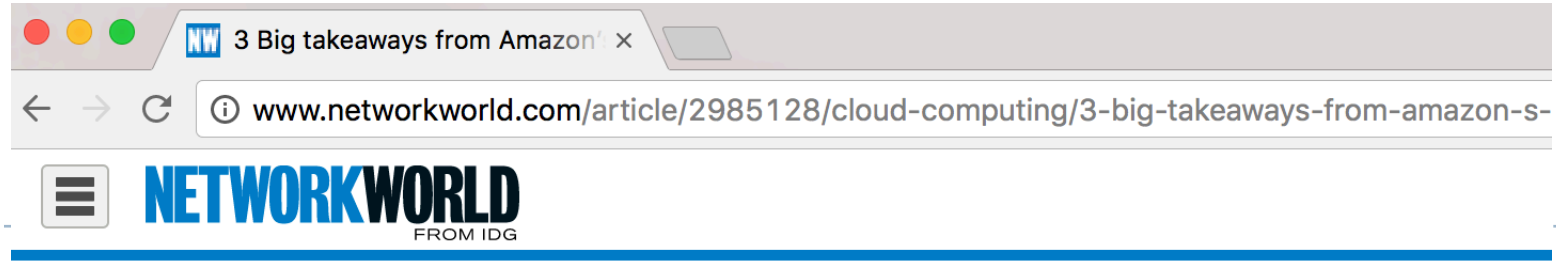
Cristina Nita-Rotaru

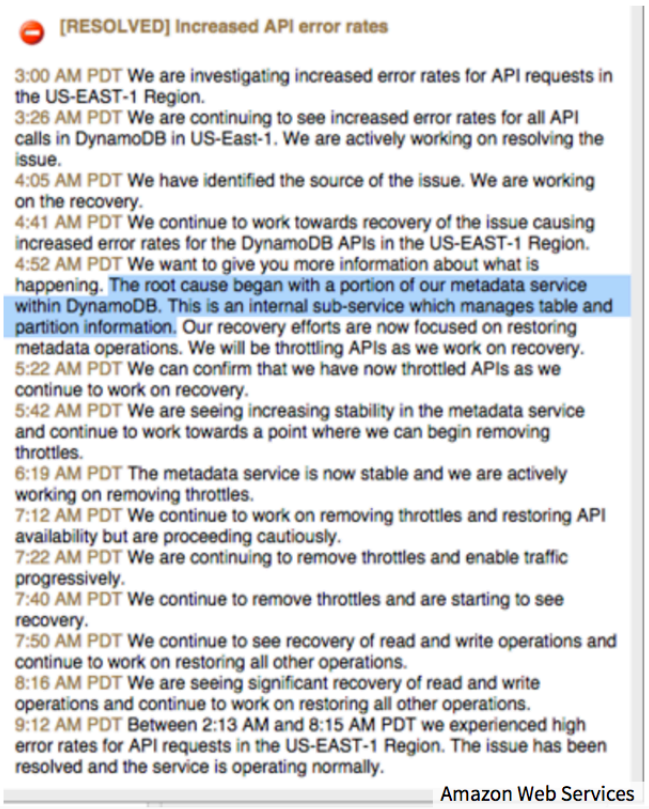
7680: Distributed Systems

Introduction. Class Policy. Examples.

1: Introduction



At 6 AM ET error rates for the company's massive NoSQL database named DynamoDB began skyrocketing in AWS's US-East Virginia region - the oldest and largest of its nine global regions. By 7:52 AM ET, AWS determined the cause of the problems: an issue with how the database manages metadata had gone awry, impacting the service's partitions and tables.



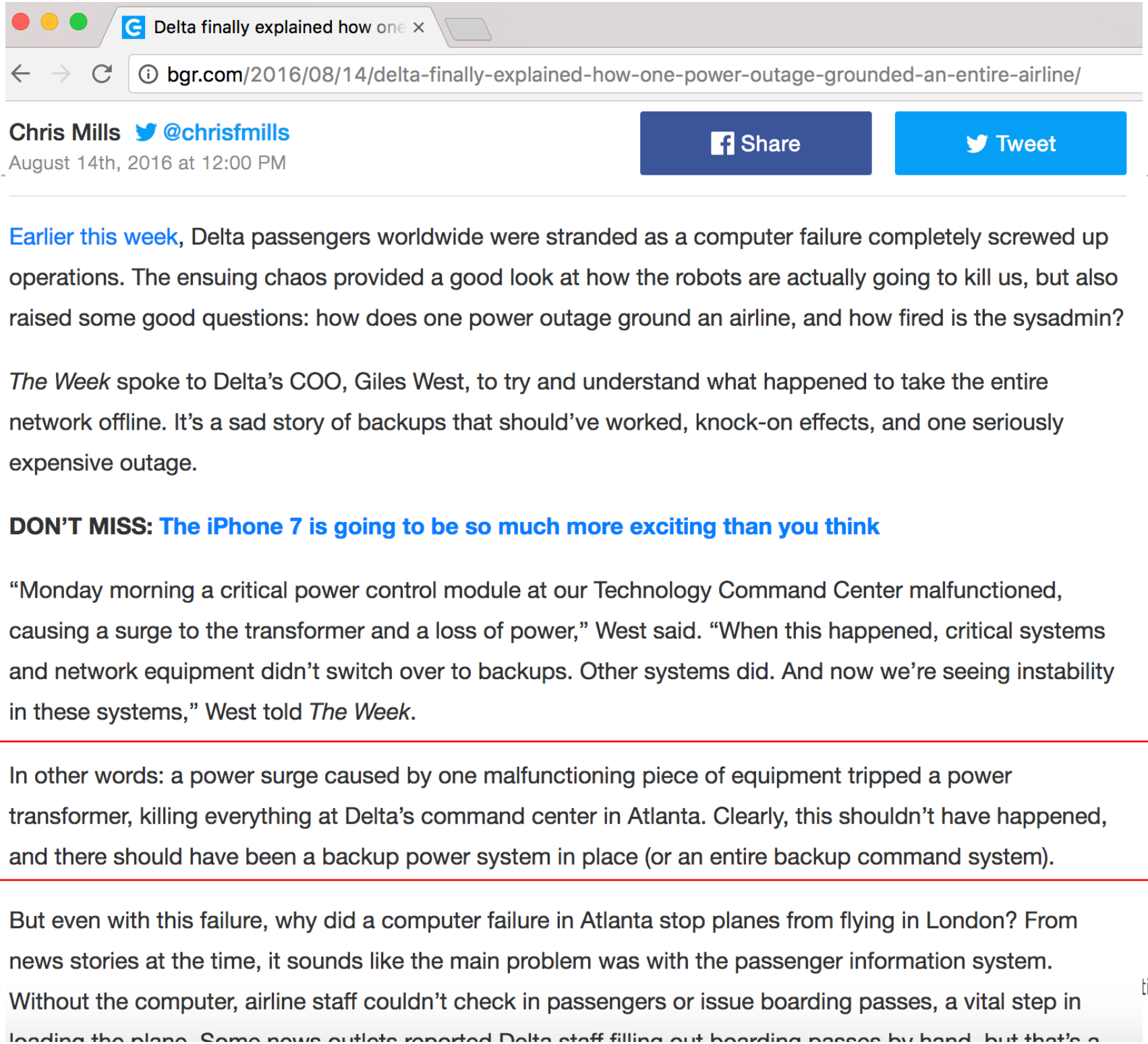
[RESOLVED] Increased API error rates

3:00 AM PDT We are investigating increased error rates for API requests in the US-EAST-1 Region.
3:26 AM PDT We are continuing to see increased error rates for all API calls in DynamoDB in US-East-1. We are actively working on resolving the issue.
4:05 AM PDT We have identified the source of the issue. We are working on the recovery.
4:41 AM PDT We continue to work towards recovery of the issue causing increased error rates for the DynamoDB APIs in the US-EAST-1 Region.
4:52 AM PDT We want to give you more information about what is happening. The root cause began with a portion of our metadata service within DynamoDB. This is an internal sub-service which manages table and partition information. Our recovery efforts are now focused on restoring metadata operations. We will be throttling APIs as we work on recovery.
5:22 AM PDT We can confirm that we have now throttled APIs as we continue to work on recovery.
5:42 AM PDT We are seeing increasing stability in the metadata service and continue to work towards a point where we can begin removing throttles.
6:19 AM PDT The metadata service is now stable and we are actively working on removing throttles.
7:12 AM PDT We continue to work on removing throttles and restoring API availability but are proceeding cautiously.
7:22 AM PDT We are continuing to remove throttles and enable traffic progressively.
7:40 AM PDT We continue to remove throttles and are starting to see recovery.
7:50 AM PDT We continue to see recovery of read and write operations and continue to work on restoring all other operations.
8:16 AM PDT We are seeing significant recovery of read and write operations and continue to work on restoring all other operations.
9:12 AM PDT Between 2:13 AM and 8:15 AM PDT we experienced high error rates for API requests in the US-EAST-1 Region. The issue has been resolved and the service is operating normally.

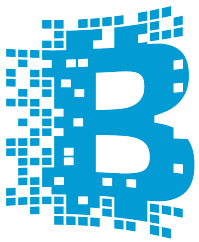
Amazon Web Services

Amazon Web Service's Health Dashboard shows

Because of the intricate interconnectivity of AWS's services, the issue snowballed to impact 34 total services (out of 117) that the company's Service Health Dashboard monitors. Everything from Elastic Compute Cloud (EC2) virtual machines to the Glacier storage service to its Relational Database Service were impacted. According to media reports, other companies that rely on AWS experienced outages too, ranging from Netflix to IMDB, to Tinder, Pocket and Buffer.


Delta finally explained how one ×
bgr.com/2016/08/14/delta-finally-explained-how-one-power-outage-grounded-an-entire-airline/
Chris Mills @chrisfmills
August 14th, 2016 at 12:00 PM
Share
Tweet
Earlier this week, Delta passengers worldwide were stranded as a computer failure completely screwed up operations. The ensuing chaos provided a good look at how the robots are actually going to kill us, but also raised some good questions: how does one power outage ground an airline, and how fired is the sysadmin?
The Week spoke to Delta's COO, Giles West, to try and understand what happened to take the entire network offline. It's a sad story of backups that should've worked, knock-on effects, and one seriously expensive outage.
DON'T MISS: The iPhone 7 is going to be so much more exciting than you think
"Monday morning a critical power control module at our Technology Command Center malfunctioned, causing a surge to the transformer and a loss of power," West said. "When this happened, critical systems and network equipment didn't switch over to backups. Other systems did. And now we're seeing instability in these systems," West told The Week.
In other words: a power surge caused by one malfunctioning piece of equipment tripped a power transformer, killing everything at Delta's command center in Atlanta. Clearly, this shouldn't have happened, and there should have been a backup power system in place (or an entire backup command system).
But even with this failure, why did a computer failure in Atlanta stop planes from flying in London? From news stories at the time, it sounds like the main problem was with the passenger information system. Without the computer, airline staff couldn't check in passengers or issue boarding passes, a vital step in

- Air Traffic Control
- Space Shuttle
- Banking Systems
- Grid Power Systems
- Cloud Computing



BLOCKCHAIN

Cloud computing

- Management infrastructure tools
  - Chubby, Group Membership, Distributed Registries, PBFT
- Scalable data sharing and event notification
  - Pub-Sub, multicast, gossip, Group Communication, Sinfonia
- Network-Level and other resource-managed technologies
  - Load balancing
- Shared global file system, or in-memory store services.
  - GFS, HDFS, S3, Memcached, Dynamo
- Scalable database systems and transactional subsystems
  - BigTable, Spanner, Casandra
- Large-scale distributed data processing
  - MapReduce, DryadLINQ, Storm, Spark

What do these services have in common?

- They rely on many resources (i.e. computers, servers, data centers)
  - Scalability
  - Availability
  - Performance

- They are located in different places
- They require coordination
- They depend on different services

What is a distributed system?

A *distributed computing system* is a set of computer programs executing on one ore more computers and coordinating actions by exchanging messages.

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

Attributed to Leslie Lamport

Distributed systems requirements

- Reliability: provide continuous service
- Availability: ready to use
- Safety: systems do what they are supposed to do, avoiding catastrophic consequences
- Security: withstands passive/active attacks from outsiders or insiders

...not easy to achieve because

- Computers and networks fail in many (often unpredictable) ways
- Computers get compromised
- Real-time constraints
- Performance requirements
- Complexity

Why Do Computer Systems Fail?

- *Why Do Computers Stop and What can be done about it? Jim Gray, 1985*
  - System administration (operator actions, system configuration and maintenance)
  - Software faults, environmental failures
  - Hardware failures (disks and communication controllers)
  - Power outages
- *Why do Internet services fail, and what can be done about it? D. Oppenheimer, A.Ganapathi and D.A. Patterson, 2003.*
  - Operator error (particularly configuration errors) is the leading cause of failures
  - Failures in custom-written front-end software
  - Not enough on-line testing

Why Do Computers Get Compromised?

- Software bugs
- Administration errors
- Lack of diversity, same vulnerability is exploited
- The explosion of the Internet facilitates the spread of malware

..how do computer system fail...

- Halting failures: no way to detect except by using timeout
- Fail-stop failures: accurately detectable halting failures
- Send-omission failures
- Receive-omission failures
- Network failures
- Network partitioning failures
- Timing failures: temporal property of the system is violated
- Byzantine failures: arbitrary failures, include both benign and malicious failures

What is this class about

THEORY + SYSTEM IMPLEMENTATION

- Theory
  - Fundamental algorithms and services
  - Impossibility results
  - Trade-offs between different characteristics designing distributed systems
- Implementation
  - What can go wrong when designing, implementing, testing and deploying a distributed service
  - Design of existing and popular software
  - Dependencies between different services
  - Interoperability issues

2: Syllabus and class policy

Course Information

- Meetings
  - MW 2:50-4:30pm Jan.9 – Apr. 30
- Professor contact info:
  - Office: WVH 258
  - Email: c.nitarotaru@neu.edu
  - Office hours: from 2 pm before each class or by appointment
- Class webpage

http://cnitarot.github.io/courses/ds_Spring_2017/index.html

- Piazza for class communication
  - Use Piazza for questions and postings
  - Hw and projects posted on piazza

Prerequisites

- Strong systems and networking background
- Socket programming
- Fluency in many languages
  - C/C++
  - Java
  - Python or some other scripting language
- Linux command line proficiency
- Some computer security and cryptography fundamentals

Course overview

- Time in distributed systems. Clock synchronization. Global states and distributed snapshots.
- Consensus: synchronous systems, asynchronous systems, byzantine failures (including randomized solutions).
- Distributed commit (2PC and 3PC). Weak consistency models. Weak and strong consistency in partitioned database systems. Linearizability. CAP Theorem
- Process Groups: Leader election, membership, reliable multicast, virtual synchrony. Gossip protocols.
- Quorums. Paxos. Viewstamped replication. BFT.
- Reliable distributed shared memory.
- Peer-to-peer file sharing, indexing, data fusion and data mining.
- Popular systems: GFS/HDFS; BigTable/Spanner/HBASE; Chubby/Zookeeper; MapReduce/Spanner; Mesos/Yarn.

Reference Material

- Textbooks
  - Ken Birman: Reliable Distributed Systems
- Recommended reading
  - Research papers that will be specified for each lecture

Grading policy

- Written assignments 25%
- Programming projects 45%
- Final project 20
- Class participation 10%

- There is no curve for grades

Written assignments

- **Purpose of the written assignments is to make you understand the theoretical results discussed in class**
  - **Read the material before solving them and solve them with closed books and notebooks**
- 4-5 written theoretical assignments
- Homework is individual
- Homework must be typed – PDF submission format only
- For submission, follow the information in the homework description

Programming projects

- **Purpose of the programming projects is to help you understand practical aspects of things discussed in class**
  - **Read all material in class and the description of the project in details before starting**
- 3 programming projects
- Programming projects are individual
- All the code must be from scratch
- Use the languages, tools, VMs, specified in the project description

Final project

- **Purpose of the final project is to help you understand some existing software or start a research project**
- No extra days for the final project
- You must work in teams of 2
  - Start shopping for a partner at the beginning of the semester, don't wait till the project is assigned/chosen
- You can choose the final project, or I can assign one
- Project proposal presentation (1 page)
- Final presentation in class + report of 3 pages submitted with the code and presentation

Late policy

- Each of you gets 5 LATE DAYS that can be used any way you want for homework and projects (but not the final project); you do not need to let me know if you plan to take any late day; just submit late
  - Keep track of your late days used
  - 20% off from grade obtained for that project or homework per day late
- Follow the requirements from project description to see how to submit
- **Assignments are due at 9:59:59 pm, no exceptions**
  - **1 second late = 1 hour late = 1 day late**

Regrading

- YOU HAVE 1 WEEK to ASK for REGRADING of a homework or project from the moment solutions were posted on piazza or discussed in class
- Make sure you read and understand the solution before asking for a regrade
- Request for a regrade will result in the regrading of the entire homework, project

Class attendance and notes

- Your are strongly recommended to attend and take notes
- If you miss class is your responsibility to go through the covered material on your own
- Slides will be made available online after lecture
- There will be assigned reading from papers and other online materials
- Class participation is 10% of your grade
  - Be active on Piazza
  - Ask questions in class
  - Answer questions in class

Individual Meeting

- You are required to meet with me at least once per semester
- I will send doodle links with available time slots that you can sign up for, in the next few weeks
- You can always set up additional appointments by sending me an email first

Academy integrity

- It is allowed to discuss homework problems before writing them down; however, WRITING IS INDIVIDUAL
  - if you look at another student's written or typed answers, or let another student look at your written or typed answers, that is considered cheating.
- Never have a copy of someone else's homework or program in your possession and never give your homework (or password) or program to someone else.
- NO CHEATHING WILL BE TOLERATED.
- ANY CHEATING WILL AUTOMATICALLY RESULT in F grade and report to the university administration

How to ask on Piazza

- Read slides, notes, homework or project description
- Use #hashtags (#lecture2, #project3, #hw1, etc.)
- Describe the problem clearly, using the right terms
- Add code in attached files
- Add output from compiler or debugginng information
- Add any other relevant information
- **Don't post solutions on piazza**
- **Anything that relates to solution post PRIVATELY**

Weather/Emergency

- In the event of a major campus emergency, course requirements, deadlines and grading percentages are subject to changes that may be necessitated by a revised semester calendar or other circumstances beyond the instructor's control.
- Monitor weather and piazza particularly if you don't live close to school.

Debugging distributed protocols

- They are known to be difficult to debug
- Write proactively – print all the info you send/receive over the network;
- Have state machine design before implementation and make sure you understand what your state machine is supposed to do before you implement your code
- Have message detailed description in design before implementation
- Focus on testcases to understand specific behavior
  - Delay, interleave, drop messages
  - Crash participants

One last word …

- **No meetings will be accepted with the TA or instructor the day homework or projects are due**
- Start early, plan carefully
- Develop your solution gradually, test gradually so you always have functionality for which you can receive a grade; YOUR CODE MUST WORK
- Do not wait to submit your code last minute
- Don't post solutions on piazza
- Don't cheat

PIAZZA ACCOUNTS

- All communication is on piazza, make sure you get notifications and you check piazza constantly
- If you have not received a piazza notification email me c.nitarotaru@neu.edu

1: Examples of distributed systems: Google File System (GFS)

Application Characteristics

- Hundreds of clients must perform concurrent (atomic) appends with minimal synchronization
- Sustained bandwidth more important than latency
- Response time for individual read/write not important
- Non-traditional access patterns
- Files are very big, several gigs

Design Choices

- Fault-tolerance is a must; Constant monitoring, error detection, automatic recovery part of the design
- Designed for big files. I/O operations and block sizes have to be revisited.
- Non-traditional access patterns:
  - Most files are modified by appending rather than overwriting;
  - Large repositories that must be scanned (archival, streams, intermediate data)
  - Result: appending is the focus of optimization
- Co-design applications and file system; looser consistency

Interface Design

- Not standard API (such as POSIX)
- Supports file/directory hierarchy and the usual: {create, delete, open, close, read, write}
- Additionally:
  - **snapshot: low cost file / directory tree copying**
  - **record append: concurrent appends, no locks!**

Architecture Overview

- No files, base unit is **chunk**:
  - Fixed-part of a file, typically 64 MB
  - Global ID: 64 bit, unique "chunk handle", assigned by master server upon chunk creation
  - Read/Write: need chunk handle + byte range
- Servers:
  - Single master
  - Multiple backups (chunkservers)
- Multiple clients

Design Overview: Consistency Model

- File namespace modifications: atomic & handled only by master server
- Chunks:
  - "consistent" if all clients see same data, no matter which replica they ask
  - "defined" if it is consistent and known, i.e. some modification done w/o interruption
  - "undefined" if concurrent modifications are successful
  - "inconsistent" on any failed modification
  - Inconsistent regions may be padded or contain duplicates

Design Overview: Consistency Model

- File namespace modifications: atomic & handled only by master server
- Chunks:
  - "consistent" if all clients see same data, no matter which replica they ask
  - "defined" if it is consistent and known, i.e. some modification done w/o interruption
  - "undefined" if concurrent modifications are successful
  - "inconsistent" on any failed modification
  - Inconsistent regions may be padded or contain duplicates

New Google Storage System

- New requirements: highly interactive applications (for example email)
- Available all the time
- Wide-area network
- One of the biggest changes was using active replication – based on PAXOS, proven, optimal, fault-tolerant consensus algorithm with no requirement for a distinguished master

Required Reading

- Chapter 1 and 2 from Reliable Distributed Systems
- Why do Internet services fail, and what can be done about it? D. Oppenheimer, A. Ganapathi and D. A. Patterson, 2003.
- Why Do Computers Stop and What can be done about it? Jim Gray, 1985.