Storage Systems

NPTEL Course

Jan 2012

(Lecture 42)

K. Gopinath

Indian Institute of Science

GFS Probs

- Design: High sustained bw more imp than low latency
  - batch-oriented appls such as web crawling/indexing
- Reality now: Gmail or YouTube (~RT)
    - developer base moved from MapReduce to interactive apps (using BigTable)
  - SPOF (master) not a disaster for batch-oriented appls, but unacceptable for latency-sensitive appls, such as video serving
    - Early: No automatic failover scenario if master crashes
      - Manual restart (upto 1hour)
    - Now: automatic failover but takes 10 min (early) to 10 secs (now)
    - Write to 3 chunkservers: if one “slow”, limits to 5-10MB/s
      - Pullchunk: allocates new chunk, etc. : takes 10 secs + (for 64MB) to a minute
  - Extensive Qing in the design (for bulk perf) but Qing delay kills

More Probs

- Limited number of files (all metadata in master in memory!)
    - 64MB to 1MB?
    - OK upto multiple 10's of TB.
    - With 10-100's of PB? 10GB-100's GB of mem.
- Single master a scanning bottleneck (recovery, etc)
- Metadata serving bottleneck with many Map/Reduce clients
    - Scaling from 1000's of ops to 10000's and beyond
- Batch op does not allow for large-scale incremental processing using distributed transactions and notifications
    - Freshness of web results

Consistency Probs

- Clients push write until it succeeds
  - Client failures cause indefinite state
- RecordAppend interface for multiple writers to append to a log concurrently
  - Primary selects offset to write
  - Primary can change and new offset taken
- RecordAppend does not offer any replay protection
  - Some piece of data multiple times in the file or data in a different order
    - multiple times in one chunk replica, but not necessarily in all
  - Read of file: data in different ways at different times
  - At record level: records in different orders depending on which chunks read

Other probs

- snapshot a chunk: to replace a replica, or whenever some chunkserver goes down and need to replace some of its files
  - Actually cloning
  - Difficult...

Solutions

- one master per cell to one master per data center
  - put multiple GFS masters on top of a pool of chunkservers
  - appl responsible for partitioning data across diff cells
- combine some number of underlying objects into larger files
  - with quotas on both file counts and storage space: typ exhausted file count quota first
  - new design point: 100 million files per master with 100's of masters

Appl Workarounds

- Gmail uses a multihomed model: if one instance of a Gmail account not avlbl, move to another data center
- BigTable transaction log a big bottleneck for getting a txn logged
    - two logs open at any one time
    - write to one and if that gets stuck, write to other
    - merge logs but only on replay
- or, use BigTable for any app with lots of small data items

BigTable

- a structured storage system with many key-value pairs and a schema
  - For web indexing, Google Earth, etc.
- sparse, distributed, persistent multi-dimensional sorted map
  - map indexed by a row key (upto 64KB, typ 10-100B), column key, and a timestamp: each value in map an uninterpreted array of bytes

    (row:string, column:string, time:int64) → string

    - Eg: URLs: row keys, metadata categories of web pages: column names, contents of web pages: contents: column, timestamps when fetched
  - data in lexicographic order by row key
- every read or write of data under a single row key is atomic (regardless of the number of different columns being read or written in the row)

BigTable Design

- Not a full relational data model
  - provides clients with a simple data model that supports dynamic control over data layout and format
  - allows clients to reason about the locality properties of the data represented in the underlying storage
    - Clients can control locality of their data through careful choices in their schemas
- Data indexed using row and column names that can be arbitrary strings
- Treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings
- Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk

Design (contd)

- Row range for a table dynamically partitioned.
  - Each row range a *tablet:* unit of distribution and load balancing
  - reads of short row ranges efficient: typically require comm with only a small number of machines
- Clients can select their row keys for good locality of data accesses
  - For storing Web pages, pages in the same domain grouped together into contiguous rows by reversing the hostname components of the URLs

Column Families

- Column keys grouped into sets (column families): *family:qualifier*
  - basic unit of access control
  - All data stored in a column family typ same type (eg. language)
    - compress data in the same column family together
    - number of distinct column families small (100's) but a table may have an unbounded number of columns
    - families rarely change during operation

BigTable, GFS and Chubby

- GFS provides only two basic data structures: logs and SSTables (Sorted String Tables)
    - majority of data in protocol buffers (data description lang) in these two structures.
  - SSTables are immutable, while BigTable provides mutable key value storage
  - Stores incoming data into transaction log files.
  - “compacted” into a series of SSTables, which in turn get compacted together over time (~LFS)
    - Sort and index
- Uses a highly-available and persistent distributed lock service called Chubby that uses Paxos
  - 5 active replicas, with one elected master to actively serve requests

API

- Creating and deleting tables and column families
- Changing cluster, table, and column family metadata, such as access control rights
- Client applications can
  - write or delete values in Bigtable
  - look up values from individual rows
  - iterate over a subset of the data in a table

Use cases

- Supports single-row transactions to perform atomic RMW sequences on data stored under a single row key
  - No support for general transactions across row keys
  - Bigtable allows cells to be used as integer counters
- Supports execution of client-supplied scripts in addr spaces of servers
  - Allows data transformation, filtering based on arbitrary exprs, and summarization via operators but no write back into BigTable
- Both an input source and as an output target for MapReduce jobs

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