Pat Helland on Failure and Resilience in Distributed Systems

Edaena Salinas

From the Editor

PAT HELLAND OF SALESFORCE (disclosure: Salesforce employs both Helland and me) talks with show-host Edaena Salinas about failure and resilience in distributed systems. The interview covers the growth of distributed computing on the public cloud, the origin of failure in the manufacturing process of hardware, the lifecycle of a server, why servers need to be replaced every three years, self-healing systems, replication, and the consistency, availability, and partition-tolerance (CAP) theorem. Sections that were not transcribed, due to space limits, cover how mutability makes life harder, append-only data, how electricity costs dominate, “cattle versus pets,” the rise of DevOps automation, testing, testing in production, and monitoring. The entire interview is available on our website at http://www.se-radio.net/2018/10/se-radio-episode-344-pat-helland-on-web-scale/, in Really Simple Syndication, and in the iTunes store. —Robert Blumen

Edaena Salinas: I have with me Pat Helland. Pat is a principal software architect at Salesforce, where he works on cloud-based multitenant database systems. Pat has been working on databases, transactions, and distributed systems since 1978. Previously, he worked on the big data plumbing for Bing at Microsoft. Pat coauthored the article “Too Big Not to Fail: Embrace Failure, so It Doesn’t Embrace You” for ACM Queue, where Pat also writes a regular column. Pat, welcome to the show. Today, we’re going to be talking about web scale, failure, and recovery. What is web-scale computing?

Pat Helland: Google, Amazon, Salesforce do work on demand. These sites get larger and smaller over time—throughout the day and the seasons. Shopping season can make a big difference. You have to provision your data centers to accept a lot of traffic and to manage larger and smaller amounts. You end up with hundreds or thousands of servers working together.

[Web-scale computing is] the management of their lifecycle.

Edaena Salinas: Web-scale computing involves hardware and software. We also have VMs [virtual machines] and containers. Can you explain these components?

Pat Helland: Sure. A server is a physical box. You can walk into a data center and kick a server; it takes electricity. A VM is a virtual machine. That allows software to pretend that the VM is a server. You can run lots of VMs—things
that look like physical servers—on the same [physical] server, sharing the resources, and using [the server’s operating system] to protect one VM from another. Containers are a technology that has emerged over the last 10–15 years to allow the easy deployment of software. They are lighter weight than VMs. They run inside of VMs or on a physical server as microservices.

Edaena Salinas: You have spent the majority of your almost-40-year career working on transactions and distributed systems. In your ACM column “Life Beyond Distributed Transactions,” you talk about a thought experiment. Can you explain?

Pat Helland: A transaction is when all of it or none of it must happen. That is so the programmer doesn’t have to think about partial results. They don’t have to worry about “this” happening without “that” happening. When you’re making a bunch of changes, it’s hard to make sure [the computation] keeps going when work is split into pieces across multiple servers and one piece of it fails. It’s hard to ensure this. There are some people who argue it’s possible, but it’s not easy. Certainly [it’s very difficult to do] across trust environments [not all components are part of the same department or company]. How do you commit a transaction? Did it all happen or none of it happen? Marriage means that both of them are married or neither. What if you’re getting married on the telephone? The person you want to marry is in another country. [The person conducting it] asks you: “Will you take this person to be your wife or husband?” You say: “I do.” But then [the call drops]. There’s a real dilemma here: Is the husband married, but the wife is not married? Are you allowed to go on a date tonight? An atomic transaction depends on being able to communicate. The CAP conjecture has proven that you can’t maintain availability when there are failures in between the servers. Consistency, availability, and partition tolerance: Can you have all three? The classic notion of consistency means that you and I both want to make a change where we agree on what that change is.

Edaena Salinas: You are talking about consistency, availability, and partition tolerance. Is the phone call about consistency?

Pat Helland: Partition tolerance. When I got the dial tone, I got a partition. I can’t communicate over the telephone when the call dropped. Either we’re married or we’re not married. Now, I need to find out: Did I and the person I want to marry both have the same outcome? The typical answer is, you wait until you reconnect to find out the other party said [before the call dropped]. Availability suffers until you get a consistent answer. If you want to do your own thing without waiting for the reconnection, [then you have] availability; but if you don’t know the outcome, then you give up consistency.

Edaena Salinas: This example you just gave illustrates a failure happening when a component breaks. What are some of the reasons why a server in a data center can fail?

Pat Helland: Servers are incredibly reliable. But there are lots of them. A lot of things can cause a physical box that you purchased from the vendor to no longer function. Disk drives are spinning things; they fail at a rate of one in four or five years. Power supplies can go out. The way I think about servers in a data center is like a bag of nails on a construction site. They’re all the same. Once in a while one’s bent, so you throw it aside. They don’t have an individual personality. No one walks into a construction site loving and caring for each individual nail. You make sure you have enough nails by weight to have extra so you can cope with things going wrong. Some nails get thrown away.

When an individual server breaks, a cluster of computers in a data center must continue to provide the web service that it signed up for. If you have thousands of servers, when you go home for the weekend, there are going to be some broken Monday morning. On Monday you figure which ones to unplug, and then replace them.

Edaena Salinas: That’s a good analogy, because in the construction, if you get a broken nail, the whole construction is not shut down, right?

Pat Helland: They better not, because you’re going to have broken nails. You don’t get emotional about it. You don’t hold a funeral for the nails. You bring extra nails.

Edaena Salinas: What is an example of designing to expect failure without having human intervention to maintain availability?

Pat Helland: Let’s talk about a distributed system that holds data. HDFS [Hadoop Distributed File System] is a great example. Every piece of data is in a block, and every block is put on three different servers. One part of the system is constantly asking: “Can I still talk to all the servers that have got my data?” I may have an HDFS cluster of 20,000 servers. The main node is talking to them to make sure they’re alive. If one of them is not, then it knows where the copies of the
data are. The second and third copies of the blocks on the failed server are on different servers, spread around the data center.

If you drop those server replicas from three to two, you have a problem, not a crisis, but you have a problem. The problem gets solved by finding a place to put a third copy for each one of them. The main node tells each of the [surviving] servers containing those blocks: “Put [a copy of] that block in a [new] third location.” You want to spread it around.

If you do this well, and HDFS does, it takes less time to get back to having three copies of every block than it would to copy all of the data out of the one server that died. The reason is that the second and third copies are in many different servers, which can talk to so many other servers in parallel. You can cope with the loss of a server in much less time than it would take to read and write a single server. When you are back to three replicas of every block of data, you can continue. My joking way of describing this is, walk into a data center with a baseball bat and whack a rack of servers. That drops the replicas [data on that server] down to two.

The system makes additional replicas spread out across lots of different racks. By the time you raise the baseball bat to hit another rack, everything is back to three, so it can take another hit. That’s, of course, an exaggeration, but it shows how to think about failure. The failure is quickly repaired. The faster it’s repaired, the more available the underlying data are.

Edaena Salinas: I see at least two design principles here. One is, first of all, have your data replicated. Have more than one copy of it available and also recover quickly, if those data are not available.

Pat Helland: Correct. Availability is a function of how often failures occur. We characterize this as *mean time between failure* and, how quickly you can recover from the failure, *mean time to repair*. If you can recover quickly, your overall availability goes up.

The algorithms for dealing with data are easier if the underlying data are not changing—data you wrote once and you’re going to read a bunch of times, maybe over years, and eventually throw away. But you’re not going to change it. You write other data rather than changing the original. What that means is, as you lose replicas and create new ones, you don’t have to worry about the data being changed. That’s a huge benefit.

Edaena Salinas: One example I can think of is looking up a book. It’s already been written. It’s not the same as a bank-account balance.

Pat Helland: The revision of the book is actually a different book in this approach. The book has two revisions—revision one and revision two. Each revision is a unique thing.

You mentioned bank accounts. Bank accounts, it turns out, are handled as immutable data. If you’re looking at your May bank statement, to understand it you have to look through the January-to-April statements to understand what’s going on. If there is a problem in your May bank statement, you call the bank and they correct the problem. But they don’t issue a new May bank statement. They issue a change in the June bank statement that has not yet been sent to you. From the perspective of the bank, once a
transaction is written, it is never going to be changed. Changes are new things that are written and then they show up later on your statement.

Immutability works much better in a distributed system. I wrote the paper called “Immutability Changes Everything” to raise awareness about this. We’re evolving to more and more data being treated as immutable data.

Edaena Salinas: I want to talk more about the hardware in the data centers. In “Too Big Not to Fail,” you talk about the cost of changing hardware every three years. Why is it not cost-effective to keep the hardware longer than that?

Pat Helland: The new hardware is more efficient than the old hardware. If you had a car from the 1960s and it was getting 10 miles to the gallon, and gasoline was the largest cost of owning the car, then you might want to go down and get yourself a new car that gets 40 miles to the gallon, because that lowers your cost of ownership.

In data centers today, electricity and the physical building are larger components to the lifetime cost of a server than the cost of the server itself. If you can only fit so many racks of servers into your building, and you can only wire so much electricity, then electricity costs dominate. If a computer does twice as much computation with the same amount of electricity, then when does it become a good idea to spend money to get that computer sooner? There’s no perfect answer here. A ballpark is about every three years. The costs of replacing include decommissioning the old server and commissioning the new server. To decommission it, you have to get all the traffic off, get all the data you want to keep off it, unplug it, and, in many cases, put the physical drives through a shredder to prevent data being snooped on.

When you commission the machine, you bring it to the data center, verify it, install software, load data, and then send traffic to it. Each of those things takes time. If I’ve only got three years between buying a server and throwing it away, how long does it take me to set it up and commission it, and how long do I spend decommissioning it? Those steps consume part of your three-year window. This is the routine of cycling through hardware in a data center. I’m fond of saying electronics and roast beef are worthless next year.

If you plot time on the x-axis against failure on the y-axis, you get what’s called a bathtub failure curve. When you plug in a brand-new server some of them fail right away. As you use them for months the [failure rate] drops; they’re very reliable; not much breaks. When you start getting past three-to-five years, the failure rate goes up. The shape of the curve looks like a bathtub. You want to test your way out of the initial failures that are happening. And you want to retire the older servers before they start getting flaky later on in their lifetime.

Edaena Salinas: With the rise of cloud computing, the commission and decommission are handled by the cloud provider. For example, Amazon, Google.

Pat Helland: Correct. Amazon, Google, Microsoft. That’s part of the cloud IaaS [infrastructure-as-a-service] approach. You’re buying virtual machines. The cloud provider will take care of hardware and electricity. They worry about getting Internet coming in and going out. They can do it more cost-effectively at volume.

Edaena Salinas: What I’m trying to understand is when you see failures at the beginning, is that because the setup wasn’t correct? Was there a misconfiguration?

Pat Helland: It’s due to the funkiness of manufacturing. The factory burns it in for a while before they deliver it to you. When you get it into the data center, you stack it in a rack, and then you burn it in some more before you send traffic to it. You try to eliminate the initial failures. But you can’t burn it in long enough that every server you put in production doesn’t have initial failure windows.

Edaena Salinas: Why is the failure rate higher on the brand-new servers?

Pat Helland: It’s common for manufactured stuff to fail promptly in its lifetime: cars, TVs, toasters. That’s why you see the 30-day warranty on a lot of things. You take it home, and within days it doesn’t work. The ones that work, as time goes on, wear out different parts in different ways. As they wear out, they fail more often.

**ABOUT THE AUTHOR**

EDAENA SALINAS is a software engineer at Microsoft, where she has worked on web applications on front- and back-end development since 2014. Her research interests include software engineering, web development, artificial intelligence, testing, and DevOps. Contact her at edaena@thewomenintechshow.com.