CS 4530: Fundamentals of Software Engineering

Module 9.1 Distributed Systems: Goals and Challenges

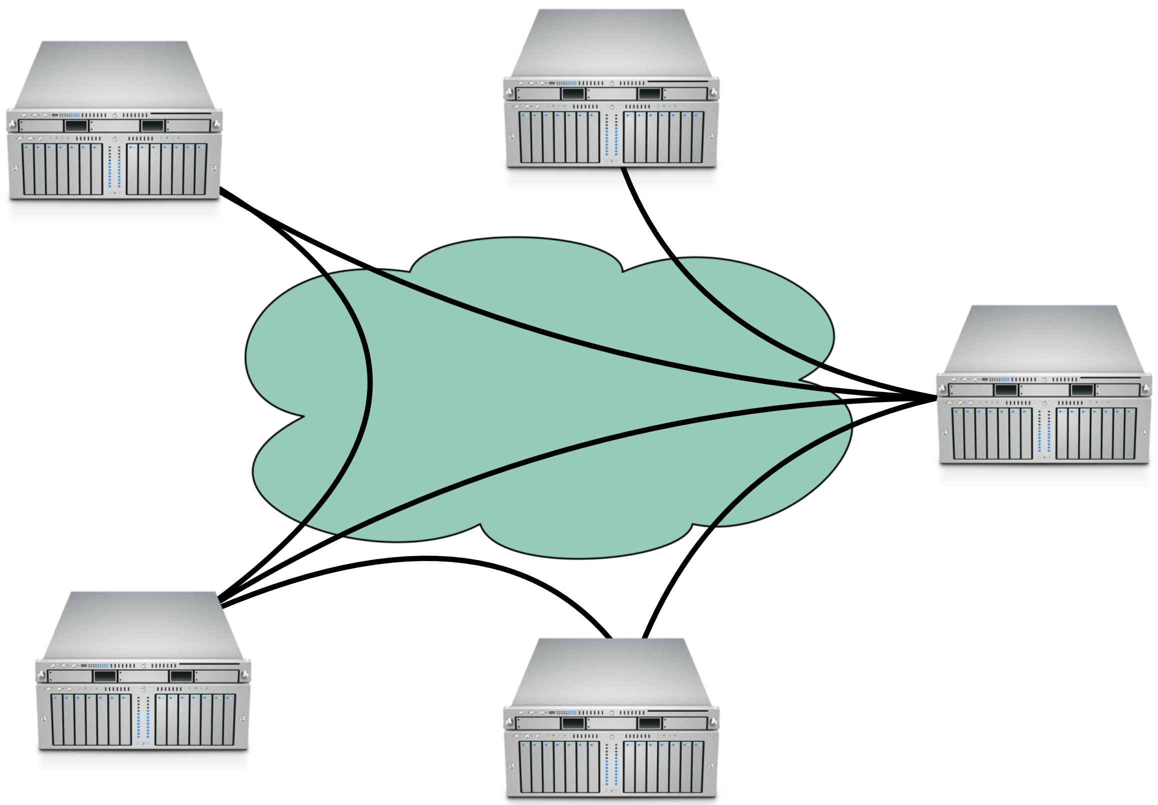
Adeel Bhutta, Mitch Wand

Khoury College of Computer Sciences

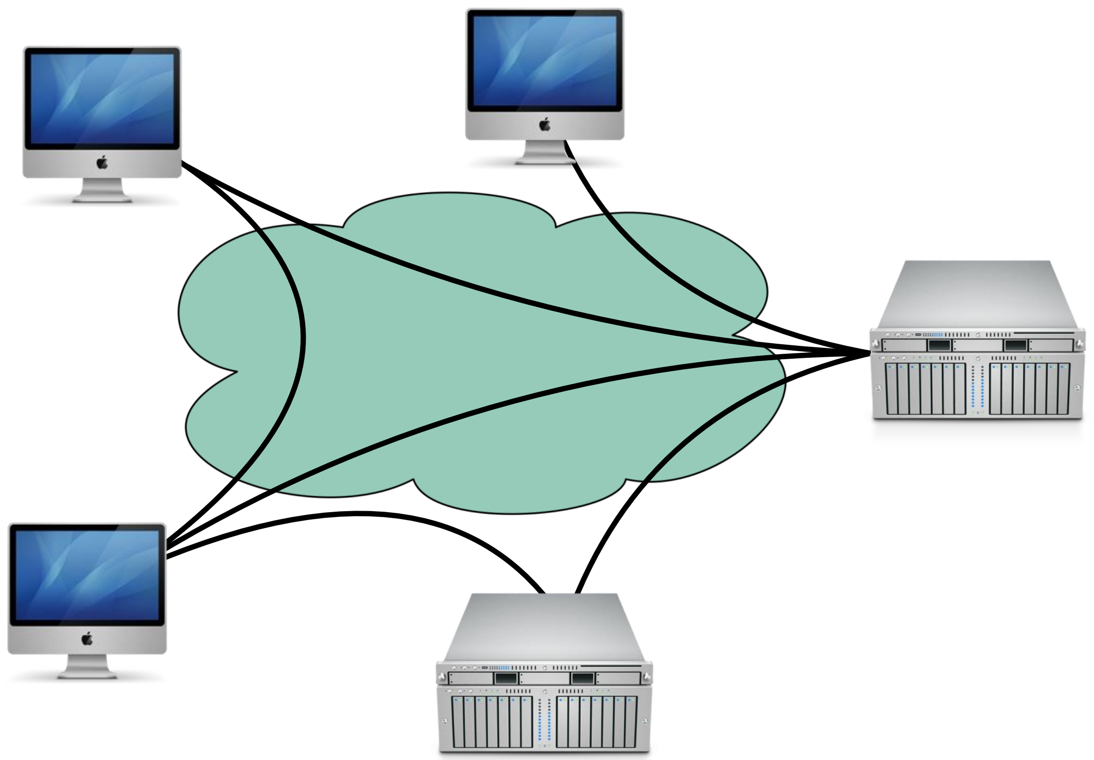
Learning Goals for this Lesson

- At the end of this lesson you should be able to
  - List and define 5 goals of using distributed systems
  - List 4 major challenges inherent in distributed systems

What is a distributed system?



Model One:
Many servers talking through a network



Model Two:
Many servers and clients talking through a network

Distributed Systems Goals

- Scalability
- Performance
- Latency
- Availability
- Fault Tolerance

“Distributed Systems for Fun and Profit”, Takada

Distributed Systems Goals

- **Scalability**
- Performance
- Latency
- Availability
- Fault Tolerance

"the ability of a system, network, or process, to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth."

"Distributed Systems for Fun and Profit", Takada

Distributed Systems Allow Horizontal Scaling

- “Vertical” scaling: add more resources to existing server
  - Faster CPUs, more CPU cores, more RAM, more storage
  - Becomes ineffective : Clock speed plateaus; difficult to write applications that utilize 256 CPU cores (though adding 2TB RAM to a server *can* often help)
- “Horizontal” scaling: add more servers
  - Rely on “commodity” servers rather than state-of-the-art hardware
  - Allows for dynamic addition of resources as needed by load

Distributed Systems Goals

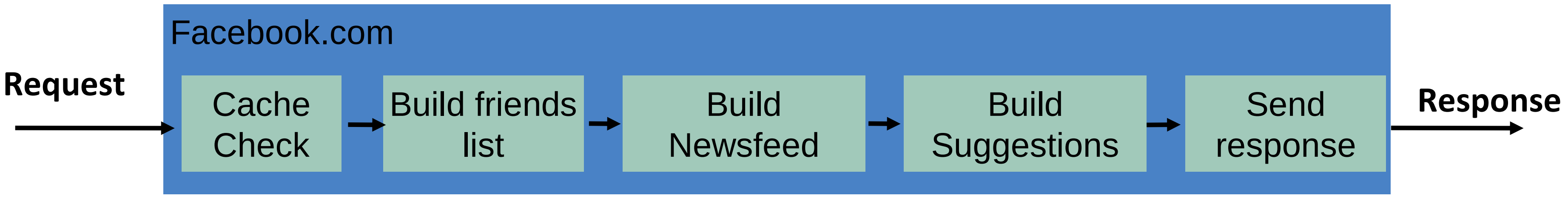
- Scalability
- **Performance**
- Latency
- Availability
- Fault Tolerance

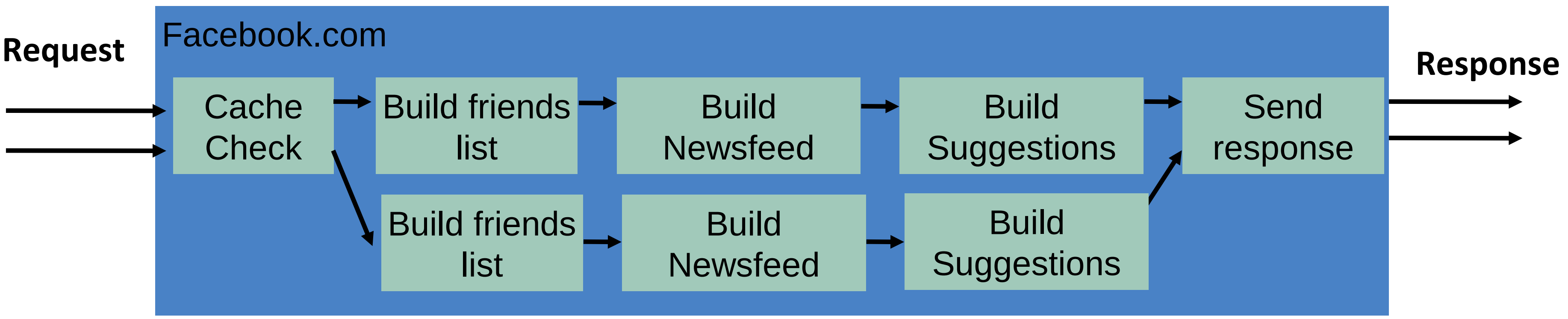
"The amount of useful work accomplished by a computer system compared to the time and resources used."

"Distributed Systems for Fun and Profit", Takada

Multiple Servers Can Improve Throughput With Concurrency

Throughput: total requests that can be processed per unit-time





Distributed Systems Goals

- Scalability
- Performance
- **Latency**
- Availability
- Fault Tolerance

The time during which something that has already happened is concealed from view.

In a multi-server system, we can select a server that is closer to the user.

“Distributed Systems for Fun and Profit”, Takada

Reduce latency by distributing data

- Move or replicate the data
  - Avoid bottlenecks
  - Decrease transmission time

“Distributed Systems for Fun and Profit”, Takada

Distributed Systems Goals

- Scalability
- Performance
- Latency
- **Availability**
- Fault Tolerance

"the proportion of time a system is in a functioning condition."

Availability = uptime / (uptime + downtime).

Often measured in "nines"

Availability %	Downtime/year
90%	>1 month
99%	< 4 days
99.9%	< 9 hours
99.99%	<1 hour
99.999%	5 minutes
99.9999%	31 seconds

"Distributed Systems for Fun and Profit", Takada

Distributed Systems can improve availability by replicating servers

- A single-server system is either up or down.
- If you have many servers, the probability that some server is down increases
- BUT: the probability that all servers are down decreases (exponentially!)

Here’s a crude quantitative model

- Say there’s a 1% chance of having some hardware failure occur to a machine in a given month (power supply burns out, hard disk crashes, etc)
- Now I have 10 machines
  - Probability(at least one fails during the month) = 1 - Probability(no machine fails) = $1-(1-.01)^{10}$ = 10%
- 100 machines -> 63% chance that at least one fails
- Chance that all machines fail during the month: $(.01)^{10} = 10^{-12}$

Distributed Systems Goals

- Scalability
- Performance
- Latency
- Availability
- **Fault Tolerance**

"ability of a system to behave in a well-defined manner once faults occur"

[“Distributed Systems for Fun and Profit”, Takada]

Design to expect faults

- “Define what faults you expect and then design a system or an algorithm that is tolerant of them. You can't tolerate faults you haven't considered.”

What kind of faults?

Disks fail

Power supplies fail

Power goes out

Networking fails

Security breached

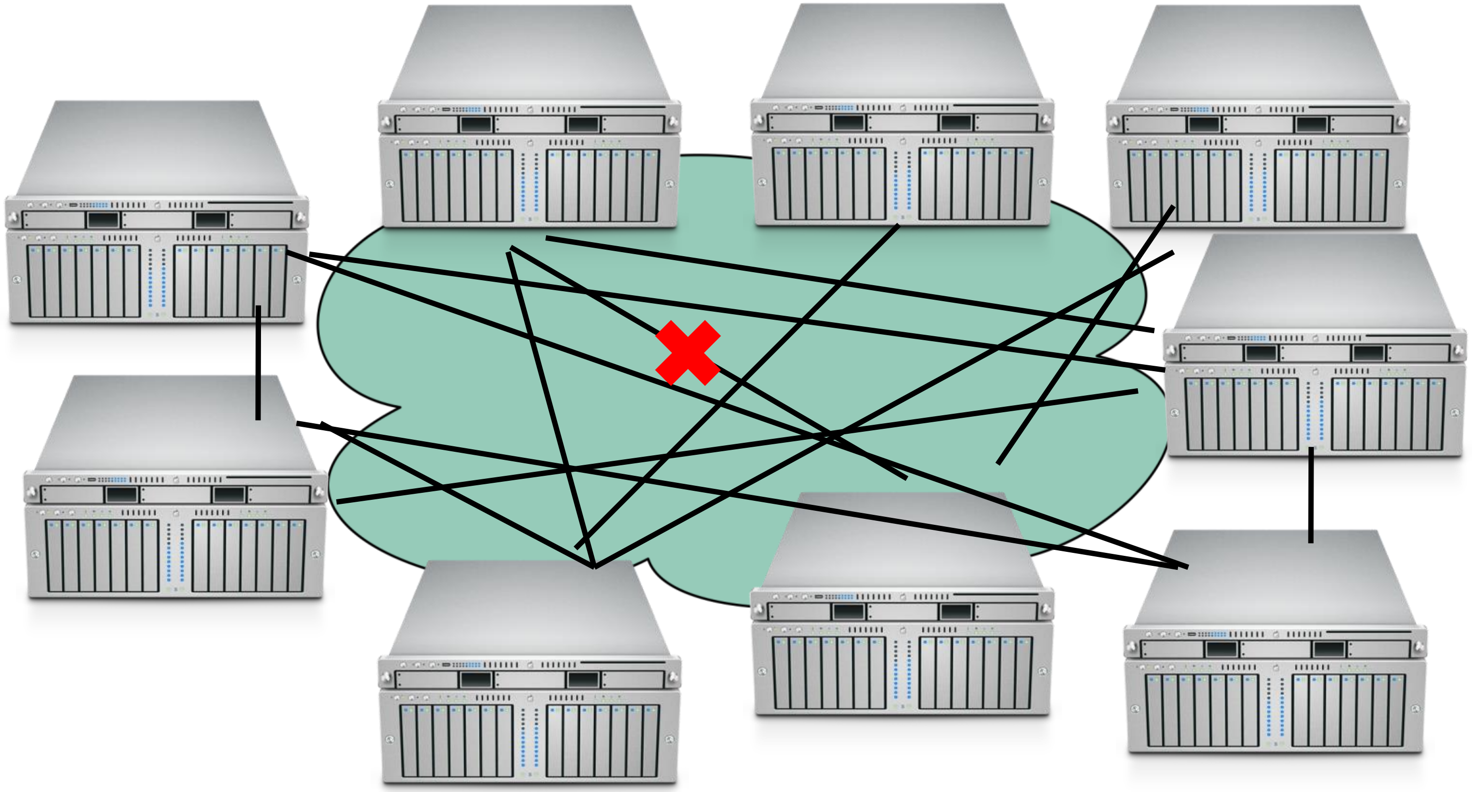
Datacenter goes offline

“Distributed Systems for Fun and Profit”, Takada

Distributed Systems Challenges

More machines means more links that might fail.

- Number of nodes + distance between them



Networks introduce delays

- Cannot expect network to be a perfect analog for communication within a single computer because:
  - Speed of light (1 foot/nanosecond)
  - Communication links exist in uncontrolled/hostile environments
  - Communication links may be bandwidth limited (tough to reach even 100MB/sec)
- In contrast to a single computer, where:
  - Distances are measured in mm, not feet
  - Physical concerns can be addressed all at once
  - Bandwidth is plentiful (easily GB/sec)

Worse yet, networks still fail, intermittently and for prolonged periods


arstechnica.com

ars TECHNICA SUBSCRIBE SIGN IN

BIZ & IT —

The discovery of Apache ZooKeeper's poison packet

How PagerDuty found four different bugs.

EVAN GILMAN - 5/13/2015, 9:00 AM

Evan Gilman is an operations engineer at PagerDuty, responsible for designing and automating the company's resilient infrastructure. Prior to PagerDuty, he operated AS4511 at the University of Miami, and has a passion for all things network. This story originally appeared on PagerDuty.

ZooKeeper, for those who are unaware, is a well-known open source project that enables highly reliable distributed coordination. It is trusted by many around the world, including PagerDuty. It provides high availability and linearizability through the concept of a leader, which can be dynamically re-elected, and ensures consistency through a majority quorum.

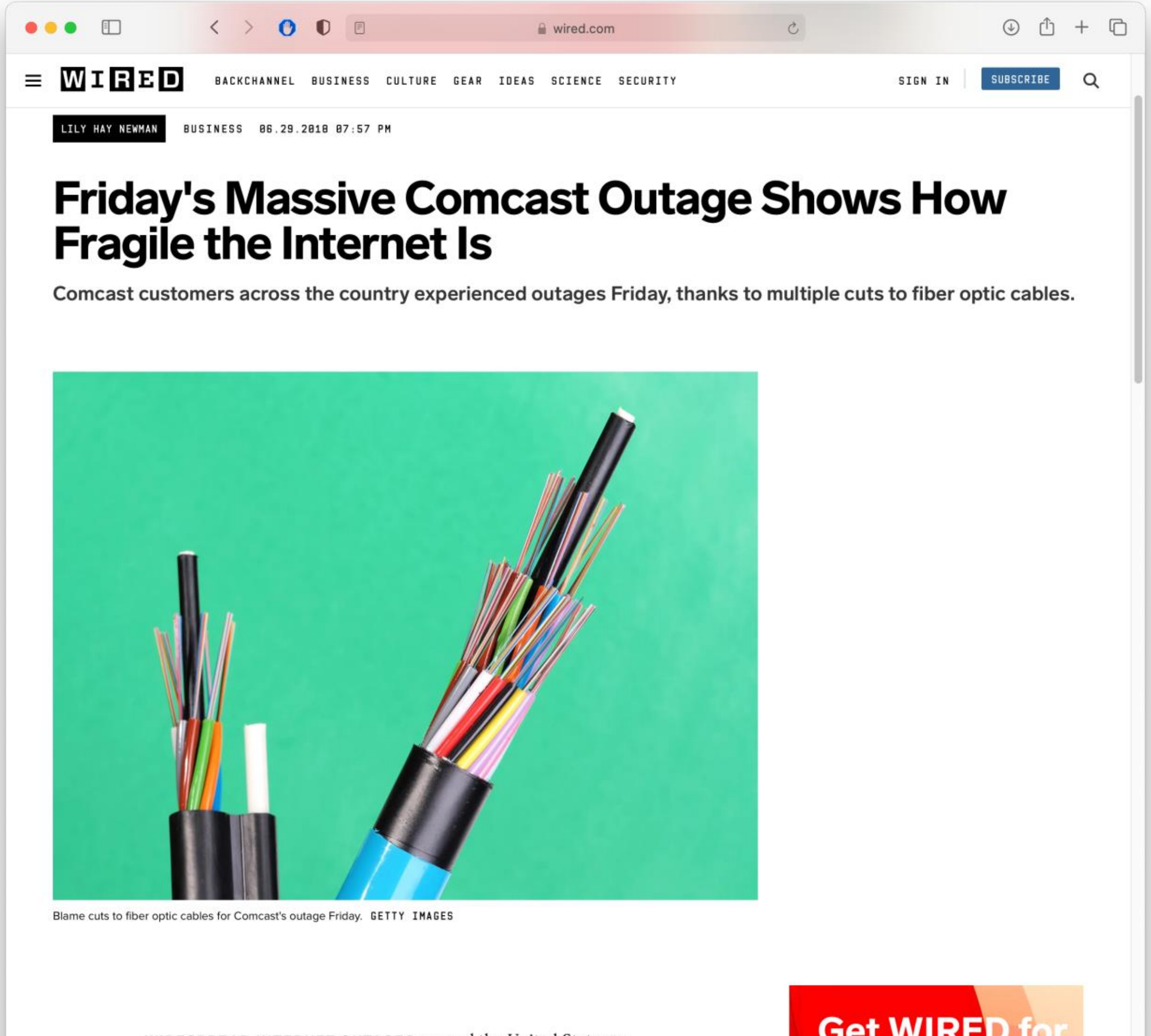
The leader election and failure detection mechanisms are fairly mature, and typically just work... until they don't. How can this be? Well, after a lengthy investigation, we managed to uncover four different bugs coming together to conspire against us, resulting in random cluster-wide lockups. Two of those bugs lay in ZooKeeper, and the other two were lurking in the Linux kernel. This is our story.

Background: The use of ZooKeeper at PagerDuty

Here at PagerDuty, we have several disparate services that power our alerting pipeline. As events are received, they traverse these services as a series of tasks that get picked up off of various work queues. Each one of these services leverages a dedicated ZooKeeper cluster to coordinate which application host processes each task. As such, you can imagine that ZooKeeper operations are absolutely critical to the reliability of PagerDuty at large.

Part I: The ZooKeeper bugs

One day last year, an engineer noticed that one of the ZooKeeper clusters in our load test environment was broken. It manifested itself as lock timeouts in the dependent application. We confirmed that the


wired.com

WIRED BACKCHANNEL BUSINESS CULTURE GEAR IDEAS SCIENCE SECURITY SIGN IN SUBSCRIBE

LILY HAY NEWMAN BUSINESS 06.29.2018 07:57 PM

Friday's Massive Comcast Outage Shows How Fragile the Internet Is

Comcast customers across the country experienced outages Friday, thanks to multiple cuts to fiber optic cables.

Blame cuts to fiber optic cables for Comcast's outage Friday. GETTY IMAGES

Get WIRED for

We still rely on other administrators, who are not infallible

Amazon Web Services outage takes a portion of the internet down with it

Zack Whittaker

Comment

@zackwhittaker / 12:32 PM EST • November 25, 2020

Image Credits: David Becker / Getty Images

Amazon Web Services is currently having an outage, taking a chunk of the internet down with it.

Several AWS services were experiencing problems as of early Wednesday, according to its status page. That means any app, site or service that relies on AWS might also be down, too. (As I found out the hard way this morning when

aws.amazon.com

aws

Contact Sales Support English My Account Sign In to the Console

Products Solutions Pricing Documentation Learn Partner Network AWS Marketplace Customer Enablement Events Explore More

Summary of the Amazon Kinesis Event in the Northern Virginia (US-EAST-1) Region

November, 25th 2020

We wanted to provide you with some additional information about the service disruption that occurred in the Northern Virginia (US-EAST-1) Region on November 25th, 2020.

Amazon Kinesis enables real-time processing of streaming data. In addition to its direct use by customers, Kinesis is used by several other AWS services. These services also saw impact during the event. The trigger, though not root cause, for the event was a relatively small addition of capacity that began to be added to the service at 2:44 AM PST, finishing at 3:47 AM PST. Kinesis has a large number of "back-end" cell-clusters that process streams. These are the workhorses in Kinesis, providing distribution, access, and scalability for stream processing. Streams are spread across the back-end through a sharding mechanism owned by a "front-end" fleet of servers. A back-end cluster owns many shards and provides a consistent scaling unit and fault-isolation. The front-end's job is small but important. It handles authentication, throttling, and request-routing to the correct stream-shards on the back-end clusters.

The capacity addition was being made to the front-end fleet. Each server in the front-end fleet maintains a cache of information, including membership details and shard ownership for the back-end clusters, called a shard-map. This information is obtained through calls to a microservice vending the membership information, retrieval of configuration information from DynamoDB, and continuous processing of messages from other Kinesis front-end servers. For the latter communication, each front-end server creates operating system threads for each of the other servers in the front-end fleet. Upon any addition of capacity, the servers that are already operating members of the fleet will learn of new servers joining and establish the appropriate threads. It takes up to an hour for any existing front-end fleet member to learn of new participants.

At 5:15 AM PST, the first alarms began firing for errors on putting and getting Kinesis records. Teams engaged and began reviewing logs. While the new capacity was a suspect, there were a number of errors that were unrelated to the new capacity and would likely persist even if the capacity were to be removed. Still, as a precaution, we began removing the new capacity while researching the other errors. The diagnosis work was slowed by the variety of errors observed. We were seeing errors in all aspects of the various calls being made by existing and new members of the front-end fleet, exacerbating our ability to separate side-effects from the root cause. At 7:51 AM PST, we had narrowed the root cause to a couple of candidates and determined that any of the most likely sources of the problem would require a full restart of the front-end fleet, which the Kinesis team knew would be a long and careful process. The resources within a front-end server that are used to populate the shard-map compete with the resources that are used to process incoming requests. So, bringing front-end servers back online too quickly would create contention between these two needs and result in very few resources being available to handle incoming requests, leading to increased errors and request latencies. As a result, these slow front-end servers could be deemed unhealthy and removed from the fleet, which in turn, would set back the recovery process. All of the candidate solutions involved changing every front-end server's configuration and restarting it. While the leading candidate (an issue that seemed to be creating memory pressure) looked promising, if we were wrong, we would double the recovery time as we would need to apply a second fix and restart again. To speed restart, in parallel with our investigation, we began adding a configuration to the front-end servers to obtain data directly from the authoritative metadata store rather than from front-end server neighbors during the bootstrap process.

At 9:39 AM PST, we were able to confirm a root cause, and it turned out this wasn't driven by memory pressure. Rather, the new capacity had caused all of the servers in the fleet to exceed the maximum number of threads allowed by an operating system configuration. As this limit was being exceeded,

Learning Goals for this Lesson

- You should now be able to
  - List and define 5 goals of using distributed systems
  - List 4 major challenges inherent in distributed systems