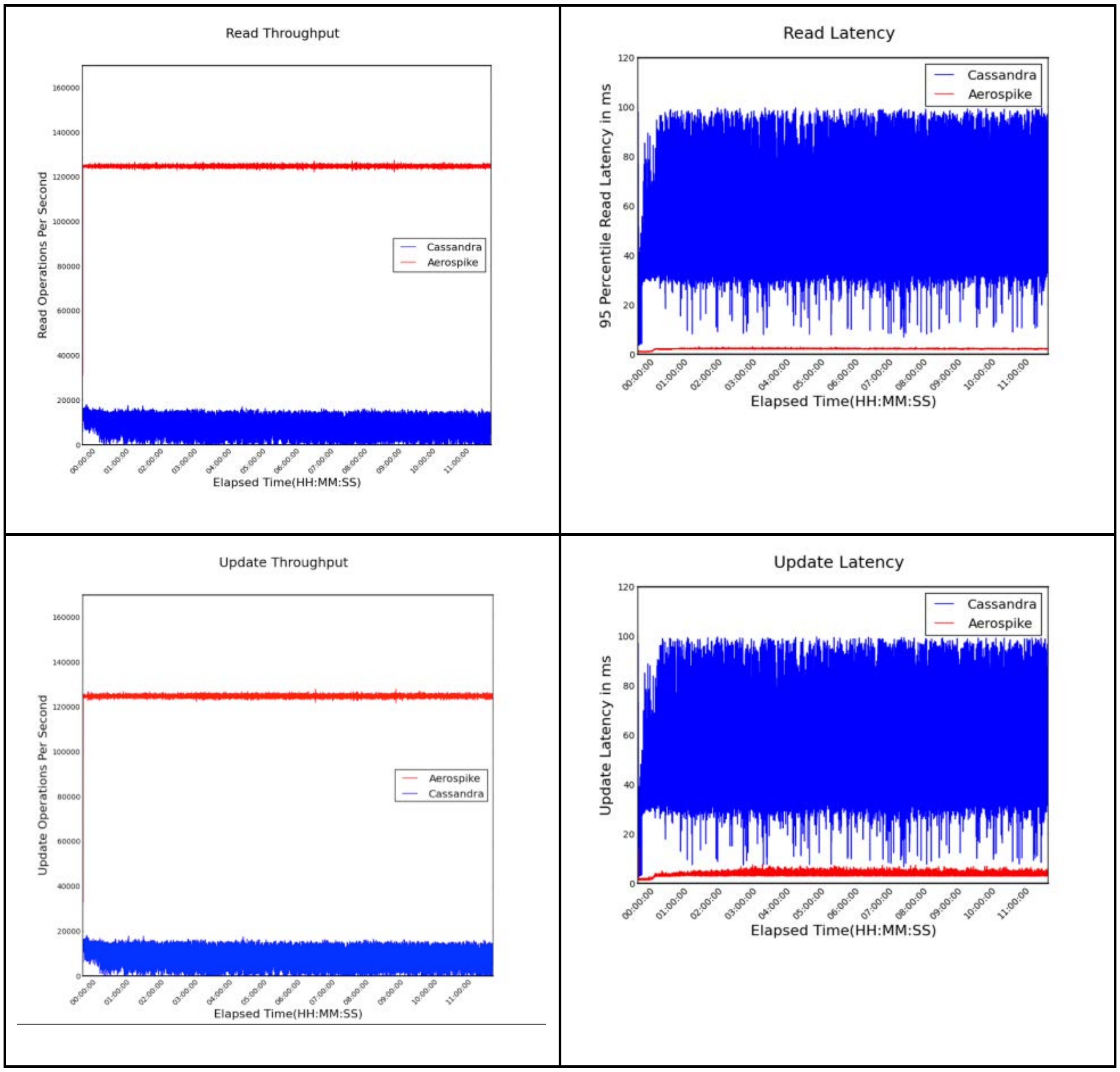


Figure 2. Measured variance in read/write throughput and latency

As Figure 2 illustrates, during the twelve hours that the benchmark ran, the variance in throughput and latency for both read and write operations varied hugely for Cassandra (marked in blue). In contrast, this variance remained in a very narrow band for Aerospike (marked in red). What this translates to is consistent response and throughput characteristics for your applications: Aerospike is very predictable, making it trivial for you to meet your application SLAs not just now, but also in the future.

Proven Reliability

Aerospike uses a shared-nothing architecture, where all nodes are peers, without any node having a specific role. It is built with a unique master-based cluster algorithm. A single node will be the owner or the primary node for that partition. If a replication factor is defined, then there will be N number of other nodes that have a copy of that record for reliability. If you lose a node, then you have another copy. And unlike other systems, Aerospike writes synchronously across all copies of the data.

When a node fails or is removed from the cluster, any node that has a secondary copy of the partition can be instantly promoted to be the master of that partition, without the typical delays imposed by a consensus algorithm. Aerospike uses the Paxos algorithm to ensure consensus across the cluster, but since each node is equal in its role and equal with the current state of the data, any of the secondary nodes can be promoted. This has allowed an organization like AppNexus to minimize downtime, as expressed by its CTO, Geir Magnusson:

> *"2.5 million impressions a second at peak, although we can go much higher, and we see north of 90 Billion impressions per day and this is a 24×7 business with 100% uptime with Aerospike."* [56]

Valery Vybornov, Head of R&D at IMHO Vi's, has a very similar story:

> *"We also considered the Aerospike database's maturity and stability—most notably that it has been running in production non-stop for almost four years."* [57]

Beyond "fair-weather" performance and availability, you also need a system that can deal with human (e.g., "phat fingering") or natural disasters (e.g., the weather). Aerospike has the ability to ensure availability across regions using Cross Datacenter Replication (XDR). During Superstorm Sandy that hit the East Coast of the United States in 2012, this feature allowed adMarketplace to maintain availability without a hitch, as their company's CTO, Mike Yudin, narrates:

> *"We do a 100% uptime. We lost one of our data centers in the flood [Hurricane Sandy], and it's not just the data center itself that lost power, it's the entire network infrastructure of the tri-state major area, all the backbones... How did we do this? We do this by having redundant, not only redundant equipment within the data center, but also the globally load-balanced infrastructure across multiple locations. If one gets flooded, then traffic just gets shifted into the data center that survives. The trick here of course is to make sure that your location has all the same data and all the same intelligence as the system that got destroyed."* [58]

[56] http://www.aerospike.com/why-appnexus-uses-aerospike/
[57] http://www.aerospike.com/blog/scaling-to-meet-russias-rapid-internet-ad-trajectory/
[58] http://www.aerospike.com/blog/super-storm-sandy-and-100-uptime/

Peopleware

Hiring and retaining staff is always a critical task in any organization. You need to ensure that the services and applications you build and deploy meet the time to market (TTM) needs of the organization. Yet people costs should never be excluded from this calculation. The long-term care and maintenance of your infrastructure is a real cost which you need to drive down - if only to spend your company's energy on unique business functions, not database maintenance. From an operational perspective, Aerospike radically simplifies running and maintaining a distributed database, as expressed by BlueKai’s VP of Delivery, Ted Wallace:

> *“It just works and then you move on. We’ve done that repeatedly over the course of the past 2.5 years and it’s always just worked. That’s been awesome.”* [59]

Contrast this with the typical advice from the Cassandra community - for instance, Sam Bisbee, CTO of Threat Stack:

> *“Don’t under staff Cassandra. This is hard as a start up, but recognize going in that it could require 1 to 2 FTEs as you ramp up, maybe more depending on how quickly you scale up.”* [60]

Operational Simplicity

The true value of complex technology is its simplicity of usage, especially in challenging production environments. You do not want to consume time and energy as the cluster expands, or when you refresh the hardware or migrate to a new Data Center. You want to use your database infrastructure like a utility, adding capacity as and when you need to, without the need to perform extensive planning for maintenance windows. As Tapad’s CEO, Dag Liodden, states:

> *“Aerospike makes upgrading simple. That’s the beauty of this product. There’s no planning required. You can take servers down, and still have the system running.”* [61]

Aerospike achieves this with the operational simplicity of self-forming and self-healing clusters, which are both rack-aware and data center-aware. No downtime is required to add or remove nodes from a cluster; it automatically re-distributes partitions of data to ensure the new compute resources are efficiently used. Rack awareness also ensures that data is correctly separated to avoid compounding failures. Using a proprietary algorithm, nodes can be restarted - for example, to perform a software upgrade - but the memory contents are preserved, allowing the process to restart without the need to warm the memory buffers and caches in just seconds, as the data was never evicted from DRAM.

[59] http://www.aerospike.com/blog/bluekai-nosql-speed-scale-simplicity/
[60] http://blog.threatstack.com/scaling-cassandra-lessons-learned
[61] http://s3-us-west-1.amazonaws.com/aerospike-fd/wp-content/uploads/2015/02/Tapad_CaseStudy_101012.pdf

As Ted Wallace from BlueKai opines:

> *"In the case of Aerospike, when we need more capacity in our cluster, we get a machine ready, add it to the cluster and step away. It just works. That is very empowering and very rewarding when you don't have to have somebody spend days creating a documented process, to get approvals, you don't have to send notifications out to your customers because there is downtime because you have to do some massive database maintenance. It just works and then you move on."* [62]

Amey Patil, Big Data Engineer at Crowdfire adds:

> *"Due to its master-master replication model, we do not have to worry about rebalancing, failover or recovery! This has definitely pleased our DevOps team. ;-)"* [63]

Complex technology does not have to be complex to use and operate. Period.

[62] http://www.aerospike.com/blog/bluekai-nosql-speed-scale-simplicity/
[63] https://crowdfire.engineering/why-we-chose-aerospike-over-other-databases-1dfa2d66a292#.27jfii8t1

Summary

In Table 4 below, we contrast the five signs you've outgrown Cassandra with their corresponding solution using Aerospike:

Aerospike	Attributes	Cassandra
• Scaling up before scaling out • Server count reduction	Total Cost of Ownership	• Data volume & usage drive larger cluster • Server sprawl • Significant cost growth
• Predictable, consistent, low-variance performance	Performance	• Missed application SLAs • Provisioning more hardware and caches to meet SLAs
• 99.999+ uptime (five 9's)	Reliability	• Experiencing memory pressure or other computing resource pressures • Fighting compactions
• Setup and forget	Peopleware	• Can't find dedicated Cassandra skills • Can't justify Apache committers
Operational Simplicity: • Simple to grow clusters online • Simple to perform rolling upgrades	Operational Usage	• Fighting garbage collection, tombstones and JVM tuning • Re-tuning for new workloads or hardware • Advance planning required to grow clusters

Table 4. Characteristics of Aerospike vs. Cassandra across key attributes

Aerospike believes in three core principles:

1. Being built for modern computing architectures, and ready for the next generation of hardware
2. Master-based clustering, which means simple scaling and failover
3. Simple Developer Experience (DX)

Being built for modern computing architectures, and ready for the next generation of hardware - By designing early and understanding the deep and significant technology trends, Aerospike has positioned itself as the only NoSQL database system designed and equipped to fully utilize Flash/SSD and storage class memory systems. By being able to fully parallelize reads and writes to storage, Aerospike can drive 12, 14, even 16 SSD/Flash devices per chassis; it runs out of IOPs before you run out of CPU cycles. This is a huge cost savings over trying to use DRAM and cache-based solutions [64]. These characteristics make cloud-based deployments viable for high transactional loads typically only reserved for on-premise deployments. High-memory instances and in-memory solutions are not a cost-effective way to go; you need the ability to blend the speed of DRAM access with the cost effectiveness of Flash/SSD (as Aerospike does). Increasingly, cloud providers are moving to Flash/SSD-based systems to drive better utilization and reduce power and cooling costs versus traditional HDD systems. They are a leading indicator of where the future of compute architectures will look like for everybody in the next 18-24 months. You need a solution designed and optimized for modern computing architectures.

Master-based clustering, which means simple scaling and failover - Availability is a key ingredient for most applications. It's a form of brand insurance, because anytime your infrastructure (of which your database is a part) is unavailable, this impacts your brand perception. Whether this manifests as a decline when swiping a credit card because the system cannot process a “rainy day” load, or as the inability to provide suitable recommendations (and the resulting lost opportunity cost) - it comes down to brand and perception. It's no longer simply getting an empty or system maintenance page when the web site is unavailable - it's all about ensuring your audience remains engaged, regardless of whether your infrastructure is having a rainy day or not. Master-based clustering and Paxos consensus algorithms make up the core technical reasons why Aerospike provides near-instantaneous failover. Aerospike can therefore sustain mixed read/write workloads with ease and with predictable throughput and latency, even on rainy days. This not only satisfies your key availability requirements; it also significantly reduces your overall TCO when compared to eventually consistent systems like Cassandra. As you have seen, Aerospike has unmatched availability. And availability is a core attribute of your customers’ perception of your services and offerings. This is no lab experiment - these are real-world examples of applications handling the most demanding workloads, 24x7.

[64] http://www.aerospike.com/blog/bluekai-flashssd-speed-at-scale/

Simple Developer Experience (DX) - Developers have become the early adopters of technology, often choosing a technology before the operations (or "DevOps") team are aware of a new project. APIs have to be natural, simple and current. You don't want your vendor to be a laggard - which is why Aerospike published its DX Manifesto in 2015. We support the languages and frameworks you need to build efficient & flexible applications, cutting the time to market that your business is demanding, and ensuring that you remain competitive.

Aerospike is the next-generation, enterprise-grade NoSQL solution. Aerospike has a fundamentally unique architecture that has helped customers like BlueKai, Applovin, ShareThis, AdForm, InMobi, PubMatic, NexTag and Curse cost-effectively convert significant applications from Cassandra to Aerospike. Converting to Aerospike has allowed these organizations to achieve more predictable performance, improve uptime and availability, and significantly decrease TCO.

Our recent YCSB benchmark goes into great detail on the performance gains of an Aerospike solution. If you're concerned about your Cassandra implementation, contact Aerospike at info@aerospike.com and ask for a free one-on-one consultation with our Solution Architect staff to evaluate your specific situation. Or simply download your free Enterprise Trial from our website, and join the Aerospike revolution.

Appendix

Product Comparison

Please refer to the chart below for a summary of key product differences between Aerospike and Cassandra:

	Aerospike	Cassandra
Features		
Data Model	Key-Value	Wide Column Store
Consistency	Master-Based strong consistency	Eventual Consistency
Architecture	Shared-nothing	Redundant coordinator nodes
Developer Experience (vendor-supported)		
Languages Supported	C#, C/C++, Erlang, Go, Java, Node.js, Perl, PHP, Python, Ruby	C#, C++, Java, Node.js, PHP, Python, Ruby
Frameworks Supported	Spring Data, Play, ASP.net Session State, Express Session Store, Shiro Session management	Community only
Server-Side User-Defined Functions	LUA	Java JavaScript
Operational Experience[65]		
Supported Platforms	Redhat 6, 7 Ubuntu 12.04, 14.04, 16.04 Debian 7, 8 Generic Linux	Generic Linux Generic Debian OS-X 10.x
Ecosystem (vendor-supported)		
Hadoop Integration	Cloudera Certified HortonWorks Certified	HortonWorks Certified
Spark Integration	Yes	Yes
Docker Integration	Yes	Yes
Hardware Partnerships	Intel Micro OCZ / Toshiba SanDisk / Western Digital	EMC [2] Hewlett Packard Enterprise

[65] Platforms support through Apache http://www.planetcassandra.org/cassandra/