Cassandra – A Decentralized Structured Storage System

Avinash Lakshman, Prashant Malik

LADIS 2009

Anand Iyer

CS 294-110, Fall 2015

Historic Context

- Early & mid 2000:
  - Web applications grow at tremendous rates
  - Big data
  - 3 V's (Volume, Velocity, Variety) places lots of demands
  - Need for scalable and flexible data storage solutions
- Traditional solutions
  - Open source (MySQL, Postgres)
    - Scalability, elasticity
  - Commercial (Oracle, SQL Server)
    - Cost, dependency
  - SQL not really amenable to distributed operations
- Leads to NoSQL revolution

Historic Context

Big Table (OSDI 2006)

- Multiple attribute based access
- Master slave architecture
- Consistency over availability

Dynamo (SOSP 2007)

- User shopping cart storage
- Decentralized
- High availability ($$$)
- Simpler KV model

Problem

- Facebook Inbox Search
  - Enable users to search through inbox
    - Multiple attributes
  - Manage data spread across multiple datacenters
  - Provide high availability and no single point of failure (Why?)
  - Treat failures as the norm.

BigTable + Dynamo = 

Key Idea (1)

- Leverage partitioning and replication techniques from Dynamo
  - Partition based on consistent hashing (like Dynamo)
  - Load balance by moving lightly loaded nodes on the ring
  - Replicate based on policies
    - Dynamo style replication for simple policies
    - Zookeeper based for more involved policies (e.g. Rack Aware)

Key Idea (2)

- Use a richer data model similar to Big Table
  - Distributed multidimensional map
  - Column Families (similar to BigTable) and Supercolumns
  - Persistence using commit logs, memtables and SSTable compaction
- Later versions of Cassandra made changes to many of these

Fundamental Tradeoffs

- Tradeoff #1: Between consistency and availability in the face of network partitions (CAP theorem)
- Favor availability over consistency
- Allow tunable consistency
  - Quorum based
    - Strict quorum => strong consistency (R + W > N)
    - Partial quorum => eventual consistency (R+W <= N)
- Tradeoff #2: Latency v/s consistency guarantee during normal operations

Influence

- Highly popular, one of the most popular NoSQL stores.
- Installation at 1500+ companies including eBay, Netflix, Github, etc.
- Largest known deployment at Apple, with over 75,000 nodes storing over 10 PB of data.
  - But also acquired FoundationDB recently!
- Influenced the development of several other NoSQL databases.

Impact of NoSQL

- Of course DB folks are not happy
  - "Eventual consistency = creates garbage" (Michael Stonebraker, LISA 2011)
- Several attempts to scale traditional DBMS
  - MySQL Cluster
  - VoltDB
    - Claims 5-7X improvements over Cassandra
- NewSQL
  - Offer SQL and ACID with NoSQL's scalability
  - Focus on reducing overheads using systems techniques
    - But don't give up SQL or ACID
    - Mostly in-memory

At the same time…

- Cassandra 2.0:
  - Secondary index
  - CQL (SQL like interface)
  - Lightweight transactions (ACID)
  - Triggers (stored procedures)
- Cassandra 3.0:
  - Materialized views
  - UDF, UDAs

Going Forward

- Storage problem will likely worsen
  - Easier to collect data
    - Machine generated
  - Mobile App Era
    - Billions of smartphone users
    - Mobile technology improving
  - Internet-of-Things
    - Variety will increase, flexible schema more important
- BigTable/Cassandra will remain influential, but what about the consistency semantics?

Is eventual consistency really garbage?

- PBS (Bailis et. al.): *"in practice, and in the average case, eventually consistent data stores often deliver consistent data."*
- Facebook's move to HBase
  - *"We found Cassandra's eventual consistency model to be a difficult pattern to reconcile for our new Messages infrastructure."*
- Newer systems have transactions as a primary goal
  - Google Spanner (OSDI 12, later in the class)
    - *"We believe it is better to have application programmers deal with performance problems due to overuse of transactions as bottlenecks arise, rather than always coding around the lack of transactions."*
  - Google F1 (SIGMOD 12, later in the class)
    - *"We also have a lot of experience with eventual consistency systems at Google. In all such systems, we find developers spend a significant fraction of their time building extremely complex and error-prone mechanisms to cope with eventual consistency and handle data that may be out of date.*
- Is this a characteristic of new emerging workloads/settings?