Storage Systems

NPTEL Course

Jan 2012

(Lecture 43)

K. Gopinath

Indian Institute of Science

BigTable Impl

- 3 main components: a library that is linked into every client, one master server, and many tablet servers
- tablet servers dynamically added (or removed) from a cluster to accommodate changes in workloads.
- master responsible for
  - assigning tablets to tablet servers
  - detecting addition and expiration of tablet servers
  - balancing tablet-server load
  - garbage collection of files in GFS.
  - also handles schema changes such as table and column family creations
- persistent state of a tablet stored in GFS thru memtable

Interactions betw GFS and BigTable

- Persistent state of a tablet stored in GFS
- Incoming writes committed to a commit log that stores redo records
  - recently committed ones sorted and buffered in memory as memtables
  - older updates stored in a seq of SSTables (sorted string tables)
- Incoming reads checked in memtables and SSTables
  - sorted tables: efficient op
- To recover a tablet:
  - a tablet server reads its metadata: the list of SSTables that comprise a tablet and a set of a redo points (pointers into any commit logs that may contain data for the tablet)
  - server reads indices of SSTables into memory and reconstructs memtable by applying all committed updates since the redo points

Hadoop Distributed File System

- Rack-aware filesystem
  - To help run work on the node where the data is, and, failing that, on the same rack/switch, to reduce backbone traffic.
- HDFS stores large files (an ideal file size is a multiple of 64 MB) across multiple machines.
- Reliability by replicating data across multiple hosts (no RAID storage on hosts)
- Above HDFS runs MapReduce Engine
  - Client applications submit MapReduce jobs to JobTracker

Yahoo! PNUTS

- focuses on data serving for web applications
  - workloads mostly of queries that read and write single records or small groups of records
  - not for complex queries, e.g., offline analysis of web crawls
- data storage organized as hashed or ordered tables
- designed for low latency for large numbers of concurrent requests including updates and queries
  - all high latency operations asynchronous
  - support record-level mastering (local ops afap)
- per-record consistency guarantees: all replicas of a given record apply all updates to the record in the same order

Timeline consistency

- Object timelines
  - Reads are served using a local copy; may be stale
  - But application can get current version or any   vers > N
  - While copies may lag master record, every copy goes through same sequence of changes
  - Test-and-set writes facilitate per-record transactions

PNUTS design

- uses a guaranteed message-delivery service rather than a persistent log
- trigger-like notifications
  - imp for some apps that must invalidate cached copies after some time (eg. ad serving with a time contract)
  - users subscribe to stream of updates on a table
    - asynch publish-subscribe message system
      - can be optimized for geographically distant replicas and replicas do not need to know locs of other replicas
      - contrast to gossip protocols

Windows Azure

- Designed as a scalable cloud storage system
  - cloud storage in the form of blobs (user files), tables (structured storage) and queues (msg delivery)
- Blobs for incoming and outgoing data, Queues for overall workflow for processing the Blobs, intermediate service state and final results in Tables or Blobs
- Publically searchable content (via Bing) within 15 secs of a Facebook/Twitter user’s posting or status update
- “Strong consistency”: same as others (within a “stamp”)
  - Intra-stamp synch repl
  - Inter-stamp asynch repl
- Global and Scalable Namespace/Storage

Another Model for Storage: Consistent Hashing

- Hash both objects and devices using same hash function
- Map each obj to a point on the edge of a circle
    - Equivalently, to a specific angle
- Map each device (eg. storage bucket) also pseudo-randomly mapped on to a series of points around circle
- An obj stored by selecting the closest mapped device on the circle
- Each device contains the resources mapped to an angle between it and the next smallest angle

If a device added or removed, only nearby objs remapped

Used by Amazon and Facebook

Distr Hash Table (DHT)

- Uses some variant of consistent hashing to map keys to nodes
- A node with ID i_x owns all keys k_m for which i_x the closest ID, measured according to $\delta(k_m,i_x)$
- To store a file with filename *fn* and *data* in DHT
  - Calculate, say 160-bit hash key, *k* = SHA-1(*fn*)
  - A message *put(k,data)* sent to any node in DHT
  - msg forwarded from node to node thru overlay network (connecting nodes) until it reaches single node responsible for key *k* as specified by the keyspace partitioning
- To retrieve file *fn*, *get*(SHA-1(*fn*))

Cassandra

- Distributed multi dimensional map indexed by a key
  - row key a string with no size restrictions (typ 16-36B)
  - value is a highly structured object
- Similar to Bigtable: every op under a single row key atomic per replica no matter how many columns are being read or written into
  - Columns grouped together into sets called column families
- Data partitioned across cluster using consistent hashing but uses an order preserving hash function to do so
- Cassandra API: three simple methods:
  - insert(table,key,rowMutation)
  - get(table,key,columnName)
  - delete(table,key,columnName)

Server-side Consistency Models

- N = number of nodes that store replicas of data
- W = number of replicas that need to ack receipt of update before update completes
- R = number of replicas that are contacted when a data object is accessed through a read op
- W+R > N: write set & read set always overlap and one can guarantee strong consistency
- W+R <= N: Weak/eventual consistency
- R=1, W=N: optimize for read

  W=1, R=N for very fast write

Client-side Consistency Models

- Strong: After an update completes, any subsequent access will return the updated val
- Weak: May return stale value. *Special case*:
- Eventual: storage system guarantees that if no new updates made to the object, eventually all accesses will return last updated value eg. DNS
  - *Variations*:
    - Causal consistency (CC)
    - Read-your-writes consistency (RyWC)
    - Session consistency (SnC)
    - Monotonic read consistency (MRC)
    - Monotonic write consistency (MWC)
  - If client connects to a server only, RyWC/MRC easy

Haystack: Facebook’s photo storage

- Std solutions (eg. NFS): excessive number of disk operations because of metadata lookups
  - to read a single photo: 1+ disk ops to translate filename to an inode #, 1 to read inode from disk, and one to read file itself
- reduce per photo metadata so that all metadata lookups in main memory
  - rwx perms not needed; 128-256B inode size too big
  - now disk ops only for reading *actual data:* increases overall throughput
  - high throughput and low latency: at most one disk op per read
- Why does caching not work?

Haystack Design

- Content Distr Networks (CDN) effective for serving “hot photos”
  - recently uploaded & popular
- But social networking sites also see a large # of requests for less popular (often older) content: long tail
- Requests from long tail account for a significant amount of Facebook traffic
  - almost all access the backing photo storage hosts as these requests typically miss in the CDN
- Haystack: each usable TB costs approx 28% less and processes 4x more reads per sec than an equiv TB on a NAS appliance

Summary

- Scalability inducing many cross-layer designs
  - Have to pay attention to overheads and remove them
  - Handling failures imp, so repl (or erasure coding) critical in design
    - Distribution of data a necessity
    - Coord of updates a necessity