CS252
Graduate Computer Architecture

Lecture 7: I/O 3: a little Queueing Theory and I/O benchmarks

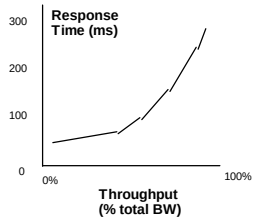
February 7, 2001
Prof. David A. Patterson
Computer Science 252
Spring 2001

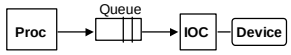
Summary: Dependability

- Fault => Latent errors in system => Failure in service
- Reliability: quantitative measure of time to failure (MTTF)
  - Assuming expoentially distributed independent failures, can calculate MTTF system from MTTF of components
- Availability: quantitative measure % of time delivering desired service
- Can improve Availability via greater MTTF or smaller MTTR (such as using standby spares)
- No single point of failure a good hardware guideline, as everything can fail
- Components often fail slowly
- Real systems: problems in maintenance, operation as well as hardware, software

Review: Disk I/O Performance

Metrics:
Response Time
Throughput



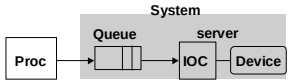


Response time = Queue + Device Service time

Introduction to Queueing Theory

- More interested in long term, steady state than in startup => Arrivals = Departures
- Little's Law:
  Mean number tasks in system = arrival rate x mean reponse time
  - Observed by many, Little was first to prove
- Applies to any system in equilibrium, as long as nothing in black box is creating or destroying tasks

A Little Queuing Theory: Notation

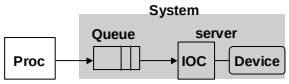


- Queuing models assume state of equilibrium: input rate = output rate
- Notation:

r	average number of arriving customers/second
T_{ser}	average time to service a customer (tradtionally $\mu = 1/T_{ser}$)
u	server utilization (0..1): $u = r \times T_{ser}$ (or $u = r / T_{ser}$)
T_q	average time/customer in queue
T_{sys}	average time/customer in system: $T_{sys} = T_q + T_{ser}$
L_q	average length of queue: $L_q = r \times T_q$
L_{sys}	average length of system: $L_{sys} = r \times T_{sys}$

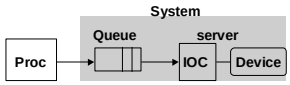
- Little's Law: $Length_{server} = rate \times Time_{server}$
  (Mean number customers = arrival rate x mean service time)

A Little Queuing Theory



- Service time completions vs. waiting time for a busy server: randomly arriving event joins a queue of arbitrary length when server is busy, otherwise serviced immediately
  - Unlimited length queues key simplification
- A *single server queue*: combination of a servicing facility that accomodates 1 customer at a time (*server*) + waiting area (*queue*): together called a *system*
- Server spends a variable amount of time with customers; *how do you characterize variability?*
  - Distribution of a random variable: histogram? curve?

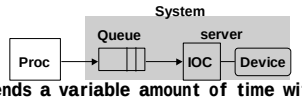
A Little Queuing Theory



- Server spends variable amount of time with customers
  - Weighted mean $m1$ = (f1 x T1 + f2 x T2 +...+ fn x Tn)/F (F=f1 + f2...)
  - $variance$ = (f1 x $T1^2$ + f2 x $T2^2$ +...+ fn x Tn^2)/F – $m1^2$
    - » Must keep track of unit of measure (100 ms^2 vs. 0.1 s^2)
  - *Squared coefficient of variance*: $C^2 = variance/m1^2$
    - » Unitless measure (100 ms^2 vs. 0.1 s^2)

- Exponential distribution $C^2 = 1$: most short relative to average, few others long; 90% < 2.3 x average, 63% < average
- Hypoexponential distribution $C^2 < 1$: most close to average, C^2=0.5 => 90% < 2.0 x average, only 57% < average
- Hyperexponential distribution $C^2 > 1$: further from average C^2 =2.0 => 90% < 2.8 x average, 69% < average

A Little Queuing Theory: Variable Service Time



- Server spends a variable amount of time with customers
  - Weighted mean $m1$ = (f1xT1 + f2xT2 +...+ fnXTn)/F (F=f1+f2+...)
- Usually pick $C = 1.0$ for simplicity
- Another useful value is average time must wait for server to complete task: $m1(z)$
  - Not just 1/2 x m1 because doesn't capture variance
  - Can derive $m1(z) = 1/2 \times m1 \times (1 + C^2)$
  - *No variance* => $C^2 = 0$ => $m1(z) = 1/2 \times m1$

A Little Queuing Theory: Average Wait Time

- Calculating average wait time in queue T_q
  - If something at server, it takes to complete on average $m1(z)$
  - Chance server is busy = u; average delay is u x $m1(z)$
  - All customers in line must complete; each avg T_{ser}

$T_q = u \times m1(z) + L_q \times T_{ser} = 1/2 \times u \times T_{ser} \times (1 + C) + L_q \times T_{ser}$
$T_q = 1/2 \times u \times T_{ser} \times (1 + C) + r \times T_q \times T_{ser}$
$T_q = 1/2 \times u \times T_{ser} \times (1 + C) + u \times T_q$
$T_q \times (1 - u) = T_{ser} \times u \times (1 + C) /2$
$T_q = T_{ser} \times u \times (1 + C) / (2 \times (1 - u))$

- Notation:

r	average number of arriving customers/second
T_{ser}	average time to service a customer
u	server utilization (0..1): $u = r \times T_{ser}$
T_q	average time/customer in queue
L_q	average length of queue:$L_q = r \times T_q$

A Little Queuing Theory: M/G/1 and M/M/1

- Assumptions so far:
  - System in equilibrium, number sources of requests unlimited
  - Time between two successive arrivals in line are exponentially distrib.
  - Server can start on next customer immediately after prior finishes
  - No limit to the queue: works First-In-First-Out "discipline"
  - Afterward, all customers in line must complete; each avg T_{ser}
- Described "memoryless" or Markovian request arrival (M for C=1 exponentially random), General service distribution (no restrictions), 1 server: *M/G/1 queue*
- When Service times have C = 1, *M/M/1 queue*
  $T_q = T_{ser} \times u \times (1 + C) /(2 \times (1 - u)) = T_{ser} \times u / (1 - u)$

T_{ser}	average time to service a customer
u	server utilization (0..1): $u = r \times T_{ser}$
T_q	average time/customer in queue

A Little Queuing Theory: An Example

- processor sends 10 x 8KB disk I/Os per second, requests & service exponentially distrib., avg. disk service = 20 ms
- On average, how utilized is the disk?
  - What is the number of requests in the queue?
  - What is the average time spent in the queue?
  - What is the average response time for a disk request?
- Notation:

r	average number of arriving customers/second = 10
T_{ser}	average time to service a customer = 20 ms (0.02s)
u	server utilization (0..1): $u = r \times T_{ser}$= 10/s x .02s = 0.2
T_q	average time/customer in queue = $T_{ser} \times u / (1 - u)$ = 20 x 0.2/(1-0.2) = 20 x 0.25 = 5 ms (0 .005s)
T_{sys}	average time/customer in system: $T_{sys} = T_q + T_{ser}$= 25 ms
L_q	average length of queue:$L_q = r \times T_q$ = 10/s x .005s = 0.05 requests in queue
L_{sys}	average # tasks in system: $L_{sys} = r \times T_{sys}$ = 10/s x .025s = 0.25

A Little Queuing Theory: Another Example

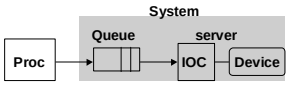
- processor sends 20 x 8KB disk I/Os per sec, requests & service exponentially distrib., avg. disk service = 12 ms
- On average, how utilized is the disk?
  - What is the number of requests in the queue?
  - What is the average time a spent in the queue?
  - What is the average response time for a disk request?
- Notation:

r	average number of arriving customers/second= 20
T_{ser}	average time to service a customer= 12 ms
u	server utilization (0..1): $u = r \times T_{ser}$= ___/s x .___s =
T_q	average time/customer in queue = $T_{ser} \times u / (1 - u)$ = ___ x ___/(____) = ___ x _____ = _____ ms
T_{sys}	average time/customer in system: $T_{sys} = T_q + T_{ser}$= 16 ms
L_q	average length of queue:$L_q = r \times T_q$ = ___/s x _____s = _____ requests in queue
L_{sys}	average # tasks in system : $L_{sys} = r \times T_{sys}$ = _____/s x ___s =

Pitfall of Not using Queuing Theory

- 1st 32-bit minicomputer (VAX-11/780)
- How big should write buffer be?
  - Stores 10% of instructions, 1 MIPS
- Buffer = 1
- => Avg. Queue Length = 1 vs. low response time

Summary: A Little Queuing Theory



- Queuing models assume state of equilibrium: input rate = output rate
- Notation:
  - r average number of arriving customers/second
  - T_{ser} average time to service a customer (tradtionally $\mu = 1/T_{ser}$)
  - u server utilization (0..1): $u = r \times T_{ser}$
  - T_q average time/customer in queue
  - T_{sys} average time/customer in system: $T_{sys} = T_q + T_{ser}$
  - L_q average length of queue: $L_q = r \times T_q$
  - L_{sys} average length of system : $L_{sys} = r \times T_{sys}$
- Little's Law: $Length_{system} = rate \times Time_{system}$ (Mean number customers = arrival rate x mean service time)

I/O Benchmarks

- For better or worse, benchmarks shape a field
  - Processor benchmarks classically aimed at response time for fixed sized problem
  - I/O benchmarks typically measure throughput, possibly with upper limit on response times (or 90% of response times)
- What if fix problem size, given 60%/year increase in DRAM capacity?

Benchmark	*Size of Data*	*% Time I/O*	*Year*
I/OStones	1 MB	26%	1990
Andrew	4.5 MB	4%	1988

  - Not much time in I/O
  - Not measuring disk (or even main memory)

I/O Benchmarks: Transaction Processing

- Transaction Processing (TP) (or On-line TP=OLTP)
  - Changes to a large body of shared information from many terminals, with the TP system guaranteeing proper behavior on a failure
  - If a bank's computer fails when a customer withdraws money, the TP system would guarantee that the account is debited if the customer received the money and that the account is unchanged if the money was not received
  - Airline reservation systems & banks use TP
- Atomic transactions makes this work
- Each transaction => 2 to 10 disk I/Os & 5,000 and 20,000 CPU instructions per disk I/O
  - Efficiency of TP SW & avoiding disks accesses by keeping information in main memory
- Classic metric is Transactions Per Second (TPS)
  - Under what workload? how machine configured?

I/O Benchmarks: Transaction Processing

- Early 1980s great interest in OLTP
  - Expecting demand for high TPS (e.g., ATM machines, credit cards)
  - Tandem's success implied medium range OLTP expands
  - Each vendor picked own conditions for TPS claims, report only CPU times with widely different I/O
  - Conflicting claims led to disbelief of all benchmarks=> chaos
- 1984 Jim Gray of Tandem distributed paper to Tandem employees and 19 in other industries to propose standard benchmark
- Published "A measure of transaction processing power," Datamation, 1985 by Anonymous et. al
  - To indicate that this was effort of large group
  - To avoid delays of legal department of each author's firm
  - Still get mail at Tandem to author

I/O Benchmarks: TP1 by Anon et. al

- DebitCredit Scalability: size of account, branch, teller, history function of throughput

TPS	Number of ATMs	Account-file size
10	1,000	0.1 GB
100	10,000	1.0 GB
1,000	100,000	10.0 GB
10,000	1,000,000	100.0 GB

  - Each input TPS =>100,000 account records, 10 branches, 100 ATMs
  - Accounts must grow since a person is not likely to use the bank more frequently just because the bank has a faster computer!
- Response time: 95% transactions take Š 1 second
- Configuration control: just report price (initial purchase price + 5 year maintenance = cost of ownership)
- By publishing, in public domain

I/O Benchmarks: TP1 by Anon et. al

- Problems
  - Often ignored the user network to terminals
  - Used transaction generator with no think time; made sense for database vendors, but not what customer would see
- Solution: Hire auditor to certify results
  - Auditors soon saw many variations of ways to trick system
- Proposed minimum compliance list (13 pages); still, DEC tried IBM test on different machine with poorer results than claimed by auditor
- Created Transaction Processing Performance Council in 1988: founders were CDC, DEC, ICL, Pyramid, Stratus, Sybase, Tandem, and Wang; ~40 companies today
- Led to TPC standard benchmarks in 1990,www.tpc.org

Unusual Characteristics of TPC

- Price is included in the benchmarks
  - cost of HW, SW, and 5-year maintenance agreements included => price-performance as well as performance
- The data set generally must scale in size as the throughput increases
  - trying to model real systems, demand on system and size of the data stored in it increase together
- The benchmark results are audited
  - Must be approved by certified TPC auditor, who enforces TPC rules => only fair results are submitted
- Throughput is the performance metric but response times are limited
  - eg, TPC-C: 90% transaction response times < 5 seconds
- An independent organization maintains the benchmarks
  - COO ballots on changes, meetings, to settle disputes...

TPC Benchmark History/Status

Benchmark	Data Size (GB)	Performance Metric	1st Results
A: Debit Credit (retired)	0.1 to 10	transactions/ second	Jul-90
B: Batch Debit Credit (retired)	0.1 to 10	transactions per second	Jul-91
C: Complex Query OLTP	100 to 3000 (min.07 * tpm)	new order trans/min.	Sep-92
D: Decision Support (retired)	100, 300, 1000	queries/hour	Dec-95
H: Ad hoc decision support	100, 300, 1000	queries/hour	Oct-99
R: Business reporting decision support	1000	queries/hour	Aug-99
W: Transactional web benchmark	~ 50, 500	web inter-actions/sec.	Jul-00

I/O Benchmarks: TPC-C Complex OLTP

- Models a wholesale supplier managing orders
- Order-entry conceptual model for benchmark
- Workload = 5 transaction types
- Users and database scale linearly with throughput
- Defines full-screen end-user interface
- Metrics: new-order rate (tpmC) and price/performance ($/tpmC)
- Approved July 1992

I/O Benchmarks: TPC-W Transactional Web Benchmark

- Represent any business (retail store, software distribution, airline reservation, ...) that markets and sells over the Internet/ Intranet
- Measure systems supporting users browsing, ordering, and conducting transaction oriented business activities.
- Security (including user authentication and data encryption) and dynamic page generation are important
- Before: processing of customer order by terminal operator working on LAN connected to database system
- Today: customer accesses company site over Internet connection, browses both static and dynamically generated Web pages, and searches the database for product or customer information. Customer also initiate, finalize & check on product orders & deliveries
- Started 1/97; hoped to release Fall, 1998?Jul 2000!

1998 TPC-C Performance tpm(c)

Rank	Config	tpmC	$/tpmC	Database
1	IBM RS/6000 SP (12 node x 8-way)	57,053.80	$147.40	Oracle8 8.0.4
2	HP HP 9000 V2250 (16-way)	52,117.80	$81.17	Sybase ASE
3	Sun Ultra E6000 c/s (2 node x 22-way)	51,871.62	$134.46	Oracle8 8.0.3
4	HP HP 9000 V2200 (16-way)	39,469.47	$94.18	Sybase ASE
5	Fujitsu GRANPOWER 7000 Model 800	34,116.93	$57,883.00	Oracle8
6	Sun Ultra E6000 c/s (24-way)	31,147.04	$108.90	Oracle8 8.0.3
7	Digital AlphaS8400 (4 node x 8-way)	30,390.00	$305.00	Oracle7 V7.3
8	SGI Origin2000 Server c/s (28-way)	25,309.20	$139.04	INFORMIX
9	IBM AS/400e Server (12-way)	25,149.75	$128.00	DB2
10	Digital AlphaS8400 5/625 (10-way)	24,537.00	$110.48	Sybase SQL

- Notes: 7 SMPs , 3 clusters of SMPs,
- avg 30 CPUs/system

1998 TPC-C Price/Performance $/tpm(c)

Rank		Config	$/tpmC	tpmC	Database
1	Acer	AcerAltos 19000Pro4	$27.25	11,072.07	M/S SQL 6.5
2	Dell	PowerEdge 6100 c/s	$29.55	10,984.07	M/S SQL 6.5
3	Compaq	ProLiant 5500 c/s	$33.37	10,526.90	M/S SQL 6.5
4	ALR	Revolution 6x6 c/s	$35.44	13,089.30	M/S SQL 6.5
5	HP	NetServer LX Pro	$35.82	10,505.97	M/S SQL 6.5
6	Fujitsu	teamserver M796i	$37.62	13,391.13	M/S SQL 6.5
7	Fujitsu	GRANPOWER 5000 Model 670	$37.62	13,391.13	M/S SQL 6.5
8	Unisys	Aquanta HS/6 c/s	$37.96	13,089.30	M/S SQL 6.5
9	Compaq	ProLiant 7000 c/s	$39.25	11,055.70	M/S SQL 6.5
10	Unisys	Aquanta HS/6 c/s	$39.39	12,026.07	M/S SQL 6.5

- Notes: all Microsoft SQL Server Database
- All uniprocessors?

2001 TPC-C Performance Results

Ra	Company	System	tpmC	$/tpmC	CPUs	Database Softw
1	Compaq	ProLiant 8500-700-192P	505,303	$19.80	192	SQL Server 200
2	IBM	Netfinity 8500R c/s	440,880	$32.28	128	DB2 UDB 7.1
3	Compaq	ProLiant 8500-X700-96P	262,244	$20.24	96	SQL Server 200
4	Compaq	ProLiant 8500-X550-96P	229,914	$23.08	96	SQL Server 200
5	Bull	Escala EPC2450	220,807	$43.31	24	Oracle 8i Enterp
6	IBM	IBM eServer pSeries 680	220,807	$43.30	24	Oracle 8i Enterp
7	HP	HP 9000 Superdome Ente	197,024	$66.27	48	Oracle8 Enterpr
8	Fujitsu	PRIMEPOWER 2000 c/s	183,771	$56.16	64	SymfoWARE S
9	Compaq	ProLiant 8500-X700-64P	179,658	$19.75	64	SQL Server 200
10	IBM	IBM eServer iSeries 840-2	163,776	$58.88	24	DB2 for AS/40

- Notes: 4 SMPs, 6 clusters of SMPs: 76 CPUs/system
- 3 years => Peak Performance 8.9X, 2X/yr

2001 TPC-C Price Performance Results

	Compan	System	tpmC	$/tpmC	CP	Database Softwar
1	Compaq	ProLiant ML-570-6/700-3P	20,207	$ 9.51	3	SQL Server 2000
2	Dell	PowerEdge 6450/3P	24,925	$ 9.90	3	SQL Server 2000
3	Dell	PowerEdge 6400/3P	24,925	$ 9.91	3	SQL Server 2000
4	Dell	PowerEdge 6400	30,231	$ 11.07	4	SQL Server 2000
5	Dell	PowerEdge 6450	30,231	$ 11.08	4	SQL Server 2000
6	HP	NetServer LH 6000	33,136	$ 11.85	6	SQL Server Enter
7	Compaq	ProLiant ML-570-6/700	32,328	$ 12.49	4	SQL Server 2000
8	HP	HP NetServer LXr 8500	43,047	$ 12.76	8	SQL Server 2000
9	Compaq	ProLiant 8500-6/700-4	34,600	$ 12.89	4	SQL Server 2000
10	Compaq	ProLiant 8500-550-6P	33,617	$ 12.91	6	SQL Server Enter

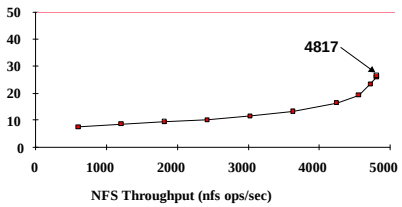
- Notes: All small SMPs, all running M/S SQL server
- 3 years => Cost Performance 2.9X, 1.4X/yr

SPEC SFS/LADDIS

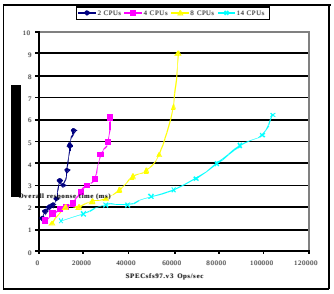
- 1993 Attempt by NFS companies to agree on standard benchmark: Legato, Auspex, Data General, DEC, Interphase, Sun. Like NFSstones but
  - Run on multiple clients & networks (to prevent bottlenecks)
  - Same caching policy in all clients
  - Reads: 85% full block & 15% partial blocks
  - Writes: 50% full block & 50% partial blocks
  - Average response time: 50 ms
  - Scaling: for every 100 NFS ops/sec, increase capacity 1GB
  - Results: plot of server load (throughput) vs. response time & number of users
    - » Assumes: 1 user => 10 NFS ops/sec

1998 Example SPEC SFS Result: DEC Alpha

- 200 MHz 21064: 8KI + 8KD + 2MB L2; 512 MB; 1 Gigaswitch
- DEC OSF1 v2.0
- 4 FDDI networks; 32 NFS Daemons, 24 GB file size
- 88 Disks, 16 controllers, 84 file systems



SPEC sfs97 for EMC Celera NFS servers: 2, 4, 8, 14 CPUs; 67, 133, 265, 433 disks 15,700, 32,000, 61,800 104,600 ops/sec



SPEC WEB99

- Simulates accesses to web service provider, supports home pages for several organizations. File sizes:
  - less than 1 KB, representing an small icon: 35% of activity
  - 1 to 10 KB: 50% of activity
  - 10 to 100 KB: 14% of activity
  - 100 KB to 1 MB: a large document and image,1% of activity
- Workload simulates dynamic operations: rotating advertisements on a web page, customized web page creation, and user registration.
- workload gradually increased until server software is saturated with hits and response time degrades significantly.

SPEC WEB99 for Dells in 2000

System Name	Resul	HTTP Version/OS	CPU	CPU typ	DRAM
PowerEdge 2400/667	732	IIS 5.0/Windows 2000	1	667 MH	2 GB
PowerEdge 2400/667	1270	TUX 1.0/Red Hat Linux	1	667 MH	2 GB
PowerEdge 4400/800	1060	IIS 5.0/Windows 2000	2	800 MH	4 GB
PowerEdge 4400/800	2200	TUX 1.0/Red Hat Linux	2	800 MH	4 GB
PowerEdge 6400/700	1598	IIS 5.0/Windows 2000	4	700 MH	8 GB
PowerEdge 6400/700	4200	TUX 1.0/Red Hat Linux	4	700 MH	8 GB

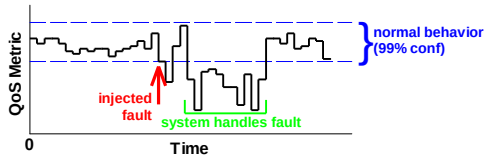
- Each uses 5 9GB, 10,000 RPM disks except the 5th system, which had 7 disks, and the first 4 have 0.25 MB of L2 cache while the last 2 have 2 MB of L2 cache
- Appears that the large amount of DRAM is used as a large file cache to reduce disk I/O, so not really an I/O benchmark

Availability benchmark methodology

- Goal: quantify variation in QoS metrics as events occur that affect system availability
- Leverage existing performance benchmarks
  - to generate fair workloads
  - to measure & trace quality of service metrics
- Use fault injection to compromise system
  - hardware faults (disk, memory, network, power)
  - software faults (corrupt input, driver error returns)
  - maintenance events (repairs, SW/HW upgrades)
- Examine *single-fault* and *multi-fault* workloads
  - the availability analogues of performance micro- and macro-benchmarks

Benchmark Availability? Methodology for reporting results

- Results are most accessible graphically
  - plot change in QoS metrics over time
  - compare to "normal" behavior
    - » 99% confidence intervals calculated from no-fault runs



Case study

- Availability of software RAID-5 & web server
  - Linux/Apache, Solaris/Apache, Windows 2000/IIS
- Why software RAID?
  - well-defined availability guarantees
    - » RAID-5 volume should tolerate a single disk failure
    - » reduced performance (degraded mode) after failure
    - » may automatically rebuild redundancy onto spare disk
  - simple system
  - easy to inject storage faults
- Why web server?
  - an application with measurable QoS metrics that depend on RAID availability and performance

Benchmark environment: faults

- Focus on faults in the storage system (disks)
- *Emulated disk* provides reproducible faults
  - a PC that appears as a disk on the SCSI bus
  - I/O requests intercepted and reflected to local disk
  - fault injection performed by altering SCSI command processing in the emulation software
- Fault set chosen to match faults observed in a long-term study of a large storage array
  - media errors, hardware errors, parity errors, power failures, disk hangs/timeouts
  - both transient and "sticky" faults

Single-fault experiments

- "Micro-benchmarks"
- Selected 15 fault types
  - 8 benign (retry required)
  - 2 serious (permanently unrecoverable)
  - 5 pathological (power failures and complete hangs)
- An experiment for each type of fault
  - only one fault injected per experiment
  - no human intervention
  - system allowed to continue until stabilized or crashed

Multiple-fault experiments

- "Macro-benchmarks" that require human intervention
- Scenario 1: reconstruction
  (1) disk fails
  (2) data is reconstructed onto spare
  (3) spare fails
  (4) administrator replaces both failed disks
  (5) data is reconstructed onto new disks
- Scenario 2: double failure
  (1) disk fails
  (2) reconstruction starts
  (3) administrator accidentally removes active disk
  (4) administrator tries to repair damage

Comparison of systems

- Benchmarks revealed significant variation in failure-handling policy across the 3 systems
  - transient error handling
  - reconstruction policy
  - double-fault handling
- Most of these policies were undocumented
  - yet they are critical to understanding the systems' availability

Transient error handling

- Transient errors are common in large arrays
  - example: Berkeley 368-disk Tertiary Disk array, 11mo.
    - » 368 disks reported transient SCSI errors (100%)
    - » 13 disks reported transient hardware errors (3.5%)
    - » 2 disk failures (0.5%)
  - isolated transients do not imply disk failures
  - but streams of transients indicate failing disks
    - » both Tertiary Disk failures showed this behavior
- Transient error handling policy is critical in long-term availability of array

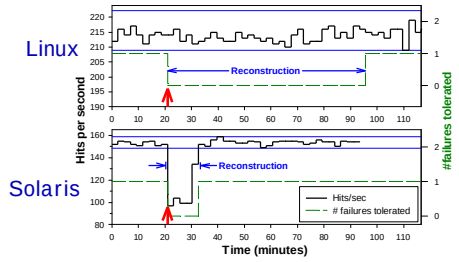
Transient error handling (2)

- Linux is *paranoid* with respect to transients
  - stops using affected disk (and reconstructs) on *any* error, transient or not
    - » fragile: system is more vulnerable to multiple faults
    - » disk-inefficient: wastes two disks per transient
    - » but no chance of slowly-failing disk impacting perf.
- Solaris and Windows are more forgiving
  - both ignore most benign/transient faults
    - » robust: less likely to lose data, more disk-efficient
    - » less likely to catch slowly-failing disks and remove them
- Neither policy is ideal!
  - need a hybrid that detects streams of transients

Reconstruction policy

- Reconstruction policy involves an availability tradeoff between performance & redundancy
  - until reconstruction completes, array is vulnerable to second fault
  - disk and CPU bandwidth dedicated to reconstruction is not available to application
    - » but reconstruction bandwidth determines reconstruction speed
  - policy must trade off *performance availability* and *potential data availability*

Example single-fault result



- Compares Linux and Solaris reconstruction
  - Linux: **minimal performance impact but longer window of vulnerability to second fault**
  - Solaris: **large perf. impact but restores redundancy fast**

Reconstruction policy (2)

- **Linux:** favors performance over data availability
  - **automatically-initiated reconstruction, idle bandwidth**
  - **virtually no performance impact on application**
  - **very long window of vulnerability (>1hr for 3GB RAID)**
- **Solaris:** favors data availability over app. perf.
  - **automatically-initiated reconstruction at high BW**
  - **as much as 34% drop in application performance**
  - **short window of vulnerability (10 minutes for 3GB)**
- **Windows:** favors neither!
  - ***manually-initiated* reconstruction at moderate BW**
  - **as much as 18% app. performance drop**
  - **somewhat short window of vulnerability (23 min/3GB)**

Double-fault handling

- **A double fault results in unrecoverable loss of some data on the RAID volume**
- **Linux:** blocked access to volume
- **Windows:** blocked access to volume
- **Solaris:** silently continued using volume, delivering *fabricated* data to application!
  - **clear violation of RAID availability semantics**
  - **resulted in corrupted file system and garbage data at the application level**
  - **this *undocumented* policy has serious availability implications for applications**

Availability Conclusions: Case study

- **RAID vendors should expose and document policies affecting availability**
  - **ideally should be user-adjustable**
- **Availability benchmarks can provide valuable insight into availability behavior of systems**
  - **reveal undocumented availability policies**
  - **illustrate impact of specific faults on system behavior**
- **We believe our approach can be generalized well beyond RAID and storage systems**
  - **the RAID case study is based on a general methodology**

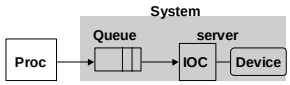
Conclusions: Availability benchmarks

- **Our methodology is best for *understanding* the availability behavior of a system**
  - **extensions are needed to distill results for automated system comparison**
- **A good fault-injection environment is critical**
  - **need realistic, reproducible, controlled faults**
  - **system designers should consider building in hooks for fault-injection and availability testing**
- **Measuring and understanding availability will be crucial in building systems that meet the needs of modern server applications**
  - **our benchmarking methodology is just the first step towards this important goal**

Summary: I/O Benchmarks

- **Scaling to track technological change**
- **TPC: price performance as nomalizing configuration feature**
- **Auditing to ensure no foul play**
- **Throughput with restricted response time is normal measure**
- **Benchmarks to measure Availability, Maintainability?**

Summary: A Little Queuing Theory



- Queuing models assume state of equilibrium: input rate = output rate
- Notation:
  - r average number of arriving customers/second
  - T_{ser} average time to service a customer (tradtionally $\mu = 1/ T_{ser}$)
  - u server utilization (0..1): $u = r \times T_{ser}$
  - T_q average time/customer in queue
  - T_{sys} average time/customer in system: $T_{sys} = T_q + T_{ser}$
  - L_q average length of queue: $L_q = r \times T_q$
  - L_{sys} average length of system : $L_{sys} = r \times T_{sys}$
- Little's Law: $Length_{system} = rate \times Time_{system}$ (Mean number customers = arrival rate x mean service time)