CS4740
CLOUD COMPUTING

Reliability

Prof. Chang Lou, UVA CS, Spring 2024
AGENDA

– What is reliability
– Motivation for reliability research
– Software techniques to improve cloud reliability
  – Testing
  – Program analysis
  – Formal methods
  – ...

– End of semester concluding remarks :)}
WHAT IS RELIABILITY

— What are some common qualities we measure on systems?
WHAT IS RELIABILITY

— Reliability is not
  — Performance: make systems faster
  — Usability: make systems more user-friendly
  — Security: make systems safer against intrusions
  — Cost-effectiveness: make systems more affordable

— Reliability is
  — the system's ability to consistently perform its intended function without failure over a given period.
WHAT IS RELIABILITY

— Reliability
  — measured with the probability that a system operates without failure in a given period of time.
  — how to compute probability: Mean Time Between Failures (MTBF)

\[
\text{Reliability} = 1 - \frac{1}{\text{MTBF}} = 1 - \frac{\text{NumofBreakdowns}}{E[\text{uptime}]}
\]
CLOUD FAILURES

– Cloud failures are prevalent
CLOUD FAILURES

— Bad user experience

#Facebook is not a Law Enforcement issue, please don't call us about it being down, we don't know when FB will be back up!

Reddit when youtube's been down for 5 min

I'm sitting here in the dark in my toddler's room because the light is controlled by @Google Home. Rethinking... a lot right now.

Everybody right now.
#AWS #awscloud #awsoutage #awsdown #S3 #AWSs3 #Amazon
CLOUD FAILURES

– Huge economic loss and service unavailability

Microsoft’s MFA is so strong, it locked out users for 8 hours

3 difficult days for Rackspace Cloud Load Balancers

posted by iwgcr

After almost 24 hours of technical difficulties, Facebook is back

Facebook blamed the issue on a “server configuration change.”

Amazon ‘missed out on $34m in sales during internet outage’

The e-commerce giant generates $9,615 in sales per second – but not when its website is down

Ben Chapman • Tuesday 08 June 2021 16:54 • 0 Comments

Millions online hit by Microsoft 365 outages

911 emergency services go down across the US after CenturyLink outage

Zack Whittaker • @zackwhittaker • 4 months ago
CLOUD FAILURES

— Cloud systems fail due to different root causes

CLOUD FAILURES

– .. sometimes very weird root causes

Google Fiber Shot Down By 'Bored' Hunters

Google reinforces undersea cables after shark bites

Sharks have been biting on fibre optic cables under the Pacific, possibly confused by electrical signals that resemble fish
REMAINING PART OF LECTURE

— We focus on solutions for software bugs
# Tackling Software Issues in Different Ways

<table>
<thead>
<tr>
<th>Bug finding</th>
<th>Formal methods</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzz testing</td>
<td>Model checking</td>
<td>Failure detection</td>
</tr>
<tr>
<td>Static analysis</td>
<td>Symbolic execution</td>
<td>Failure diagnosis</td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td>Theorem proving</td>
<td>Failure recovery</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Can we automatically find bugs in the codes?  
Can we prove the codes are bug-free?  
Can we better handle failures at runtime?
Testing (fuzzy)
func TestPersist12C(t *testing.T) {
    servers := 3
    cfg := make_config(t, servers, unreliable: false, snapshot: false)
    defer cfg.cleanup()

    cfg.begin(description: "Test (2C): basic persistence")

    cfg.one(cmd: 11, servers, retry: true)

    // crash and re-start all
    for i := 0; i < servers; i++ {
        cfg.start1(i, cfg.applier)
    }

    for i := 0; i < servers; i++ {
        cfg.disconnect(i)
        cfg.connect(i)
    }

    cfg.one(cmd: 12, servers, retry: true)

    leader1 := cfg.checkOneLeader()
    cfg.disconnect(leader1)
    cfg.start1(leader1, cfg.applier)
    cfg.connect(leader1)

    cfg.one(cmd: 13, servers, retry: true)
func TestPersist12C(t *testing.T) {
    servers := 3
    cfg := make_config(t, servers, unreliable: false, snapshot: false)
    defer cfg.cleanup()

    cfg.begin(description: "Test (2C): basic persistence")

    cfg.one(cmd: 11, servers, retry: true)

    // crash and re-start all
    for i := 0; i < servers; i++ {
        cfg.start1(i, cfg.applier)
    }
    for i := 0; i < servers; i++ {
        cfg.disconnect(i)
        cfg.connect(i)
    }

    cfg.one(cmd: 12, servers, retry: true)
**TESTING**

```go
func TestPersist12(t *testing.T) {
    servers := 3
    cfg := make_config(t, servers, unreliable: false, snapshot: false)
    defer cfg.cleanup()

    cfg.begin(description: "Test (2C): basic persistence")

    cfg.one(cmd: 11, servers, retry: true)

    // crash and re-start all
    for i := 0; i < servers; i++ {
        cfg.start1(i, cfg.applier)
    }
    for i := 0; i < servers; i++ {
        cfg.disconnect(i)
        cfg.connect(i)
    }

    cfg.one(cmd: 12, servers, retry: true)
}
```

Test passed, does that mean your program has no bug?

Tests only cover a small portion of possibilities!
FUZZ TESTING

— Goal:
  — To find program inputs that reveal a bug

— Approach:
  — Generate inputs randomly until program reports errors
FUZZ TESTING EXAMPLE

- Standard HTTP GET request
  - § GET /index.html HTTP/1.1

- Fuzzing HTTP GET request
  - § AAAAAA...AAAA /index.html HTTP/1.1
  - § GET //////index.html HTTP/1.1
  - § GET %n%n%n%n%n%n.html HTTP/1.1
  - § GET /AAAAAAAAAAAAAA.html HTTP/1.1
  - § GET /index.html HTTTTTTTTTTTTTTTP/1.1
FUZZ TESTING EXAMPLE 2: OPEN-SOURCE SOFTWARE

---

Many open-sourced fuzzer implementation

- e.g., Atheris: A Coverage-Guided, Native Python Fuzzer from Google

---

**Maya: Datetimes for Humans™**

Datetimes are very frustrating to work with in Python, especially when dealing with different locales on different systems. This library exists to make the simple things much easier, while admitting that time is an illusion (timezones doubly so).

Datetimes should be interacted with via an API written for humans.

Maya is mostly built around the headaches and use-cases around parsing datetime data from websites.
FUZZ TESTING EXAMPLE 2: OPEN-SOURCE SOFTWARE

– Many open-sourced fuzzer implementation
  – e.g., Atheris: A Coverage-Guided, Native Python Fuzzer from Google

```python
>>> scraped = '2016-12-16 18:23:45.423992+00:00'
>>> maya.parse(scraped).datetime()
datetime.datetime(2016, 12, 16, 13, 23, 45, 423992)

>>> maya.parse('may15,2021').datetime()
datetime.datetime(2022, 5, 15, 0, 0, tzinfo=None)
```

Maya: Python Datetimes Library

Applying fuzzer to find a triggering input
FUZZ TESTING EXAMPLE

— How to fuzz testing a distributed system?

— Very challenging, especially considering all concurrency and non-determinism
  — here we show an intuitive approach
FUZZ TESTING

```go
func TestPersist12C(t *testing.T) {
    servers := 3
    cfg := make_config(t, servers, unreliable)
    defer cfg.cleanup()

    cfg.begin(description: "Test (2C): basic persist")
    cfg.one(cmd: 11, servers, retry: true)

    // crash and re-start all
    for i := 0; i < servers; i++ {
        cfg.start1(i, cfg.applier)
    }
    for i := 0; i < servers; i++ {
        cfg.disconnect(i)
        cfg.connect(i)
    }
    cfg.once(cmd: 12, servers, retry: true)
}
```
FUZZ TESTING

```go
func TestPersist12C(t *testing.T) {
    servers := 3
    cfg := make_config(t, servers, unreliable)
    defer cfg.cleanup()

    cfg.begin(description: "Test (2C): basic params")
    cfg.one(cmd: 11, servers, retry: true)

    // crash and re-start all
    for i := 0; i < servers; i++ {
        cfg.start1(i, cfg.applier)
    }
    for i := 0; i < servers; i++ {
        cfg.disconnect(i)
        cfg.connect(i)
    }
    cfg.one(cmd: 12, servers, retry: true)
}
```

- Execute program
- Check result
- Collect & analyze
- Mutate
- Use as new seed!

Initial seed:

```
one(11)
disconnect(1)
connect(1)
disconnect(2).. ...
```

Search space:

```
one(11)
disconnect(1)
disconnect(2)...
```

Set input:

```
one(11)
disconnect(1)
disconnect(2)...
```
FUZZ TESTING

— Strength
  — low cost, easy-to-implement
  — practical for large programs

— Weakness
  — randomness
  — complexity of structured input
  — wasted efforts on rejected input
Static analysis
func (rf *Raft) RequestVote(args *RequestVoteArgs, reply *RequestVoteReply) {
    rf.mu.Lock()
    log.Printf("Worker%d: receive %v \n", rf.me, args)
    rf.CheckBehind(args.Term)
    reply.Term = rf.currentTerm
    if (rf.votedFor == -1 || rf.votedFor == args.CandidateId) && (args.LastLogTerm > rf.log[len(rf.log)-1].Term ||
        (args.LastLogTerm == rf.log[len(rf.log)-1].Term && args.LastLogIndex >= len(rf.log)-1)) {
        log.Printf("Worker%d: grant true %v %v %v \n", rf.me, rf.votedFor, rf.currentTerm, rf.commitIndex)
        rf.votedFor = args.CandidateId
        rf.currentTerm = args.Term
        rf.ifeaderAlive = true
        rf.recentVoted = true
        log.Printf("Worker%d: become follower\n", rf.me)
        rf.role = Follower
        rf.persist()
    } else { // reply.VoteGranted = true
        return
    } // reply.VoteGranted = false
    log.Printf("Worker%d: grant false %v %v %v \n", rf.me, rf.votedFor, rf.currentTerm, rf.commitIndex)
    rf.mu.Unlock()}

anything wrong with this code?
func (rf *Raft) RequestVote(args *RequestVoteArgs, reply *RequestVoteReply) {
    rf.mu.Lock()
    log.Printf("Worker%d: receive %v \n", rf.me, args)
    rf.CheckBehind(args)
    reply.Term = rf.currentTerm
    if (rf.votedFor == -1 || rf.votedFor == args.CandidateId) && (args.LastLogTerm > rf.log[len(rf.log)-1].Term ||
    (args.LastLogTerm == rf.log[len(rf.log)-1].Term && args.LastLogIndex >= len(rf.log)-1)) {
        log.Printf("Worker%d: grant true %v %v %v \n", rf.me, rf.votedFor, rf.currentTerm, rf.commitIndex)
        rf.votedFor = args.CandidateId
        rf.currentTerm = args.Term
        rf.ifLeaderAlive = true
        rf.recentVoted = true
        log.Printf("Worker%d: become follower\n", rf.me)
        rf.role = Follower
        rf.persist()
        reply.VoteGranted = true
        return
    }
    reply.VoteGranted = false
    log.Printf("Worker%d: grant false %v %v %v \n", rf.me, rf.votedFor, rf.currentTerm, rf.commitIndex)
}

rf.mu.Unlock()
ANOTHER EXAMPLE

x = 10;
y = x;
z = 0;
while (y > -1) {
    x = x / y;
    y = y - 1;
    z = 5;
}
can x be zero?
ANOTHER EXAMPLE

```plaintext
x = 10;
y = x;
z = 0;
while (y > -1) {
    x = x / y;
y = y - 1;
z = 5;
}
```
ANOTHER EXAMPLE

x = 10
y = x
z = 0
y > -1
x = x / y
y = y - 1
z = 5
(exit)

x:NZ
x:NZ, y:NZ
x:NZ, y:NZ, z:Z
x:NZ, y:MZ, z:Z
x:NZ, y:MZ, z:MZ
x:NZ, y:MZ, z:NZ
x:NZ, y:MZ, z:Z
x:NZ, y:MZ, z:MZ
x:NZ, y:MZ, z:NZ
# SOUNDNESS, COMPLETENESS

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soundness</td>
<td>“Sound for reporting correctness”</td>
</tr>
<tr>
<td></td>
<td>Analysis says no bugs → No bugs</td>
</tr>
<tr>
<td></td>
<td>or equivalently</td>
</tr>
<tr>
<td></td>
<td>There is a bug → Analysis finds a bug</td>
</tr>
<tr>
<td>Completeness</td>
<td>“Complete for reporting correctness”</td>
</tr>
<tr>
<td></td>
<td>No bugs → Analysis says no bugs</td>
</tr>
</tbody>
</table>

Recall: $A \rightarrow B$ is equivalent to $(\neg B) \rightarrow (\neg A)$
in practice, often settle for unsound and incomplete analysis
STATIC ANALYSIS

— Strength
   — scalability
   — fault localization

— Weakness
   — require specific bug pattern (false negative)
   — lack runtime information (false positive)
Model checking
TESTING IS USEFUL, HOWEVER..

—“Testing can only show the presence of errors, not their absence.”

Edsger Dijkstra
1930–2002
Many techniques focus on checking implementation, not design.

What if the system design is incorrect?

Example: Microwave oven
  - Start: “start” button pressed
  - Close: is door closed?
  - Heat: microwave active
  - Error: error state

Safety property: the oven doesn’t heat up until the door is closed
  - $(\neg \text{Heat}) \cup \text{Close}$
MOTIVATION EXAMPLE
DEMO: CHECK CHANG’S MICROWAVE OVEN WITH TLA+
MODEL CHECKING PROBLEM

— Given state transition graph M
— Let $\phi$ be specification (a temporal logic formula)
— Find all states $s$ of $M$ such that for all execution sequences $x$ starting from $s$, $x, 0 \models \phi$
MODEL CHECKING STEPS

— 1. Write a specification of the system in a formal specification language (think math).

— 2. Specify correctness properties as invariants on states or behaviors.

— 3. Use a model checker to exhaustively check that every state/behavior of the system, within a bounded range of configurations, satisfies your invariants.
   — e.g., TLA+ (by Leslie Lamport)
\* UNCHANGED <<ackeed, leaderVars, logVars, restartCtrl>>

\* ACTION: AppendEntries
\* Leader i sends j an AppendEntries request containing up to 1 entry.
\* While implementations may want to send more than 1 at a time, this spec uses
\* just 1 because it minimizes atomic regions without loss of generality.

\* AppendEntries(i, j) ==
\* i /= j
\* state[i] = Leader
\* pendingResponse[i][j] = FALSE \* not already waiting for a response
\* LET prevLogIndex == nextIndex[i][j] - 1
\* prevLogTerm == IF prevLogIndex > 0 THEN
\* log[i][prevLogIndex].term
\* ELSE
\* 0
\* \* Send up to 1 entry, constrained by the end of the log.
\* lastEntry == Min{Len(log[i]), nextIndex[i][j]}
\* entries == SubSeq(log[i], nextIndex[i][j], lastEntry)
\* IN
\* pendingResponse' = [pendingResponse EXCEPT ![i][j] = TRUE]
\* Send(nextTerm |-> AppendEntriesRequest,
\*       mterm |-> currentTerm[i],
\*       mprevLogIndex |-> prevLogIndex,
\*       mprevLogTerm |-> prevLogTerm,
\*       mentries |-> entries,
\*       mcommitIndex |-> Min{commitIndex[i], lastEntry},
\*       msource |-> i,
\*       mdest |-> j)]
\* UNCHANGED <<serverVars, candidateVars, nextIndex, matchIndex, logVars, auxVars>>
Concluding remarks
IT HAS BEEN A LONG JOURNEY..
WE BUILT TWO CLOUD SYSTEMS.

Raft Protocol Summary

**Recover/Vote RPC**
- Invoked by candidate to gather votes.
- **Arguments:**
  - candidate
  - candidate’s term
  - prevLogId
- **Returns:**
  - true if term == candidate’s term, false otherwise
- **Implementation:**
  1. If term == candidate’s term, return true
  2. Return false

**AppendEntries RPC**
- Invoked by leader to replicate log entries and discover inconsistencies, also used as heartbeat.
- **Arguments:**
  - leader
  - prevLogId
- **Returns:**
  - true if follower consent to update its log
  - false otherwise
- **Implementation:**
  1. Return true if term == leader's term and prevLogId == leader's last entry known to be committed
  2. AppendLogId’s entry log entries to own log (copy for heartbeat)
  3. Return false otherwise

**AppendEntries**
- Each entry indicates the following in stable storage
- **currentTerm**
- **leaderId**
- **prevLogId**
- **term**
- **commitIndex**

**MapReduce**
- How MapReduce Works?
PLAYED WITH COMMERCIAL CLOUD SYSTEMS..

Google File System

ZooKeeper (Lab Day I, Lab Day II)
GAMES..

Green cup, Red cup

Consensus

Consensus (w/ malicious peers)

2PC failures w/ Donut

Gandalf

YOU SHALL NOT PASS!
DISCUSSION WITH CLOUD EXPERTS..

"Managing Cloud Health with AIOps" (Microsoft Azure)

"Block Store over the Cloud" (Alibaba Cloud)
What if I'd like to learn more
FUTURE STUDY

– 1. Online resources
– cloud/distributed system course, e.g., MIT 6.824
– follow up latest progress on top system conferences, e.g., SOSP/OSDI

6.5840 Schedule: Spring 2024

E25-111, TR1-2:30

Here is the tentative schedule of lectures and due dates. The lecture notes and paper questions for future dates are copies from previous years, and may change. Lectures are in E25-111, Tues/Thurs 1:00 to 2:30.

<table>
<thead>
<tr>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feb 5</td>
<td>First day of classes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feb 8</td>
<td>LEC 1 (rm): Introduction, video Preparation: Handout/Notes/Slides Assigned: Lab 1: Load/Leader</td>
<td>Feb 7</td>
<td>LEC 2 (rm): Multi-core, video examples Assigned: Lab 2: Load/Leader</td>
<td>Feb 9</td>
</tr>
<tr>
<td>Feb 19</td>
<td>Presidents' day Assigned: Lab 3.1: null Monday schedule</td>
<td>Feb 21</td>
<td>LEC 5 (guest lecture): Relax GarFKP of Cooperate Go service Preparation: Read: The Go Programming Language and Environment FAQ (Question)</td>
<td>Feb 23</td>
</tr>
</tbody>
</table>

SOSP 2024
The 30th Symposium on Operating Systems Principles
November 4–6, 2024 · Hilton Austin, Texas, USA
FUTURE STUDY

2. Contribute to open-source cloud software
   - for example, download and play with Kubernetes today
   - even submitting a small PR is a big achievement and a good start!
FUTURE STUDY

3. Continue exploring cloud in our grad-level course!
   - Focus on Reliability
   - Paper reading + Project
   - No exam :)
   - Undergraduate students are welcomed

CS6501: Cloud System Reliability
Fall 2024, UVA CS
.. A FEW MORE WORDS

— This is a class in “progress.”

you  

me

— Thank you so much for supporting and improving this course!
Share your thoughts for future students on Student Experiences of Teaching!

https://go.blueja.io/34k62-FXlkKIGXYdtxOxJw

Extra credits for Completed SET!

CS 4740 - 001 Cloud Computing
TAKEAWAYS

— Next class: **Final Review**
— **Deadline** of Lab2C: 4/29, Monday
— Today's office hour -> Friday 4-5pm
ACKNOWLEDGEMENT

THIS COURSE IS DEVELOPED HEAVILY BASED ON COURSE MATERIALS SHARED BY PROF. INDRANIL GUPTA, PROF. ROBERT MORRIS, PROF. MICHAEL FREEDMAN, PROF. KYLE JAMIESON, PROF. WYATT LLOYD AND PROF. ROXANA GEAMBASU. MANY APPRECIATIONS FOR GENEROUSLY SHARING THEIR MATERIALS AND TEACHING INSIGHTS.

SOME CONTENTS ARE FROM OSDI'21 PREVIEW SESSION VIDEO MADE BY CHENGCHENG WAN AND LEFAN ZHANG