QA: Advanced Automated Testing and Dynamic Analysis

17-313 Fall 2023

Foundations of Software Engineering

https://cmu-313.github.io

Andrew Begel and Rohan Padhye

Learning Goals

- Describe random test-input generation strategies such as fuzz testing
- Characterize challenges of performance testing and suggest strategies
- Reason about failures in microservice applications how chaos engineering can be applied to test resiliency of cloud-based applications
- Describe A/B testing for usability

Automated Analysis for Functional and Non-Functional Properties

- Correctness – Static Analysis and Testing
- Robustness – Fuzzing
- Performance – Profiling
- Scalability – Stress testing
- Resilience – Soak testing
- Reliability – Chaos Engineering
- Usability – A/B testing

Puzzle: Find x such p1(x) returns True

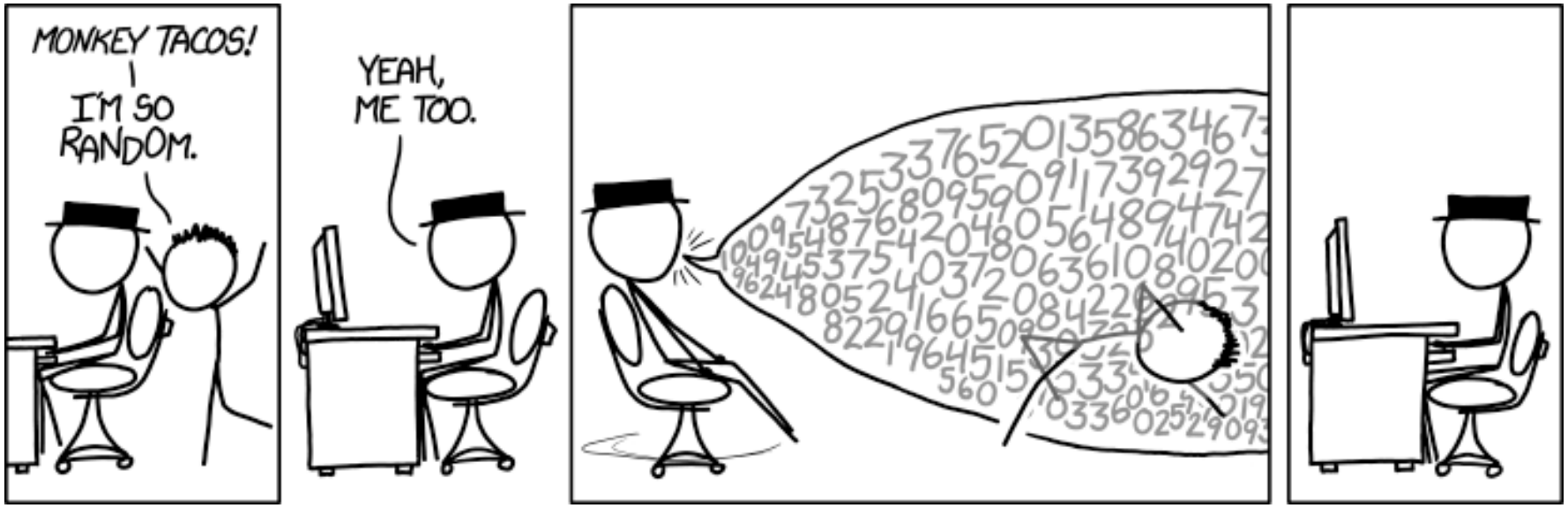
```
def p1(x):
  if x * x - 10 == 15:
    return True
  return False
```

Puzzle: Find x such p2(x) returns True

```
def p2(x):
  if x > 0 and x < 1000:
    if ((x - 32) * 5/9 == 100):
      return True
  return False
```

Puzzle: Find x such p3(x) returns True

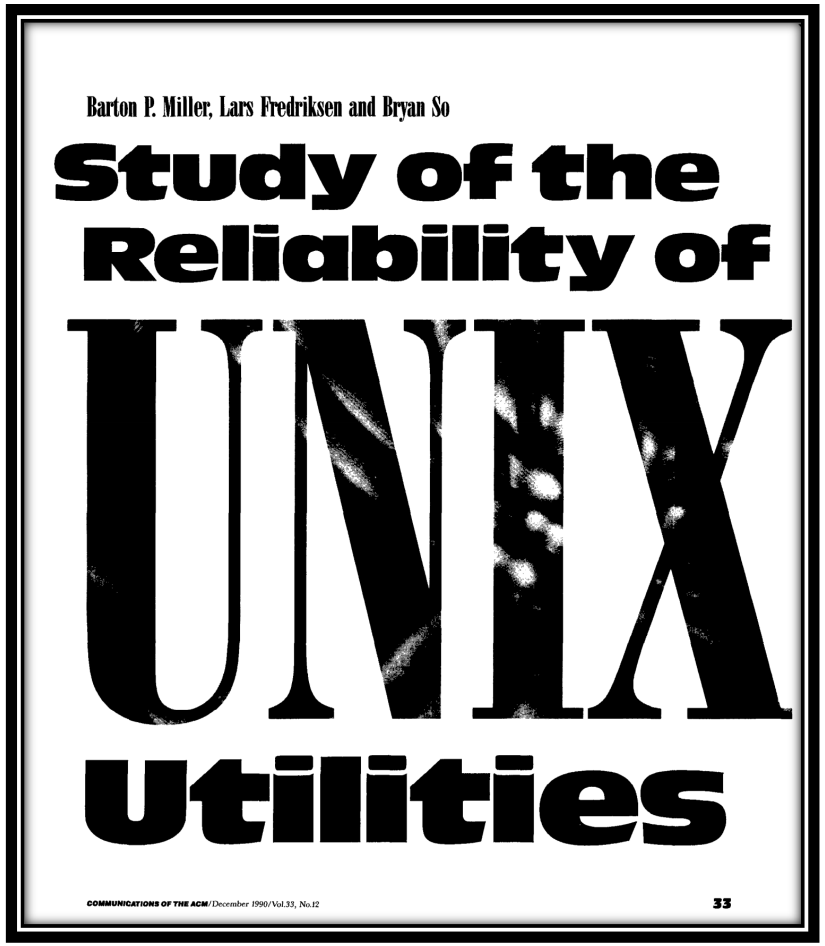
```
def p3(x):
  if x > 3 and x < 100:
    z = x - 2
    c = 0
    while z >= 2:
      if ((z ** (x - 1)) % x) == 1:
        c = c + 1
      z = z - 1
    if c == x - 3:
      return True
  return False
```



Original: https://xkcd.com/1210 CC-BY-NC 2.5

Fuzz Testing

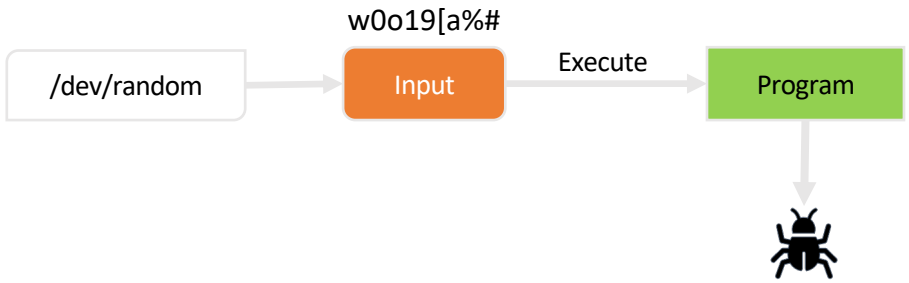
Security and Robustness



Communications of the ACM (1990)

> On a dark and stormy night one of the authors was logged on to his work-station on a dial-up line from home and the rain had affected the phone lines; there were frequent spurious characters on the line. The author had to race to see if he could type a sensible sequence of characters before the noise scrambled the command. This line noise was not surprising; but we were surprised that these spurious characters were causing programs to crash.

Fuzz Testing



A 1990 study found crashes in:
adb, as, bc, cb, col, diction, emacs, eqn, ftp, indent, lex, look, m4, make, nroff, plot, prolog, ptx, refer!, spell, style, tsort, uniq, vgrind, vi

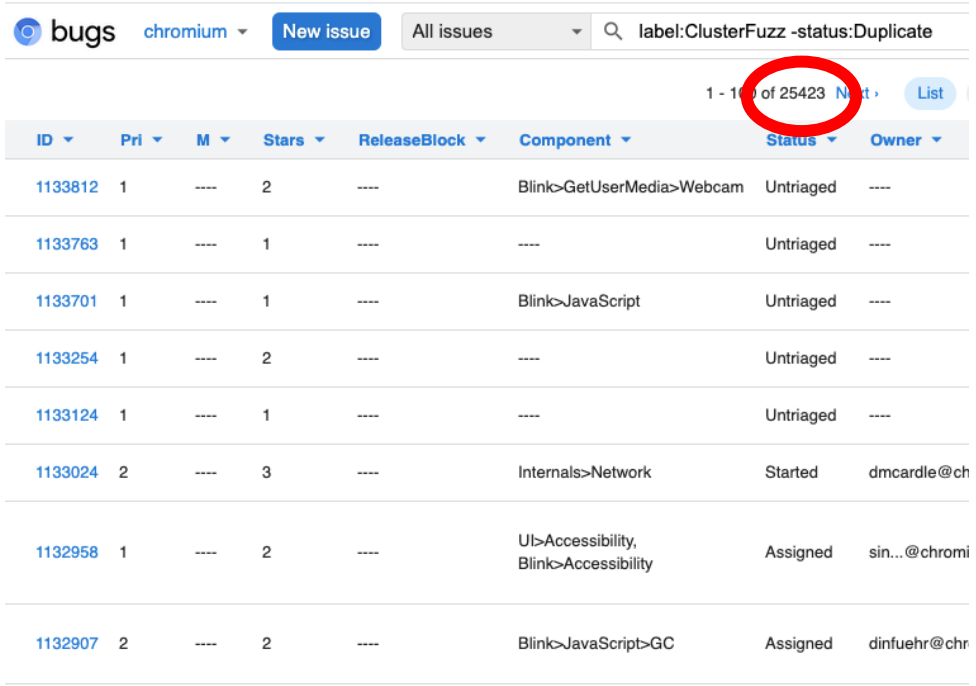
Common Fuzzer-Found Bugs in C/C++

Causes: incorrect arg validation, incorrect type casting, executing untrusted code, etc.

Effects: buffer-overflows, memory leak, division-by-zero, use-after-free, assertion violation, etc. (“crash”)

Impact: security, reliability, performance, correctness

ClusterFuzz @ Chromium



ID	Pri	M	Stars	ReleaseBlock	Component	Status	Owner
1133812	1	----	2	----	Blink>GetUserMedia>Webcam	Untriaged	----
1133763	1	----	1	----	----	Untriaged	----
1133701	1	----	1	----	Blink>JavaScript	Untriaged	----
1133254	1	----	2	----	----	Untriaged	----
1133124	1	----	1	----	----	Untriaged	----
1133024	2	----	3	----	Internals>Network	Started	dmcardle@ch
1132958	1	----	2	----	UI>Accessibility, Blink>Accessibility	Assigned	sin...@chromi
1132907	2	----	2	----	Blink>JavaScript>GC	Assigned	dinfuehr@chr

Strengths and Limitations

- **Exercise**: Write down one strength and one weakness of fuzzing as a means of finding bugs.

  Bonus: Write down one assumption about the program that fuzzing depends on.

Strengths and Limitations

- Strengths:
  - Cheap to generate inputs
  - Easy to debug when a failure is identified
  - Finds bugs that are hard to imagine with manual testing

- Limitations:
  - Randomly generated inputs don't make sense most of the time.
    - E.g. Imagine testing a browser and providing some "input" HTML randomly: **dgsad5135o gsd;gj lsdkg3125j@!T%#(W+123sd asf j**
  - Unlikely to exercise interesting behavior in the web browser
  - Can take a long time to find bugs. Not sure when to stop.

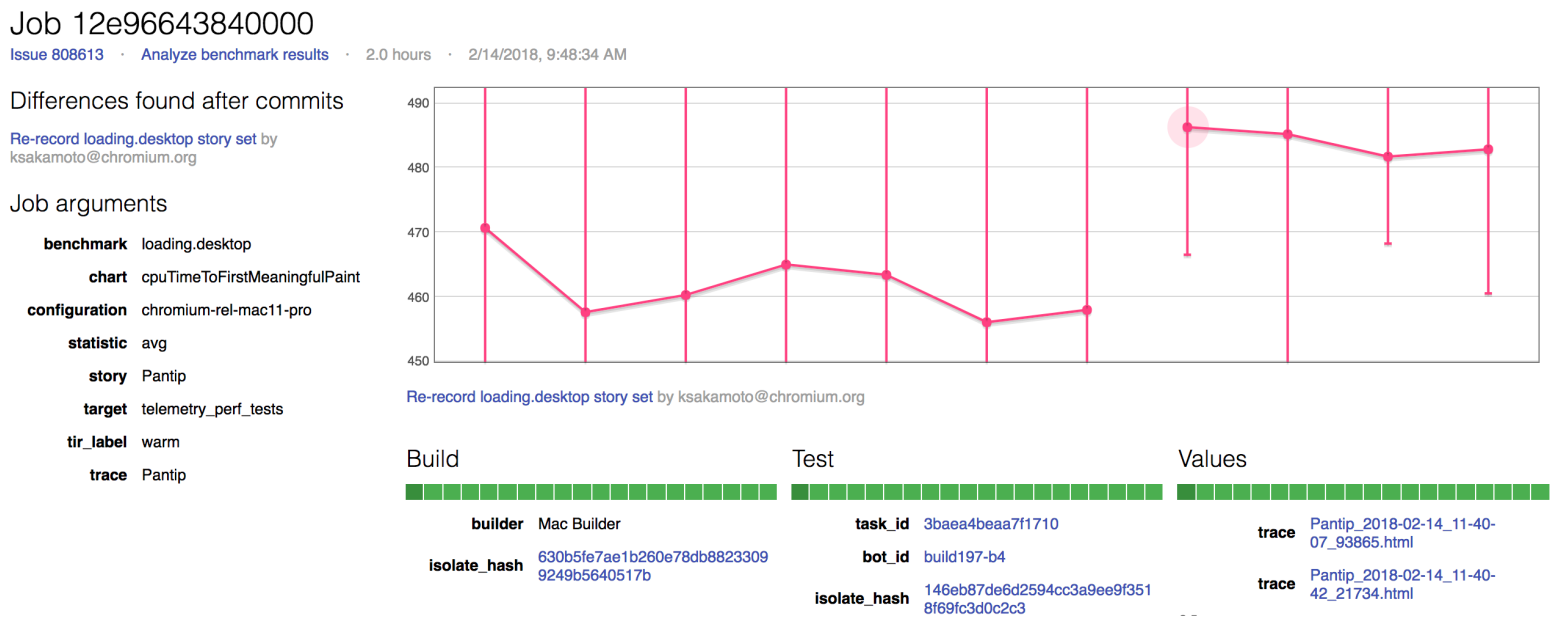
Performance Testing and Debugging

Performance Testing

- Goal: Identify *performance bugs*. What are these?
  - Unexpected bad performance on some subset of inputs
  - Performance degradation over time
  - Difference in performance across versions or platforms
- Not as easy as functional testing. What's the oracle?
  - Fast = good, slow = bad // but what's the threshold?
  - How to get reliable measurements?
  - How to debug where the issue lies?

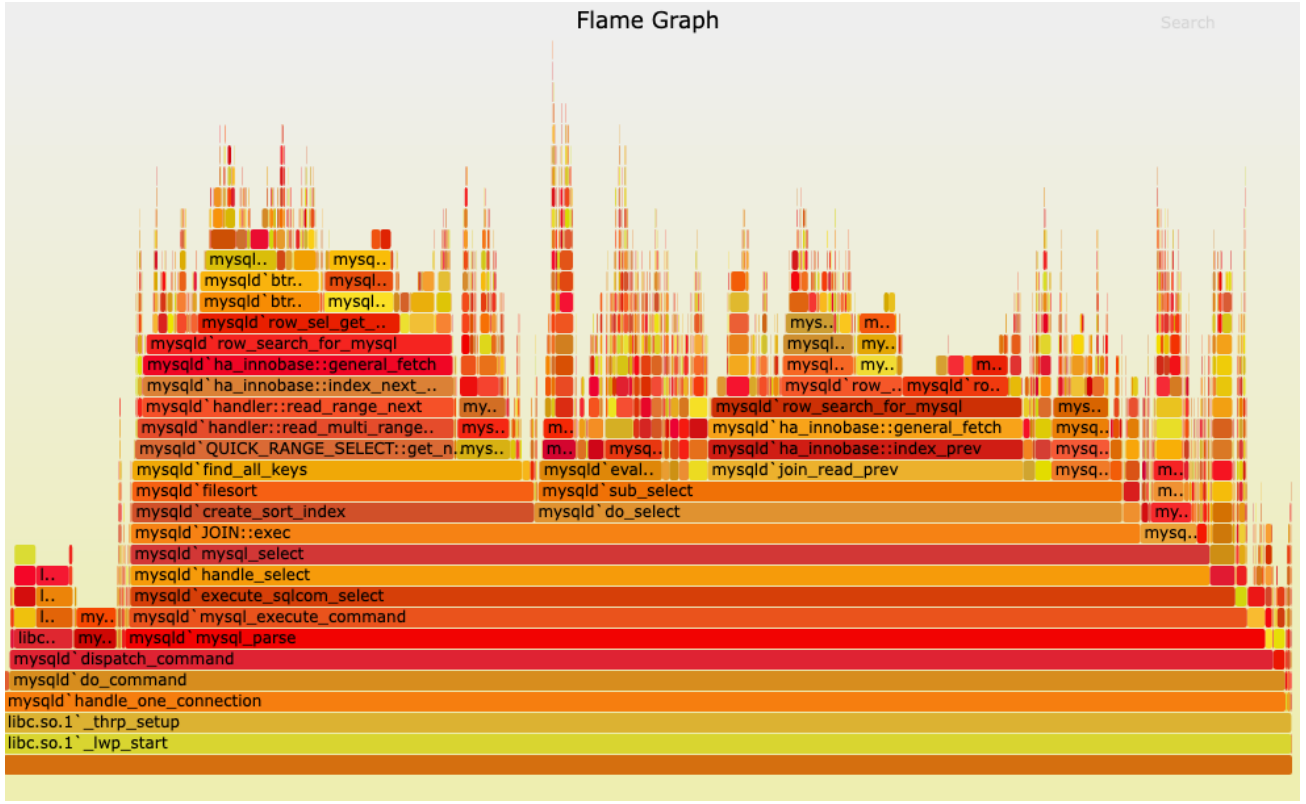
Performance regression testing helps identify trends

- Measure execution time of critical components
- Log execution times and compare over time

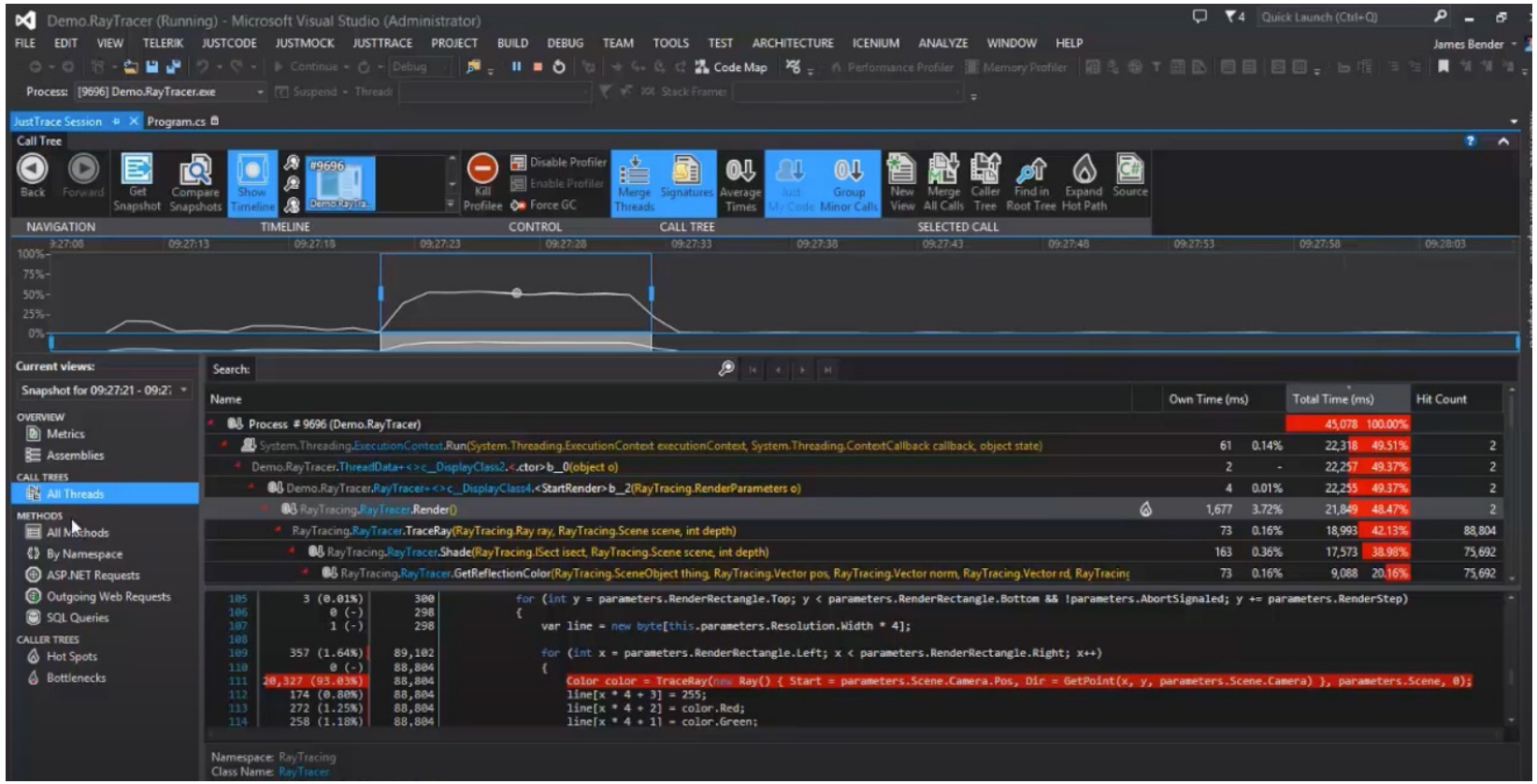


Profiling helps debug performance issues

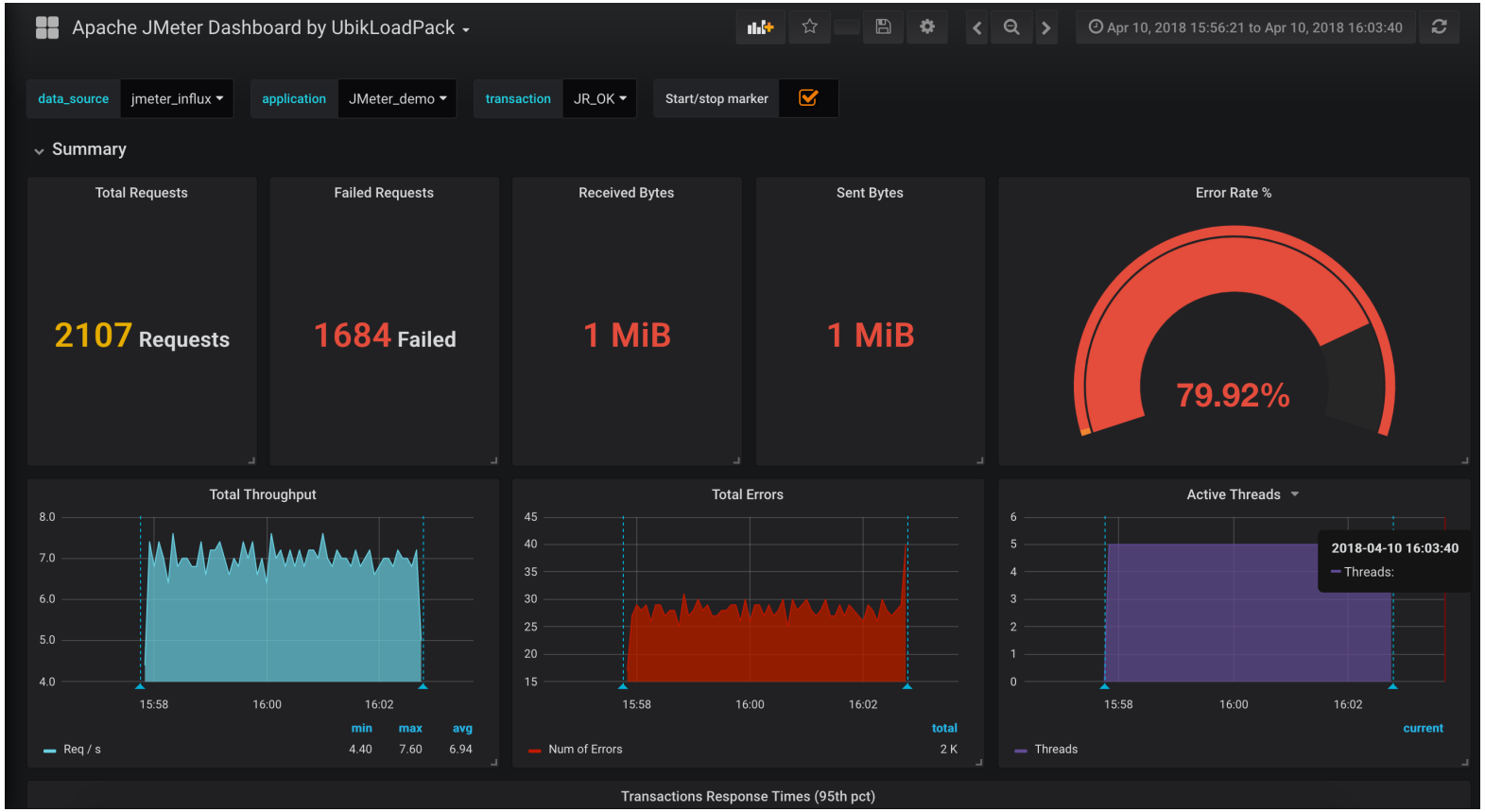
- Profiling is a form of dynamic analysis – measures and logs run-times for every function invocation
- Helps identify bottlenecks in execution time and memory
- Flame graphs are a popular visualization of resource consumption by call stack.



Profilers often included in IDEs



Domain-Specific Perf Testing
(e.g. JMeter for Java web apps)



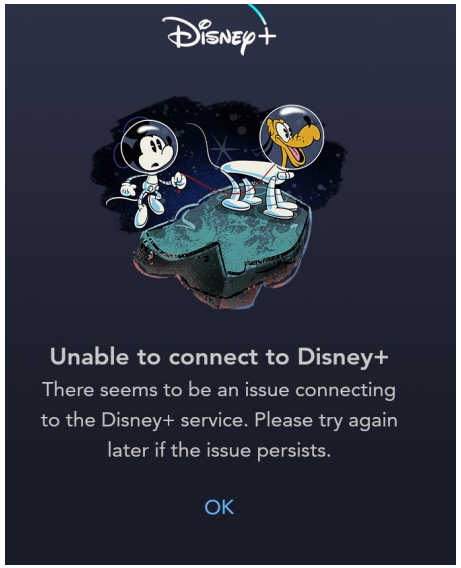
http://jmeter.apache.org

Stress testing

- Scalability/Robustness testing technique: test beyond the limits of normal operation.
- Can apply at any level of system granularity.
- Key idea: throw large amounts of input / requests and see how the program behaves
- Often a way to test the error-handling capabilities of the application

Real Issues: Disney+ Launch

- Lots of issues reported on launch day.
- Disney had planned for a spike in traffic.
  - Tested massive concurrent video streaming capability.
- BUT: the stress was in paths other than streaming
  - User account creation
  - Logins and auth
  - Browsing old titles



Soak testing

- A system may behave exactly as expected under artificially limited execution conditions, but fail in production after extended use.
  - E.g., Memory leaks may take longer to lead to failure
- **Soak testing** a system involves applying a significant load over a significant period of time and observing system resilience.
- Time-consuming to run but useful to apply at big release milestones or when making infrastructure changes.

Reliability testing

- What happens when some components of a large complex system fail? Can the system recover and keep working?
- How can you test the reliability of something as complex as Netflix or Google maps or Instagram?
- One idea: simulate a large-scale deployment and induce random failures in various components
- Another idea…

Chaos Engineering: Testing in Production

- Purposefully take down components in a **live deployment**.
- Observe system response. Do failovers work correctly?
- Tests the failure-handling and fallback capabilities of large systems.
- Useful in preparing for natural disasters or cyberattacks.

Example: Google

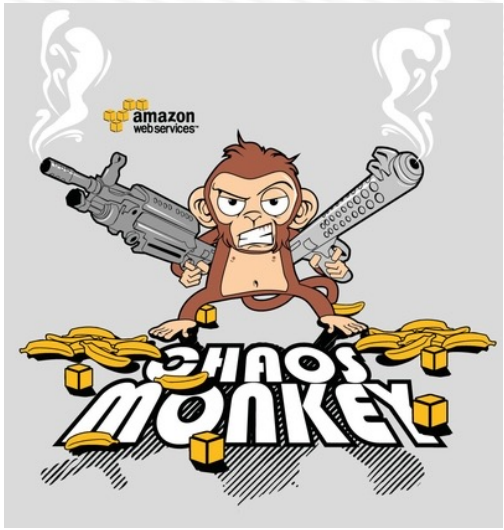
Terminate network in Sao Paulo for testing:

- Hidden dependency takes down links in Mexico which would have remained undiscovered without testing

Turn off data center to find that machines won't come back:

- Ran out of DHCP leases (for IP address allocation) when a large number of machines come back online unexpectedly.

Example: Netflix



Significant deployment on AWS cloud.
Hundreds of updates to microservices and infrastructure through the day.

Chaos Monkey randomly takes down AWS instances or network connections or randomly changes config files.

How to tell ”are we still good?”
Key metric: Stream Starts per Second (SPS)
Measures *availability*

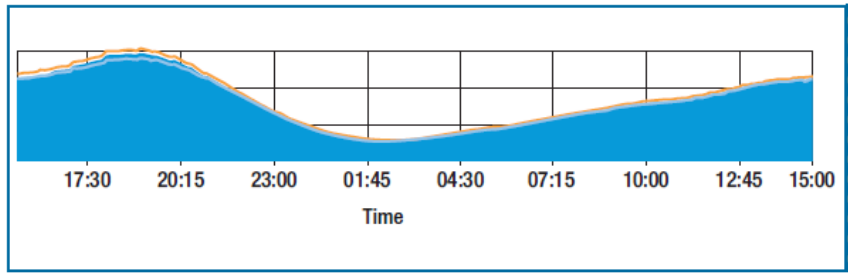


FIGURE 2. A graph of SPS ([stream] starts per second) over a 24-hour period. This metric varies slowly and predictably throughout a day. The orange line shows the trend for the prior week. The *y*-axis isn't labeled because the data is proprietary.

Testing GUIs and Usability

Automating GUI/Web Testing

- This is hard
- Capture and Replay Strategy
  - mouse actions
  - system events
- Test Scripts: (click on button labeled "Start" expect value X in field Y)
- Lots of tools and frameworks
  - e.g. Selenium for browsers
- Can avoid load on GUI testing by separating model from GUI
- Beyond functional correctness?

Usability: A/B testing

- Controlled randomized experiment with two variants, A and B, which are the control and treatment.
- One group of users given A (current system); another random group presented with B; outcomes compared.
- Often used in web or GUI-based applications, especially to test advertising or GUI element placement or design decisions.

Example

- A company sends an advertising email to its customer database, varying the photograph used in the ad…

Example: group A (99% of users)

Act now!
Sale ends soon!

Example: group B (1%)

Act now!
Sale ends soon!

A/B Testing

- Requires good metrics and statistical tools to identify significant differences.
- E.g. clicks, purchases, video plays
- Must control for confounding factors

Summary

- Automatic testing for non-functional properties requires coming up with creative “test oracles”.
- Dynamic analysis is often the only viable approach for assessing many of these qualities (e.g., usability or scalability). Statically figuring this out is almost impossible.
- Corollary: Tools are great, but you need to have good test inputs / scenarios to make the most of them.