Voldemort

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

Outline

Data Intensive Web Sites

Many data intensive sites contain the following features:

- People you may know ...
- Items you may like (recommendations)
- Relationships between pairs of people
- LinkedIn (135 million users) features many more such relationships

Phases

Three phases : data collection, processing, serving

LinkedIn

- Voldemort is a key-value system for finally serving the data.
- LinkedIn uses Voldemort (which is now open source)
- Several other sites such eHarmony, and Nokia use Voldemort
- A Hadoop engine processes all the data in LinkedIn's data store, and creates a read-only database that Voldemort uses
- Support for : quick updates , load balancing
- Optimized for serving bulk read-only data (quickly)
- Twice as fast as MySQL (4TB of new data every day)

Related Work

- Two MySQL solutions: MyISAM and InnoDB
- MyISAM
  - Compact on-disk data structure
  - Creates index after loading the data file
  - Locks the complete table (during loading)
- InnoDb
  - Supports fine-grained row-level locking
  - Very slow
  - Requires a lot of disk space
- PNUTS (Yahoo)
  - CPUs are shared between data loading modules and data serving modules
  - Reduces the performance of data serving modules

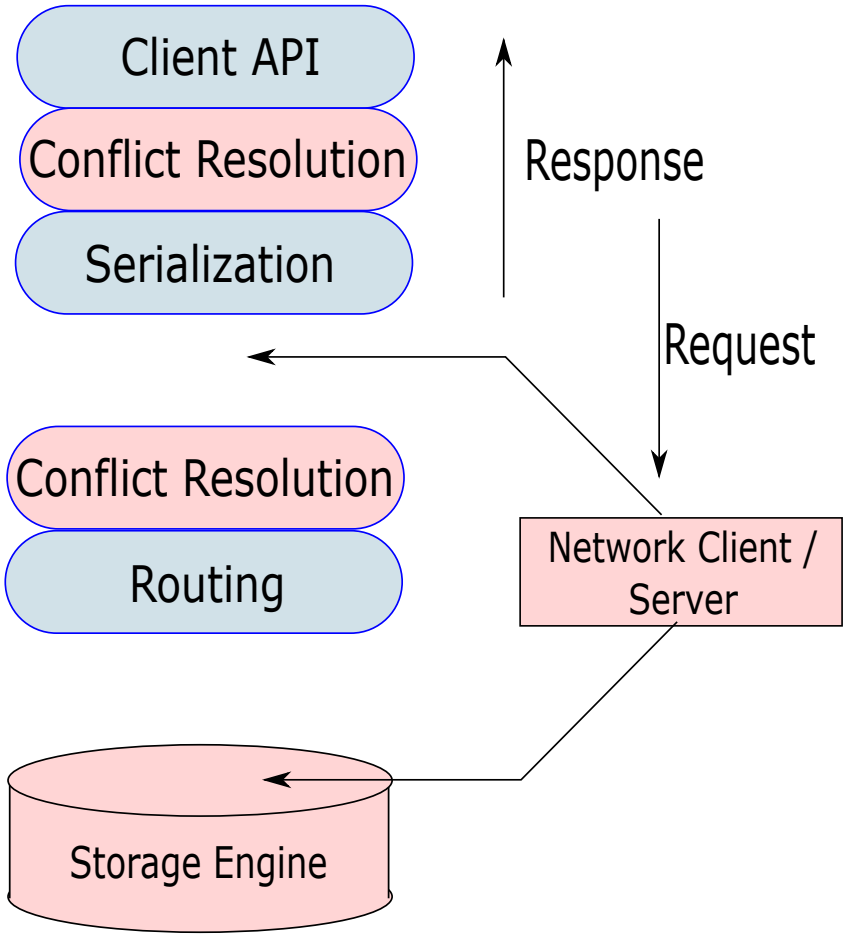
Outline

Overall Structure

Clusters, Nodes and Stores

- A Voldemort *cluster* contains multiple *nodes*
- A physical host can run multiple nodes
- Each node has a given number of stores (database table)
- Example:
  - Store : member id $\rightarrow$ recommended group ids
  - Store : group id $\rightarrow$ description
- Attributes of a store:
  - Replication factor $\rightarrow$ number of nodes that contain this store
  - Read/write quorum size
  - Serialization (XML/JSON) and compression
  - Storage engine: Berkeley DB or MySQL

Responses and Requests



Outline

Routing Module

- Deals with partitioning and replication
- Splits the hash ring into equal size partitions
- Maps partitions to nodes
- Each key is map to a preference list (of partitions)
  - Map to a primary partition (based on its hash)
  - $N - 1$ subsequent partitions (clockwise)
- Pluggable Storage Layer
  - Traditional get and put functions
  - Block-read functions (streaming)
  - Read-only operations
  - Administrative Functions

Routing Modes

Definition (Routing Modes)

Client Side Routing Retrieves meta-data and cluster topology. Makes the routing decisions locally.

Server Side Routing Server takes all the routing decisions.

Outline

Shortcomings of MySQL and BDB

- For bulk loading, multiple *put* requests is not the correct solution.
- We need to update the underlying B+ tree multiple times.
- Alternative Solution
  - Maintain a separate server that maintains a copy of the database
  - Switch to the new copy instantaneously
  - DISADVANTAGE : Double the resources, bulk copy overhead
- Alternative Solution 2
  - Run Hadoop to generate indices offline
  - Atomically switch to the new set of indices
  - Problem of additional resources

Requirements

- Minimize the performance overhead of live requests.
- Scaling and fault tolerance
- Fast rollback capability
- Ability to handle large datasets

List of Steps

1. Driver program sends a message to HDFS – Trigger Build
2. The Hadoop+HDFS systems starts the build.
3. Driver program sends a message to the Voldemort cluster – Trigger Fetch
4. The Voldemort cluster initiates a parallel fetch from the Hadoop nodes
5. Driver program sends a message to the Voldemort cluster – Trigger Swap
6. Voldemort executes the swap

Storage Format

- Voldemort is Java based (on a JVM), and uses the OS to manage its memory
- The input data destined for a node is split into multiple chunk buckets
- Each chunk bucket is split into multiple chunk sets
- A chunk bucket is defined by the partition id and replica id
- Each chunk set has a : data file and an index file
- Naming convention of a chunk set file
  - partition id_replica id_chunk set id.{data,index}
- Entry in the index file: top 8 bytes of MD5 signature + 4 byte offset in the data file

Structure of the Data File

- Stores the number of collided tuples
- Plus, list of collided tuples
  - key size, value size, key value

Chunk Set Generation

- Inputs: chunk sets per bucket, cluster topology, store definitions, input data locations in HDFS
- Mapper
  - Emits upper 8 bytes of the MD5 key along with node id, partition id, replica id, key, and value
- Partitioner
  - Route data to the correct reducer based on the chunk set id
- Reducer
  - Every reducer is responsible for only one chunk set
  - Hadoop sorts the inputs based on the keys
  - Each Voldemort node is a directory in HDFS files with files for each chunk set.

Data Versioning

- Every store is represented by a directory
- Every version of a store has a unique directory
- The current version of a store points to the right directory using a symbolic link
- For moving to a new version:
  - Get a read-write lock on the previous version's directory
  - Close all the files
  - Open the files in the new directory and map them to memory
  - Switch the symbolic link

Data Loading

- The driver triggers a Hadoop job to create a new version.
- It then triggers a fetch request on all the Voldemort nodes.
- Use a pull based modeling (traffic based fetch throttling)
- Swap the version's directory
- Global atomic semantics
- Takes 0.05 ms in LinkedIn

Outline

Retrieval

- Calculate the MD5 of the key
- Generate primary partition id, any replica id, chunk set id (first 4 bits of MD5)
- Find the chunk set index file
- Locate the value in the chunk set data file
- Searching in the index file is the most time consuming process
  - Ensure that they are in main memory by fetching them at the end
  - User interpolation search (O(log(log(N))))

Schema Upgrades and Rebalancing

Schema Upgrades

- Use version bits with each JSON file
- The mapping of the version bits to the schema is saved in the meta-data section of the store definition.

Rebalancing

- We can dynamically add partitions.
- Create a plan for moving partitions and their replicas.
- Start moving the partitions, and lazily propagate information about temporary topologies.

Setup

- Simulated data set: Keys (random 1024 byte strings)
- Linux based setup : Dual CPU
- 8 cores, 24 GB RAM
- Number of nodes: 25, 940 GB data size per node, 123 stores, replication factor: 2, store size range (700 KB to 4.15 TB), maximum number of store swaps per day: 76

Build Time and Read Latency

- If we increase the file size from 1 GB to 1700 GB
  - The build time for MySQL increases lineary from 0 till 350 minutes (for 125 GB). Results are not shown beyond 125 GB.
  - For Voldemort, the build time increases linearly from 0 to 40 mins.
- Read latency (y axis) vs time since swap in minutes (x axis)
  - The MySQL median read latency reduces from 32ms (1 min) to 2ms (180 mins)
  - The interpolated and binary values for Voldemort reduce from 2ms to 0.1 ms (same range)

Queries Per Second

- For the same 100 GB dataset, the throughput (queries per second) was varied and the latency was measured.
- For MySQL the latency increased from 1.7 ms (100 qps) to 3.3 ms (420 qps). The rate followed increased steeply after 300 qps
- Voldemort's latency increased from 1.2 ms (100 qps) to 3.5 ms (700 qps).
- Conclusion: For the same latency, Voldemort can support twice the throughput.

People You May Know and Collaborative Filtering

- The latency was measured after a swap for two data sets: People You May Know (PYMK) and Collaborative Filtering (CF)
  - PYMK : A suggested set of users that the given user may like to establish connections with.
  - CF : Profiles similar to the visited member's profile that were viewed in the same session.
- The latency was plotted after a swap .
- In both cases the latencies decrease sub-linearly .
- CF has a larger latency than PYMK because of the larger size of the value .


Sumbaly, Roshan, et al. "Serving large-scale batch computed data with project voldemort." Proceedings of the 10th USENIX conference on File and Storage Technologies. USENIX Association, 2012.