CMPT 473
Software Testing, Reliability and Security

Chaos Engineering

Nick Sumner
wsumner@sfu.ca

Distributed systems are challenging & pervasive

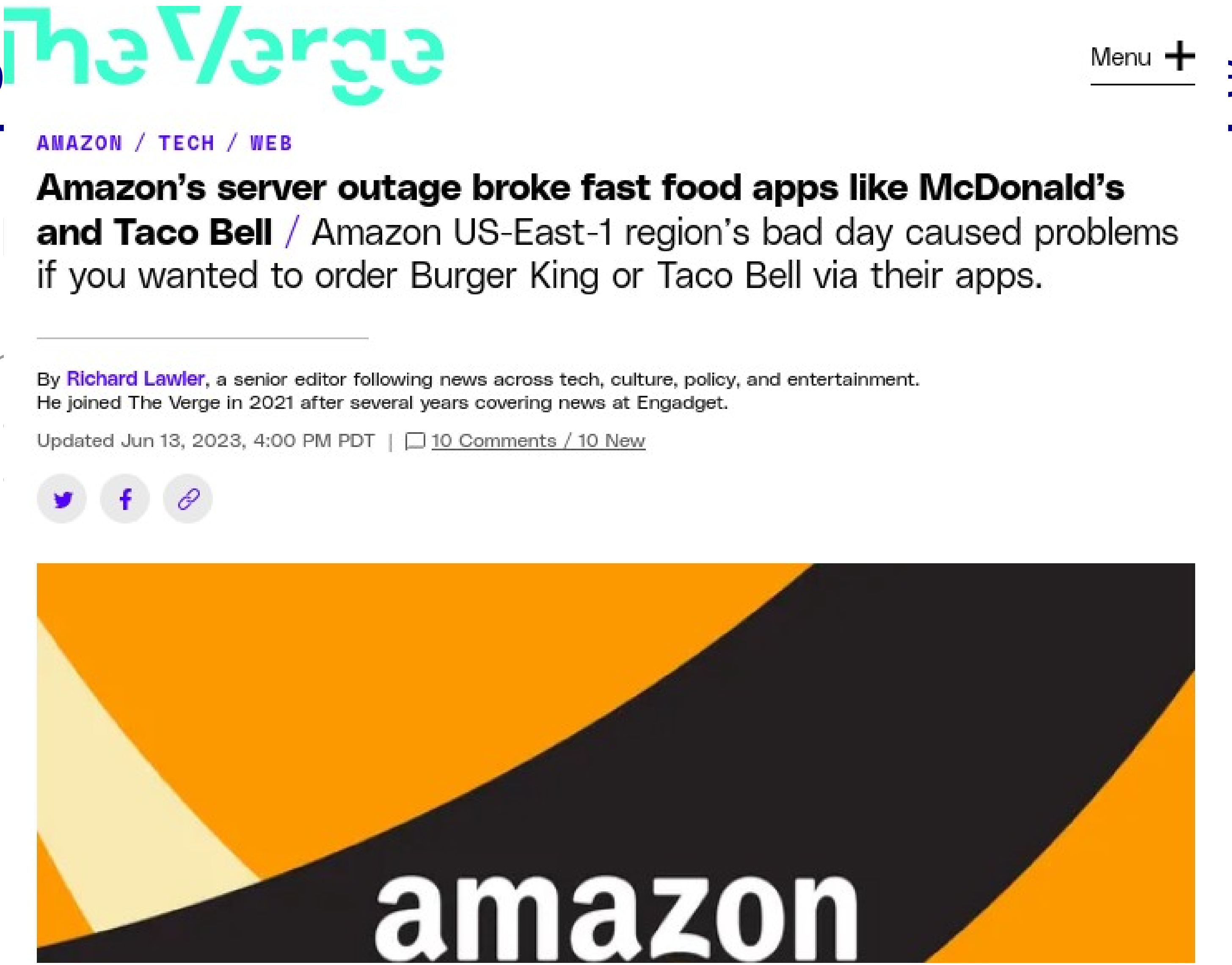
- Distributed applications face many hurdles

Distributed systems are challenging & pervasive

- Distributed applications face many hurdles
  - Multiple participants
  - Unreliable communication channels
  - May be allowed to crash
  - May need to tolerate malicious participants
  - Must eventually agree an some set of decisions

Distrib

- Distri
  - M
  - Ur
  - M
  - M
  - M



Menu +

AMAZON / TECH / WEB

Amazon's server outage broke fast food apps like McDonald's and Taco Bell / Amazon US-East-1 region's bad day caused problems if you wanted to order Burger King or Taco Bell via their apps.

By Richard Lawler, a senior editor following news across tech, culture, policy, and entertainment. He joined The Verge in 2021 after several years covering news at Engadget.

Updated Jun 13, 2023, 4:00 PM PDT | 10 Comments / 10 New

Distributed systems are challenging & pervasive

- Distributed applications face many hurdles
  - Multiple participants
  - Unreliable communication channels
  - May be allowed to crash
  - May need to tolerate malicious participants
  - Must eventually agree an some set of decisions
- Every one of these challenges makes application writing harder

Distributed systems are challenging & pervasive

- Distributed applications face many hurdles
  - Multiple participants
  - Unreliable communication channels
  - May be allowed to crash
  - May need to tolerate malicious participants
  - Must eventually agree an some set of decisions
- Every one of these challenges makes application writing harder
- And yet the trends (good or bad) are pushing in this direction
  - SOA & Microservices
  - IoT
  - Control systems
  - *coin & smart contracts
  - …

How do the solutions we know fit in?

- In general
  - Failure is always an option
  - Ordering is hard
  - Agreement is hard
  - The software and what you think it does may differ

How do the solutions we know fit in?

- In general
  - Failure is always an option
  - Ordering is hard
  - Agreement is hard
  - The software and what you think it does may differ
- How do TLA+ and similar tools fit into the picture?
  - Safety
  - Liveness
  - Fairness
  - ***Actual behavior?***
  - ***Performance?***

How do the solutions we know fit in?

- In general
  - Failure is always an option
  - Ordering is hard
  - Agreement is hard
  - The software and what you think it does may differ
- How do TLA+ and similar tools fit into the picture?
  - Safety
  - Liveness
  - Fairness
  - ***Actual behavior?***
  - ***Performance?***
- Spec. verification still faces challenges on more empirical issues

Focus on experimentation

- Instead, we can again perform *experiments*
  on the *live* system
  targeted at particular problems or goals

Focus on experimentation

- Instead, we can again perform *experiments*
  on the *live* system
  targeted at particular problems or goals
- *Chaos engineering*
  The discipline of **experimenting** on system
  in order to **build confidence** in the system's capability to
  withstand turbulent conditions in production [Principles Of Chaos]

Focus on experimentation

- Instead, we can again perform *experiments*
  on the *live* system
  targeted at particular problems or goals
- *Chaos engineering*
  The discipline of experimenting on system
  in order to build confidence in the system's capability to
  withstand turbulent conditions in production [Principles Of Chaos]
- You can think about chaos engineering as
  A/B testing for distributed systems

Focus on experimentation

- Instead, we can again perform *experiments*
  on the *live* system
  targeted at particular problems or goals
- *Chaos engineering*
  The discipline of experimenting on system
  in order to build confidence in the system's capability to
  withstand turbulent conditions in production [Principles Of Chaos]
- You can think about chaos engineering as
  A/B testing for distributed systems
  where tests focus on pathologies of system reliability

Focus on experimentation

- Instead, we can again perform *experiments*
  on the *live* system
  targeted at particular problems or goals
- *Chaos engineering*
  The discipline of experimenting on system
  in order to build confidence in the system's capability to
  withstand turbulent conditions in production [Principles Of Chaos]
- You can think about chaos engineering as
  A/B testing for distributed systems
  where tests focus on pathologies of system reliability
- Instead of looking for improvements, you look for degradation

Focus on experimentation

- Instead, we can again perform *experiments*
  on the *live* system
  targeted at particular problems or goals
- *Chaos engineering*
  The discipline of experimenting on system
  in order to build confidence in the system's capability to
  withstand turbulent conditions in production [Principles Of Chaos]
- You can think about chaos engineering as
  A/B testing for distributed systems
  where tests focus on pathologies of system reliability
- Instead of looking for improvements, you look for degradation
- Chaos engineering is about finding the latent chaos in the system

The 8 fallacies of distributed computing

- Common mistakes from Lyon, Deutsch, & Gosling
  1) The network is reliable
  2) Latency is zero
  3) Bandwidth is infinite
  4) The network is secure
  5) Topology doesn't change
  6) There is one administrator
  7) Transport cost is zero
  8) The network is homogeneous

The 8 fallacies of distributed computing

- Common mistakes from Lyon, Deutsch, & Gosling
  1) The network is reliable
  2) Latency is zero
  3) Bandwidth is infinite
  4) The network is secure
  5) Topology doesn't change
  6) There is one administrator
  7) Transport cost is zero
  8) The network is homogeneous
- Originally, experiments targeted these,
  but others are inspired by fault injection, race conditions, …

Coping with failure

- How failure is handled varies depending on a system
  - Logging & continue?
  - Rerouting?
  - Approximation and quality of service degradation?
  - Error reporting?
  - Terminal failure?

Coping with failure

- How failure is handled varies depending on a system
  - Logging & continue?
  - Rerouting?
  - Approximation and quality of service degradation?
  - Error reporting?
  - Terminal failure?

} fallback strategies are common

Coping with failure

- How failure is handled varies depending on a system
  - Logging & continue?
  - Rerouting?
  - Approximation and quality of service degradation?
  - Error reporting?
  - Terminal failure?

fallback strategies are common

- What impact might fallback strategies have on business performance?

The structure of chaos experimentation

- Four common steps for a chaos experiment

The structure of chaos experimentation

- Four common steps for a chaos experiment
    1) Measure & define the baseline behavior of the system

The structure of chaos experimentation

- Four common steps for a chaos experiment
    1) Measure & define the baseline behavior of the system
    2) Hypothesize that the baseline should continue under stress

The structure of chaos experimentation

- Four common steps for a chaos experiment
  1) Measure & define the baseline behavior of the system
  2) Hypothesize that the baseline should continue under stress
  3) Simulate pathological behaviors on the deployed systems

The structure of chaos experimentation

- Four common steps for a chaos experiment
  1) Measure & define the baseline behavior of the system
  2) Hypothesize that the baseline should continue under stress
  3) Simulate pathological behaviors on the deployed systems
  4) Try to disprove your hypothesis (show that there is a difference)

The structure of chaos experimentation

- Four common steps for a chaos experiment
  - Measure & define the baseline behavior of the system
  - Hypothesize that the baseline should continue under stress
  - Simulate pathological behaviors on the deployed systems
  - Try to disprove your hypothesis (show that there is a difference)
- The harder it is to show a difference,
  the more confidence you have in the robustness of your system

The structure of chaos experimentation

- Four common steps for a chaos experiment
  - Measure & define the baseline behavior of the system
  - Hypothesize that the baseline should continue under stress
  - Simulate pathological behaviors on the deployed systems
  - Try to disprove your hypothesis (show that there is a difference)
- The harder it is to show a difference,
  the more confidence you have in the robustness of your system
- NOTE:
  Just as in sequential hypothesis testing, you might want an "early out"
  - Managing the risks is critical even to getting management buy in

Defining your baseline

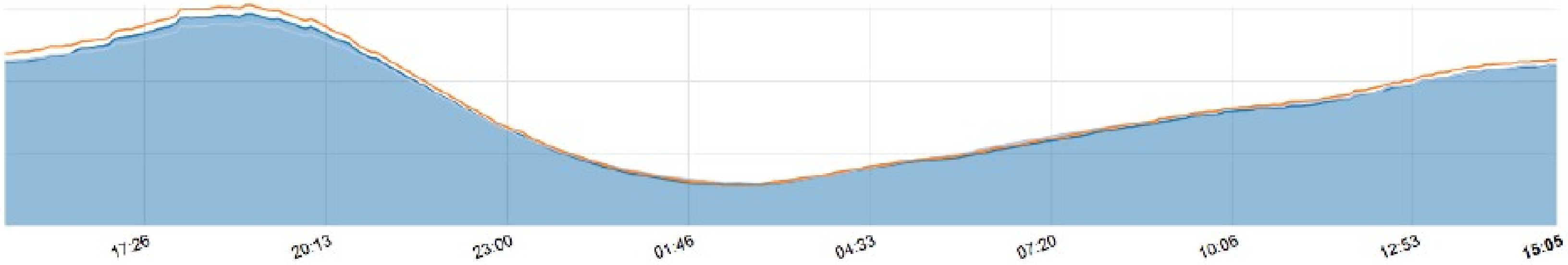
- Just like our discussion on performance,
  if you measure the wrong thing then your results won't make sense

Defining your baseline

- Just like our discussion on performance,
  if you measure the wrong thing then your results won't make sense
- Identify the key metrics that matter
  - Common attributes like throughput, latency, availability are good
  - The key business measures are even better
    (clicks/sec, successful purchases, video views, …)

Defining your baseline

- Just like our discussion on performance,
  if you measure the wrong thing then your results won't make sense
- Identify the key metrics that matter
  - Common attributes like throughput, latency, availability are good
  - The key business measures are even better
    (clicks/sec, successful purchases, video views, ...)
- Recognize that the baseline captures a distribution with trends



[Netflix SPS, 2016]

Defining your baseline

- Just like our discussion on performance,
  if you measure the wrong thing then your results won't make sense
- Identify the key metrics that matter
  - Common attributes like throughput, latency, availability are good
  - The key business measures are even better
    (clicks/sec, successful purchases, video views, ...)
- Recognize that the baseline captures a distribution with trends
- Coarser grained metrics focus on business value and avoid getting distracted by details
  - Netflix: CPU load vs SPS? SPS captures availability & business demands

Choosing your stressors

- Choose “very real world events” and simulate them

Choosing your stressors

- Choose “very real world events” and simulate them
- These drive away from the happy path and force fallbacks to be explored in practice
  - 92% of distributed system failures come from poor error handling
  - One form of failure leads to another, causing failure cascades

Choosing your stressors

- Choose “very real world events” and simulate them
- These drive away from the happy path and force fallbacks to be explored in practice
  - 92% of distributed system failures come from poor error handling
  - One form of failure leads to another, causing failure cascades
- Examples:
  - Inject random latency on requests
  - Terminate VM instances
  - Force request failures
  - Make entire Amazon regions unavailable
  - Corrupt headers & communication
  - Double send requests, permute orders, etc.

Managing risk

- The chaos community calls this “limiting the blast radius”

Managing risk

- The chaos community calls this “limiting the blast radius”
- Choose your population based on service tolerances

Managing risk

- The chaos community calls this “limiting the blast radius”
- Choose your population based on service tolerances
- Design early exit strategies and circuit breakers into the process

Managing risk

- The chaos community calls this “limiting the blast radius”
- Choose your population based on service tolerances
- Design early exit strategies and circuit breakers into the process
- Start in test environments & work toward production

Refining the objective of chaos

- Be careful that the goal is not to add instability to a system

Refining the objective of chaos

- Be careful that the goal is not to add instability to a system
- You are engineering the chaos already in the system,
  and you want a methodical process to expose it

Refining the objective of chaos

- Be careful that the goal is not to add instability to a system
- You are engineering the chaos already in the system, and you want a methodical process to expose it
- The process should be one of discovery, uncovering unknowns, and making a system more resilient

Refining the objective of chaos

- Be careful that the goal is not to add instability to a system
- You are engineering the chaos already in the system, and you want a methodical process to expose it
- The process should be one of discovery, uncovering unknowns, and making a system more resilient
- The goal is to uncover the latent chaos early in a controlled setting
  - By identifying unlikely problems early, you can prevent uncontrolled risk

Popular Tools

- Several tools are available
  - Chaos Monkey (Netflix)
  - Gremlin
  - Chaos Mesh (Kubernetes)
  - ToxiProxy (Shopify)
  - …

Popular Tools

- Several tools are available
  - Chaos Monkey (Netflix)
  - Gremlin
  - Chaos Mesh (Kubernetes)
  - ToxiProxy (Shopify)
  - …
- They focus on different strategies & potential injection abilities
  - e.g. Chaos Monkey just terminates VMs

Popular Tools

- Several tools are available
  - Chaos Monkey (Netflix)
  - Gremlin
  - Chaos Mesh (Kubernetes)
  - ToxiProxy (Shopify)
  - ...
- They focus on different strategies & potential injection abilities
  - e.g. Chaos Monkey just terminates VMs
- Several are open source

Popular Tools

- Several tools are available
  - Chaos Monkey (Netflix)
  - Gremlin
  - Chaos Mesh (Kubernetes)
  - ToxiProxy (Shopify)
  - ...
- They focus on different strategies & potential injection abilities
  - e.g. Chaos Monkey just terminates VMs
- Several are open source
- We can explore examples through:
  - (1) Problems, (2) Likely outcomes, and (3) Experiments to test them

Examples: unreliable networks [Gremlin]

- What happens when your channel to a service fails?

Examples: unreliable networks [Gremlin]

- What happens when your channel to a service fails?
- Likely outcomes:
  - Traffic may be rerouted to alternates
  - Fire alarms may trigger if critical
  - Application level metrics should be preserved, ***but*** …

Examples: unreliable networks [Gremlin]

- What happens when your channel to a service fails?
- Likely outcomes:
  - Traffic may be rerouted to alternates
  - Fire alarms may trigger if critical
  - Application level metrics should be preserved, ***but*** …
- An experiment can simply make a service unreachable

Examples: resource exhaustion [Gremlin]

- What happens when you saturate a resource like CPU, Memory, I/O?

Examples: resource exhaustion [Gremlin]

- What happens when you saturate a resource like CPU, Memory, I/O?
- Likely outcomes:
  - Increased error rates
  - Increased latency
  - QoS degradation if possible
  - Load balancer invocation
  - Fire alarm triggers

Examples: resource exhaustion [Gremlin]

- What happens when you saturate a resource like CPU, Memory, I/O?
- Likely outcomes:
  - Increased error rates
  - Increased latency
  - QoS degradation if possible
  - Load balancer invocation
  - Fire alarm triggers
- An experiment can simply consume CPU cycles

Examples: datastore saturation [Gremlin]

- What happens when a data service specifically becomes saturated?

Examples: datastore saturation [Gremlin]

- What happens when a data service specifically becomes saturated?
- Likely effects:
  - Increased application latency on data dependent paths
  - Metrics on other paths ideally unaffected
  - Fire alarms when critical

Examples: datastore saturation [Gremlin]

- What happens when a data service specifically becomes saturated?
- Likely effects:
  - Increased application latency on data dependent paths
  - Metrics on other paths ideally unaffected
  - Fire alarms when critical
- This can be implemented by
  - Making the datastore unavailable
  - Increasing latency to the datastore
  - Actually consuming bandwidth to the store

Summary

- Chaos engineering builds upon other techniques we have seen to explore distributed system reliability in practice

Summary

- Chaos engineering builds upon other techniques we have seen to explore distributed system reliability in practice
- It can discover problems at a small scale before they become larger

Summary

- Chaos engineering builds upon other techniques we have seen to explore distributed system reliability in practice
- It can discover problems at a small scale before they become larger
- By managing the existing chaos in your apps, you can produce more reliable apps in general