Parallel & Distributed Data Management

Kai Shen

Data Management

- Data management
  - Efficiency: fast reads/writes
  - Durability and consistency: data is safe and sound despite failures
  - Usability: convenient interfaces for application developers
- Scalable data management
  - Large volume and high volume of accesses, for example, in data centers
  - Parallelism enables high efficiency
  - Distributed systems are necessary for reliability

Google File System

- Google File System (or Colossus) works on such assumptions
  - A modest number of large files (100s of MBs or GBs or more); large-grained accesses are common ⇨ throughput is more important than latency
  - Reads are more often than writes ⇨ replication is cost-effective
- Design/implementation
  - Chunk-based (64MB per chunk), a single large file contains multiple chunks which are distributed (allowing parallel accesses)
  - Many chunk servers, but one master node

Reliability

- Google file system employs replication (default 3) for reliability, but consistent replication for mutable data is challenging:
  - Writes have to reach every replica
  - Writes are done in the same order on all replicas
  - A client may read stale data from a replica who is behind on applying writes
  - What consistency model is supported?
- Availability
  - Single master design is simpler (no need for distributed consensus), but requires quick, consistent recovery from failure
  - Replication of the master, but only one allows writes

Unstructured vs. Structured Data

- Is file system API convenient for application developers?
- Data model
  - Unstructured data: General byte stream in a file
  - Structured data: Specific, regular data organization that contains semantics to enable powerful/flexible search and update
- Most prominent example of structured data management: relational database
  - Data is organized into tables, views, keys (references), indexes etc. that allow SQL queries and updates with potential multiple table joins

Example SQL Statement (TPC-H Q2)

```
select
    s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
from
    part, supplier, partsupp, nation, region
where
    p_partkey = ps_partkey and s_suppkey = ps_suppkey and p_size = 22
    and p_type like '%COPPER' and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey and r_name = 'AFRICA'
    and ps_supplycost = (
        select min(ps_supplycost)
        from partsupp, supplier, nation, region
        where p_partkey = ps_partkey and s_suppkey = ps_suppkey
            and s_nationkey = n_nationkey and n_regionkey = r_regionkey
            and r_name = 'AFRICA'
    )
```


Relational Databases

- Your favorite relational database?
- Database typically runs on one machine.
- For scalability and reliability ⇨ parallel/distributed databases:
  - Data partitioning makes distributed query complex and expensive
  - Data replication makes it challenging to maintain data consistency in updates
  - Transactions are already complex (much more complex if transactions commit in a distributed fashion)

Key-value/Nosql Store

- There is space between no-structure (plain file system) and strongly-structured data (relational DBs supporting SQL)
- Key-value or hash table store
  - Data is organized into sets of key-value pairs
  - Support lookup(key), insert(key,value), delete(key), and replace(key,new_value)
- What data fits the key-value model?
  - Webtable: key is the URL, value is the web page content
  - What else?
- Nosql data stores

Specific Key-value Stores

- Before the times of relational databases
  - dbm, developed by Ken Thompson (AT&T), 1970s
- After the wide uses of relational databases
  - TokyoCabinet/KyotoCabinet, 2000s
  - LevelDB, 2010s
  - … …

Key-value Stores vs. Relational DBs

- Data semantics isn't as flexible
  - What if your data semantics does fit into one single table; semantic linking/joining of multiple tables are needed to answer queries or support updates?
- Easier to scale up
  - Operations are uniformly simple, each operating on a single data item (no whole table scans or cross-table joins)
  - Data partitioning won't cause distributed operations
  - Replication consistency is easier to maintain
- And leaner and faster
  - Really? Some argue SQLite is lean, fast, AND supporting SQL.

Scalable Structured Store: Bigtable

- Developed at Google [Chang et al. 2006]
- Data model
  - Enhanced key-value model
  - (row:string, column:string, time:int64) → string
  - Two-dimensional key allows multiple attributes for each key: in Webtable, a web page has content, incoming references, other labels (spam, …), see Figure 1 of the paper
  - Timestamp allows version management (earlier versions may be useful; allows garbage collection)

Bigtable: Data Locality

- Data layout
  - Data in lexicographic order by row key
  - Applications should devise the row key in a way such that rows often referenced together have lexicographically nearby keys
    - Reversing URL hostname components for row keys in Webtable "www.google.com/index.html" → "com.google.www/index.html"
- Processing near data
  - Data processing scripts can be supplied to run at data server

Bigtable: Distributed Organization

- Centralized or decentralized management?
  - A master server and many tablet servers
  - A special METADATA table that records the location of user tables
- Scalability and robustness in centralized management
  - Relieve the master from common tasks → won't become the scaling bottleneck
  - Fast recovery in case of master failure

Bigtable: Distributed Consistency

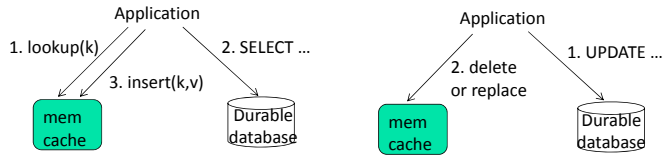
- Maintain consistency in distributed system
  - Ensure that everyone agrees with one master at a time
  - Ensure that everyone agrees with the root METADATA table
  - ⇨ Distributed consensus
- Paxos distributed consensus
  - Google's implementation: Chubby lock service

Bigtable: Performance

- Figure 6 of the paper

memcache

- In-memory key-value store, not durable
- Limited memory space, older things are evicted (following LRU order) when space runs out, effectively a cache
- Typically work together with a durable store

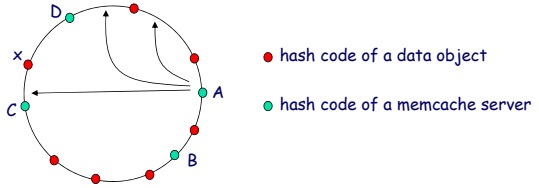


Distributed memcache [Nishtala et al. 2013]

- Key idea:
  - Decouple the performance/scalability from I/O and durability, if the workload is read-mostly
- A two-layer data management system: distributed memcache and durable data store (relational databases or …)
  - Both layers handle writes
  - Distributed memcache handles reads alone (mostly)
  - If the workload is read-mostly, only the distributed memcache layer needs to be scalable and fast
  - Can work with legacy databases that don't scale well
- Used in many places including Facebook

Distributed memcache: Scalabilty

- Since memcache isn't concerned with durability or I/O, its scalability is easier to accomplish:
  - A balanced data partition to many memcache servers
  - A scalable, robust, distributed algorithm to tell which memcache server has your data



Bigtable vs. Distributed memcache

- Performance / scalability?
- A memcache system is fundamentally constrained by the underlying durable data store if application performs a nontrivial amount of writes.

Megastore

- A recent Google system [Baker et al. 2011] that is a step closer to relational databases
  - Support relational DB-like data model but do not support operations that hinder scalability (e.g., SQL table joins)
  - SQL table joins can be implemented by applications after initial lookups on the based tables with indexes
- A new point in the flexibility vs. scalability design space:
  - Bridge to the SQL world, but require applications to explicitly support expensive operations

Summary

- A variety of data management systems
  - Unstructured file system
  - Relational databases with SQL support (MySQL, SQLite, Oracle, Microsoft SQL Server, …)
  - Single machine key-value stores (KyotoCabinet, LevelDB, …)
  - Bigtable
  - Distributed memcache
  - Megastore
  - … …
- What is right for my scalable data management application?
  - Data model and data access semantics
  - Scalability / performance requirements
  - Ease of development and ease of use