Creating cloud object storage infrastructure without getting hurt

https://www.host-telecom.com/blog/creating-cloud-object-storage-infrastructure-without-getting-hurt/

December 21, 2017 by Denise Boehm

When creating cloud object storage with open source software, a strong understanding of the underlying hardware infrastructure is key to creating a high-functioning, scalable solution. While it's good to know that OpenStack Swift can deliver high input/output operations per second (IOPS) and that Ceph tends to deliver better data consistency, object storage success is highly dependent on the ability of your hardware infrastructure to scale the types of workloads you're running.

To illustrate that point, we consider here the anonymous, true story of a young developer we'll call Antoine and his then-employer, which we'll refer to as Services 'R Us. Early in his career with Services 'R Us, Antoine learned the hard way about interactions between software, servers, and disks and the hardware's critical role in creating high-performing object storage solutions that scale. This story is the cautionary tale of lessons learned followed by some really excellent advice informed by cured cluelessness.

Software is king…

Services 'R Us provided multiple cloud services and seeing a gap in storage, instructed young engineer Antoine and his team to come up with a cloud object storage solution. The team, not unreasonably, made their first task identifying a software development platform. Reviewing the options, Antoine decided to use open source for its power and flexibility saying, "We chose OpenStack Swift for its IOPS performance and because it was a non-proprietary platform."

However, he and his team were novices to the technology, Antoine admitting, "We didn't have any experience with it (Swift) or real knowledge of how to build an architecture that would allow us to easily scale when we hit a certain point." Nevertheless, they built the software, configured the hardware, launched the product, and customers started using it.

And the good times rolled – until they didn’t

Of the initial release from Services ’R Us, Antoine noted, “We had a successful launch and got good response from our users. We thought it would be really easy to scale our object storage solution on different hardware configurations.” Indeed, for about a year, Services ’R Us had an increasing list of happy customers and no problems supporting various hardware configurations. That’s when the hammer dropped, and it dropped fast.

Failure at scale without warning

After a year of steady, incident-free user adoption, performance declined suddenly and dramatically. Object storage was approaching 1 PB of data with a replication factor of 3, and the number of stored objects was in the several billions when alarming increases in latency and resulting user time-outs quickly created a critical situation. But the Services ’R Us team couldn’t pinpoint the cause of the breakdown.

Crisis mode

With great initial performance driving high demand for its storage services, Services ’R Us was now looking at a brick wall with continued growth. “Continuing to scale at that point was starting to kill us, and we really had to scramble,” Antoine remembers. However, with the source of the performance problems not immediately evident and as the situation became critical, Antoine said the team really started to question the viability of Swift. “We did not have monitoring enabled and couldn’t pinpoint the slowdown and were really considering Ceph despite its slower IOPS performance. But because latency and not data integrity was the big issue, Ceph didn’t seem like a plausible cure,” he said.

Admitting that he and his colleagues had flown a bit by the seat of their pants during development with a myopic view on software, Antoine said, “We really weren’t prepared for how much the demand on storage would increase and how quickly we’d outgrow our infrastructure. Our servers were at capacity with drives, processors, and the network all maxed out.” The team started working the hardware angle.

Bad gets worse

Correctly diagnosing the problem as rooted in hardware was half of the solution, but Antoine was facing some additional challenges. While some of the original engineering team moved forward to fix the object storage solution, some of them moved out of

Services ’R Us completely. To make matters worse, no one had properly documented the project before the exodus, creating the perfect storm.

Just add some more disks and servers, right?

To Antoine and what was left of the original team operating with incomplete information, it made sense to implement a major upgrade of the hardware infrastructure now that they knew it was the root cause of their object storage performance degradation. But there was a huge hitch. “We got the brilliant idea to upgrade the servers with a lot more disks, Antoine said, “But when we disconnected one server for the upgrade, we had a complete degradation in storage performance. It turned out that changes made to the proxy server had screwed up the cluster.” He noted that they couldn’t tell if the cluster had rebalanced, but that they saw a huge data loss and data inconsistency after they renewed the Swift cluster. The remaining team continued on with different repair efforts, and it was a long road.

Six months to normal

Tweaking for improvements in latency, the team saw minor performance increases but a huge slowdown in data replication. Increased replication speeds brought compromises in data integrity and more customer timeouts. Picking the problems apart and finding solutions that didn’t introduce more problems was a painstaking process that took six months to untangle.

During this time, the Services ’R US team completely reworked the hardware architecture and the Swift software solution running on it, eliminating all superfluous functionality to ensure strong object storage performance. The move to rebuild from the ground up required a major paradigm shift to avoid their original, eventually crippling mistakes. So now we’ll look at how they approached the problem with wisdom gained from the school of hard knocks.

The do-over

As Antoine and his Services ’R Us colleagues rethought their object storage offering, they focused first on how their customers were using the solution, then the physical resources necessary, and finally the best software solution for the hardware that would scale together. So we begin with their customer profiles.

Defining customer types

The first object storage customer group Antoine and team defined was web application users. The data for web apps are static on high-load, scalable applications and mobile applications, using CSS, JSON, graphics, photos, animation, and flash games. Data storage requirements for static data range between a kilobyte to several megabytes. In general, users can access this data from storage, and workloads tend to be much higher during business hours with a significant effect on IOPS.

The second defined customer group was users storing larger amounts of archived data and backups, ranging in size from MBs to GBs. Examples of this type of user are healthcare institutions and businesses with digitized data, often documents, who require writing functionality often for nighttime backups. Capacity is a bigger issue than IOPS for this type of user.

With the variations in data types, sizes, and user requirements, Antoine and his group began planning their revamped object storage solution to include specific customizations for both static and archived data response times and consistency. They then turned to hardware infrastructure and the best software to run on it to accommodate customer needs.

Hardware basics

Antoine's first step was considering the basics about data read/write for his first audience of static data users, concluding that capacity and efficiency were of primary importance, rather than speed. In Figure 1 is a simple load executing triple replication. The proxy server receives the "PUT" query for the 3 data copies and creates 3 requests to the object server. With two successful replications, the object server returns a response of 200 to the client. Storage's disk load and bandwidth thus increased times three. With the GET request, the load was proportional to the incoming request.

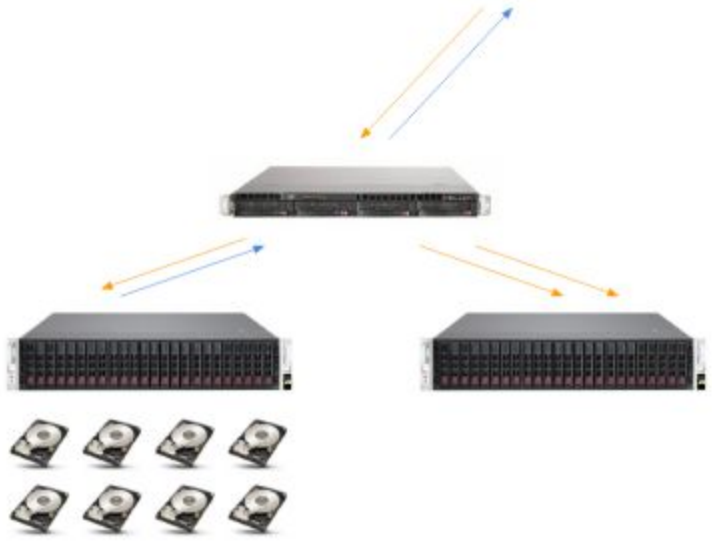

Fig. 1 — Simple load executing triple replication

Beyond the basics

Not designing for simple loads, Antoine and his team looked at other ways to achieve high productivity knowing that disk capacity didn't necessarily correlate with increase performance. An alternative they considered was using multiple disks for increased volume to avoid single points of failure. While some developers get around this problem using larger servers, for example 2TB in a 2-unit server, server storage can be an issue, and power consumption is greater with higher associated costs.

With IOPS the key to balancing data volume, they looked for a cost-effective solution. Antoine decided to use JBODs containing 44 disks at 10 terabytes each with an occupied space of four units. He also created safeguards against points of failure, maximizing IOPS on similar storage volumes, which is discussed below. Now comfortable in the object storage infrastructure they'd designed, Antoine and the team turned to data performance and software analysis.

Maximizing data performance

With the understanding that IOPS was the big bottleneck, the Services 'R Us team considered the remaining decisions necessary to build an optimal object storage solution. To ensure system integrity and performance, Antoine referenced Consistency, Availability, and Partition Tolerance (CAP) Theorem, as shown in Figure 2, which states

that a distributed data store cannot simultaneously provide all three qualities. He shot for availability and consistency.

Fig. 2 — CAP Theory

A second look at CEPH

With CAP Theorem in mind, the team returned to software options, with a second, more thorough analysis of CEPH. Created for block storage and limited volumes, CEPH is used for clustered file systems as well as object storage. Its strength is maintaining data consistency, and when an object storage device fails or stops, the Controlled Replication Under Scalable Hashing (CRUSH) algorithm automatically starts the rebalancing process to redistribute data in the remaining OSDs.

Because rebalancing requires time and resources, and its functions are prioritized over client requests, the possibility of slowdowns and customer timeouts increases. Having faced serious latency and timeouts on the first object storage go-round, the team now took a deeper look at its original software development platform, OpenStack Swift.

Swift on second glance

While Swift hadn't panned out well in the first object storage release, IOPS were still top-of-mind for Service's 'R Us, and Swift's first priority is client traffic, followed by asynchronous object writing and deletion, then data consistency and data replication. Certainly CEPH had the upper hand on the two last items of the priority list, but Swift looked to deliver on the first three. In addition, the team had addressed the major issue

of hardware infrastructure, so Swift was back. Now time to look at increasing and maintaining performance.

Optimizing object storage integrated components

Keeping their top two CAP goals of consistency and availability in mind, Antoine led the team in delivering high performing object storage deployments with the re-engineered infrastructure using JBODs. It wasn't all smooth sailing, though.

Challenges on the way

The team found that cluster performance took a bit of a hit under high load, with most disks loaded to IO, two replications instead of three, and the third delayed to the asynchronous queue line, causing software performance to dip. In a server or disk failure scenario, replication was interrupted with associated declines in data consistency. On another front, the team found that expanding the cluster caused traffic disruptions.

Making the infrastructure work

Antoine and his colleagues found that adding new servers to the cluster ring and rebalancing it enabled client traffic to process normally. Without rebalancing, IOPS and physical storage decreased and replication and consistency suffered with slow system restoration.

Segregating customer types improves performance in ways not readily apparent

While they found no miracle cures for improving object storage performance, dividing the customers by use types was a huge advantage, but not for the reasons that Services 'R Us developers thought.

With the first tier of web app customers using static data for read/write functions mainly during business hours and the second tier of customers storing larger amounts of archived data and backups with higher use rates at night, it initially seemed that putting both types on the same disks would work out, but it didn't. Instead, disks performing client operations constrained IOPS around replication and had a negative effect on data

consistency. The solution was keeping customers on different disks with hot and cold clusters, one for IOPS performance and the other for data backup.

Cache systems for bottlenecks

Antoine and his colleagues also found that adding a cache system for web app read/write was helpful in preventing bottlenecks, using enhanced IO and placing cache on NVME disks. The cache partition connects and disconnects from disks with live data. With triple replication, Antoine found that project cache must be ⅓ the size of the hot tier accommodating read/write functionality to accommodate upticks in client traffic and obviate the need for huge SATA disks.

Double server connections

A final point concerns inaccessible Swift disks that are lost if shut down. This is where Antoine's JBOD solution really delivers. Adding dual-ported SAS disks to a server can extend JBOD to two servers and switch them to others servers as needed, minimizing downtown and preserving data integrity.

No pain no gain

While neither he nor his colleagues would do it the same way again, Antoine knows he learned more about object storage architecture than if he'd done it right the first time. Hopefully you can avoid similar pitfalls from the perspective of hard-won experience.

BY

Denise Boehm