Distributed Systems

19. Spanner

Paul Krzyzanowski
Rutgers University
Fall 2017

Spanner

Spanner (Google’s successor to Bigtable ... sort of)

Spanner

• Globally distributed multi-version database
• ACID (general purpose transactions)
• Schematized tables (Semi-relational)
  - Built on top of a key-value based implementation
  - SQL-like queries
• Lock-free distributed read transactions

Goal: make it easy for programmers to use
Working with eventual consistency & merging is hard => don't make developers deal with it

Data Storage

• Universe holds 1 or more databases
  - Database holds 1 or more tables
  - Table = arbitrary number of rows and columns
  • Table storage may be interleaved
  • All data in a table has version information (timestamp)
• Shards (tables) are replicated
  - Synchronous replication via Paxos
• Transactions across shards use 2-phase commit
• Directory = set of contiguous keys
  - Unit of data allocation
  - Granularity for data movement between Paxos groups
  • Done in background

Transactions

• ACID properties
• Transactions are serialized: strict 2-phase locking used

1. Acquire all locks
   - do work
2. Get a commit timestamp
3. Log the commit timestamp via Paxos to majority of replicas
4. Do the commit
   - Apply changes locally & to replicas
5. Release locks

Data Storage

• Tables sharded across rows into tablets (like Bigtable)
• Tablets stored in spanservers
• 1000s of spanservers per zone
  - Collection of servers – can be run independently

• Zonemaster allocates data to spanservers
• Location proxies – Used by clients to locate spanservers that hold the data they need
• UniverseMaster – status of all zones
• Placement driver – transfers data between zones
2-Phase locking can be slow

We can use read locks and write locks
But
  - read locks block behind write locks
  - write locks block behind read locks

Multiversion concurrency to the rescue!
  - Take a snapshot of the database for transactions up to a point in time
  - You can read old data without getting a lock
  - Because you are reading before a specific point in time
    • Results are consistent

We need commit timestamps that will enable meaningful snapshots

Getting good commit timestamps

• Vector clocks work
  - Pass along current server's notion of time with each message
  - Receiver updates its concept of time (if necessary)

• But not feasible in large systems
  - Pain in HTML (have to embed vector timestamp in HTTP transaction)
  - Doesn’t work if you introduce things like phone call logs

• Spanner: use physical timestamps
  - If $T_1$ commits before $T_2$, $T_1$ must get a smaller timestamp
  - Commit order matches global wall-time order

TrueTime

Remember: we can’t know global time across servers!

• Global wall-clock time = time + interval of uncertainty
  - $TT.now().earliest$ = time guaranteed to be $<=$ current time
  - $TT.now().latest$ = time guaranteed to be $>=$ current time

• Each data center has a GPS receiver & atomic clock
  - Atomic clock synchronized with GPS receivers
    • Validates GPS receivers
  - Spanservers periodically synchronize with time servers
    • Know uncertainty based on interval
    • Synchronize ~ every 30 seconds: clock uncertainty < 10 ms

Commit Wait

We don’t know the exact time
  ... but we can wait out the uncertainty

1. Acquire all locks
  - do work
2. Get a commit timestamp: $t = TT.now().latest$
3. Commit wait: wait until $TT.now().earliest$ $>$ $t$
4. Commit
5. Release locks

Integrate replication with concurrency control

1. Acquire all locks
   - do work
2. Get a commit timestamp: $t = TT.now().latest$
3. (a) Start consensus for replication
   (b) Commit wait (in parallel)
4. Commit
5. Release locks

Make the replicas & wait for all to finish

Spanner Summary

• Semi-relational database of tables
  - Supports externally consistent distributed transactions
  - No need for users to try deal with eventual consistency

• Multi-version database
  - Synchronous replication

• Scales to millions of machines in hundreds of data centers
  - SQL-based query language

• Used in F1, the system behind Google’s Adwords platform
• May be used in Gmail & Google search
Spanner Conclusion

- **ACID semantics not sacrificed**
  - Life gets easy for programmers
  - Programmers don’t need to deal with eventual consistency

- **Wide-area distributed transactions built-in**
  - Bigtable did not support distributed transactions
  - Programmers had to write their own
  - Easier if programmers don’t have to get 2PC right

- **Clock uncertainty is known to programmers**
  - You can wait it out