Building and operating a pretty big storage system called S3

July 27, 2023 • 6277 words

Today, I am publishing a guest post from Andy Warfield, VP and distinguished engineer over at S3. I asked him to write this based on the Keynote address he gave at USENIX FAST '23 that covers three distinct perspectives on scale that come along with building and operating a storage system the size of S3.

In today's world of short-form snackable content, we're very fortunate to get an excellent in-depth exposé. It's one that I find particularly fascinating, and it provides some really unique insights into why people like Andy and I joined Amazon in the first place. The full recording of Andy presenting this paper at fast is embedded at the end of this post.

–W

Building and operating a pretty big storage system called S3

I've worked in computer systems software — operating systems, virtualization, storage, networks, and security — for my entire career. However, the last six years

working with Amazon Simple Storage Service (S3) have forced me to think about systems in broader terms than I ever have before. In a given week, I get to be involved in everything from hard disk mechanics, firmware, and the physical properties of storage media at one end, to customer-facing performance experience and API expressiveness at the other. And the boundaries of the system are not just technical ones: I've had the opportunity to help engineering teams move faster, worked with finance and hardware teams to build cost-following services, and worked with customers to create gob-smackingly cool applications in areas like video streaming, genomics, and generative AI.

What I'd really like to share with you more than anything else is my sense of wonder at the storage systems that are all collectively being built at this point in time, because they are pretty amazing. In this post, I want to cover a few of the interesting nuances of building something like S3, and the lessons learned and sometimes surprising observations from my time in S3.

17 years ago, on a university campus far, far away…

S3 launched on March 14th, 2006, which means it turned 17 this year. It's hard for me to wrap my head around the fact that for engineers starting their careers today, S3 has simply existed as an internet storage service for as long as you've been working with computers. Seventeen years ago, I was just finishing my PhD at the University of Cambridge. I was working in the lab that developed Xen, an open-source hypervisor that a few companies, including Amazon, were using to build the first public clouds. A group of us moved on from the Xen project at Cambridge to create a startup called XenSource that, instead of using Xen to build a public cloud, aimed to commercialize it by selling it as enterprise software. You might say that we missed a bit of an opportunity there. XenSource grew and was eventually acquired by Citrix, and I wound up learning a whole lot about growing teams and growing a business (and negotiating commercial leases, and fixing small server room HVAC systems, and so on) – things that I wasn't exposed to in grad school.

But at the time, what I was convinced I really wanted to do was to be a university professor. I applied for a bunch of faculty jobs and wound up finding one at UBC (which worked out really well, because my wife already had a job in Vancouver and we love the city). I threw myself into the faculty role and foolishly grew my lab to 18 students, which is something that I'd encourage anyone that's starting out as an assistant professor to never, ever do. It was thrilling to have such a large lab full of amazing people and it was absolutely exhausting to try to supervise that many graduate students all at once, but, I'm pretty sure I did a horrible job of it. That said, our research lab was an incredible community of people and we built things that I'm still really proud of today, and we wrote all sorts of really fun papers on security, storage, virtualization, and networking.

A little over two years into my professor job at UBC, a few of my students and I decided to do another startup. We started a company called Coho Data that took

advantage of two really early technologies at the time: NVMe SSDs and programmable ethernet switches, to build a high-performance scale-out storage appliance. We grew Coho to about 150 people with offices in four countries, and once again it was an opportunity to learn things about stuff like the load bearing strength of second-floor server room floors, and analytics workflows in Wall Street hedge funds – both of which were well outside my training as a CS researcher and teacher. Coho was a wonderful and deeply educational experience, but in the end, the company didn't work out and we had to wind it down.

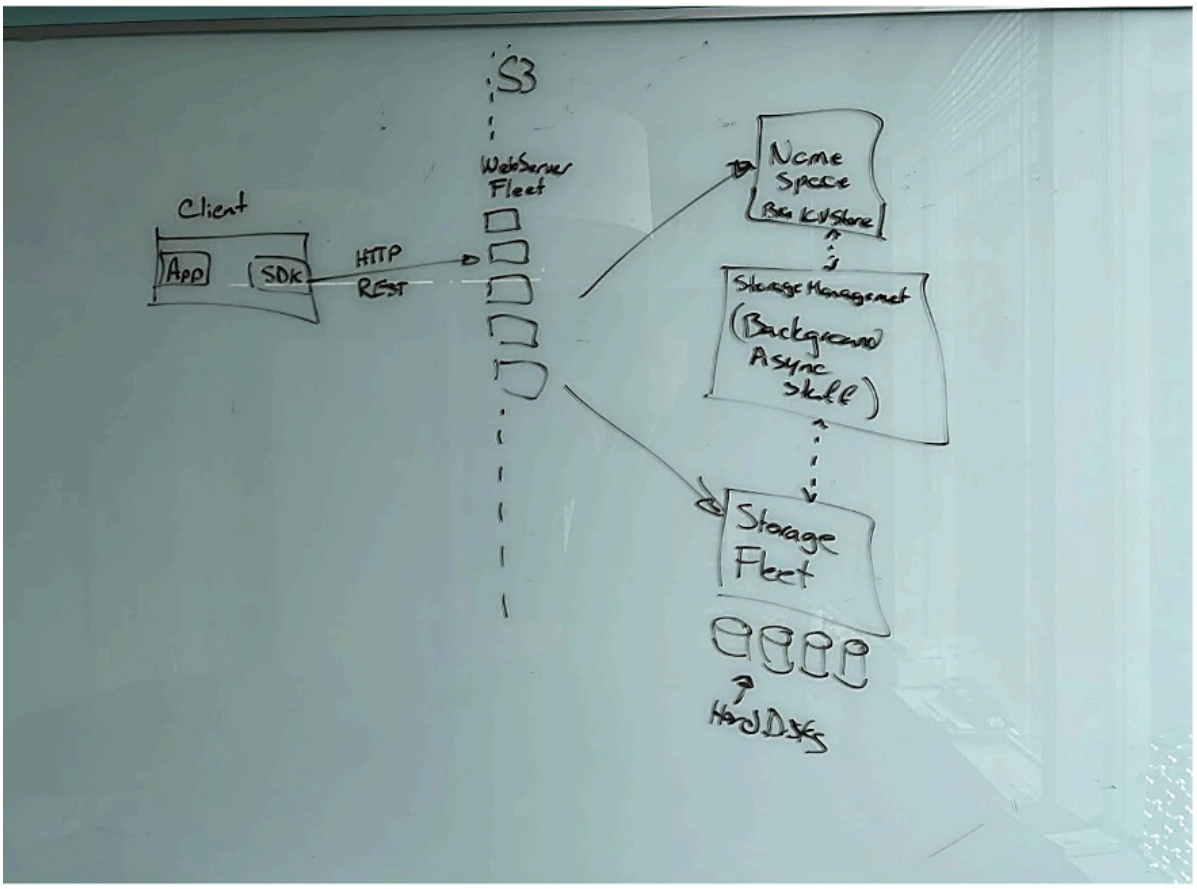
And so, I found myself sitting back in my mostly empty office at UBC. I realized that I'd graduated my last PhD student, and I wasn't sure that I had the strength to start building a research lab from scratch all over again. I also felt like if I was going to be in a professor job where I was expected to teach students about the cloud, that I might do well to get some first-hand experience with how it actually works.

I interviewed at some cloud providers, and had an especially fun time talking to the folks at Amazon and decided to join. And that's where I work now. I'm based in Vancouver, and I'm an engineer that gets to work across all of Amazon's storage products. So far, a whole lot of my time has been spent on S3.

How S3 works

When I joined Amazon in 2017, I arranged to spend most of my first day at work with Seth Markle. Seth is one of S3's early engineers, and he took me into a little room with a whiteboard and then spent six hours explaining how S3 worked.

It was awesome. We drew pictures, and I asked question after question non-stop and I couldn't stump Seth. It was exhausting, but in the best kind of way. Even then S3 was a very large system, but in broad strokes — which was what we started with on the whiteboard — it probably looks like most other storage systems that you've seen.



Amazon Simple Storage Service - Simple, right?

S3 is an object storage service with an HTTP REST API. There is a frontend fleet with a REST API, a namespace service, a storage fleet that's full of hard disks, and a fleet that does background operations. In an enterprise context we might call these background tasks "data services," like replication and tiering. What's interesting here, when you look at the highest-level block diagram of S3's technical design, is the fact that AWS tends to ship its org chart. This is a phrase that's often used in a pretty disparaging way, but in this case it's absolutely fascinating. Each of these broad components is a part of the S3 organization. Each has a leader, and a bunch of teams that work on it. And if we went into the next level of detail in the diagram, expanding one of these boxes out into the individual components that are inside it, what we'd find is that all the nested components are their own teams, have their own fleets, and, in many ways, operate like independent businesses.

All in, S3 today is composed of hundreds of microservices that are structured this way. Interactions between these teams are literally API-level contracts, and, just like the code that we all write, sometimes we get modularity wrong and those team-level interactions are kind of inefficient and clunky, and it's a bunch of work to go and fix it, but that's part of building software, and it turns out, part of building software teams too.

Two early observations

Before Amazon, I'd worked on research software, I'd worked on pretty widely adopted open-source software, and I'd worked on enterprise software and hardware appliances that were used in production inside some really large

businesses. But by and large, that software was a thing we designed, built, tested, and shipped. It was the software that we packaged and the software that we delivered. Sure, we had escalations and support cases and we fixed bugs and shipped patches and updates, but we ultimately delivered software. Working on a global storage service like S3 was completely different: S3 is effectively a living, breathing organism. Everything, from developers writing code running next to the hard disks at the bottom of the software stack, to technicians installing new racks of storage capacity in our data centers, to customers tuning applications for performance, everything is one single, continuously evolving system. S3's customers aren't buying software, they are buying a service and they expect the experience of using that service to be continuously, predictably fantastic.

The first observation was that **I was going to have to change, and really broaden how I thought about software systems and how they behave**. This didn't just mean broadening thinking about software to include those hundreds of microservices that make up S3, it meant broadening to also include all the people who design, build, deploy, and operate all that code. It's all one thing, and you can't really think about it just as software. It's software, hardware, and people, and it's always growing and constantly evolving.

The second observation was that despite the fact that this whiteboard diagram sketched the broad strokes of the organization and the software, it was also wildly misleading, because it completely obscured the scale of the system. Each one of the boxes represents its own collection of scaled out software services, often themselves built from collections of services. **It would literally take me years to come to terms with the scale of the system that I was working with, and even today I often find myself surprised at the consequences of that scale.**

S3 by the numbers (as of publishing this post).

Technical Scale: Scale and the physics of storage

It probably isn't very surprising for me to mention that S3 is a really big system, and it is built using a LOT of hard disks. Millions of them. And if we're talking about S3, it's worth spending a little bit of time talking about hard drives themselves. Hard drives are amazing, and they've kind of always been amazing.

The first hard drive was built by Jacob Rabinow, who was a researcher for the predecessor of the National Institute of Standards and Technology (NIST). Rabinow was an expert in magnets and mechanical engineering, and he'd been asked to build a machine to do magnetic storage on flat sheets of media, almost like pages in a book. He decided that idea was too complex and inefficient, so, stealing the idea of a spinning disk from record players, he built an array of spinning magnetic disks that could be read by a single head. To make that work, he cut a pizza slice-style notch out of each disk that the head could move through to reach the appropriate platter. Rabinow described this as being like "like reading a book

without opening it." The first commercially available hard disk appeared 7 years later in 1956, when IBM introduced the 350 disk storage unit, as part of the 305 RAMAC computer system. We'll come back to the RAMAC in a bit.

The first magnetic memory device. Credit: https://www.computerhistory.org/storageengine/rabinow-patents-magnetic-disk-data-storage/

Today, 67 years after that first commercial drive was introduced, the world uses lots of hard drives. Globally, the number of bytes stored on hard disks continues to grow every year, but the applications of hard drives are clearly diminishing. We just seem to be using hard drives for fewer and fewer things. Today, consumer devices are effectively all solid-state, and a large amount of enterprise storage is similarly switching to SSDs. Jim Gray predicted this direction in 2006, when he very presciently said: "Tape is Dead. Disk is Tape. Flash is Disk. RAM Locality is King." This quote has been used a lot over the past couple of decades to motivate flash storage, but the thing it observes about disks is just as interesting.

Hard disks don't fill the role of general storage media that they used to because they are big (physically and in terms of bytes), slower, and relatively fragile pieces of media. For almost every common storage application, flash is superior. But hard drives are absolute marvels of technology and innovation, and for the things they are good at, they are absolutely amazing. One of these strengths is cost efficiency, and in a large-scale system like S3, there are some unique opportunities to design around some of the constraints of individual hard disks.

The anatomy of a hard disk. Credit: https://www.researchgate.net/figure/Mechanical-components-of-a-typical-hard-disk-drive_fig8_224323123

As I was preparing for my talk at FAST, I asked Tim Rausch if he could help me revisit the old plane flying over blades of grass hard drive example. Tim did his PhD at CMU and was one of the early researchers on heat-assisted magnetic recording (HAMR) drives. Tim has worked on hard drives generally, and HAMR specifically for most of his career, and we both agreed that the plane analogy – where we scale up the head of a hard drive to be a jumbo jet and talk about the relative scale of all the other components of the drive – is a great way to illustrate the complexity and mechanical precision that's inside an HDD. So, here's our version for 2023.

Imagine a hard drive head as a 747 flying over a grassy field at 75 miles per hour. The air gap between the bottom of the plane and the top of the grass is two sheets of paper. Now, if we measure bits on the disk as blades of grass, the track width would be 4.6 blades of grass wide and the bit length would be one blade of grass. As the plane flew over the grass it would count blades of grass and only miss one blade for every 25 thousand times the plane circled the Earth.

That's a bit error rate of 1 in 10^15 requests. In the real world, we see that blade of grass get missed pretty frequently – and it's actually something we need to

account for in S3.

Now, let's go back to that first hard drive, the IBM RAMAC from 1956. Here are some specs on that thing:

Storage Capacity:	3.75 MB
Weight:	1,730 lbs.
Size:	60" L x 68" H x 29" W
Disks:	50–24 inch disks
Cost:	~$9,200/megabyte
Latency:	600 ms

Now let's compare it to the largest HDD that you can buy as of publishing this, which is a Western Digital Ultrastar DC HC670 26TB. Since the RAMAC, capacity has improved 7.2M times over, while the physical drive has gotten 5,000x smaller. It's 6 billion times cheaper per byte in inflation-adjusted dollars. But despite all that, seek times – the time it takes to perform a random access to a specific piece of data on the drive – have only gotten 150x better. Why? Because they're mechanical. We have to wait for an arm to move, for the platter to spin, and those mechanical aspects haven't really improved at the same rate. If you are doing random reads and writes to a drive as fast as you possibly can, you can expect about 120 operations per second. The number was about the same in 2006 when S3 launched, and it was about the same even a decade before that.

This tension between HDDs growing in capacity but staying flat for performance is a central influence in S3's design. We need to scale the number of bytes we store by moving to the largest drives we can as aggressively as we can. Today's largest drives are 26TB, and industry roadmaps are pointing at a path to 200TB (200TB drives!) in the next decade. At that point, if we divide up our random accesses fairly across all our data, we will be allowed to do 1 I/O per second per 2TB of data on disk.

S3 doesn't have 200TB drives yet, but I can tell you that we anticipate using them when they're available. And all the drive sizes between here and there.

Managing heat: data placement and performance

So, with all this in mind, one of the biggest and most interesting technical scale problems that I've encountered is in managing and balancing I/O demand across a really large set of hard drives. In S3, we refer to that problem as heat management.

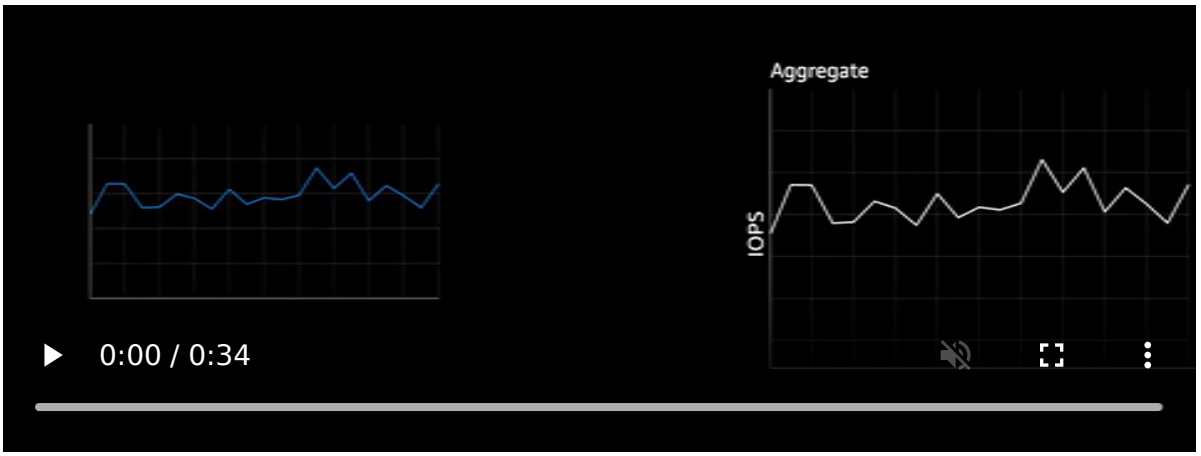
By heat, I mean the number of requests that hit a given disk at any point in time. If we do a bad job of managing heat, then we end up focusing a disproportionate

number of requests on a single drive, and we create hotspots because of the limited I/O that's available from that single disk. For us, this becomes an optimization challenge of figuring out how we can place data across our disks in a way that minimizes the number of hotspots.

Hotspots are small numbers of overloaded drives in a system that ends up getting bogged down, and results in poor overall performance for requests dependent on those drives. When you get a hot spot, things don't fall over, but you queue up requests and the customer experience is poor. Unbalanced load stalls requests that are waiting on busy drives, those stalls amplify up through layers of the software storage stack, they get amplified by dependent I/Os for metadata lookups or erasure coding, and they result in a very small proportion of higher latency requests — or "stragglers". In other words, hotspots at individual hard disks create tail latency, and ultimately, if you don't stay on top of them, they grow to eventually impact all request latency.

As S3 scales, we want to be able to spread heat as evenly as possible, and let individual users benefit from as much of the HDD fleet as possible. This is tricky, because we don't know when or how data is going to be accessed at the time that it's written, and that's when we need to decide where to place it. Before joining Amazon, I spent time doing research and building systems that tried to predict and manage this I/O heat at much smaller scales – like local hard drives or enterprise storage arrays and it was basically impossible to do a good job of. But this is a case where the sheer scale, and the multitenancy of S3 result in a system that is fundamentally different.

The more workloads we run on S3, the more that individual requests to objects become decorrelated with one another. Individual storage workloads tend to be really bursty, in fact, most storage workloads are completely idle most of the time and then experience sudden load peaks when data is accessed. That peak demand is much higher than the mean. But as we aggregate millions of workloads a really, really cool thing happens: the aggregate demand smooths and it becomes way more predictable. In fact, and I found this to be a really intuitive observation once I saw it at scale, once you aggregate to a certain scale you hit a point where it is difficult or impossible for any given workload to really influence the aggregate peak at all! So, with aggregation flattening the overall demand distribution, we need to take this relatively smooth demand rate and translate it into a similarly smooth level of demand across all of our disks, balancing the heat of each workload.



Replication: data placement and durability

In storage systems, redundancy schemes are commonly used to protect data from hardware failures, but redundancy also helps manage heat. They spread load out and give you an opportunity to steer request traffic away from hotspots. As an example, consider replication as a simple approach to encoding and protecting data. Replication protects data if disks fail by just having multiple copies on different disks. But it also gives you the freedom to read from any of the disks. When we think about replication from a capacity perspective it's expensive. However, from an I/O perspective – at least for reading data – replication is very efficient.

We obviously don't want to pay a replication overhead for all of the data that we store, so in S3 we also make use of erasure coding. For example, we use an algorithm, such as Reed-Solomon, and split our object into a set of k "identity" shards. Then we generate an additional set of m parity shards. As long as k of the (k+m) total shards remain available, we can read the object. This approach lets us reduce capacity overhead while surviving the same number of failures.

The impact of scale on data placement strategy

So, redundancy schemes let us divide our data into more pieces than we need to read in order to access it, and that in turn provides us with the flexibility to avoid sending requests to overloaded disks, but there's more we can do to avoid heat. The next step is to spread the placement of new objects broadly across our disk fleet. While individual objects may be encoded across tens of drives, we intentionally put different objects onto different sets of drives, so that each customer's accesses are spread over a very large number of disks.

There are two big benefits to spreading the objects within each bucket across lots and lots of disks:

1. A customer's data only occupies a very small amount of any given disk, which helps achieve workload isolation, because individual workloads can't generate a hotspot on any one disk.

2. Individual workloads can burst up to a scale of disks that would be really difficult and really expensive to build as a stand-alone system.

Here's a spiky workload

For instance, look at the graph above. Think about that burst, which might be a genomics customer doing parallel analysis from thousands of Lambda functions at once. That burst of requests can be served by over a million individual disks. That's not an exaggeration. Today, we have tens of thousands of customers with S3 buckets that are spread across millions of drives. When I first started working on S3, I was really excited (and humbled!) by the systems work to build storage at this scale, but as I really started to understand the system I realized that it was the scale of customers and workloads using the system in aggregate that really allow it to be built differently, and building at this scale means that any one of those individual workloads is able to burst to a level of performance that just wouldn't be practical to build if they were building without this scale.

The human factors

Beyond the technology itself, there are human factors that make S3 - or any complex system - what it is. One of the core tenets at Amazon is that we want engineers and teams to fail fast, and safely. We want them to always have the confidence to move quickly as builders, while still remaining completely obsessed with delivering highly durable storage. One strategy we use to help with this in S3 is a process called "durability reviews." It's a human mechanism that's not in the statistical 11 9s model, but it's every bit as important.

When an engineer makes changes that can result in a change to our durability posture, we do a durability review. The process borrows an idea from security research: the threat model. The goal is to provide a summary of the change, a comprehensive list of threats, then describe how the change is resilient to those threats. In security, writing down a threat model encourages you to think like an adversary and imagine all the nasty things that they might try to do to your system. In a durability review, we encourage the same "what are all the things that might go wrong" thinking, and really encourage engineers to be creatively critical of their own code. The process does two things very well:

1. It encourages authors and reviewers to really think critically about the risks we should be protecting against.
2. It separates risk from countermeasures, and lets us have separate discussions about the two sides.

When working through durability reviews we take the durability threat model, and then we evaluate whether we have the right countermeasures and protections in place. When we are identifying those protections, we really focus on identifying coarse-grained "guardrails". These are simple mechanisms that protect you from a

large class of risks. Rather than nitpicking through each risk and identifying individual mitigations, we like simple and broad strategies that protect against a lot of stuff.

Another example of a broad strategy is demonstrated in a project we kicked off a few years back to rewrite the bottom-most layer of S3's storage stack – the part that manages the data on each individual disk. The new storage layer is called ShardStore, and when we decided to rebuild that layer from scratch, one guardrail we put in place was to adopt a really exciting set of techniques called "lightweight formal verification". Our team decided to shift the implementation to Rust in order to get type safety and structured language support to help identify bugs sooner, and even wrote libraries that extend that type safety to apply to on-disk structures. From a verification perspective, we built a simplified model of ShardStore's logic, (also in Rust), and checked into the same repository alongside the real production ShardStore implementation. This model dropped all the complexity of the actual on-disk storage layers and hard drives, and instead acted as a compact but executable specification. It wound up being about 1% of the size of the real system, but allowed us to perform testing at a level that would have been completely impractical to do against a hard drive with 120 available IOPS. We even managed to publish a paper about this work at SOSP.

From here, we've been able to build tools and use existing techniques, like property-based testing, to generate test cases that verify that the behaviour of the implementation matches that of the specification. The really cool bit of this work wasn't anything to do with either designing ShardStore or using formal verification tricks. It was that we managed to kind of "industrialize" verification, taking really cool, but kind of research-y techniques for program correctness, and get them into code where normal engineers who don't have PhDs in formal verification can contribute to maintaining the specification, and that we could continue to apply our tools with every single commit to the software. Using verification as a guardrail has given the team confidence to develop faster, and it has endured even as new engineers joined the team.

Durability reviews and lightweight formal verification are two examples of how we take a really human, and organizational view of scale in S3. The lightweight formal verification tools that we built and integrated are really technical work, **but they were motivated by a desire to let our engineers move faster and be confident even as the system becomes larger and more complex over time.** Durability reviews, similarly, are a way to help the team think about durability in a structured way, but also to make sure that we are always holding ourselves accountable for a high bar for durability as a team. There are many other examples of how we treat the organization as part of the system, and it's been interesting to see how once you make this shift, you experiment and innovate with how the team builds and operates just as much as you do with what they are building and operating.

Scaling myself: Solving hard problems starts and ends with “Ownership”

The last example of scale that I’d like to tell you about is an individual one. I joined Amazon as an entrepreneur and a university professor. I’d had tens of grad students and built an engineering team of about 150 people at Coho. In the roles I’d had in the university and in startups, I loved having the opportunity to be technically creative, to build really cool systems and incredible teams, and to always be learning. But I’d never had to do that kind of role at the scale of software, people, or business that I suddenly faced at Amazon.

One of my favourite parts of being a CS professor was teaching the systems seminar course to graduate students. This was a course where we’d read and generally have pretty lively discussions about a collection of “classic” systems research papers. One of my favourite parts of teaching that course was that about half way through it we’d read the SOSP Dynamo paper. I looked forward to a lot of the papers that we read in the course, but I really looked forward to the class where we read the Dynamo paper, because it was from a real production system that the students could relate to. It was Amazon, and there was a shopping cart, and that was what Dynamo was for. It’s always fun to talk about research work when people can map it to real things in their own experience.

But also, technically, it was fun to discuss Dynamo, because Dynamo was eventually consistent, so it was possible for your shopping cart to be wrong.

I loved this, because it was where we’d discuss what you do, practically, in production, when Dynamo was wrong. When a customer was able to place an order only to later realize that the last item had already been sold. You detected the conflict but what could you do? The customer was expecting a delivery.

This example may have stretched the Dynamo paper’s story a little bit, but it drove to a great punchline. Because the students would often spend a bunch of discussion trying to come up with technical software solutions. Then someone would point out that this wasn’t it at all. That ultimately, these conflicts were rare, and you could resolve them by getting support staff involved and making a human decision. It was a moment where, if it worked well, you could take the class from being critical and engaged in thinking about tradeoffs and design of software systems, and you could get them to realize that the system might be bigger than that. It might be a whole organization, or a business, and maybe some of the same thinking still applied.

Now that I’ve worked at Amazon for a while, I’ve come to realize that my interpretation wasn’t all that far from the truth — in terms of how the services that we run are hardly “just” the software. I’ve also realized that there’s a bit more to it than what I’d gotten out of the paper when teaching it. Amazon spends a lot of

time really focused on the idea of “ownership.” The term comes up in a lot of conversations — like “does this action item have an owner?” — meaning who is the single person that is on the hook to really drive this thing to completion and make it successful.

The focus on ownership actually helps understand a lot of the organizational structure and engineering approaches that exist within Amazon, and especially in S3. To move fast, to keep a really high bar for quality, teams need to be owners. They need to own the API contracts with other systems their service interacts with, they need to be completely on the hook for durability and performance and availability, and ultimately, they need to step in and fix stuff at three in the morning when an unexpected bug hurts availability. But they also need to be empowered to reflect on that bug fix and improve the system so that it doesn’t happen again. Ownership carries a lot of responsibility, but it also carries a lot of trust – because to let an individual or a team own a service, you have to give them the leeway to make their own decisions about how they are going to deliver it. It’s been a great lesson for me to realize how much allowing individuals and teams to directly own software, and more generally own a portion of the business, allows them to be passionate about what they do and really push on it. It’s also remarkable how much getting ownership wrong can have the opposite result.

Encouraging ownership in others

I’ve spent a lot of time at Amazon thinking about how important and effective the focus on ownership is to the business, but also about how effective an individual tool it is when I work with engineers and teams. I realized that the idea of recognizing and encouraging ownership had actually been a really effective tool for me in other roles. Here’s an example: In my early days as a professor at UBC, I was working with my first set of graduate students and trying to figure out how to choose great research problems for my lab. I vividly remember a conversation I had with a colleague that was also a pretty new professor at another school. When I asked them how they choose research problems with their students, they flipped. They had a surprisingly frustrated reaction. “I can’t figure this out at all. I have like 5 projects I want students to do. I’ve written them up. They hum and haw and pick one up but it never works out. I could do the projects faster myself than I can teach them to do it.”

And ultimately, that’s actually what this person did — they were amazing, they did a bunch of really cool stuff, and wrote some great papers, and then went and joined a company and did even more cool stuff. But when I talked to grad students that worked with them what I heard was, “I just couldn’t get invested in that thing. It wasn’t my idea.”

As a professor, that was a pivotal moment for me. From that point forward, when I worked with students, I tried really hard to ask questions, and listen, and be excited and enthusiastic. But ultimately, my most successful research projects were never

mine. They were my students and I was lucky to be involved. The thing that I don't think I really internalized until much later, working with teams at Amazon, was that one big contribution to those projects being successful was that the students really did own them. Once students really felt like they were working on their own ideas, and that they could personally evolve it and drive it to a new result or insight, it was never difficult to get them to really invest in the work and the thinking to develop and deliver it. They just had to own it.

And this is probably one area of my role at Amazon that I've thought about and tried to develop and be more intentional about than anything else I do. As a really senior engineer in the company, of course I have strong opinions and I absolutely have a technical agenda. But If I interact with engineers by just trying to dispense ideas, it's really hard for any of us to be successful. It's a lot harder to get invested in an idea that you don't own. So, when I work with teams, I've kind of taken the strategy that my best ideas are the ones that other people have instead of me. I consciously spend a lot more time trying to develop problems, and to do a really good job of articulating them, rather than trying to pitch solutions. **There are often multiple ways to solve a problem, and picking the right one is letting someone own the solution.** And I spend a lot of time being enthusiastic about how those solutions are developing (which is pretty easy) and encouraging folks to figure out how to have urgency and go faster (which is often a little more complex). But it has, very sincerely, been one of the most rewarding parts of my role at Amazon to approach scaling myself as an engineer being measured by making other engineers and teams successful, helping them own problems, and celebrating the wins that they achieve.

Closing thought

I came to Amazon expecting to work on a really big and complex piece of storage software. What I learned was that every aspect of my role was unbelievably bigger than that expectation. I've learned that the technical scale of the system is so enormous, that its workload, structure, and operations are not just bigger, but foundationally different from the smaller systems that I'd worked on in the past. I learned that it wasn't enough to think about the software, that "the system" was also the software's operation as a service, the organization that ran it, and the customer code that worked with it. I learned that the organization itself, as part of the system, had its own scaling challenges and provided just as many problems to solve and opportunities to innovate. And finally, I learned that to really be successful in my own role, I needed to focus on articulating the problems and not the solutions, and to find ways to support strong engineering teams in really owning those solutions.

I'm hardly done figuring any of this stuff out, but I sure feel like I've learned a bunch so far. Thanks for taking the time to listen.

Related posts (and papers)

- Amazon's Dynamo
- Eventually consistent - revisited
- Using lightweight formal methods to validate key-value storage node in Amazon S3
