Paper Review –8

Voldemort Extension

Title: Serving Large-Scale Batch Computed Data with Project Voldemort

Summary

The current serving system lacks the ability to load bulk data without affecting performance of a cluster. Since database engines tend to focus on consistency rather than availability. This leads to performance degradation and are not suitable for certain use cases at LinkedIn. In order to support bulk loading without losing on performance, Voldemort has been extended to handle large scale loading of bulk data by constructing indexes offline. By far this extension has helped reduce overall latency and increased throughput at LinkedIn as compared to other relational based database engines.

Good Aspects

In order to meet requirements such as PYMK (People you may know) and Collaborative filtering, the engineers have built an extension on Project Voldemort to handle large volume of data. The paper has captured the internal mechanisms and has provided analysis of existing solution by showing how insufficient they are for bulk loading and serving read only data. The paper introduces the internal storage format and indexes are created offline using Hadoop. Appropriate evidence has also been shown both in production and experimental results that how well it performs when compared to existing solutions.

Review

Following details will cover all aspect of the paper providing crucial and also interesting aspects of the paper that appealed the reader.

Related Work

- Before going into the concepts, its important to see what others have done and how far they have succeeded in developing a way to load huge data.
  - MySQL is a common serving system that has two commonly used engines called MyISAM and InnoDB. Bulk support is provided by them via the LOAD DATA INFILE. However, the implementations differ in both the engines where MyISAM locks the entire table while construction and InnoDB does row locking and takes lot of disk space.
  - PNUTS – Uses range-partitioned tables which tries to optimize data movement but how ever incur latency degradation on live serving due to multitenant issues.

Key Contributions

- A scalable offline index construction, based on MapReduce which produces partitioned data for online consumption.
- Complete data cycle to refresh terabytes of data with minimum effect on existing serving latency.
- Custom storage format for static data, which leverages the operating system's page cache for cache management.
- Provide O(1) lookup time.

Architecture

- A cluster can contain multiple nodes and a physical node can run multiple nodes. All nodes in a may cluster may have same number of stores(=tables).

For example - feature dealing with group recommendations will map to two stores:

1. recording a member id to recommended group ids
2. recording a group id to its corresponding description.

- Every store has the following list of configurable parameters
  - Replication Factor (N): Number of nodes where each key-value tuple is replicated.
  - Required Reads: Number of nodes Voldemort reads from, in parallel during a get before declaring a success.
  - Required Writes: Number of node responses Voldemort blocks for, before declaring success during a put.
  - Key/Value serialization and compression: Uses custom BSON format
  - Storage Engine type: Supports any engine.

API

- get
- put

System Requirements

- These are the following characteristics required in a serving system at LinkedIn
    - Minimal performance impact on live requests - the number of get requests must not be impacted during the bulk load.
    - Fault tolerance and scalability
    - Rollback capability – To minimize time in error, it must support very fast rollback to a previous good state.
    - Ability to handle large data sets.

- **Storage Format**

    - Since storage structures try to build data structures in memory, there are several advantages of maintaining the structure in the page cache.
    - Voldemort maps the entire index into the address space and also delegates memory management to the operating system.
    - The input data is split into multiple chunk buckets, which in turn are split into multiple chunk sets.
    - A chunk bucket is defined by the primary partition id and replica id, there by given a unique identifier.
    - The number of chunk sets per bucket is decided by driver program in Hadoop.
    - The index file is a compact structure containing the sorted upper 8 bytes of the MD5 of the key followed by the 4 byte offset of the corresponding value in the data file.

- **Chunk Set Generation**

    - The Hadoop job takes as its input the number of chunk sets per bucket, cluster topology, store definition and the input data location on HDFS.
    - The mapper phase deals with the partitioning of data depending on the routing strategy; The partitioner phase redirects the key to the correct reducer and the reducer phase deals with writing the data to a single chunk set.

- **Data Versioning**
    - Every store is represented by a directory which in turn contain directories corresponding to versions of the data. A symbolic link per store is used to point to the current serving version directory.
    - Every version directory has a configurable number associated to it.

- **Data loading**
    - The initiator of the complete construction is a standalone driver program that constructs, fetches and swaps the data.
    - Once the Hadoop job is complete, the driver triggers a fetch request on all Voldemort nodes.
    - While the data is being streamed from HDFS, the checksum is validated with the checksum from the build step.
    - After the data is available on each node in their new version directory, the driver triggers a swap operation on all nodes.

- **Retrieval**
    - Calculate the MD5 of the key.
    - Generate the primary partition ID, replica ID, chunk set ID
    - Find the corresponding active chunk set files using the 3 variables from the previous step.
    - Performing a search using the top 8 bytes - using interpolation search or binary search.

- **Performance**
    - As compared to the build times of MySQL using the LOAD DATA command, MySQL exhibits very slow build times because it buffers changes to the index before flushing it to the disk.
    - MySQL requires 1.4 times more I/O than Voldemort implementation.
    - Its found that binary search exhibits better median latency since its lookups cache in memory unlike Interpolation which has less lookups hence less data in cache.

Production Applications

- People You May Know (PYMK) data:
    - Users are presented with a suggested set of other users they might know and would like to connect with.
- Collaborative filtering (CF) data set:
    - This feature shows other profiles viewed in the same session as the visited members profile.

Conclusion and Summary

- In conclusion the Voldemort extension has helped LinkedIn achieve high throughput and low latency by using their custom database engine.
- The paper captures the weakness of relational data bases and how they fail badly when scalability arises with huge data.
- Its in production for 2 years and become an integral part of their ecosystem.
- The source is open source and part of Apache software foundation