Storage Architecture and Challenges

Google

Faculty Summit, July 29, 2010
Andrew Fikes, Principal Engineer
fikes@google.com

Introductory Thoughts

Google operates **planet-scale** storage systems

What keeps us programming:

- Enabling application developers
- Improving data locality and availability
- Improving performance of shared storage

A note from the trenches: "You know you have a large storage system when you get paged at 1 AM because you only have a few petabytes of storage left."

The Plan for Today

- Storage Landscape
- Storage Software and Challenges
- Questions (15 minutes)

Storage Landscape: Hardware

A typical warehouse-scale computer:

- 10,000+ machines, 1GB/s networking
- 6 x 1TB disk drives per machine

What has changed:

- Cost of GB of storage is lower
- Impact of machine failures is higher
- Machine throughput is higher

What has not changed:

- Latency of an RPC
- Disk drive throughput and seek latency

Storage Landscape: Development

Product success depends on:
- Development speed
- End-user latency

Application programmers:
- Never ask simple questions of the data
- Change their data access patterns frequently
- Build and use APIs that hide storage requests
- Expect uniformity of performance
- Need strong availability and consistent operations
- Need visibility into distributed storage requests

Storage Landscape: Applications

Early Google:

- US-centric traffic
- Batch, latency-insensitive indexing processes
- Document "snippets" serving (single seek)

Current day:

- World-wide traffic
- Continuous crawl and indexing processes (Caffeine)
- Seek-heavy, latency-sensitive apps (Gmail)
- Person-to-person, person-to-group sharing (Docs)

Storage Landscape: Flash (SSDs)

Important future direction:

- Our workloads are increasingly seek heavy
- 50-150x less expensive than disk per random read
- Best usages are still being explored

Concerns:

- Availability of devices
- 17-32x more expensive per GB than disk
- Endurance not yet proven in the field

Storage Landscape: Shared Data

Scenario:

- Roger shares a blog with his 100,000 followers
- Rafa follows Roger and all other ATP players
- Rafa searches all the blogs he can read

To make search fast, do we copy data to each user?

- YES: Huge fan-out on update of a document
- NO: Huge fan-in when searching documents

To make things more complicated:

- Freshness requirements
- Heavily-versioned documents (e.g. Google Wave)
- Privacy restrictions on data placement

Storage Landscape: Legal

- Laws and interpretations are constantly changing
- Governments have data privacy requirements
- Companies have email and doc. retention policies
- Sarbanes-Oxley (SOX) adds audit requirements

Things to think about:

- Major impact on storage design and performance
- Are these storage- or application-level features?
- Versioning of collaborative documents

Storage Software: Google's Stack

Tiered software stack

- Node
  - Exports and verifies disks
- Cluster
  - Ensures availability within a cluster
  - File system (GFS/Colossus), structured storage (Bigtable)
  - 2-10%: disk drive annualized failure rate
- Planet
  - Ensures availability across clusters
  - Blob storage, structured storage (Spanner)
  - ~1 cluster event / quarter (planned/unplanned)

Storage Software: Node Storage

Purpose: Export disks on the network

- Building-block for higher-level storage
- Single spot for tuning disk access peformance
- Management of node addition, repair and removal
- Provides user resource accounting (e.g. I/O ops)
- Enforces resource sharing across users

Storage Software: GFS

The basics:

- Our first cluster-level file system (2001)
- Designed for batch applications with large files
- Single master for metadata and chunk management
- Chunks are typically replicated 3x for reliability

GFS lessons:

- Scaled to approximately 50M files, 10P
- Large files increased upstream app. complexity
- Not appropriate for latency sensitive applications
- Scaling limits added management overhead

Storage Software: Colossus

- Next-generation cluster-level file system
- Automatically sharded metadata layer
- Data typically written using Reed-Solomon (1.5x)
- Client-driven replication, encoding and replication
- Metadata space has enabled availability analyses

Why Reed-Solomon?

- Cost. Especially w/ cross cluster replication.
- Field data and simulations show improved MTTF
- More flexible cost vs. availability choices

Storage Software: Availability

Tidbits from our Storage Analytics team:

- Most events are transient and short (90% < 10min)
- Pays to wait before initiating recovery operations

Fault bursts are important:

- 10% of faults are part of a correlated burst
- Most small bursts have no rack correlation
- Most large bursts are highly rack-correlated

Correlated failures impact benefit of replication:

- Uncorrelated R=2 to R=3 => MTTF grows by 3500x
- Correlated R=2 to R=3 => MTTF grows by 11x

source: Google Storage Analytics team
D.Ford, F.Popovici, M.Stokely, and V-A. Truong, F. Labelle, L. Barroso, S. Quinlan, C. Grimes

Storage Software: Bigtable

The basics:

- Cluster-level structured storage (2003)
- Exports a distributed, sparse, sorted-map
- Splits and rebalances data based on size and load
- Asynchronous, eventually-consistent replication
- Uses GFS or Colossus for file storage

The lessons:

- Hard to share distributed storage resources
- Distributed transactions are badly needed
- Application programmers want sync. replication
- Users want structured query language (e.g. SQL)

Storage Challenge: Sharing

Simple Goal: Share storage to reduce costs

Typical scenario:

- Pete runs video encoding using CPU & local disk
- Roger runs a MapReduce that does heavy GFS reads
- Rafa runs seek-heavy Gmail on Bigtable w/ GFS
- Andre runs seek-heavy Docs on Bigtable w/ GFS

Things that go wrong:

- Distribution of disks being accessed is not uniform
- Non-storage system usage impacts CPU and disk
- MapReduce impacts disks and buffer cache
- GMail and Buzz both need hundreds of seeks NOW

Storage Challenge: Sharing (cont.)

How do we:

- Measure and enforce usage? Locally or globally?
- Reconcile isolation needs across users and systems?
- Define, implement and measure SLAs?
- Tune workload dependent parameters (e.g. initial chunk creation)

Storage Software: BlobStore

The basics:

- Planet-scale large, immutable blob storage
- Examples: Photos, videos, and email attachments
- Built on top of Bigtable storage system
- Manual, access- and auction-based data placement
- Reduces costs by:
  - De-duplicating data chunks
  - Adjusting replication for cold data
  - Migrating data to cheaper storage

Fun statistics:

- Duplication percentages: 55% - Gmail, 2% - Video
- 90% of Gmail attach. reads hit data < 21 days old

Storage Software: Spanner

The basics:

- Planet-scale structured storage
- Next generation of Bigtable stack
- Provides a single, location-agnostic namespace
- Manual and access-based data placement

Improved primitives:

- Distributed cross-group transactions
- Synchronous replication groups (Paxos)
- Automatic failover of client requests

Storage Software: Data Placement

- End-user latency really matters
- Application complexity is less if close to its data
- Countries have legal restrictions on locating data

Things to think about:

- How do we migrate code with data?
- How do we forecast, plan and optimize data moves?
- Your computer is always closer than the cloud.

Storage Software: Offline Access

- People **want** offline copies of their data
- Improves speed, availability and redundancy

Scenario:

- Roger is keeping a spreadsheet with Rafa
- Roger syncs copy to his laptop and edit
- Roger wants to see data on laptop from phone

Things to think about:

- Conflict resolution increases application complexity
- Offline codes is often very application specific
- Do users really need peer-to-peer synchronization?

Questions

Round tables at 4 PM:

- Using Google's Computational Infrastructure
  - Brian Bershad & David Konerding
- Planet-Scale Storage
  - Andrew Fikes & Yonatan Zunger
- Storage, Large-Scale Data Processing, Systems
  - Jeff Dean

Additional Slides

Storage Challenge: Complexity

Scenario: Read 10k from Spanner

1. Lookup names of 3 replicas
2. Lookup location of 1 replica
3. Read data from replicas
    1. Lookup data locations from GFS
    2. Read data from storage node
        1. Read from Linux file system

Layers:

- Generate API impedence mismatches
- Have numerous failure and queuing points
- Make capacity and perf. prediction super-hard
- Make optimization and tuning very difficult

Storage Software: File Transfer

Common instigators of data transfer:

- Publishing production data (e.g. base index)
- Insufficient cluster capacity (disk or CPU)
- System and software upgrades

Moving data is:

- Hard: Many moving parts, and different priorities
- Expensive & time-consuming: Networks involved

Our system:

- Optimized for large, latency-insensitive networks
- Uses large windows and constant-bit rate UDP
- Produces smoother flow than TCP