IOWA

CS5630

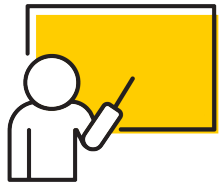
Discussion: Midterm and Research Papers

Prof. Supreeth Shastri
Computer Science
The University of Iowa

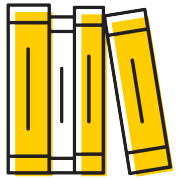
Midterm Format and Question Categories

Category	Example questions	Weight
Cloud **Concepts**	*What is tail-latency and how does it affect cloud applications?*	20%
Cloud **Services**	*Name three types of IaaS contracts. Compare their cost-availability features.*	20%
Cloud **Tradeoffs**	*Tradeoff in fault-tolerance techniques, cloud contract design, etc.,*	20%
Cloud **Programming**	*Given a program or set of CLIs, explain the behavior (or fix the error)*	20%
Cloud **Research**	*What is special about disks in datacenters?*	20%

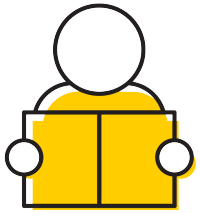
How to Prepare



*Revisit the **lectures and slides**:*
https://shastri.info/teaching/cs5630



*Read the reference **books**:*
Armbrust, Foster, Borroso



*Brush up on **research**:*
Cooper, Brewer, Irwin

It is a pen-and-paper exam; We will provide the answer sheets but bring a bunch of pens

Ground rules and policies

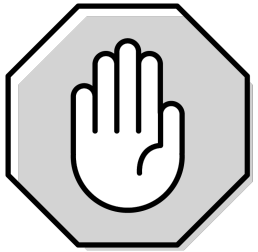
Time management

is completely your responsibility. Keep your answers brief and accurate. The exam is designed to require an hour.

Proctoring

TA and I will be around during the exam time to help with any questions or clarifications

Please bring your student ID card.



Accommodations

If you need any special accommodations for the exam, please discuss with me at least a week before.

Code of conduct

During the exam, you cannot use books, notes, Internet resources, or seek help from others.

After the exam, you should not share exam details with anyone.

Key Dates

Oct-4 6:30pm SH40

Midterm

There is an overlapping ***make up midterm*** scheduled for those with other academic conflicts

Remember, we don't have our regular lecture on Oct-4

Oct-11 lecture slot

Paper Presentation

Johnny and **Yuou** will present Google's paper on *Long-Term SLOs*

followed by

David and **Jamil**, who will present *Google's Borg*

IOWA

CS5630

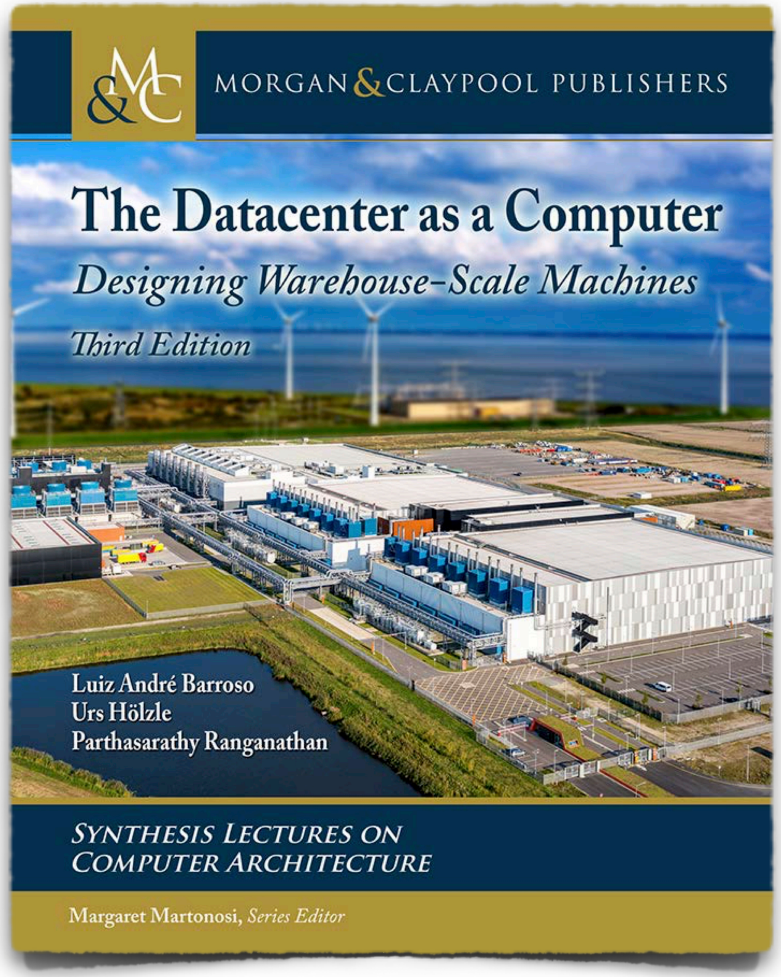
Datacenters (4): Fault-Tolerance

Prof. Supreeth Shastri
Computer Science
The University of Iowa

Lecture goals

Technical overview of fault-tolerance in datacenters and WSC systems

- *Faults-tolerance fundamentals*
- *Understanding faults in WSC*
- *Fault-tolerant designs*



Chapter 7

A fundamental choice in WSC design

Buy reliable hardware

Build fault-tolerant software

Approach-1:
buy reliable hardware

- A common approach followed for enterprise-grade systems
- Investment in highly-reliable hardware is justified when:
  - *the cost of failure is high (e.g., in enterprise or life-critical applications)*
  - *running legacy applications that are not designed to tolerate frequent hardware faults*

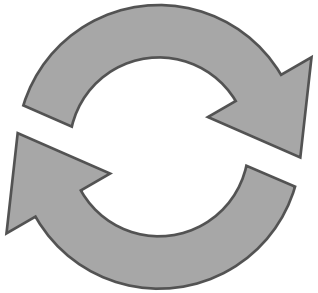
How well does this approach work for WSC?

- Suppose we use ultra-reliable server nodes with an MTBF (mean time between failures) of 30 years (or 10K days)
- However, realize that our WSC has 10,000 servers —> on average, we can expect a failure ***once per day***!
- As the WSC scale grows, it is normal to expect server failures *once every few hours*

WSCs do not benefit significantly from approach-1

why approach-1 is insufficient

Hardware failure
Software errors
Planned maintenance



Repairs, replacements,
Restarts, bug-fixes

Availability

the fraction of time during which a system/service is available for use

Unavailability

the fraction of time during which a system/service cannot be used

Approach-2:
building fault-tolerant software

- Fault-tolerant software is more complex to build and maintain than software that assumes fault-free operation
- It is best to implement fault-tolerance at the systems level than application level (why?)

Advantages of building fault-tolerance in software

- ***Datacenter designers*** can now optimize for cost efficiency (e.g., buy cheaper servers vs. enterprise grade servers)
- ***System administrators*** have more leeway in performing software upgrades and hardware replacements
- ***Software designers*** can make use of reliability techniques that span multiple datacenters

Approach-2:
building fault-tolerant software

Since software is fault-tolerant anyway, is it necessary for hardware to have any error correction or error detection capabilities?

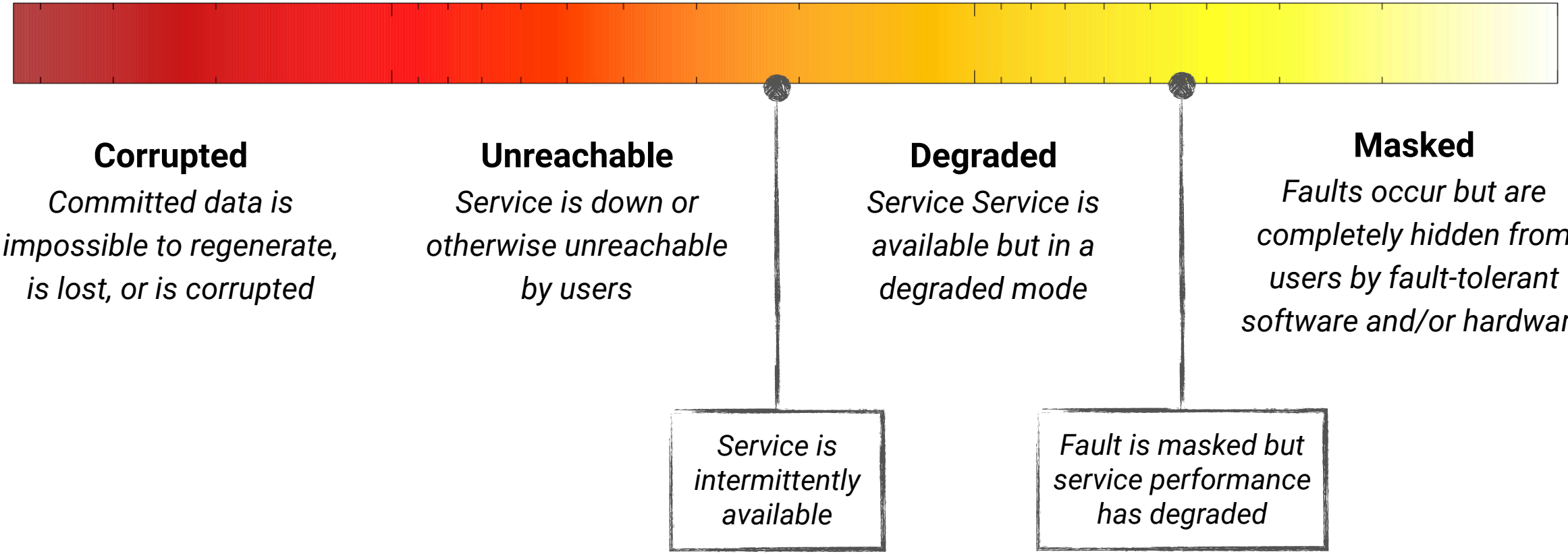
- It is ***not necessary*** that all hardware ***faults are corrected*** transparently; if error correction functionality can be offered *within a reasonable cost or complexity*, it often pays to support it
- However, the hardware ***must*** have the ability to ***detect and report failures***; if not, the software systems incur extraordinary performance burden and still may not be able to offer correct results

Google story

Google was using inexpensive DRAMs that lacked even basic error detection such as parity checking.
In 2000, Google's web index started returning seemingly random documents
Investigations revealed that a data structure had a particular set of bits stuck at zero!
Adding bit parity checking at software-level turned out to be an highly non-performant task
Finally, Google moved to buying memory chips that included basic error detection (DRAM ECC)

Understanding Faults

Fault Severity



Corrupted

Committed data is impossible to regenerate, is lost, or is corrupted

Unreachable

Service is down or otherwise unreachable by users

Service is intermittently available

Degraded

Service Service is available but in a degraded mode

Fault is masked but service performance has degraded

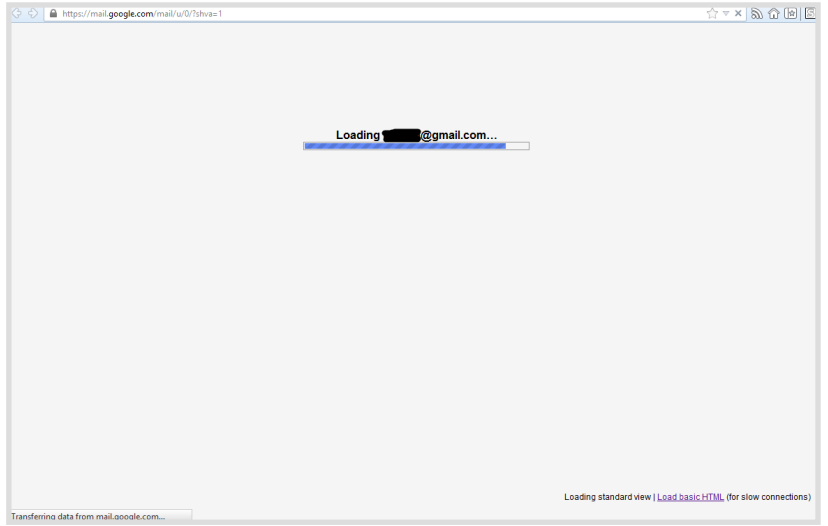
Masked

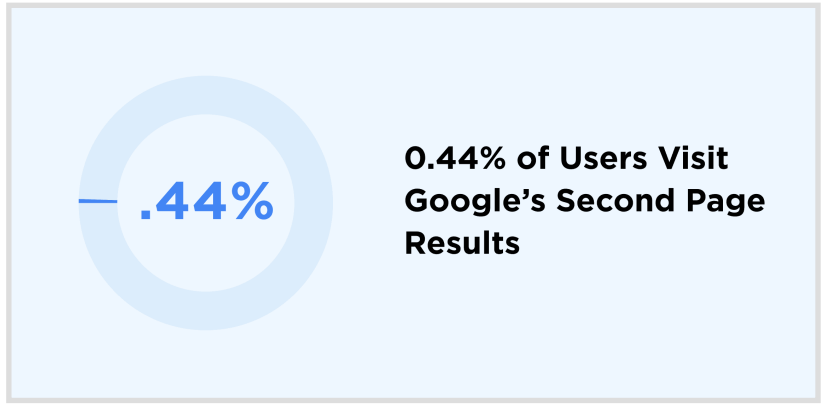
Faults occur but are completely hidden from users by fault-tolerant software and/or hardware

Fault Severity

Graceful degradation

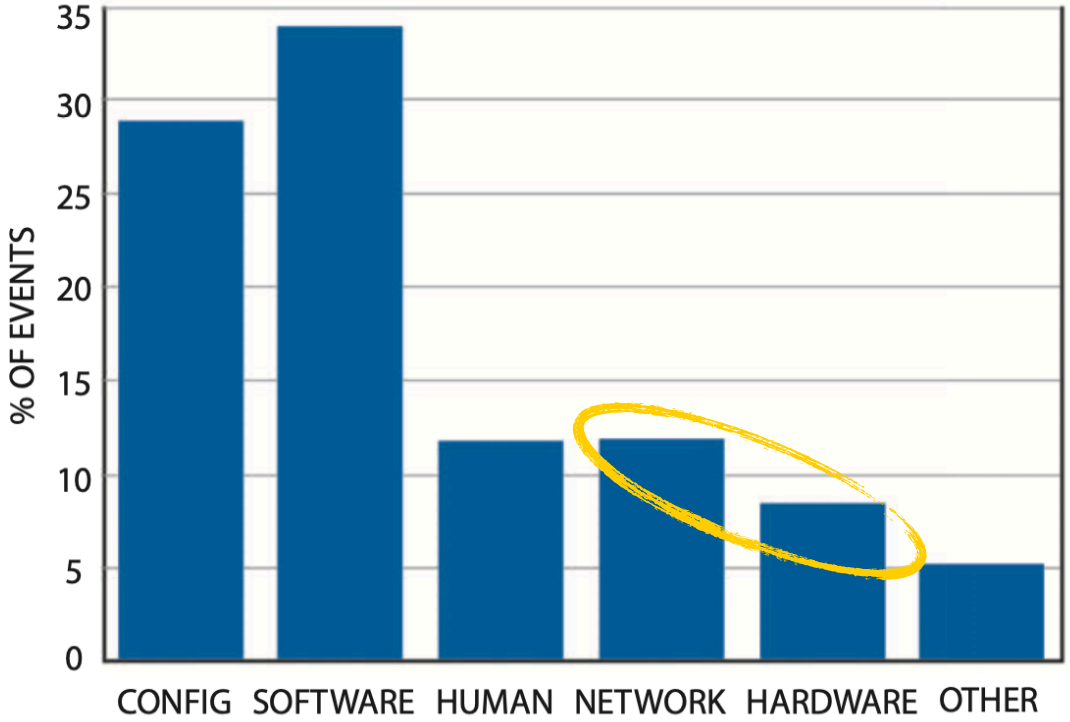
- While “Masked” is the ideal point in the spectrum of fault severity, several other points may be acceptable
- E.g., an email service that takes longer than usual (say, 5 sec), or a search that returns less results
- ***Key Insight:*** most applications **can** benefit from graceful degradation
  - *Research has established that internet endpoints may be unable to reach each other between 1% and 2% of the time (i.e., connectivity issues are persistent)*
  - *This means, an Internet service that provides 99% availability is virtually indistinguishable from a perfectly reliable system!*





Causes of Faults

- The service is less likely to be disrupted by hardware or networking faults than by software errors, faulty configuration, and human mistakes
- But, why?
  - *existence of fault-tolerant software prevents low-level hardware failures from affecting apps*
  - *Software-, Config-, and Operator-level failures tend to have a cascaded effect (i.e., cause multiple failures in a correlated fashion)*
- These findings are NOT unique to WSC systems; Similar results have been reported in HPC systems, enterprise systems, and other distributed systems



The reasons attributed to service level disruptions/ degradations of a major Google service (collected over a period of six week)

Fault-tolerant Designs

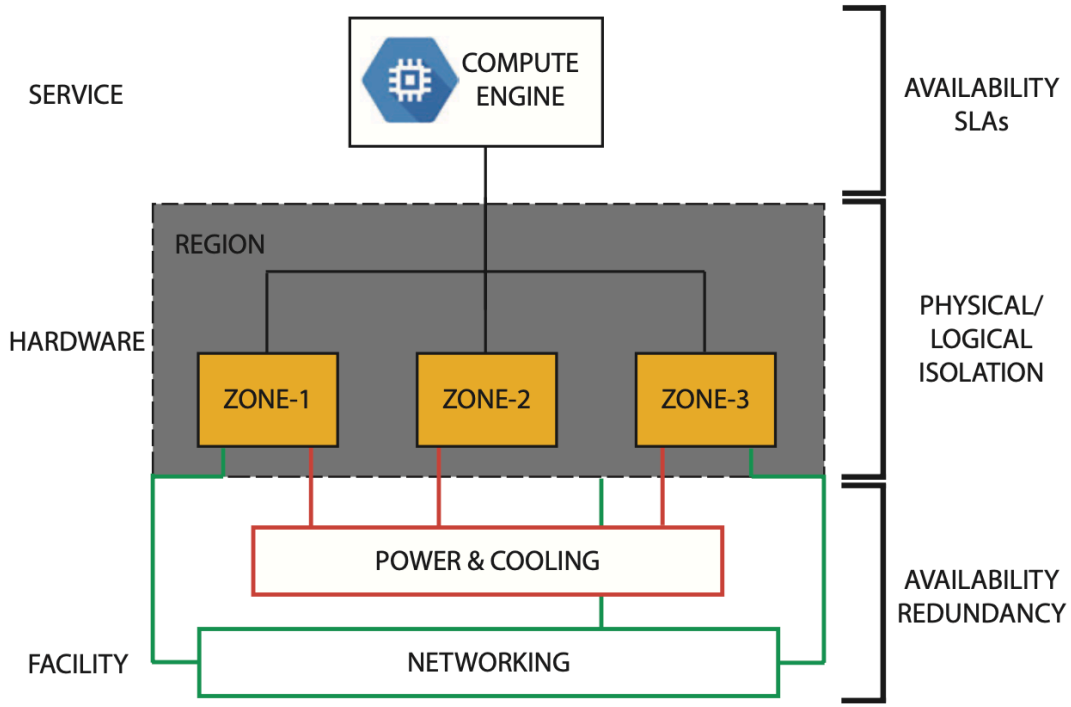
Fault-tolerant WSCs

Techniques
Replication
Sharding
Encoding
Redundant execution
Load balancing
Eventual consistency

- **Guiding principle**: why hide faults when you can tolerating them
  - *Case in point: any typical WSC application, say web server, will incorporate several of the reliability techniques (in the table)*
- Still, broken machines must be eventually replaced
  - *in contrast to traditional enterprise IT, repairs in datacenters can be handled with some delay and in batches*
- Key aspects of a fault-tolerant WSC
  - ***Power and cooling**: multiple sources of electricity and cooling, redundant connections to each rack*
  - ***Networking**: full bisection bandwidth within the datacenter, high-speed interconnect with other datacenters*
  - ***Servers and Storage**: spare capacity, fine-grained monitoring infrastructure*
  - ***Job scheduling**: spread jobs across failure units (machines, racks, and power domains), automatically restart evicted jobs*

Fault-tolerant Applications

- Understand the Service Level Agreements (SLAs) before designing applications
- Cloud providers expose to users their hardware-level isolations
  - *Regions and zones (across datacenters)*
  - *Machines and racks (within the datacenter)*
- Other alternatives:
  - *using services/platforms of multiple cloud providers (i.e., supercloud architectures)*
  - *hybrid applications with public- and private-cloud components*



Essay (ICON)

No spot quiz this week
(since we had an essay worth 2x last lecture)