1 lec2-VM

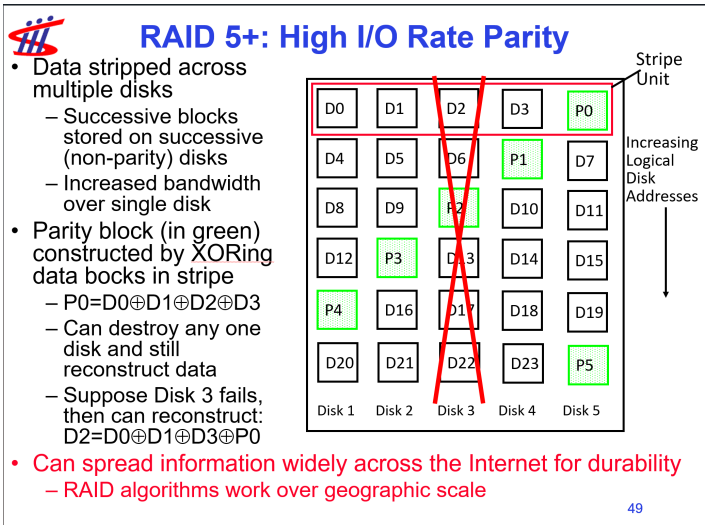
simple base and bound CRAY-1: alter address of every load/store by adding "base"(error is bigger than limit). In program address do not need reallocate for different DRAM.
Multiple segement: segement number, base/limit pair, mark as V/N. physical address has range base → base+limit.
Problem: space become segmented, hard to inter-process sharing.
Swapping: move entire/part of the process to disk.
virtual seg, virtual page, offset. Easy allocation and sharing. Need to be contiguous, two lookups.
Use hash table "inverted hash table" to map virtual page to physical page.

	Advantages	Disadvantages
Segmentation	Fast context switching: Segment mapping maintained by CPU	External fragmentation
Paging (single-level page)	No external fragmentation, fast easy allocation	Large table size ~ virtual memory
Paged segmentation	Table size ~ # of pages in virtual memory, fast easy allocation	Multiple memory references per page access
Two-level pages		
Inverted Table	Table size ~ # of pages in physical memory	Hash function more complex

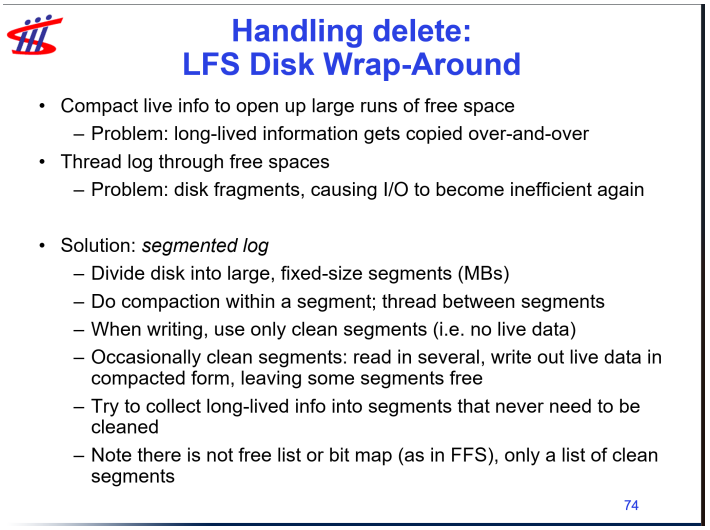
cache miss: compulsory, capacity, conflict; coherence, policy.
Page Fault: Choose old page. If modified, write content back. Change PTE to invalid.
Second Chance: access a page means set 1, if at head, put to tail if 1 and set 0, if 0, replace it.
Clock: put all pages in a circle, repeat second chance.
N th chance: N chance, if 1, change to 0, reset cnt, if 0, cnt++, if cnt=N, replace.
Give dirty page more chance: set N=2 for dirty page, when 1, write back.
Thrashing: process is busy swapping pages in and out Compulosory miss: Clustering: On a page-fault, bring in multiple pages "around" the faulting page,Working Set Tracking: Use algorithm to try to track working set of application.
Read Ahead Prefetching: fetch sequential blocks early. Delayed Writes: write to buffer cache, periodically flush to disk.

2 lec3-file sys

open-file system: per-process, system-wide (different process could simply add a entry points to system-wide file sys), system-wide has a reference counter to count how much process use this file. If 0, remove the entry.
Unordered list of $\langle name, pointer\rangle$, FAT: record free files, last block points to next.
Time = seek time + rotational delay + transfer time + controller overhead.
RAID:Redundant Arrays of file system.
RAID 1: Each disk is fully duplicated onto its "shadow". Bandwidth sacrificed on write. Reads may be optimized.



RAID 6 and others: m nodes could recover the data, $n-m$ failure tolerated.
LFS: log-structured file system. Write to log, periodically merge to disk.



$$\text{write cost} = \frac{\text{total bytes read and written}}{\text{new data written}}$$

$$= \frac{\text{read segs} + \text{write live} + \text{write new}}{\text{new data written}}$$

$$= \frac{N + N*u + N*(1-u)}{N*(1-u)} = \frac{2}{1-u}$$

New Technology File System (NTFS), master file table(MFT) and many extends for each file. Can extent attribute, data to other blocks, form different size of files.

3 lec4-scheduling queue

Multi-Level Feedback Scheduling: first used in Cambridge Time Sharing System(CTSS), use different priority of queue. If timeout at one level, fall down to next level.
Fixed priority scheduiling; Time Slice: each queue gives a certain portion of CPU utilization.
Lottery Scheduling: each process has a number of tickets, scheduler draw a ticket, the process with the ticket will be executed.(To approximate SRTF, short running jobs get more, long running jobs

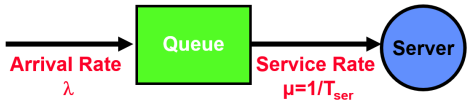
get fewer)

T_A = arrive time → end time, T_S = server start time → end time

Service rate $\mu = \frac{1}{T_S}$, arrival time $\lambda = \frac{1}{T_A}$, Utilization $U = \frac{\lambda}{\mu}$

exponential distribution $f(x) = \lambda e^{-\lambda x}$. Average $m_1 = \sum p(T) \times T$, Variance $\sigma^2 = \sum p(T) \times (T - m_1)^2$.Squared coefficient of variance: $C = \frac{\sigma^2}{m_1^2}$. No variance/deterministic: $C = 0$, "memoryless" or exponential: $C = 1$l. Disk response time $C \approx 1.5$

- Assumptions:
  - System in equilibrium; No limit to the queue
  - Time between successive arrivals is random and memoryless

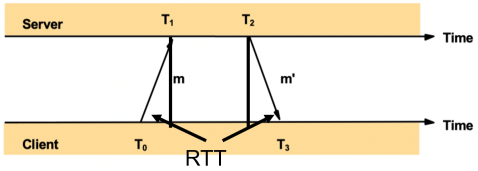


- Parameters that describe our system:
  - λ: mean number of arriving customers/second
  - T_{ser}: mean time to service a customer ("m1")
  - C: squared coefficient of variance = $\sigma^2/m1^2$
  - μ: service rate = $1/T_{ser}$
  - u: server utilization (0≤u<1): $u = \lambda/\mu = \lambda \times T_{ser}$
- Parameters we wish to compute:
  - T_q: Time spent in queue
  - L_q: Length of queue = $\lambda \times T_q$ (by Little's law)

$$T_q = T_{\text{ser}} \times \frac{1}{2}(1 + C) \times \left(\frac{u}{1 - u}\right)$$

4 lec5,6 Distributed

UTC Cristian: $\frac{\text{RTT}}{2}$, accuracy $\frac{\text{RTT}}{2}$ – min response time. Berkeley: use average adjustment, change clock rate rather than clock time.
Network Time Protocol(NTP) have three classes, first level directly connect to UTC, second level connect to first level and second level, third level connect to any other one.



Observe: We can measure server processing time accurately!

RTT = wait_time_client – server_proc_time
= $(t_3\text{-}t_0) - (t_2\text{-}t_1)$
Time adjustment at client = $(t_3$ + Offset) = t_2 + RTT/2 (Cristain's)
So, Offset = t_2 + RTT/2 - t_3
= $((t_1\text{-}t_0) + (t_2\text{-}t_3))/2$
Why do we compute RTT and Offset?

We can take 8 measurements and take the one with minimum packet delay.
Lamport Clock: $L(e) = M \times L_i(e) + i$. where i is process ID. Vector Clock: $V(e) = [L_1(e), L_2(e), \cdots, L_n(e)]$. Corrsponding bits represents the process's tag
Totally order multicast:

1. Message to be sent is timestamped with the sender's logical time.
2. Message is multicast to all processes.
3. When a process receives a message, a. put into local queue. b. ordered according to timestamp. c. receiver multicast ACK.

Need to make sure that all receiving objects already send ACKs
Centralized Algorithm: use a queue, if lock is occupied, put into queue. If release the lock, give the lock to the first one in the queue. No coordinator tolerance fault tolerance.
Leader Election: stage1: P notices leader failed stage2: P sends ELECTION algorithm to higher numbers. If no one replies, P becomes leader. Otherwise the higher number one broadcast the ELECTION. Can tolerate corrdinator failure.
Fully Decentralized: majority vote from $m > \frac{n}{2}$ coordinators. The coordinator reply GRANT/DENY . May cause starvation.
Lamport Mutual Exclusion:1.multicast time-stamped request. If own request at the head, all replies have been received, enter(get lock) exit(release lock). (ACK only to requester, Release after finish)2.receive a request, put request in queue. reply with timestamps. Unbounded wait time.
Ricart-Agrawala Algorithm: 1. send request to all other processes. 2. receive a reply from all other processes, if receive $n - 1$ replies, get the lock. 3. enter critical section. 4. release lock, broadcast to others.
Token Ring Algorithm: put all client on a ring, push the lock towards.

Algorithm	# Messages per cycle	Delay before entry	Problems
Centralized	*3*	*2*	Coordinator Crash
Decentralized	*2mk + m, k≥1*	*2mk*	Starvation
Lamport	*3(N-1)*	*2(N-1)*	Crash of any process, inefficient
Ricart & Agrawala	*2(N-1)*	*2(N-1)*	Crash of any process
Token Ring	*1 to infinite*	*0 to (N-1)*	Lost token, process crash

- k = number of retries in getting majority

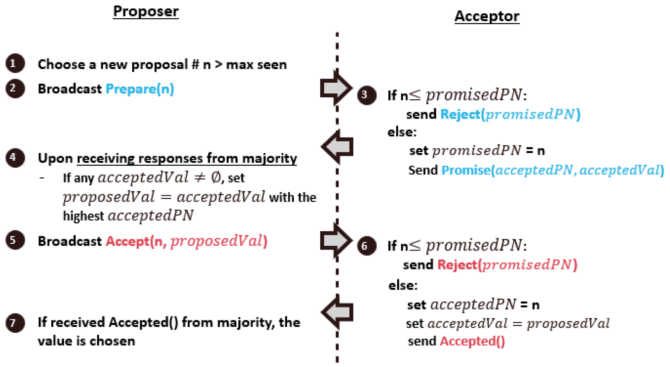
5 lec7 Consensus

CAP-consistency, Availability, Partition-tolerance.
avoid slowest node: put():wait for ACKs frin $\geq W$ replica, get(): wait for responses from $\geq R$ replica. $W + R > N$ – avoid slowest node.
Correct Consensus:

1. termination: all correct processes decide
2. agreement: all correct processes decide the same value
3. integrity: if all correct processes propose the same value, then any correct process decides that value

strict consistency:

1. Linearizability: if somebody see the latest, other's can't see the older.
2. Sequential Consistency: if A before B, all see A before B
3. Causal Consistency: if A before B, all see A before B
4. Eventual Consistency: if no update, all see the same value.



Paxos:

1. Propose state: propose a value with proposal number: Round Number + Process ID, higher proposal number have higher priority.

2. Accept state: if receive a proposal with higher number and it has not accept any proposal, send ACK. If already accept a proposal, send its value, proposer will repropose the same value. If it has receive a larger number proposal, send "reject".

3. Decide state: finally a majority will accept the proposal, and the final value is decided when a majority of acceptors accept the proposal.

6 lec8 Blockchain

bitcoin: nonce, previous hash, block header. Merkle tree: a binary tree of hash transactions, only need $O(\log n)$ to store n blocks.
k-delay confirmation: after 6 blocks, miner can collect their reward. Miners need check:

1. ScriptSig—Script PubKey return true

2. TXID—index is in current UTXO set

3. sum(input vals)≥sum(output vals)

Ethereum contract has its own address, transaction is equivalent to transfer ownership.
Replay Attack: A send a transaction to B. B secretly record the transaction without response. A thought it is a timeout, resend another transaction to B, B rebroadcast the first transaction. Solution: use a nonce to mark the transaction, nonce is bringed with data block.

7 lec 9 Fault Tolerance

BDOS: attacker fins a block: only broadcast header, tell all other clients he know this block, if a miner mine this, he will race against you, have 50% win rate, a malicious miner will give this block up, causing a stall.
PBFT Practical Byzantine Fault Tolerance: with $2f+1$ good clients, f bad clients, still work.

1. pre-prepare phase: from primary to replicas, send a pre-prepare message that means which transaction we are talking about, if $2f+1$ pre-prepare message received, go to next phase.

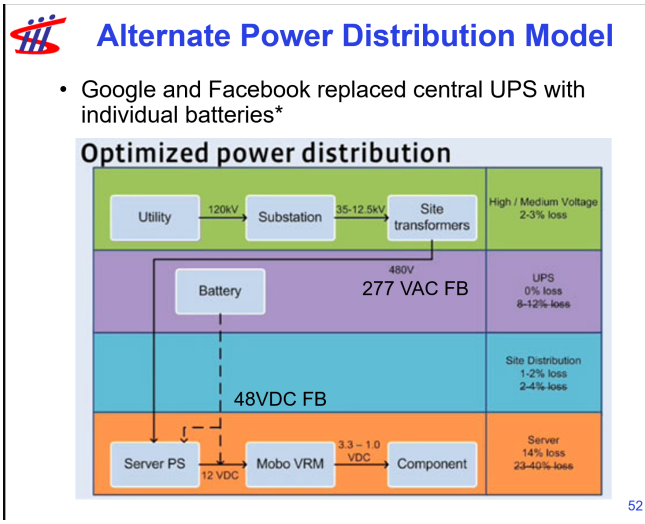
2. prepare phase: replicas send prepare message to all other replicas, if $2f+1$ distinct prepare message received, go to next phase

3. commit phase: if a replica is in PREPARED phase, broadcast a COMMIT.if $2f+1$ distinct commit message received and already in PREPARED phase, enter COMMIT-LOCAL phase. (Note that f clients can lie about PREPARED value)

MTBF: mean time between failures, **MTTR**: mean time to recover. reliability: large MTBF. Availability: small $\frac{\text{MTTR}}{\text{MTBF+MTTR}}$.

$\text{MTTR} = \text{Detect} + \text{Diagnose} + \text{Decide} + \text{Action}$, $\text{unavailability} = \frac{\text{MTTR}}{\text{MTBF}}$

Failure Rate Curve: Early "Infant Mortality" failure, constant failure, wear-out failure.
AT&T Internet DC 15mins backup source. Google&Facebook use UPS, individual battery.



SW Fault: Software Fault. Fail fast& High Availability Execution. Instant repair. sw fault Bohrhug: expectable bug, could be predicted – no repair, Heisenbug: unpredicable, and when testing may disappear – restart process.
Apache Zookeeper: Multiple servers, do leader election among servers, group membership Work Queues, Data Sharding, Event Notifications, Config, and Cluster Mgmt. Highly available, distributed coordination kernel. Using Paxos, ordered updates and strong persistence guerantees.
Testing, Control Chaos: best way is to fail constantly, inject random failure into cloud by killing VMs(most time nothing happens, occasionally something surprises)

8 lec10 Security

ACM Access Conrol Matrix: a matrix with users, files, content is access right. **ACL**: file $j \to \{(user1, access1), \ldots\}$. **Capability List**: user $i \to \{(file1, access1), \ldots\}$.
Buffer Overflow: Attacker sends large file, intended to overwrite the return address(e.g. load some executable code to stack and jump to it)
Worm Spreading

$$f = \frac{e^{k(t-T)} - 1}{e^{k(t-T)} + 1}$$

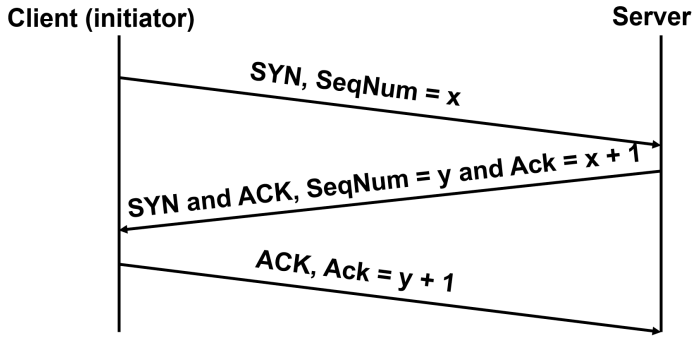
where f is fraction of hosts infeced, k is spreading rate, T is starting time of infection.

1. Morris Worm: buffer overflow in fingerd debugging routines in sendmail

2. Code Red Worm: connect to TCP 80(Microsoft IIS). Send GET to victim, buffer overflow. "bug": use same random seed generator. DOS attack on host& slow to infection.

3. MS SQL Slammer: UDP port 1434 buffer overflow. Send massive network packets. Solution: close UDP port and reboot since worm only infected in-memory processes, never write to disk.

Firewall: source address and port number, payload, stateful analysis of data.
DDOS Denial of service(SYN attack). on victim, only limit number of unfinished connections. SYN attack is based on 3-way handshaking, fake IP and request a connection, random spoofed source address, use nonce to prevent replay attack. Only when receive the nonce back, allocate memory.



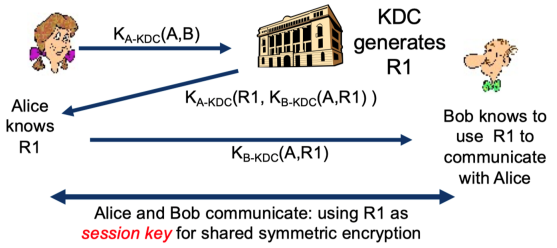
Solution: SYN Cookies

- Server: send SYN-ACK with sequence number y, where
  - y = HMAC(client_IP_addr, client_port, server_key)
  - HMAC(): Hash Message Authentication Code
- Client: send ACK containing y+1
- Sever:
  - verify if y = HMAC(client_IP_addr, client_port, server_key)
  - If verification passes, allocate memory
- Note: server doesn't allocate any memory if the client's address is spoofed
  - Why is that?
- This is done within the existing TCP specification

KDC key distribution center

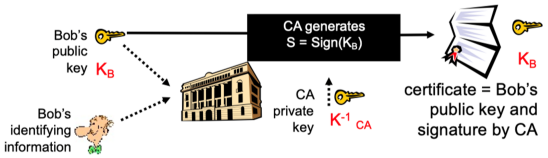
KDC: Session Key Generation

- How does KDC allow Bob, Alice to determine shared symmetric secret key to communicate with each other?



PKI: Certification Authorities (CA)

- Certification authority (CA): binds public key to particular entity, E.
- An entity E registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - Certificate contains E's public key AND the CA's signature of E's public key.



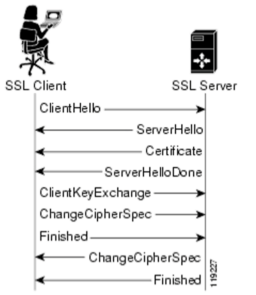
75

PKI: CA(Certification Authority): Bob's public key+signature by CA, Alice use CA's public key to verify Bob's certificate, acceot public key.

TLS: Transport Layer Security i.e. SSL(Secure Socket Layer) seperate data, add session key to encrypt, MAC each chunk form a TLS record – short header+data. Record byte stream, fed to TCP socket for transmission.

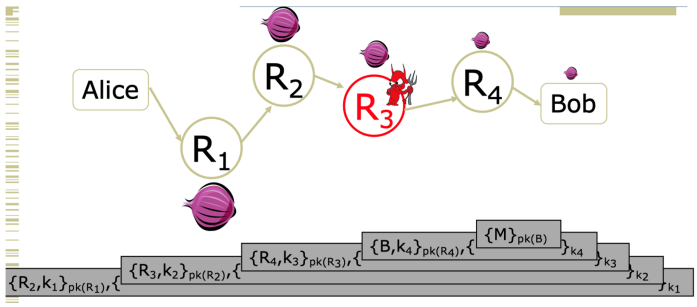
Setup Channel with TLS: "Handshake"

- Handshake Steps:
  1. Clients and servers negotiate exact cryptographic protocols
  2. Client's validate public key certificate with CA public key.
  3. Client encrypt secret random value with servers key, and send it as a challenge.
  4. Server decrypts, proving it has the corresponding private key.
  5. This value is used to derive symmetric session keys for encryption & MACs.



Diffie-Hellman: $g^a \mod p$ and $g^b \mod p$ are exchanged, $g^{ab} \mod p$ is the shared secret.
Onion Routing: use multiple layers of encryption, each layer is removed by a node. Man-in-the-middle attack: the intruder will maliciously build up a tunnel to both A and B, sending messages by decrypt and reencrypt.

Route Establishment



Routing info for each link encrypted with router's public key Each router learns only the identity of the next router

Note: k1 , k2 , k 3 etc are session keys, so when each router (R1 , R2 , .. Rn) use their private keys to decrypt the packets, they can only then get the next hop (e.g. R2) and the session key (k1) to decrypt the rest of the packet and send it along.

93