Why do Internet services fail, and what can be done about it ?

David Oppenheimer, Archana Ganapathi, and David A. Patterson

by

Pandurang Kamat

02/24/2003

The Berkeley/Stanford Recovery-Oriented Computing (ROC) Project

- David Oppenheimer
  - Ph.D. student in CS division of U.C. Berkeley
  - Interest: Techniques for benchmarking and debugging very large-scale distributed systems
- Archana Ganapathi
  - Undergraduate student at U.C. Berkeley
  - Double majoring in Computer Science and South Asian Studies
- David A. Patterson
  - Guru of Computer Architecture, faculty at UCB since 1977.
  - Leader of RISC-I project that led to the SPARC architecture
    - likely first VLSI RISC computer
  - Joint leader of RAID development team
  - NOW project -> cluster technology
  - 5 books and numerous awards

Outline

- Identify which service components are most failure prone and have highest TTR
- Discuss failure case studies
- Examine applicability of failure mitigation techniques
- Highlight need for
  - improved operator tools
  - collection of industry-wide failure data
  - creation of service level "performability" benchmarks

Services Surveyed

- Online – online service/internet portal
- Content – global content hosting service
- ReadMostly – mature readmostly internet service

Commonalities

- Physically housed in geographically distributed colocation facilities
- Use commodity hardware and networks
- Stateless front-end; stateful back-end
- Custom administrative software
- Load Balancing :
  - ReadMostly : DNS redirection
  - Online & Content : Client cooperation
- Frequent software/config updates/patches, 24x7 operation

service characteristic	Online	ReadMostly	Content
hits per day	~100 million	~100 million	~7 million
# of machines	~508, 2 sites	~2000, 4 sites	~500, ~15 sites
front-end node architecture	Solaris on SPARC and x86	open-source OS on x86	open-source OS on x86
back-end node architecture	Network Appliance filers	open-source OS on x86	open-source OS on x86
period of data studied	7 months	6 months	3 months
component failures	296	N/A	205
service failures	40	21	36

Table 1 : Differentiating characteristics of the services

Content

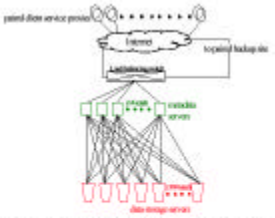


Figure 1: The architecture of one site of *Content*. Stateless metadata servers provide file metadata and route requests to the appropriate data storage servers. Persistent state is stored on commodity PC-based storage servers and is accessed via a custom protocol over UDP. Each cluster is connected to its twin site via the Internet.

Online

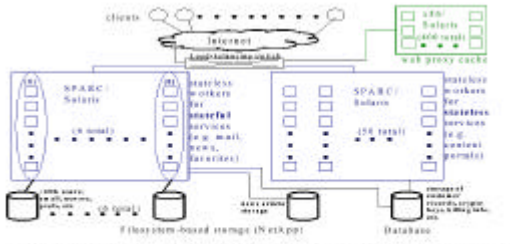


Figure 2: The architecture of one site of *Online*. Depending on the particular feature a user selects, the request is routed to any one of the web proxy cache servers, any one of 50 servers for stateless services, or any one of eight servers from a user's "service group" (a partition of one sixth of all users of the service, each with its own back-end data storage server). Persistent state is stored on Network Appliance servers and is accessed by worker nodes via NFS over UDP. This site is connected to a second site, at a colocation facility, via a leased network connection.

ReadMostly

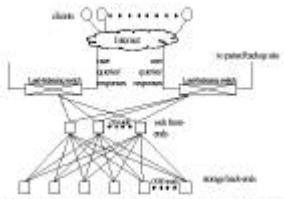
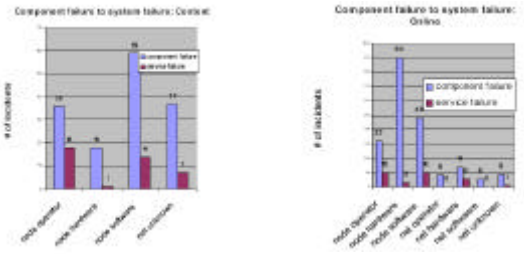


Figure 3: The architecture of one site of *ReadMostly*. A small number of web front-ends direct requests to the appropriate back-end storage servers. Persistent state is stored on commodity PC-based storage servers and is accessed via a custom protocol over TCP. A redundant pair of network switches connects the cluster to the Internet and to a twin site via a leased network connection.

Analysis of Failure Causes

1. Component failure to service failure



Analysis of Failure Causes

2. Service failure root cause

	Front-end	Back-end	Net-work	Un-known
Online	77%	3%	18%	2%
Content	60%	11%	18%	4%
Read-Mostly	6%	10%	81%	3%

Table 2: Failure cause by location

	Operator node	Operator net	H/W node	H/W net	S/W node	S/W net	Unknown node	Unknown net	Environment
Online	51%	2%	10%	15%	25%	2%	7%	3%	0%
Content	32%	4%	2%	2%	25%	6%	18%	12%	0%
Read-Mostly	5%	14%	0%	30%	5%	19%	0%	33%	0%

Table 3: Failure cause by component and type of cause

Analysis of Failure Causes

3. Service failure time to repair

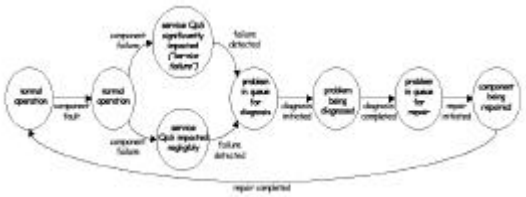
	Front-end	Back-end	Network
Online	9.4 (36)	7.3 (5)	7.8 (8)
Content	2.5 (10)	14 (3)	1.2 (2)
Read-Mostly	N/A (0)	0.2 (1)	1.2 (16)

Table 4: Average TTR by part of service, in hours

	Operator node	Operator net	H/W node	H/W net	S/W node	S/W net	Unknown node	Unknown net
Online	8.3 (16)	29 (1)	2.5 (5)	0.5 (1)	4.0 (9)	0.8 (1)	2.0 (1)	N/A (0)
Content	1.2 (8)	N/A (0)	N/A (0)	N/A (0)	0.2 (4)	N/A (0)	N/A (0)	1.2 (2)
Read-Mostly	0.2 (1)	8.1 (3)	N/A (0)	6.0 (2)	N/A (0)	1.0 (4)	N/A (0)	0.1 (6)

Table 5: Average TTR by component and type of cause in hours

Timeline of a failure



Techniques for mitigating failures

- Correctness testing
- Redundancy
- Fault injection and load testing
- Configuration checking
- Component isolation
- Proactive restart
- Exposing/monitoring failures

Technique	System state or transition avoided/ mitigated	Instances potentially avoided/ mitigated
Online correctness testing	component failure	26
Expose/monitor failures	component being repaired	12
Expose/monitor failures	problem being diagnosed	11
Redundancy	service failure	9
Config. checking	component fault	9
Online fault/load injection	component failure	6
Component isolation	service failure	5
Pre-deployment fault/load injection	component fault	3
Proactive restart	component fail	3
Pre-deployment correctness testing	component fault	2

Table 6: Potential benefit from using proposed failure mitigation techniques in *Online*

Technique	Implementation cost	Potential reliability cost	Performance impact
Online-correct	medium to high	low to moderate	low to moderate
Expose/monitor	medium	low (false alarms)	low
Redundancy	low	low	very low
Online-fault/load	high	high	moderate to high
Config	medium	zero	zero
Isolation	moderate	low	moderate
Pre-fault/load	high	zero	zero
Restart	low	low	low
Pre-correct	medium to high	zero	zero

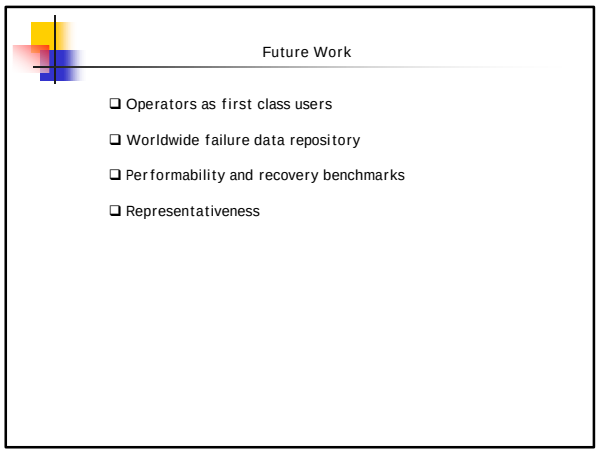
Table 7: Cost of implementing failure mitigation techniques

Failure case studies

- Operator error affecting front-end machines in Online
  lesson : sometimes replicating code and data is not enough; orthogonal redundancy such as alternate control path may help.
- Software error affecting front-end results in a cascading failure at back-end.
  lesson : Online testing could have caught this problem.
- Operator error affecting front-end machines compounded by software dropping error conditions silently
  lessons : - Software should not silently drop messages on errors
  - Operators need to understand dependencies and interactions between the various components of the service
- Problem at the interface between Online and an external service
  lesson : Online testing of configuration changes is a must
- Content's backup fiasco
  lessons : - Operator should know architectural dependencies
  - Masking small failures is not always good
  - Categorizing failures can be tricky

Conclusions

- Operator error is the largest cause of failures in two of the three services
- Operator error is the largest contributor to time to repair in two of the three services
- Configuration errors are the largest category of operator errors
- Failures in custom front-end software are significant
- More extensive online testing and more thoroughly exposing and detecting component failures would reduce failure rates in at least one service


Future Work
❑ Operators as first class users
❑ Worldwide failure data repository
❑ Performability and recovery benchmarks
❑ Representativeness