

Agenda

•Last time (Thurs night / Fri afternoon)

- Chpt 13: Transactions
- Chpt 15: Replication

•Today (Google day)

- The Anatomy of a Large-Scale Hypertextual Web Search Engine (Computer Networks, 1998)
- Google File System (SOSP 2003)
- MapReduce: Simplified Data Processing on Large Clusters (OSDI 2004)

•Thurs Apr 26: no class

•Tuesday May 1

- Finish off Google
- Class Wrap-up (and **HW#5 Due**)

•Fri Apr 11 (2-5pm): Final (closed books, closed notes)

CS451: Distributed Systems (Spring 2007)

Problem Statement

•How to design/implement search engine for Web (circa 2001):

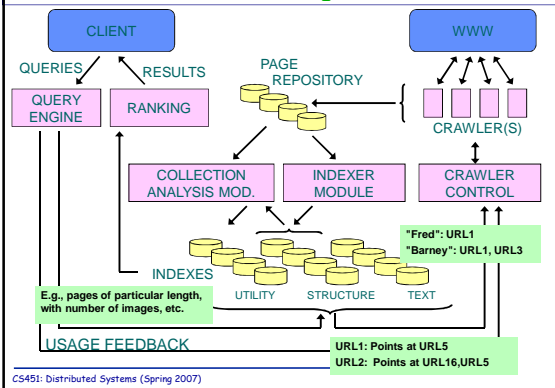
- Over 1 billion pages (and growing)
- Continuously updated
 - 23 % changed daily
 - In 10 days half the pages are gone
- Interlinked

•Basic problem statement

- Input: list of keywords
- Output: ordered list of Web pages

CS451: Distributed Systems (Spring 2007)

General Web Search Engine Architecture



CS451: Distributed Systems (Spring 2007)

Storage Considerations in the Generic Architecture

•Challenges

- Scalability
- Two Access Modes
 - Random Access
 - Streaming Access: e.g., by the indexer to process in bulk
- High rate of modifications
- Obsolete pages

•Key issues (with cluster of interconnected storage nodes)

- Page distribution across nodes (e.g., uniform, hash)
- Physical page organization within a node
- Update strategy

CS451: Distributed Systems (Spring 2007)

Our CS451 Interest

•Not ...

- ... the particular way in which the next page is selected to crawl
- ... the overall utility/correctness of response of the search engine to the particular query

•Rather:

- ... The architectural model
- ... the design goals and analysis of bottlenecks
- ... the coordination between distributed components in the architecture
- ... the organization of information at this scale
- ... the large-scale programming model
- ... what did google use that already existed, and what did they feel they needed to create?

CS451: Distributed Systems (Spring 2007)

Anatomy paper

•Computer Networks and ISDN Networks, vol 30, 1998.

•350 million pages (Jul 1998)

•State of the art at the time:

- Search engines: altavista, lycos
- Hierarchical directories: yahoo

•Metrics

- Recall: % of relevant pages that are returned
- Precision: % of returned pages that are relevant
- Precision of top 10 results

CS451: Distributed Systems (Spring 2007)

Why Google?

•Goal of Google

- “improve the quality of search engines”
- “As of November 1997, only one of the top four commercial search engines finds itself (returns its own search page in response to its name in the top ten results).”
- Key insight: use linking structure

CS451: Distributed Systems (Spring 2007)

Page Rank

•Intuitively, the number of pages pointing at the page in question or if a few important pages point at it

•Let A be the page, $T_1..T_n$ point at it, d is “damping factor” (0-1, usually 0.85), $C(A)$: # of links going out of A

$$PR(A) = (1-d) + d (PR(T_1)/C(T_1) + \dots + PR(T_n)/C(T_n))$$

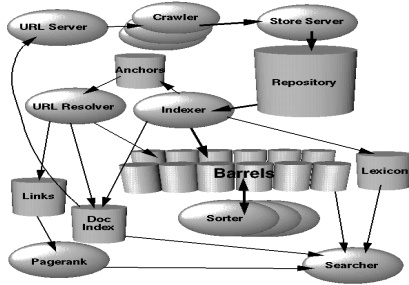
•Intuitive justification

- “We assume there is a “random surfer” who is given a web page at random and keeps clicking on links, never hitting “back” but eventually gets bored and starts on another random page”
- “The probability that the random surfer visits a page is its PageRank.”
- “the d damping factor is the probability at each page the ‘random surfer’ will get bored and request another random page.”

•“A PageRank for 26 million web pages can be computed in a few hours on a medium-size workstation”

CS451: Distributed Systems (Spring 2007)

Google Architecture



CS451: Distributed Systems (Spring 2007)

Paper #1: Why a new file system (GFS)?

•Component failures are the norm

- needs: constant monitoring, error detection, fault tolerance, automatic recovery

•Files are huge

- Unwieldy to manage billions of KB-sized files.
- Multi GB not uncommon.
- design assumptions need to be reconsidered (e.g., block sizes)

•Modification is by appending

- Reading is sequential, mostly
- no more caching data blocks in client

•Co-design of application and the file system API

- Hmmm.... Not generally a good idea*

CS451: Distributed Systems (Spring 2007)

Design Overview: Assumptions

•System built from many inexpensive commodity components (that fail)

•System stores “modest” number of large files.

- Few million, each typically 100 MB. Multi-GB common.
- Small files must be supported, but need not optimize.

•Workload is primarily:

- Large streaming reads
- Small random reads
- Many large sequential appends.

•Must efficiently implement concurrent, atomic appends.

- Producer-consumer queues.
- Many-way merging.

•High sustained bandwidth is more important than low latency

CS451: Distributed Systems (Spring 2007)

Design Overview: Interface

•POSIX-like (although NOT POSIX)

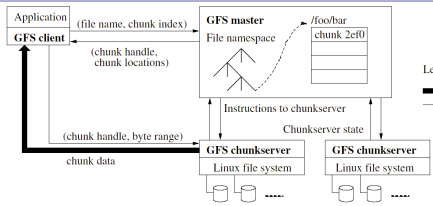
•User-level

•Two new ops:

- Snapshots*
- Record append*

CS451: Distributed Systems (Spring 2007)

Design Overview: Architecture



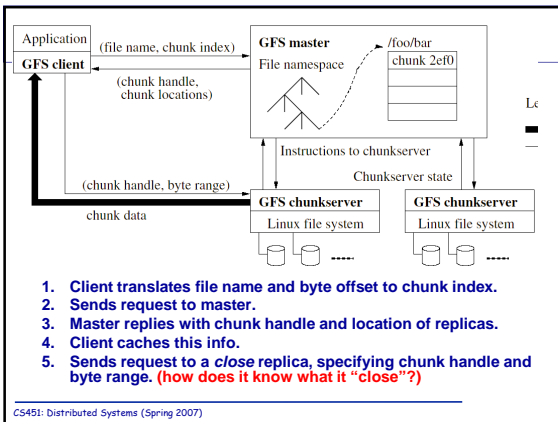
- **Single master, multiple chunkservers, multiple clients.**
- **Files divided into fixed-size chunks.**
 - Each chunk identified by immutable and globally unique chunk handle.
 - Stored by chunkservers locally as regular files.
 - Each chunk is replicated (by default 3)
- **Master maintains all metadata.**
 - Namespaces
 - Access control (*What does this say about security?*)
 - Heartbeat
- **No client side caching because streaming access.**
- **What about server side?**

CS451: Distributed Systems (Spring 2007)

Design Overview: Single Master

- **A single master simplifies, but...**
- ... **General disadvantages for distributed systems:**
 - Single point of failure
 - Bottleneck (scalability)
- **Solution**
 - Clients use master only for metadata, not reading/writing.

CS451: Distributed Systems (Spring 2007)



1. Client translates file name and byte offset to chunk index.
2. Sends request to master.
3. Master replies with chunk handle and location of replicas.
4. Client caches this info.
5. Sends request to a *close* replica, specifying chunk handle and byte range. (*how does it know what it "close"?*)

CS451: Distributed Systems (Spring 2007)

Design Overview: Chunk Size

- **Key design parameter: chose 64 MB.**
- **Each chunk is a plain Linux file.**
- **Pros and Cons of a large chunk size?**
- **Hotspots: Some files may be accessed too much, such as an executable.**
 - Fixed by storing such files with a high replication factor.

CS451: Distributed Systems (Spring 2007)

Design Overview: Metadata

- **Three types:**
 - File and chunk namespaces
 - Mapping from files to chunks
 - Locations of chunk replicas
- **All metadata is in memory.**
 - First two use an operations log for recovery.
 - Second is obtained by querying chunkservers.

CS451: Distributed Systems (Spring 2007)

Metadata: In-Memory Data Structures

- **Metadata stored in memory.**
- **Easy and efficient to periodically scan through state.**
 - Chunk garbage collection
 - Re-replication in the presence of chunkserver failure.
 - Chunk migration for load balancing.
- **Capacity of system limited by memory of master.**
- **Memory is cheap.**

CS451: Distributed Systems (Spring 2007)

Metadata: Chunk Locations

- Master polls chunkserver at startup.
- Initially kept at master, but deemed unnecessary and too complex. (*Why?*)

CS451: Distributed Systems (Spring 2007)

Metadata: Operation Log

- Historical record of metadata changes.
- Replicated on remote machines, operations are logged synchronously.
- Checkpoints used to bound startup time.
- Checkpoints created in background.

CS451: Distributed Systems (Spring 2007)

GFS Consistency: terms

Two new terms

Consistent:

All chunk servers have the same data

Defined:

The result of the "last write" is fully available
A defined chunk is also a consistent chunk

("defined" is stronger)

Questions:

- Is data corrupt if it is inconsistent?
- Is data corrupt if it is undefined?
- Can applications use data in either state?

CS451: Distributed Systems (Spring 2007)

Design Overview: Consistency Model

- Relaxed consistency model
- File creation is atomic.
 - Relatively simple, since just a single master.
- Consider a set of data modifications, and a set of reads all executed by different clients. Furthermore, assume that the reads are executed a "sufficient" time after the writes.
 - Consistent if all clients see the same thing (regardless of replica)
 - Defined if all clients see the modification in its entirety (atomic).

	Write	Record Append
Serial success	Defined	Defined, but interspersed with inconsistent
Concurrent success	Consistent but undefined	Interspersed with inconsistent
Failure	Inconsistent	

CS451: Distributed Systems (Spring 2007)

Reading Concurrently

- Apparently all bets are off.
- Clients cache chunk locations.
- Seems to not be a problem since most of their modifications are record appends.

CS451: Distributed Systems (Spring 2007)

Implications for Applications

- Some of the work that might normally be done by the file system has been moved into the application.
 - Self-validating, self-identifying records.
 - Idempotency through unique serial numbers.

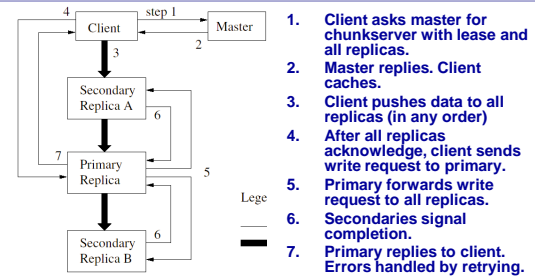
CS451: Distributed Systems (Spring 2007)

System Interactions: Leases and Mutation Order

- **How does the client, master, and chunkserver interact to implement data mutations, atomic record append, and shapshot?**
- Each modification is performed at all replicas.
- Maintain consistent order by having a single *primary* chunkserver specify the order.
- Primary chunkservers are maintained with leases (60 s default).

CS451: Distributed Systems (Spring 2007)

System Interactions: Sample Write of Data



If write straddles chunks, broken down into multiple writes, which causes **consistent but undefined** states (can contain fragments from different clients)

CS451: Distributed Systems (Spring 2007)

System Interactions: Data Flow

- To conserve bandwidth, data flows linearly along chain of chunkservers
- Pipelining is used to minimize latency and maximize throughput.
- **As opposed to what?**

CS451: Distributed Systems (Spring 2007)

System Interactions: Atomic Record Appends

- **Traditional write: client specifies offset at which data is to be written**
- 1. Client pushes data to all replicas (of the last chunk of the file)
- 2. Sends request to primary. Primary:
 - Pads current chunk if necessary, telling client to retry.
 - Writes data, tells replicas to do the same.
- 3. Failures may cause record to be duplicated. These are handled by the client.
 - Data may be different at each replica.

CS451: Distributed Systems (Spring 2007)

System Interactions: Snapshot

- A "snapshot" is a copy of a system at a moment in time.
 - When are snapshots useful?
 - Does "cp -r" generate snapshots?
- Handled using **copy-on-write (COW)** – like AFS.
 - When Master receives snapshot request, first revoke all leases.
 - Then duplicate the metadata, but point to the same chunks.
 - When a client requests a write, the master allocates a new chunk handle.
 - Each chunkserver is asked to create a duplicate chunk locally for the new chunk handle.

CS451: Distributed Systems (Spring 2007)

Master Operation: Namespace Management and Locking

- **Need locking to prevent:**
 - Two clients from trying to create the same file at the same time.
 - Changes to a directory tree during snapshotting.
- **What does the above really mean?**
- **Solution:**
 - Lock intervening directories in read mode.
 - Lock final file or directory in write mode.
 - For snapshot lock source and target in write mode.

CS451: Distributed Systems (Spring 2007)