CS162
Operating Systems and Systems Programming
Lecture 25

Why Systems Fail and What We Can Do About It

November 30, 2011
Anthony D. Joseph and Ion Stoica
http://inst.eecs.berkeley.edu/~cs162

Goals for Today

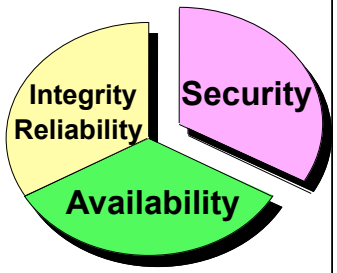
- Definitions for Fault Tolerance
- Causes of system failures
- Possible solutions
- Staying broad

"You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done." —LESLIE LAMPORT

Note: Some slides and/or pictures in the following are adapted from slides from a talk given by Jim Gray at UC Berkeley on November 9, 2000.

Dependability: The 3 ITIES

- **Reliability / Integrity:** does the right thing.
  (Also large MTBF)
- **Availability:** does it now.
  (Also small $\frac{\text{MTTR}}{\text{MTBF+MTTR}}$)

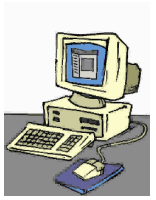


- **System Availability:** if 90% of terminals up & 99% of DB up?
  (=> 89% of transactions are serviced on time)

MTBF or MTTF = Mean Time Between (To) Failure
MTTR = Mean Time To Repair

Fault Tolerance versus Disaster Tolerance

- **Fault-Tolerance:** mask local faults
  - RAID disks
  - Uninterruptible Power Supplies
  - Cluster Failover
- **Disaster Tolerance:** masks site failures
  - Protects against fire, flood, sabotage,..
  - Redundant system and service at remote site.
  - Use design diversity



High Availability System Classes
Goal: Build Class 6 Systems

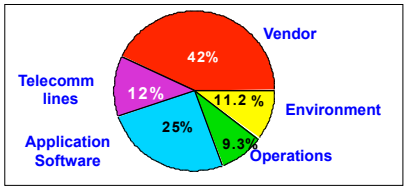
System Type	Unavailable (min/year)	Availability	Availability Class
Unmanaged	50,000	90.%	1
Managed	5,000	99.%	2
Well Managed	500	99.9%	3
Fault Tolerant	50	99.99%	4
High-Availability	5	99.999%	5
Very-High-Availability	.5	99.9999%	6
Ultra-Availability	.05	99.99999%	7

UnAvailability = MTTR/MTBF

can cut it in ½ by cutting MTTR *or* MTBF

Case Study – Japan

"Survey on Computer Security", Japan Info Dev Corp., March 1986. (trans: Eiichi Watanabe).



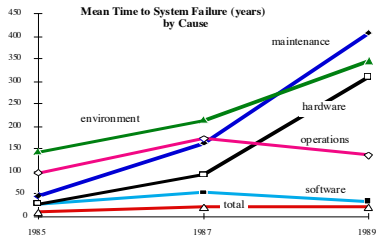
Vendor (hardware and software)	5	Months
Application software	9	Months
Communications lines	1.5	Years
Operations	2	Years
Environment	2	Years
	10	**Weeks**

1,383 institutions reported (6/84 - 7/85)

7,517 outages, MTBF ~ 10 weeks, avg duration ~ 90 MINUTES

To Get 10 Year MTBF, Must Attack All These Areas

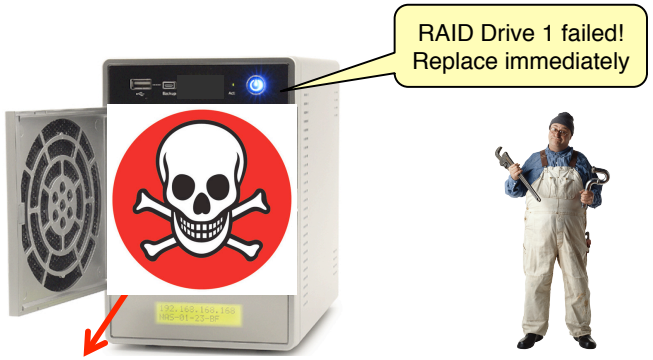
Case Studies - Tandem Trends
Reported MTBF by Component



	1985	1987	1990	
SOFTWARE	2	53	33	Years
HARDWARE	29	91	310	Years
MAINTENANCE	45	162	409	Years
OPERATIONS	99	171	136	Years
ENVIRONMENT	142	214	346	Years
SYSTEM	**8**	**20**	**21**	**Years**

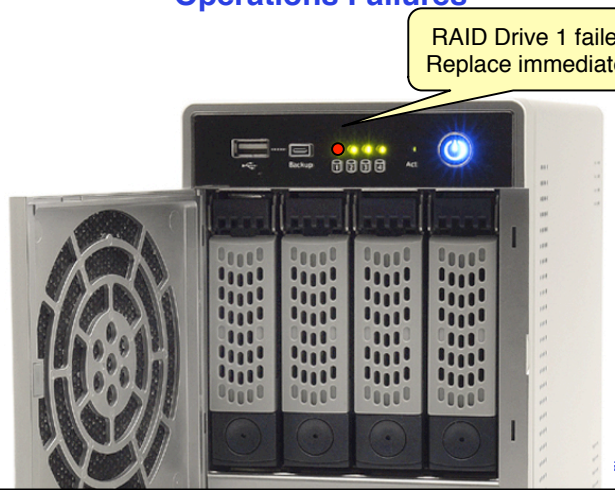
Problem: Systematic Under-reporting

Operations Failures



What went wrong??

Operations Failures



Cloud Computing Outages 2011

Vendor	When	Duration	What Happened & Why
Apple iPhone 4S Siri	November 2011	1 Day	Siri loses even the most basic functionality when Apples servers are down. Because Siri depends on servers to do the heavy computing required for voice recognition, the service is useless without that connection. Network outages caused the disruption according to Apple.
Blackberry outage	October 2011	3 Days	Outage was caused by a hardware failure (core switch failure) that prompted a "ripple effect" in RIM's systems. Users in Europe, Middle East, Africa, India, Brazil, China and Argentina initially experienced email and message delays and complete outages and later the outages spread to North America too. Main problem is message backlogs and the downtime produced a huge queue of undelivered messages causing delays and traffic jams.
Google Docs	September 2011	1 Hour	Google Docs word collaboration application cramp, shutting out millions of users from their document lists, documents, drawings and Apps Scripts. Outage was caused by a memory management bug software engineers triggered in a change designed to "improve real time collaboration within the document list.
services - Hotmail & SkyDrive	2011		Domain Name Service (DNS). Network traffic balancing tool had an update and the update did not work properly which caused the issue.
Amazon's EC2 cloud &	August 2011	1-2 days	Transformer exploded and caught fire near datacenter that resulted in power outage due to generator failure. Power back up systems at both the data centers failed causing power outages. Transformer explosion was caused by lightening strike but disputed by local utility provider.
Microsoft's BPOS	August 2011	1-2 days	Transformer exploded and caught fire near datacenter that resulted in power outage due to generator failure. Power back up systems at both the data centers failed causing power outages. Transformer explosion was caused by lightening strike but disputed by local utility provider.

From: http://analysiscasestudy.blogspot.com/

Cloud Computing Outages 2011

Vendor	When	Duration	What Happened & Why
Amazon Web Services	April, 2011	4 Days	During the upgrade, the traffic shift was executed incorrectly and rather than routing the traffic to the other router on the primary network, the traffic was routed onto the lower capacity redundant EBS network. This led to Amazon Elastic Block Store ("EBS") volumes in a single Availability Zone within the US East Region that became unable to service read and write operations. It also impacted the Relational Database Service ("RDS"). RDS depends upon EBS for database and log storage, and as a result a portion of the RDS databases hosted in the primary affected Availability Zone became inaccessible.
BPOS Outages			started getting stuck in the pipeline.
Twitter Outages	March & Feb 2011	1-4 Hours	Outages due to over capacity and moving operations to new data center.
Intuit Quick Books Online	March 2011	2 Days	Service failures on human error during scheduled maintenance operations. Inuit changed its network configuration and inadvertently blocked customer access to a portion of the company's servers. A surge in traffic overloaded the servers when connectivity was restored, so the company opted to restore service.
Google Mail and Apps Outage	February 2011	2 Days	Google mail and Google Apps users experienced login errors and empty mailboxes. Google Engineering determined that the root cause was a bug inadvertently introduced in a Gmail storage software update. The bug caused the affected users' messages and account settings to become temporarily unavailable from the datacenters.

From: http://analysiscasestudy.blogspot.com/

Cloud Computing Outages 2010

Vendor	When	Duration	What Happened & Why
Hotmail Outage	December 2010	3 Days	A number of our users reported their email messages and folders were missing from their Hotmail accounts. Error occurred from a script that was meant to delete dummy accounts
Skype Outage	December 2010	1 Day	Cluster of support servers responsible for offline instant messaging became overloaded and the P2P network became unstable and suffered a critical failure. A supernode is important to the P2P network acting like a directory, supporting other Skype clients, helping to establish connections between them etc. The failure of 25–30% of supernodes in the P2P network resulted in an increased load on the remaining supernodes.
Paypal Outage	November 2010	3 Hours	A network hardware failure was the trigger for an outage. The hardware failure was worsened by
Facebook Outage	September 2010	2 ½ Hours	Outage due to an error condition. An automated system for verifying configuration values ended up causing much more damage than it fixed. Every single client saw the invalid value and attempted to fix it that led to a query to a database cluster and cluster was overloaded with thousand of queries per second. Even after fixing problem stream of queries continued.
BPOS Outages	September 2010		period of intermittent access for BPOS organizations served from North America.
Wikipedia Outage	July & March 2010	2-3 Hours	In July, the power failure is understood to have affected Wikimedia's '*pmtpa*' cluster. Due to the temporary unavailability of several critical systems and the large impact on the available systems capacity, all Wikimedia projects went down. In March, Wikimedia servers overheated in the organization's European data center and shut themselves off automatically. Wikimedia then switched all its traffic to its server cluster in Florida, but the failover process, which involves changing servers' DNS entries, malfunctioned, knocking the organization's sites offline around the world.
Hosting.com Outage	June 2010	2 Hours	Failure of a Cisco switch at the Newark, N.J., data center caused intermittent network connectivity. Dedicated switch had failed, the second failover switch had crashed as well and the problem was caused by a software bug.
Twitter.com outage	June 2010	5 hours	Increased activity on the site, combined with system enhancements and upgrades, have uncovered networking issues. Incidences of poor site performance and a high number of errors due to one of the internal sub-networks being over-capacity.

Cloud Computing Outages 2009

Vendor	When	Duration	What Happened & Why
Salesforce.com Outage	January 2010, 2009	1-2 Hours	Outages were caused by server disruption, when a core network device failed, stopping all data from being processed in Japan, Europe, and North America. The technical reason for the outage: a core network device had failed, due to memory allocation errors. The backup plan, which was supposed to trigger a cut-over to a redundant system, also failed.
Amazon's EC2	June 2009	4-5 Hours	A lightning storm caused damage to a single Power Distribution Unit (PDU) in a single Availability Zone
eBay Paypal	August 2009	1-4 Hours	Online payments system failed a couple of times led to non completion of transactions. Network hardware issue is blamed for outage.
Twitter	August 2009	½ Day	A denial-of-service attack was blamed for the problem
Google Gmail	September 2009	2 hours 2 times	Reasons from vendors include routing errors to server maintenance issues.
Microsoft Sidekick	October 2009	6 days	Microsoft's Danger server farm, that holds the cloud T-Mobile Sidekick subscriber's data crashed, depriving users of their calendar, address book, and other key data. Critical data was lost during outage.
Rackspace.com Outage	June 2009 December 2009	1 Day 1 Hour	Power outage and subsequent power generator failures that caused servers to fail. Company was forced to pay out between $2.5 million and $3.5 million in service credits to customers. The issues resulted from a problem with a router used for peering and backbone connectivity located outside the data center at a peering facility, which handles approximately 20% of Rackspace's Dallas traffic. The router configuration error was part of final testing for data center integration between the Chicago and Dallas facilities.

From: http://analysiscasestudy.blogspot.com/

Fault Model

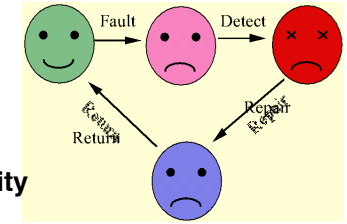
- Failures are independent*
  So, single fault tolerance is a big win
- Hardware fails fast (blue-screen, panic, …)
- Software fails-fast (or stops responding/hangs)
- Software often repaired by reboot:
  - Heisenbugs – Works On Retry
  - (Bohrbugs – Faults Again On Retry)
- Operations tasks: major source of outage
  - Utility operations – UPS/generator maintenance
  - Software upgrades, configuration changes

Some Fault Tolerance Techniques

- Fail fast modules: work or stop
- Spare modules: yield instant repair time
- Process/System pairs: Mask HW and SW faults
- Transactions: yields ACID semantics (simple fault model)

Fail-Fast is Good, Repair is Needed

Lifecycle of a module fail-fast gives short fault latency



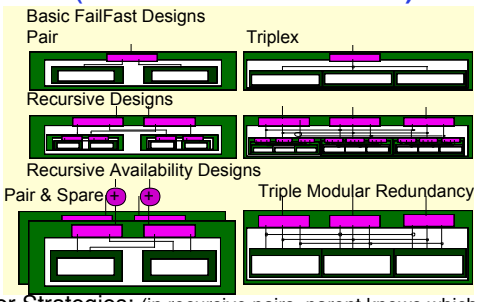
High Availability is low UN-Availability

Unavailability ~ $\frac{\text{MTTR}}{\text{MTBF}}$

Improving either MTTR or MTBF gives benefit

Simple redundancy does not help much

Hardware Reliability/Availability
(how to make HW fail fast)

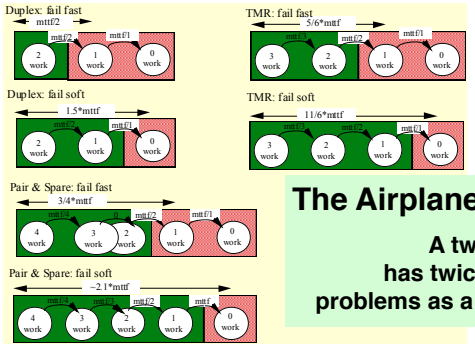


Comparitor Strategies: (in recursive pairs, parent knows which is bad)

Duplex: Fail-Fast: fail if either fails (e.g. duplexed CPUs)
vs Fail-Soft: fail if both fail (e.g. disc, network,...)

Triplex: Fail-Fast: fail if 2 fail (triplexed cpus)
Fail-Soft: fail if 3 fail (triplexed FailFast CPUs)

Redundant Designs have Worse MTBF!



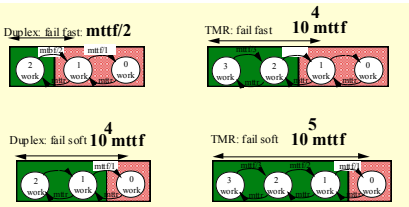
The Airplane Rule:

A two-engine airplane has twice as many engine problems as a one engine plane

THIS IS NOT GOOD: Variance is lower but MTBF is worse

Simple redundancy does not improve MTBF (sometimes hurts)

Add Repair: Get 10^4 Improvement



Availability estimates
1 year MTTF modules
12-hour MTTR

	MTTF	EQUATION	COST
SIMPLEX	1 year	MTTF	1
DUPLEX: FAIL FAST	~0.5 years	~ MTTF/2	2+ε
DUPLEX: FAIL SOFT	~1.5 years	~ MTTF(3/2)	2+ε
TRIPLEX: FAIL FAST	.8 year	~ MTTF(5/6)	3+ε
TRIPLEX: FAIL SOFT	1.8 year	~ 1.8MTTF	3+ε
Pair and spare: FAIL-FAST	~.7 year	~ MTTF(3/4)	4+ε
TRIPLEX WITH REPAIR	>10^5 years	$MTTF^3/3MTTR^2$	3+ε
Duplex fail soft + REPAIR	>10^4 years	$MTTF^2/2MTTR$	4+ε

Software Techniques: Learning from Hardware

Recall that most outages are not hardware

Most outages in Fault Tolerant Systems are SOFTWARE

Fault Avoidance Techniques: Good & Correct design

After that: Software Fault Tolerance Techniques:

- **Modularity** (isolation, fault containment)
- **N-Version Programming**: N-different implementations
- **Programming for Failures**: Programming paradigms that assume failures are common and hide them
- **Defensive Programming**: Check parameters and data
- **Auditors**: Check data structures in background
- **Transactions**: to clean up state after a failure

Paradox: Need Fail-Fast Software

Fail-Fast and High-Availability Execution

Process Pairs: Instant restart (repair)

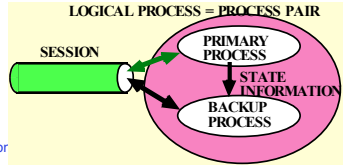
Use Defensive programming to make a process fail-fast
Have restarted process ready in separate environment
Second process "takes over" if primary faults

If software fault (bug) is a Bohrbug, then there is no repair
"wait for the next release" or "get an emergency bug fix" or
"get a new vendor"

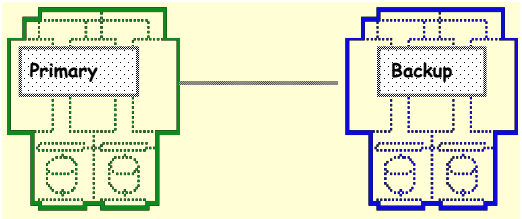
If software fault is a Heisenbug, then repair is
"reboot and retry" or "switch to backup process (instant restart)"

Tolerates HW faults too!

Repair time is seconds, could be ms if time critical



SYSTEM PAIRS FOR HIGH AVAILABILITY



- Programs, Data, Processes Replicated at 2+ sites
  - Pair looks like a single system
- System becomes logical concept
  - Like Process Pairs: System Pairs.
- Backup receives transaction log (spooled if backup down)
- If primary fails or operator switches, backup offers service

Add Geographic Diversity to *Reduce* Single Points of Failure*

Administrivia

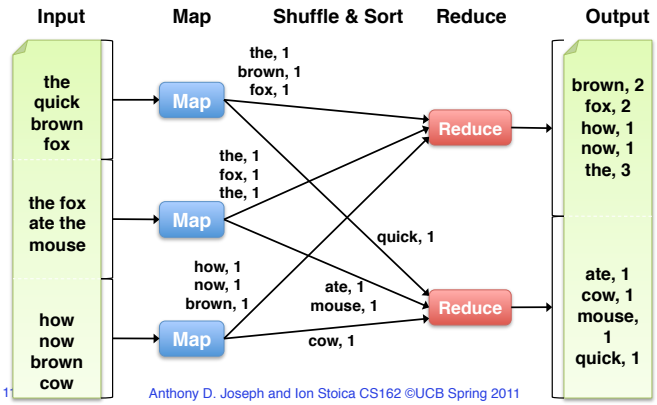
- Final:
  - Thursday, December 15, 8-11am, 155 Dwinelle
  - Closed book, **two** pages of hand-written notes (both sides)
- Topics:
  - 30% first part
  - 70% second part
- Review session: Next week TBA
- Office hours:
  - Anthony: Tuesday Dec 6, 10:30-11:30 am and Wednesday, Dec 7, 12:30-1:30pm
  - Ion: Tuesday, Dec 6, 9:30-10:30am and Wednesday, 11:30-12:30pm
- We will post example questions for the final
  - Look at previous years' second/third midterms

5min Break

MapReduce

- First widely popular programming model for data-intensive apps on commodity clusters
- Published by Google in 2004
  - Processes 20 PB of data / day
- Popularized by open-source Hadoop project
  - 40,000 nodes at Yahoo!, 70 PB at Facebook
- Programming model
  - Data type: key-value *records*
    - » Map function: (K_{in}, V_{in}) ➔ list(K_{inter}, V_{inter})
    - » Reduce function: $(K_{inter}, \text{list}(V_{inter}))$ ➔ list(K_{out}, V_{out})

Word Count Execution



Fault Tolerance in MapReduce

1. If a task crashes:
   - Retry on another node
     - » OK for a map because it had no dependencies
     - » OK for reduce because map outputs are on disk
   - If the same task repeatedly fails, fail the job

➢ **Note: For the fault tolerance to work, tasks must be *deterministic* and *side-effect-free***

Fault Tolerance in MapReduce

2. If a node crashes:
   - Relaunch its current tasks on other nodes
   - Relaunch any maps the node previously ran
     - » Necessary because their output files are lost

Fault Tolerance in MapReduce

3. If a task is going slowly (straggler):
   - Launch second copy of task on another node
   - Take output of whichever copy finishes first

- Critical for performance in large clusters

Takeaways

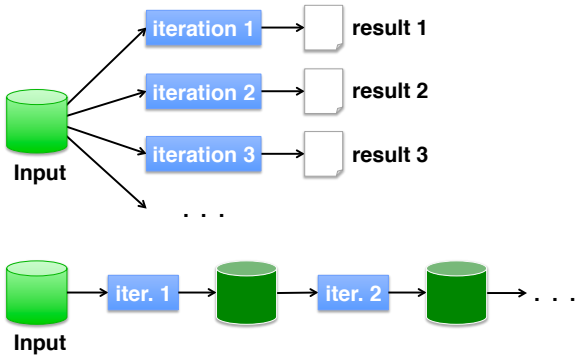
- By providing a data-parallel programming model, MapReduce can control job execution in useful ways:
  - Automatic division of job into tasks
  - Placement of computation near data
  - Load balancing
  - Recovery from failures & stragglers

Issues with MapReduce

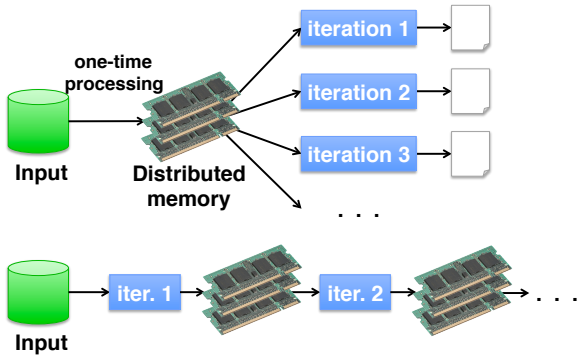
- Hard to express more complex programs
  - E.g. word count + a sort to find the top words
  - Need to write many different map and reduce functions that are split up all over the program
  - Must write complex operators (e.g. join) by hand

- Acyclic data flow -> poor support for applications that need to *reuse* pieces of data
  - Iterative algorithms (e.g. machine learning, graphs)
  - Interactive data mining (e.g. Matlab, Python, SQL)

Example: Iterative Apps



Goal: Keep Working Set in RAM



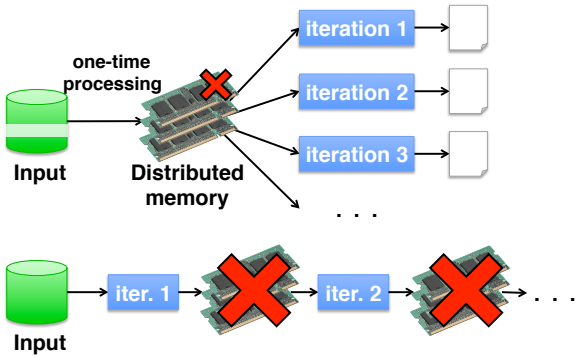
Spark Goals

- Support apps with data reuse efficiently:
  - Let them keep data in memory
- Retain the fault tolerance and automatic scheduling benefits of MapReduce
- Enhance programmability:
  - Integrate into Scala programming language
  - Allow interactive use from Scala interpreter

Key Idea: Resilient Distributed Datasets (RDDs)

- *Restricted* form of distributed shared memory
  - Read-only, partitioned collections of records
  - Can only be created through deterministic transformations (map, group-by, join, …)
- Allows efficient implementation & recovery
  - Key idea: rebuild lost data using *lineage*

RDD Recovery



Programming Model

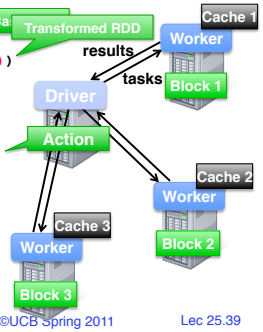
- Resilient distributed datasets (RDDs)
  - Immutable, partitioned collections of objects
  - Created through parallel *transformations* (map, filter, groupBy, join, …) on data in stable storage
  - Can be *cached* for efficient reuse
- *Actions* on RDDs
  - Count, reduce, collect, save, …

Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```



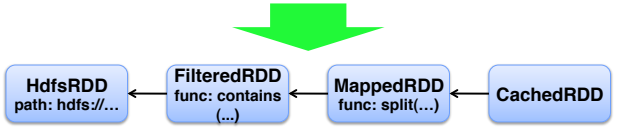
Result: scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)

RDD Fault Tolerance

RDDs maintain *lineage* information that can be used to reconstruct lost partitions

Ex:

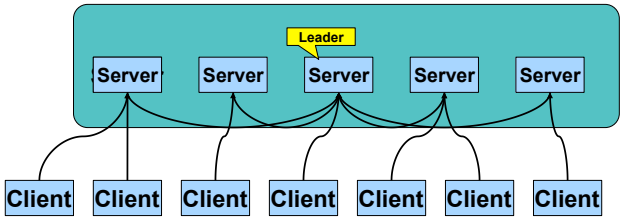
```
cachedMsgs = textFile(...).filter(_.contains("error"))
                          .map(_.split('\t')(2))
                          .cache()
```



Apache ZooKeeper

- Highly available, scalable, distributed coordination kernel
  - Leader Election, Group Membership, Work Queues, Sharding
  - Event Notifications/workflow, Config, and Cluster Mgmt
- Provides:
  - File API without partial reads/writes and no renames
  - Ordered updates and strong persistence guarantees
  - Conditional updates (version), Watches for data changes
- API:
  - String create(path, data, acl, flags)
  - void delete(path, expectedVersion)
  - Stat setData(path, data, expectedVersion)
  - (data, Stat) getData(path, watch)
  - Stat exists(path, watch)
  - String[] getChildren(path, watch)

ZooKeeper Service



- All servers store a copy of the data (in memory)
- A leader is elected at startup, or upon current leader failure
- Followers service clients, all updates go through leader
- Update responses are sent when a majority of servers have persisted the change

New CS162

- Different emphasis from CS162 in previous years; beginning in 2011 we shifted to give students a broad view on how today's systems and services
  - End-to-end system design, rather than OS only
  - More networking, database, and security concepts
  - New projects to reflect this emphasis
  - Better prepare students to design/develop such services
- Long term plan: make CS 162 a gateway course for
  - Database class (CS 186)
  - Networking class (EE 122)
  - Security class (CS 161)
  - Software engineering class (CS 169)
  - New OS class (cs16x in Fall 2012 with real OS)

New vs. Old CS162

- Curriculum: 70% overlap
  - – File systems, queuing theory, slightly fewer lectures on concurrency, caching, and distributed systems
  - + More networking, database transactions, p2p, and cloud computing

  On-going analysis of what worked and didn't work
- Different project: emphasize on how a system works end-to-end rather than focusing on implementing OS concepts in Nachos
- What if you want to do an OS project?
  - CS 16x in Fall 2012
  - Undergraduate research projects in the AMP Lab
    - » Akaros, Spartk, or Mesos projects

Stay Broad!

- Very few influential nerds – never will be
  - Why?
    - » They're too narrow
    - » Real breakthroughs tend to come from people with breadth.
- One of the most important things you should do is to force yourself to stay broad

Reasons to Stay Broad

- Breadth helps depth
  - The more you understand about many different things, the more you'll understand about each individual thing
  - Seeing how things in different areas are similar or different is a very good way of seeing what's important
- Breakthroughs often occur when people can cross traditional boundaries: compilers and architecture, graphics and VLSI, etc.
- Computers are tools: they serve people
  - To create effective tools, must understand the capabilities of computers, the capabilities of people, and the needs of the application areas where they'll be used

More Reasons to Stay Broad

- Technology is changing fast
  - If you get cubby-holed, you'll get left behind.
- Why is there a shortage of 25-year-old engineers and a surplus of 45-year-old ones?
  - Companies encourage new graduates to get so narrow (short-term focus) that they're instantly obsolete
- *If you don't keep up, you'll be left behind…*

Solutions

- Continuing education
  - Try always to be learning in whatever you do
  - Don't let your education stop when you leave Berkeley.
  - Consider getting a Masters degree or a PhD
- Explore new areas, both inside and outside Computer Science
  - Amazon AWS makes it easy to experiment
  - Everything you learn will someday be helpful, no matter how unlikely it seems – English, art, hobbies, all things are helpful
- *Bottom line: you are going to change the world!*
  - *You might not realize it yet, but the people in this classroom are going to do it*