
EEC 581 Computer Architecture

Lec 14 – Storage (Ch. 6)

Chansu Yu
Electrical and Computer Engineering
Cleveland State University

Acknowledgement ...

- **Part of class notes are from**
 - David Patterson
 - Electrical Engineering and Computer Sciences
 - University of California, Berkeley
 - <http://www.eecs.berkeley.edu/~pattsrn>
 - <http://www-inst.eecs.berkeley.edu/~cs252>

Case for Storage

- **Shift in focus from computation to communication and storage of information**
 - E.g., Cray Research/Thinking Machines vs. Google/Yahoo
 - “The Computing Revolution” (1960s to 1980s)
 - ⇒ “The Information Age” (1990 to today)
- **Storage emphasizes reliability and scalability as well as cost-performance**
- **What is “Software king” that determines which HW actually features used?**
 - Operating System for storage
 - Compiler for processor
- **Also has own performance theory—queuing theory—balances throughput vs. response time**

11/29/2007

3

Outline

- **Magnetic Disks**
- **RAID (6.2)**
- **I/O Benchmarks, Performance and Dependability (6.4)**
- **Intro to Queueing Theory (6.5)**
- **Conclusion**

11/29/2007

4

Future Disk Size and Performance

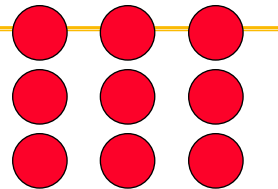
- Continued advance in capacity (60%/yr) and bandwidth (40%/yr)
- Slow improvement in seek, rotation (8%/yr)
- Time to read whole disk

Year	Sequentially	Randomly (1 sector/seek)
1990	4 minutes	6 hours
2000	12 minutes	1 week(!)
2006	56 minutes	3 weeks (SCSI)
2006	171 minutes	7 weeks (SATA)

11/29/2007

7

Disk Arrays



- **A new organization of disk storage:**
 - Arrays of small and inexpensive disks
 - Increase potential throughput by having many disk drives:
 - » Data is spread over multiple disk
 - » Multiple accesses are made to several disks
- **Reliability is lower than a single disk:**
 - But availability can be improved by adding redundant disks (RAID):
Lost information can be reconstructed from redundant information
 - MTTR: mean time to repair is in the order of hours
 - MTTF: mean time to failure of disks is tens of years

11/29/2007

8

Reliability and Availability

- **Two terms that are often confused:**
 - Reliability: Is anything broken?
 - Availability: Is the system still available to the user?
- **Availability can be improved by adding hardware:**
 - Example: adding ECC on memory
- **Reliability can only be improved by:**
 - Bettering environmental conditions
 - Building more reliable components
 - Building with fewer components
 - » Improve availability may come at the cost of lower reliability

11/29/2007

9

Array Reliability

- Reliability of N disks = Reliability of 1 Disk \div N
 - 50,000 Hours \div 70 disks = 700 hours
 - Disk system MTTF: Drops from 6 years to 1 month!
- Arrays (without redundancy) too unreliable to be useful!

Hot spares support reconstruction in parallel with access: very high media availability can be achieved

11/29/2007

10

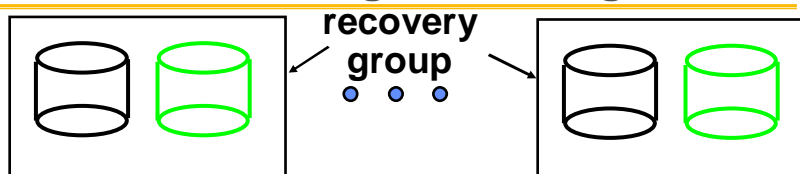
Redundant Arrays of (Inexpensive) Disks

- Files are "striped" across multiple disks
- Redundancy yields high data availability
 - Availability: service still provided to user, even if some components failed
- Disks will still fail
- Contents reconstructed from data redundantly stored in the array
 - ⇒ Capacity penalty to store redundant info
 - ⇒ Bandwidth penalty to update redundant info

11/29/2007

11

Redundant Arrays of Inexpensive Disks RAID 1: Disk Mirroring/Shadowing

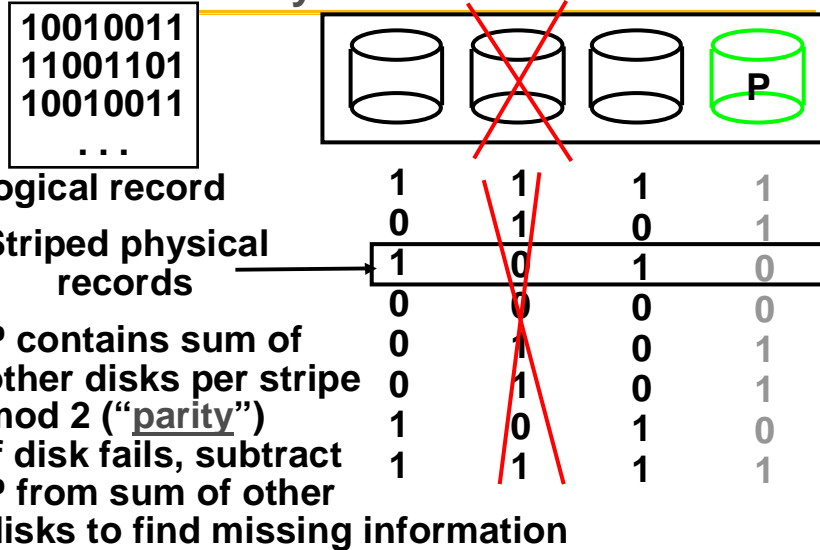


- Each disk is fully duplicated onto its "mirror"
Very high availability can be achieved
- Bandwidth sacrifice on write:
Logical write = two physical writes
 - Reads may be optimized
- Most expensive solution: 100% capacity overhead
- (RAID 2 not interesting, so skip)

11/29/2007

12

Redundant Array of Inexpensive Disks RAID 3: Parity Disk



11/29/2007

13

RAID 3

- Sum computed across recovery group to protect against hard disk failures, stored in P disk
- Logically, a single high capacity, high transfer rate disk: good for large transfers
- Wider arrays reduce capacity costs, but decreases availability
- 33% capacity cost for parity if 3 data disks and 1 parity disk

11/29/2007

14

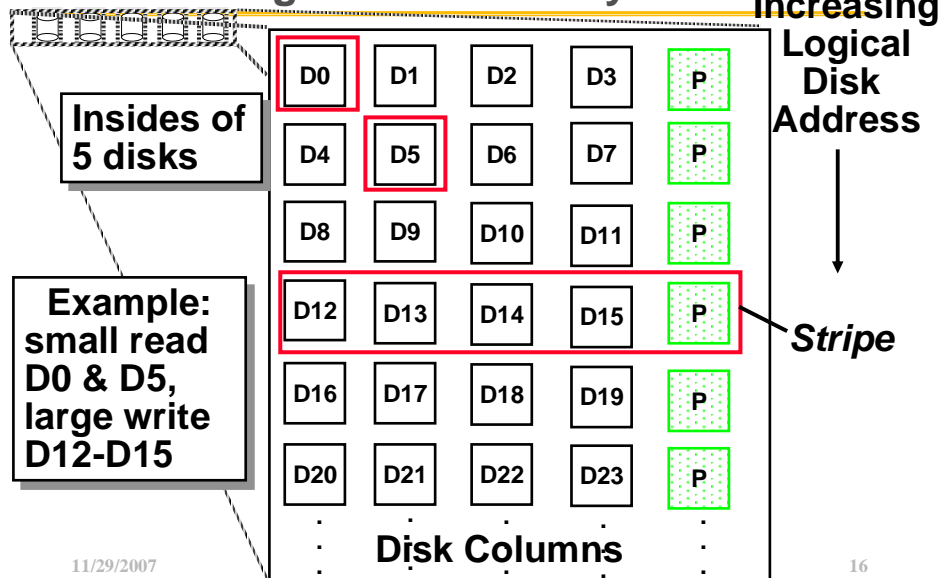
Inspiration for RAID 4

- RAID 3 relies on parity disk to discover errors on Read
- But every sector has an error detection field
- To catch errors on read, rely on error detection field vs. the parity disk
- Allows independent reads to different disks simultaneously

11/29/2007

15

Redundant Arrays of Inexpensive Disks RAID 4: High I/O Rate Parity

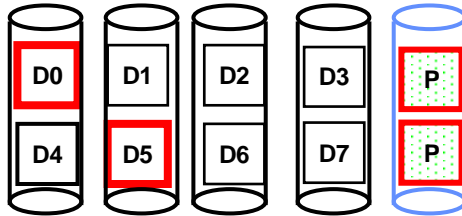


11/29/2007

16

Inspiration for RAID 5

- RAID 4 works well for small reads
- Small writes (write to one disk):
 - Option 1: read other data disks, create new sum and write to Parity Disk
 - Option 2: since P has old sum, compare old data to new data, add the difference to P
- Small writes are limited by Parity Disk: Write to D0, D5 both also write to P disk



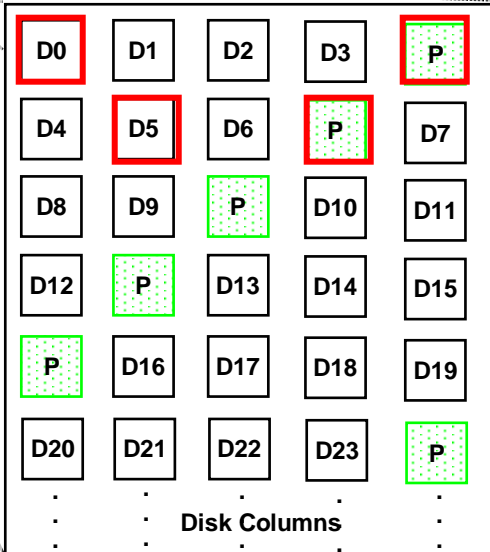
11/29/2007

17

Redundant Arrays of Inexpensive Disks RAID 5: High I/O Rate Interleaved Parity

Independent writes possible because of interleaved parity

Example: write to D0, D5 uses disks 0, 1, 3, 4



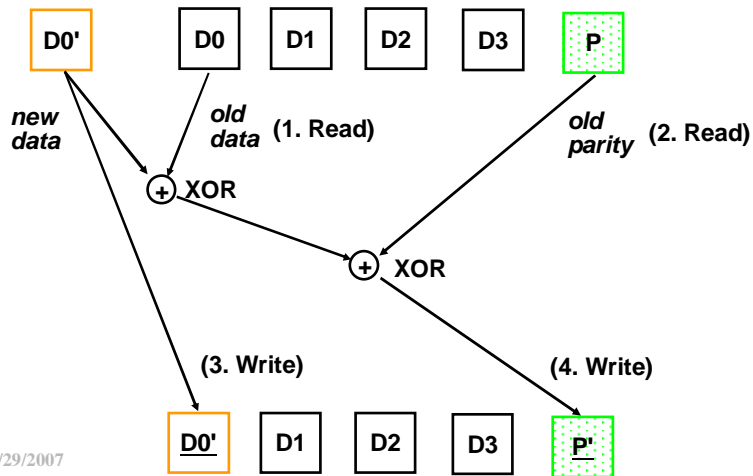
11/29/2007

18

Problems of Disk Arrays: Small Writes

RAID-5: Small Write Algorithm

1 Logical Write = 2 Physical Reads + 2 Physical Writes



RAID 6: Recovering from 2 failures

- **Why > 1 failure recovery?**
 - operator accidentally replaces the wrong disk during a failure
 - since disk bandwidth is growing more slowly than disk capacity, the MTT Repair a disk in a RAID system is increasing
 - ⇒ increases the chances of a 2nd failure during repair since takes longer
 - reading much more data during reconstruction meant increasing the chance of an uncorrectable media failure, which would result in data loss

RAID 6: Recovering from 2 failures

- Example: B0, B1, B2, B3, B4 (Parity block)
- If disk 1 breaks, $B1 = B0 \oplus B2 \oplus B3 \oplus B4$
- If disk 1 & 3 break, $B1=?$, $B3=?$
 - How about $B5 = B0 \oplus B2 \oplus B3 \oplus B4$ (B1 missing)?
 - B3 can be constructed by $B0 \oplus B2 \oplus B4 \oplus B5$
 - And, B1 can be constructed by $B0 \oplus B2 \oplus B3 \oplus B4$
- Is this OK?

11/29/2007

21

RAID 6: Recovering from 2 failures

- *Row-diagonal parity or RAID-DP*
- Row parity disk is just like in RAID 4
 - Even parity across the other 4 data blocks in its stripe
 - $4 + 1 = 5$ disks
- Diagonal parity disk (6th disk)
 - Each block of the DP disk contains the even parity of the blocks in the same diagonal

11/29/2007

22

Example $p = 5$

- Row diagonal parity starts by recovering one of the 4 blocks on the failed disk using diagonal parity
 - Since each diagonal misses one disk, and all diagonals miss a different disk, 2 diagonals are only missing 1 block
- Once the data for those blocks is recovered, then the standard RAID recovery scheme can be used to recover two more blocks in the standard RAID 4 stripes
- Process continues until two failed disks are restored

Data Disk 0	Data Disk 1	Data Disk 2	Data Disk 3	Row Parity	Diagonal Parity
0	1	2	3	4	0
1	2	3	4	0	1
2	3	4	0	1	2
3	4	0	1	2	3

Diagram illustrating a RAID 4 configuration with 5 disks (Data Disk 0, Data Disk 1, Data Disk 2, Data Disk 3, Row Parity, Diagonal Parity). The data is striped across the disks. The failed disks (Data Disk 1 and Data Disk 3) are marked with red X's. The data blocks on the failed disks are circled in blue, showing they are recovered using diagonal parity.

11/29/2007

23

Berkeley History: RAID-I

- **RAID-I (1989)**
 - Consisted of a Sun 4/280 workstation with 128 MB of DRAM, four dual-string SCSI controllers, 28 5.25-inch SCSI disks and specialized disk striping software
- Today RAID is \$24 billion dollar industry, 80% non-PC disks sold in RAIDs



11/29/2007

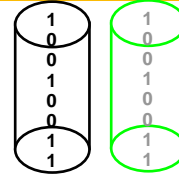
Summary: RAID Techniques: Goal was performance, popularity due to reliability of storage

- **Disk Mirroring, Shadowing (RAID 1)**

Each disk is fully duplicated onto its "shadow"

Logical write = two physical writes

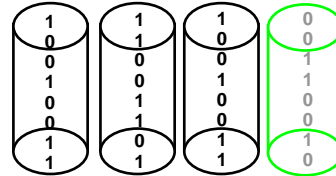
100% capacity overhead



- **Parity Data Bandwidth Array (RAID 3)**

Parity computed horizontally

Logically a single high data bw disk

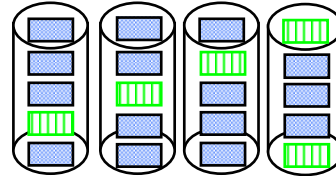


- **High I/O Rate Parity Array (RAID 5)**

Interleaved parity blocks

Independent reads and writes

Logical write = 2 reads + 2 writes



11/29/2007

Outline

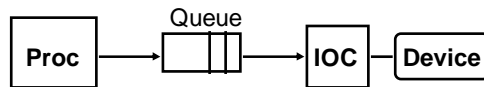
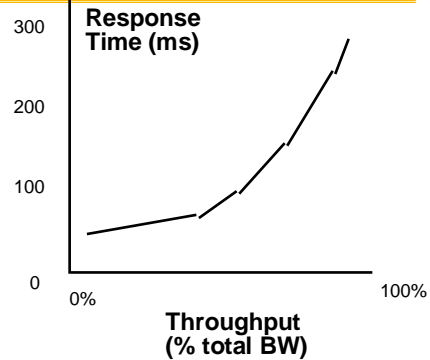
- Magnetic Disks
- RAID (6.2)
- I/O Benchmarks, Performance and Dependability (6.4)
- Intro to Queueing Theory (6.5)
- Conclusion

11/29/2007

26

I/O Performance

Metrics:
Response Time
vs. Throughput

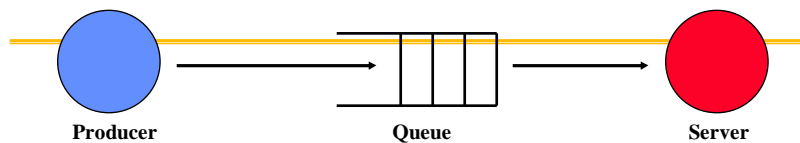


Response time = Queue + Device Service time

11/29/2007

27

Simple Producer-Server Model



- **Throughput:**

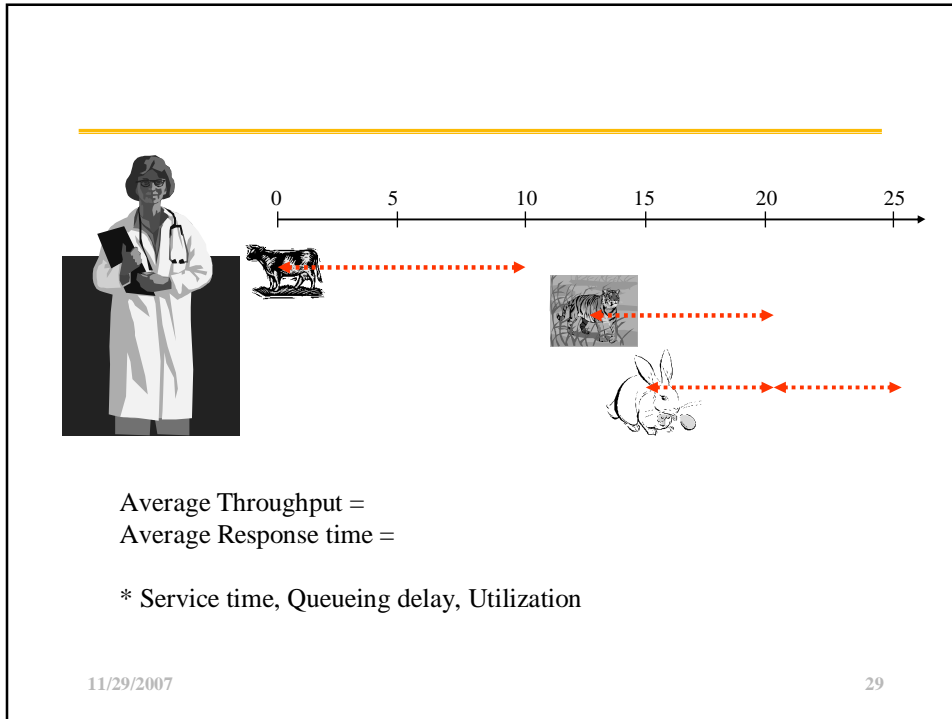
- The number of tasks completed by the server in unit time
- In order to get the highest possible throughput:
 - » The server should never be idle
 - » The queue should never be empty

- **Response time:**

- Begins when a task is placed in the queue
- Ends when it is completed by the server
- In order to minimize the response time:
 - » The queue should be empty
 - » The server will be idle

11/29/2007

28



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)
- **Transaction Processing (TP) (or On-line TP=OLTP)**
 - If bank computer fails when customer withdraw money, TP system guarantees account debited if customer gets \$ & account unchanged if no \$
 - Airline reservation systems & banks use TP
- **Atomic transactions makes this work**
- **Classic metric is Transactions Per Second (TPS)**

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 (Tandem) distributed paper to Tandem + 19 in other companies 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 "Anonymous"
- **Led to Transaction Processing Council in 1988**
 - www.tpc.org

11/29/2007

31

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 ≤ 1 second**
- **Report price (initial purchase price + 5 year maintenance = cost of ownership)**
- **Hire auditor to certify results**

11/29/2007

32

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**

11/29/2007 COO ballots on changes, meetings, to settle disputes...

33

TPC Benchmark History/Status

Benchmark	Data Size (GB)	Performance Metric	1st Results
A: Debit Credit (retired)	0.1 to 10	transactions/s	Jul-90
B: Batch Debit Credit (retired)	0.1 to 10	transactions/s	Jul-91
C: Complex Query OLTP	100 to 3000 (min. 07 * tpm)	new order trans/min (tpm)	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 (retired)	1000	queries/hour	Aug-99
W: Transactional web	~ 50, 500	web interactions/sec.	Jul-00
App: app. server & web services		Web Service Interactions/sec (SIPS)	Jun-05

11/29/2007

34

I/O Benchmarks via SPEC

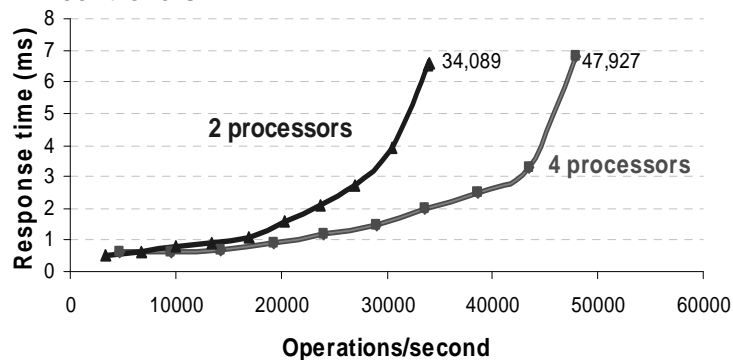
- **SFS 3.0 Attempt by companies to agree on standard benchmark to evaluate systems running NFS**
 - Run on multiple clients & networks (to prevent bottlenecks)
 - Same caching policy in all clients
 - Reads: 85% full block & 15% partial blocks
 - Writes: 50% full block & 50% partial blocks
 - Average response time: 40 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
 - 3.0 for NSF 3.0
- **Added SPECMail (mailserver), SPECWeb (webserver) benchmarks**

11/29/2007

35

2005 Example SPEC SFS Result: NetApp FAS3050c NFS servers

- 2.8 GHz Pentium Xeon microprocessors, 2 GB of DRAM per processor, 1GB of Non-volatile memory per system
- 4 FDDI networks; 32 NFS Daemons, 24 GB file size
- 168 fibre channel disks: 72 GB, 15000 RPM, 2 or 4 FC controllers



11/29/2007

36

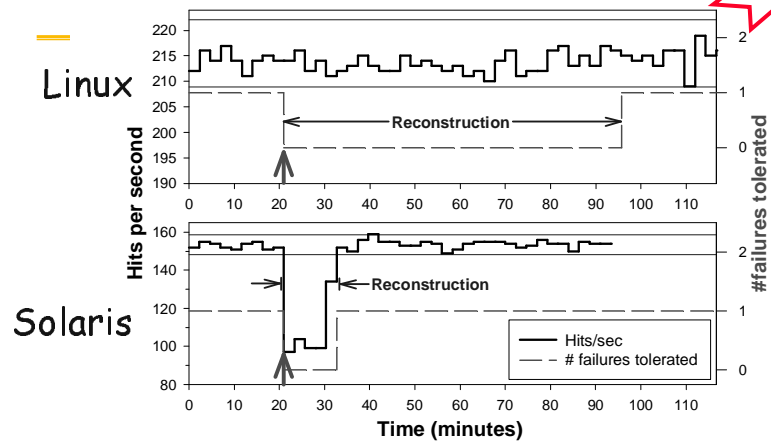
Availability benchmark methodology

- **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

11/29/2007

37

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**

11/29/2007

38

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)

11/29/2007

39

Outline

- Magnetic Disks
- RAID (6.2)
- I/O Benchmarks, Performance and Dependability (6.4)
- Intro to Queueing Theory (6.5)
- Conclusion

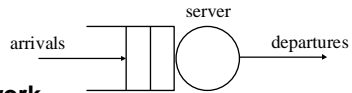
11/29/2007

40

Introduction to Queueing Theory

- **Queue: a useful abstraction of a shared resource**

- A cpu in your PC
- The memory bus in your PC
- An Ethernet line in a local area network
- Wireless medium in a wireless LAN
- An output port in a network router
- A disk in a file server
- A semaphore in a multithreaded (multi-process) program



- **A queueing system consists of**

- Customers arrive at unpredictable times
- Each customer has a service requirement
- Specified number of servers and scheduling discipline

11/29/2007

41

Introduction to Queueing Theory

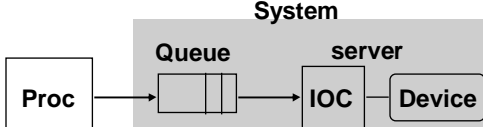


- **More interested in long term, steady state than in startup => Arrivals = Departures**
- **Little's Law:**
Mean number tasks in system = arrival rate x mean response time ($N=\lambda T$)
 - Observed by many, Little was first to prove
- **Applies to any system in equilibrium, as long as black box not creating or destroying tasks**

11/29/2007

42

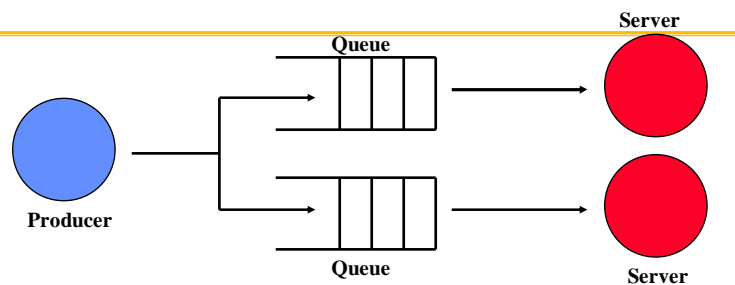
A Little Queuing Theory: Notation

- 
- **Notation:**
 - $Time_{server}$ average time to service a task
 - Average service rate = $1 / Time_{server}$ (traditionally μ)
 - $Time_{queue}$ average time/task in queue
 - $Time_{system}$ average time/task in system
 $= Time_{queue} + Time_{server}$
 - Arrival rate avg no. of arriving tasks/sec (traditionally λ)
 - $Length_{server}$ average number of tasks in service
 - $Length_{queue}$ average length of queue
 - $Length_{system}$ average number of tasks in system
 $= Length_{queue} + Length_{server}$
 - Little's Law:** $Length_{server} = Arrival\ rate \times Time_{server}$
 (Mean number tasks = arrival rate x mean service time)

11/29/2007

43

Throughput Enhancement



- In general throughput can be improved by:
 - Throwing more hardware at the problem
 - reduces load-related latency
- Response time is much harder to reduce:
 - Ultimately it is limited by the speed of light (but we're far from it)

11/29/2007

44

Review: Stochastic Process

- **Key components are**
 - Arrival process
 - Departure process
- **Stochastic process: A stochastic process is a family of random variables $\{X(t)\}$ that is also a function of time t .**

11/29/2007

45

Markov Process

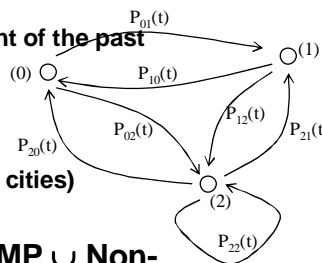
- **Stochastic process = Markov process \cup Non-Markov process**

- **Markov process**

- Future states of a process are independent of the past
- Thus, can draw the state transition graph

- **Non-Markov Process**

- Future states are dependent of the past
(think of a traveling salesman visiting 3 cities)



- **Markov process = Homogeneous MP \cup Non-Homogeneous MP**

- **Homogeneous MP**

- Transition probabilities are not functions of time ($P_{ij}(t) = P_{ij}$)

11/29/2007

46

Markov Process

- **Markov process (1907, A. A. Markov)**
 - For convenience, we will look at discrete-time, discrete-state stochastic processes $\{X_n\}$, where n is the discrete time step.
 - A set of random variables $\{X_n\}$ forms a Markov process if $P[X_{n+1} = a]$ depends only on the current state X_n and not upon any previous values.
 - I.e., $P[X_{n+1}=a \mid X_n=b, X_{n-1}=c, \dots, X_0=z] = P[X_{n+1}=a \mid X_n=b]$
 - How long the process remains in its current state before making a transition to some other states? Can it be random?

11/29/2007

47

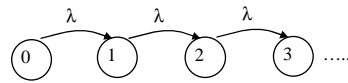
Markov Process

- **So, what is Markov ?**
 - Future state is independent of the past
 - => Future transition time is independent on how long it has been in the current state
 - => “How long will it stay in the current state” is memoryless
 - => “Time between transitions (τ)” is memoryless
 - => “New arrival or new departure” is memoryless

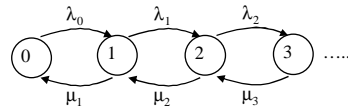
11/29/2007

48

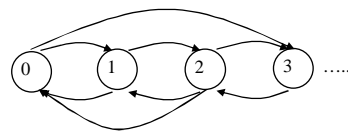
Stochastic Processes



Poisson process



Birth-death process
(M/M/1 when $\lambda_k = \lambda$, $\mu_k = \mu$)



Markov process
- inter-transition time (how long stay in state i)
does not follow memoryless distribution
- State transition is independent from the past

Semi-Markov process
- State transition is independent from the past

11/29/2007

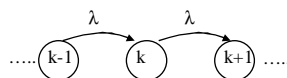
49

Poisson Process

- Birth-death process with $\lambda_k = \lambda$, $\mu_k = 0$ for all k

$$\begin{aligned} - \frac{dP_k(t)}{dt} &= -\lambda P_k(t) + \lambda P_{k-1}(t) && (k \geq 1) \\ &= -\lambda P_0(t) && (k=0) \\ \Rightarrow P_k(t) &= (\lambda t)^k e^{-\lambda t} / k! : \text{"Poisson"} \text{ (transient solution !!!)} \end{aligned}$$

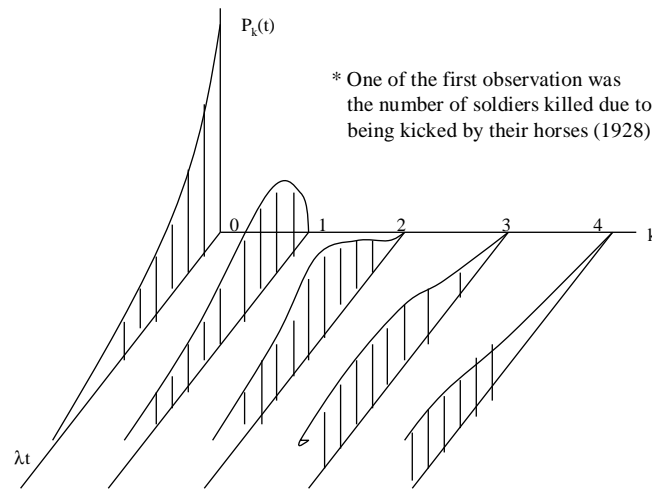
\Rightarrow Distribution of number of customers in the system is Poisson



11/29/2007

50

Poisson Process



11/29/2007

51

Poisson Process

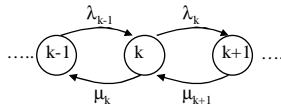
- **Most widely used exponential distribution is Poisson**
- **Described by probability mass function:**
Probability (k) = $e^{-a} \times a^k / k!$
– where a = Rate of events x Elapsed time
- **If interarrival times exponentially distributed & use arrival rate from above for rate of events, number of arrivals in time interval t is a *Poisson process***

11/29/2007

52

Birth-Death Process

- **Continuous-time (homogeneous) Markov chain** where the transitions are restricted to neighboring states only



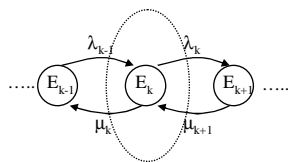
- $P_k(t) = P$ [system has k customers]
- $dP_k(t)/dt = -(\lambda_k + \mu_k) P_k(t) + \lambda_{k-1} P_{k-1}(t) + \mu_{k+1} P_{k+1}(t)$
 \Rightarrow complete solution (including transient) is too complex
- We only interested in steady-state probabilities, obtained by $dP_k(t)/dt = 0$

$$P_k = P_0 \times (\lambda_0 \lambda_1 \dots \lambda_{k-1}) / (\mu_1 \mu_2 \dots \mu_k)$$

11/29/2007

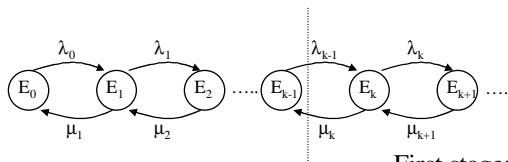
53

Local Balance Equation



Conservation of flow:

- Flow into $E_k = \lambda_{k-1} P_{k-1} + \mu_{k+1} P_{k+1}$
- Flow from $E_k = \lambda_k P_k + \mu_k P_k$
- $\Rightarrow \lambda_{k-1} P_{k-1} + \mu_{k+1} P_{k+1} = (\lambda_k + \mu_k) P_k$



First stage: $\mu_1 P_1 = \lambda_0 P_0$

Second stage: $\lambda_0 P_0 + \mu_2 P_2 = (\lambda_1 + \mu_1) P_1$

$$\Rightarrow \mu_2 P_2 = \lambda_1 P_1$$

Therefore,

$$\mu_k P_k = \lambda_{k-1} P_{k-1} \text{ (steady-state)}$$

11/29/2007

54

M/M/1: Classical Queueing System

- **Birth-death process with $\lambda_k = \lambda$, $\mu_k = \mu$ for all k**

- $p_k = p_0 (\lambda_0 \lambda_1 \dots \lambda_{k-1}) / (\mu_1 \mu_2 \dots \mu_k) = p_0 (\lambda/\mu)^k$
- $p_0 = 1 / [1 + \sum_{k=1}^{\infty} (\lambda/\mu)^k] = 1 / [1 + (\lambda/\mu) / (1 - \lambda/\mu)] = 1 - \lambda/\mu = 1 - \rho$
 $\Rightarrow p_k = (1 - \rho) \rho^k$ where $\rho = \lambda/\mu$

- average number of customers

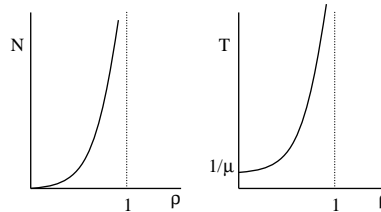
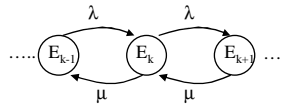
$$N = \sum_{k=1}^{\infty} k p_k = \rho / (1 - \rho)$$

- average time spent in the system

$$T = N / \lambda \text{ (Little's Law)}$$

$$= \rho / (1 - \rho) \lambda$$

$$= 1 / (1 - \rho) \mu$$



11/29/2007

55

M/M/1: Example

- **Engineers use one terminal for serious computation. Arrival pattern is Poisson with a mean of 10 people each day. The distribution of time spent at a terminal is exponential with a mean of 30 minutes. Engineers complain about the terminal service but the manager finds the terminal is in use only 5 hours out of 8-hour working day.**

- $\lambda = 10 \text{ persons/day} \times 1 \text{ day/8hours} \times 1 \text{ hour/60minutes} = 1/48 \text{ persons/min}$
- $\mu = 1/30 \text{ persons/min}$
- $\rho = \lambda / \mu = 30/48 = 0.625$: server utilization
- $N = \rho / (1 - \rho) = 1.667 \text{ persons}$: average number of customers in system
- $Nq = 1.667 - 0.625 = 1.042$: average number of customers in queue
- $T = N / \lambda = 80 \text{ minutes}$: average time spent in the system
- $Tq = 80 - 30 = 50 \text{ minutes}$: average time wasted in queue

* $Tq = Nq / \lambda$ <= Little's Law can be applied in the subsystem

11/29/2007

56

M/M/1: Another Example

- 40 disk I/Os / sec, requests are exponentially distributed, and average service time is 20 ms
⇒ Arrival rate/sec = 40, $\text{Time}_{\text{server}} = 0.02 \text{ sec}$
- 1. On average, how utilized is the disk?
 - Server utilization = Arrival rate \times $\text{Time}_{\text{server}}$
 $= 40 \times 0.02 = 0.8 = 80\%$
- 2. What is the average time spent in the queue?
 - $\text{Time}_{\text{queue}} = \frac{\text{Time}_{\text{server}} \times \text{Server utilization}}{(1 - \text{Server utilization})}$
 $= 20 \text{ ms} \times 0.8 / (1 - 0.8) = 20 \times 4 = 80 \text{ ms}$
- 3. What is the average response time for a disk request, including the queuing time and disk service time?
 - $\text{Time}_{\text{system}} = \text{Time}_{\text{queue}} + \text{Time}_{\text{server}} = 80 + 20 \text{ ms} = 100 \text{ ms}$

11/29/2007

57

How much better with 2X faster disk?

- Average service time is 10 ms
⇒ Arrival rate/sec = 40, $\text{Time}_{\text{server}} = 0.01 \text{ sec}$
- 1. On average, how utilized is the disk?
 - Server utilization = Arrival rate \times $\text{Time}_{\text{server}}$
 $= 40 \times 0.01 = 0.4 = 40\%$
- 2. What is the average time spent in the queue?
 - $\text{Time}_{\text{queue}} = \frac{\text{Time}_{\text{server}} \times \text{Server utilization}}{(1 - \text{Server utilization})}$
 $= 10 \text{ ms} \times 0.4 / (1 - 0.4) = 10 \times 2/3 = 6.7 \text{ ms}$
- 3. What is the average response time for a disk request, including the queuing time and disk service time?
 - $\text{Time}_{\text{system}} = \text{Time}_{\text{queue}} + \text{Time}_{\text{server}} = 6.7 + 10 \text{ ms} = 16.7 \text{ ms}$
 - 6X faster response time with 2X faster disk!

11/29/2007

58

Value of Queueing Theory in practice

- **Learn quickly do not try to utilize resource 100% but how far should back off?**
- **Allows designers to decide impact of faster hardware on utilization and hence on response time**
- **Works surprisingly well**