Storage Systems

NPTEL Course

Jan 2012

(Lecture 30)

K. Gopinath

Indian Institute of Science

Design Issues in Large Storage Systems

- Large Storage Systems need to be distributed and replicated
  - For throughput
  - For resilience to failures
  - For disaster recovery
- May also need to solve “(w)holistic” problems such as RT/QoS and security

  Usually, need to solve many *coordination problems*
- Many other imp aspects but difficult problem already!
- Difficulty arises from asynchronous nature and failures characteristic of real systems
  - Faced by Amazon, eBay, Skype, Gmail, Facebook...
  - (Synchronous systems typically small scale)

NFS: dir lockup

- Consider a “slow op” on a (locked) file due to NFS congestion
- LOOKUP on that file results in a lock on its dir that cannot be released until “slow op” finishes =>
  - cascade of locks upto root that hangs the system till “slow op” finishes
- lockd: similar problem but worse as lockd a user process

Recent (Sep 23, '10) 2.5 hr Facebook Outage

- Caused by an automated system to check for invalid configuration values in cache and replace them with updated values from the persistent store
  - works well for a transient problem with cache, but it doesn’t work when the persistent store is invalid
- Somebody made an “invalid change” to persistent config values
- Each alert client attempted fix of invalid cache value=> has to query cluster db that cannot scale (=>1000's queries per sec)
  - Also deleted cache key
    - Now queries do not succeed in the cache *during and just after* the fix!
    - Each new request has to go to non-scalable db again!
    - Positive feedback cycle with more requests to db
- Had to stop all reqs to cluster db to recover; *site down*

Why are failures difficult in asynch env?

- (simpler) The 2 generals coord problem: need to coord to defeat enemy in between (who can seize, dupl, corrupt any msg sent)
  - <u>*No protocol exists*</u>!
- The FLP (Fischer, Lynch, Patterson) result: impossibility of distributed consensus with 1 faulty processor (JACM'85)
- Fekete, Lynch, Mansour, Spinelli: impossibility of reliable communication in the face of crashes (JACM'93)
- No reliable data link layer can exist in CAML model (JACM'00)
  - ***c**rashes, **a**synchronous, **m**emoryless, **l**ossy model*

  System can be driven by a sequence of crashes to any global state where each node is in a state reached in some (possibly diff) run, and each link has an arbitrary mixture of packets sent in (possibly diff) runs (Jayaram/Varghese JACM'00)

Brewer's CAP Theorem

- Three important properties
  - **C**onsistency
  - **A**vailability
  - tolerance to **P**artitions due to breakdowns in communications in the system

<u>cannot all be guaranteed at the same time</u> according to a theorem in distributed systems theory (proved by Gilbert & Lynch '02)

Space of CAP

- **C** and **A**v only (no tolerance to partitions)
  - Single-site fs/db, Cluster fs/db, LDAP
  - *2-phase commit*, cache validation protocols
- **A**v and **P** only (no consistency)
  - Coda, DNS, web caching
  - *leases*, need for *conflict resolution*, *optimistic*
    - Good example*: cricinfo* scores*!*
- **C** and **P** only (no availability)
  - Distributed fs/db, Distributed locking
  - *Majority protocols*: minority partitions unavailable
  - *Pessimistic* locking

FS vs DB perspectives

- FS: a persistence and naming service for all appls
  - No information from appls on what info is critical
  - System can differentiate betw data and metadata only
  - Can guarantee
    - Consistency of metadata (needed for FS's sanity!)
      - If metadata corrupted, will fix it in some (!) way so that system can function again!
    - Optionally data consistency (using synchronous operations)
      - VERY SLOW!!! and hence not the default!
- DB: an appl from the OS/FS perspective. Uses “ACID” semantics:
  - **A**tomicity, **C**onsistency of data: DB's responsibility
  - **I**solation from other transactions also
  - **D**urability: storage system's responsibility

Remarks

- OS/Networking good at availability, but not good at consistency
  - NFS a good example here
- Can have consistency & availability within a cluster, but hard
- dfs/db better at consistency than availability
  - Wide-area DBs or Disconnected clients neither
- Durability is HARD: a large storage system itself composed of many parts
  - Recursive problem: how does it keep its own metadata or data “consistent” or “atomic” wrt changes and what persistent store can it depend on for its operations?

However!

- Consider the TCP protocol
  - TCP, the basis of Internet, finalized in RFC 793 in Sep 1981. Only clarifications to this design since then.
  - “Just Works” inspite of theoretical result of impossibility of implementing reliable comm in the face of crashes
  - Careful engineering to avoid problems due to “reincarnation” of a connection
      - A TCP cnxn identified by (IP addr, port num) of src and dst ***(finite size)***
      - *If this repeated in a new cnxn, (incorrect) packet insertion possible!*
    - but still mandates (but not followed) “cannot reboot a system faster than 2 minutes or remember last seq# before crash” !!! (see Sec 3.3, RFC793)
- So an impossibility result does not have to make us drop our plans
  - Just make sure the engineering is very good!

Ways to Get Around CAP

- Avoid failures? Impossible but can we reduce freq of failures by redundancy (say, 99.9999% uptime)?
  - Design and impl not easy + very costly
- Make sure partitions are repaired within latency requirements for a request: Too costly/Difficult?
- Assume “Timed Asynchronous Systems” (TAS): unstable periods followed by sufficiently long stable periods
  - “Failure aware” design
  - '96 FAA project based on this model but was a colossal failure as solution based on TAS subtle and needs care in impl.
- “Do not do anything when partitions present”
  - Fundamentally impossible to detect failure (aka partitions) reliably: FLP result
  - Availability is not possible

Birman: Attack Root Causes?

- As data center networks scale out, sw stack increasingly oriented towards one-to-many (multicast: MC) communication patterns: eg. Facebook/Twitter
  - Publish-subscribe and other enterprise service bus layers use multicast to push data to many receivers simultaneously
  - allows clustered appl servers to replicate state updates and heartbeats betw server instances and maintain coherent caches by invalidating or updating cached info on many nodes
- For scalability, IP multicast critical but deprecated (Amazon) as it lacks reliable packet dissemination, security, flow control and scalability in number of groups
  - IP MC addresses scarce and sensitive resources
  - If hw limits of MC addr exceeded, kernel burdened & perf decreases

Multicast Oscillatory Behaviour

- Nodes experience disturbances eg. Java gc pauses, Linux sched delays, flushing data to disk
- Prevents nodes from forwarding packets eg. when appl thread does not respond or when packets do not reach node because of a link problem.
- For a while, root continues sending, so incoming packets from the upstream link fill node’s buffers.
- Flow control causes node’s parent node to stop sending, which in turn causes its buffers to fill up.
- If node’s disturbance persists, then eventually all buffers on path from root to the node become full, and root’s sending throughput drops to zero
- In large trees (10K-60K nodes in cloud), when each node is disturbed for one sec/hour on average, throughput degradation (up to 90%) occurs even if message loss is negligible.