Paper Review –2

Bigtable

Title: Bigtable: A Distributed Storage System for Structured Data

Summary

The paper presented here provides the design, implementation and evaluation of a distributed storage system for managing structured and semi structured data called Bigtable. The Paper covers various aspects of Bigtable such as Data model and structure of the Table stored. It also presents details about the unit of storage that is a tablet and how efficiently they are stored by utilizing the power of GFS File system. It also captures refinements added to achieve more performance and reliability. At the end the performance is evaluated over different types of operations and over multitude of chubby servers to show performance gains and its applications.

Good Aspects

Bigtable has been designed to work in a distributed system where its designed to reliably scale to petabytes of data over thousands of machines. The very fact that Bigtable is used by many applications with in Google is a strong indicator of the success of Bigtable. It provides an interface that is similar to a relational database yet implements strategies such as parallelism, scalability, high performance and support of a relational model and concurrency support thereby hiding details from the Client. Overall Google have implemented their own storage solution and have great amount of flexibility over their products in a way that has benefited them.

Review

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

Bigtable is a sparse, distributed, persistent multidimensional sorted map. Bigtable itself is an array of clusters where each cluster consists of one or many computers that provide storage for Bigtable data. Bigtable uses master and slave's concept to effectively control the slaves also know as Tablet Servers to meet the storage needs.

Dependencies
Bigtable has certain dependencies in order to function completely. They are

Chubby Service – A chubby service is a service that comprises of distributed Master and Client servers providing lock implementation that helps Bigtable to perform writes in a systematic manner and handle concurrency across multiple servers.

GFS – Google File System is a scalable distributed file system to meet the storage needs of high data intensive applications such as Bigtable.

Technical Details

- **Data Model**
  - Map is indexed by row key, column key and a timestamp.
  - The value is an array of bytes.
  - **Row key** is a string type with size up to **64KB**.
  - Every load and store happens on a unit called Tablet which is a partitioned row range with a start and end position.
  - Since Bigtable stores Tablets in lexicographic order, the applications can leverage this by storing pages in same domain contiguously as they will be grouped together in a tablet server allowing for efficient retrieval.
  - **Column key** is a **string** that can be grouped into sets called Column Families. Bigtable uses smart compression algorithms so applications can take that fact to group related fields together so as to increase compression factor.
  - **Time stamp** is a **64bit** integer and can have different cells. To make management easy, Bigtable has provision to store only the recent timelines and automatically garbage collect cell versions that are older.

- **Client API**
  - Provides API for creating and deleting tables and column families.
  - Provides functions for changing cluster, table and column family metadata.
  - API supports single transactions and Group transactions which are more efficient.
  - Bigtable cells can be used as integer counters.
  - Supports client-supplied scripts that runs in the address space of servers using a language called Sawzall that is built by google to perform data transformations.

- **Architecture**
  - Bigtable depends on cluster management system to manage resources on shared machines, dealing with failures and monitoring of machine status.

  - **SSTable**
    - SSTable is a file format to internally to store Bigtable data.
    - SSTable is a persistent immutable map that maps keys to values where keys and values are arbitrary byte strings.
    - SSTable contains a sequence of blocks where each block is typically 64KB in size and the index is used to locate blocks.
    - SSTable data is stored on memory where the in memory index can be scanned to load the appropriate block from disk.

  - **Chubby Service**
    - Highly available and persistent distributed lock service that consists of five active replicas where one of them is dynamically elected to be a master.
    - Chubby uses **Paxos algorithm** to keep all replicas in sync.
    - A namespace is assigned for each Tablet that consists of directories and files. They store the lock information to make sure writes and reads are atomic.
    - Chubby clients can expire their session in which case a callback can be registered to allow chubby client to know about this and retrigger their new session.

  - **Tablet Server**
    - A Cluster consists of tablet servers that can be dynamically added or removed based on demands by a master tablet server.
    - The master is responsible for assigning tablets to tablet servers, detecting if they are offline, balancing tablet-server load in a cluster and garbage collection of un used tablets.
    - Each Tablet server manages its own tablets and has two writer threads to dispatch and balance work.
    - Clients communicate directly with tablet servers hence the master has less load.

- **Implementation**
  - **Tablet Three Level mapping**
    - A three level mapping is used to store information about a tablet and help locate it.
    - First a **chubby file** has a pointer to the **Root tablet**.
    - The root tablet contains location of all other metadata tablets in a special **METADATA table**.
    - The first entry in this METADATA table is the root tablet info.
    - The root tablet is never split to help bootstrap.
    - The METADATA table stores the location of a tablet under a row key that is an encoding of the tablet start and end row identifier.
    - Using this scheme, it can address $\mathbf{2^{34}}$ Tablets.
    - In order to round trip time, the client application can store the location once in memory and use it through the applications.
    - Sometimes this data can go stale then average of 6 round trips are required for the application to get a new location.

  - **Tablet Assignment**
    - Each tablet is assigned to one tablet server at a time by the master that keeps track of all online tablet servers.
    - It has book keeping data that has information of each tablet and if its assigned or unassigned.
    - Unassigned tablets are assigned to any free tablet server by sending a request.
    - Another use of chubby is that to keep track of all tablet servers.
    - If any tablet server goes down, then its lock namespace will have a lock released which helps identify that the tablet server is offline to help relocate all unassigned tablets of that server to another available tablet server.
    - To ensure that a Bigtable cluster is not vulnerable to any networking issues, if any issues arise between master and chubby, the master kills itself.
    - After restarting, it assigns all unassigned tablets and keeps track of it in the METDATA table.
    - Chubby servers are responsible to split tablets and once a split is done, it will ensure that METADATA table is updated according and informs the master server.

- **Tablet Serving**

  - All recent updates and reads made in the tablet servers are stored in **memtable** inside the memory of the Tablet Server.
  - Contents of memtable are sorted in order to help easy access.
  - All older contents are flushed to the disk by creating a new SSTable.
  - Following operation is done alone by the Tablet Server.
  - All read and write operations are authenticated by using the Chubby server file that stores a list of permitted users

- **Minor and Major Compaction operations**
  - When ever the memtable reaches a threshold, its written to a SSTable into the GFS. This process is called **Minor Compaction**.
  - A merging compaction occurs when few SSTables and memtables are merged to create a new SSTable. This process is called **Major Compaction.**

- **Refinements**
  A number of refinements were required in order to achieve high performance, availability and reliability.

  - **Locality Groups**
    - Segregating column families into different locality groups which are not related is a provision that helps efficient reads and mutations.
    - Useful tuning parameters can be added specifically per locality group like in-memory to help lazy load data.
  - **Compression**
    - Clients have control to compress SSTables inside a locality group.
    - Since locality groups contain related SSTables, clients can take advantage of intelligent compression algorithms that compress data that has high frequency repetitions like example Html content which has lot of repetitions and can be compressed with a large compression factor than Gzip.
  - **Two level Caching**
    - **Scan Cache –** A higher level cache that caches the key-value pairs returned by the SSTable.
    - **Block Cache** – A lower level cache that caches SSTables blocks read from GFS.
  - **Bloom Filters**
    - Bloom filters are **probabilistic data structures** that help identify membership of certain data in SSTable hence its required to check if SSTables are in memory or a disk access is required or not. Bloom filters provide efficient implementation to help determine membership.

- **Commit log implementation**
  - Since commit log for each tablet has network overhead and efficiency implications, commit logs are created per Tablet server and by recording mutations by appending to the commit logs.
  - Duplication of Log reads by sorting and partitioning them to **64MB segments** to achieve parallelism.
  - To avoid bottleneck, two log writer threads in the server switches back and forth depending on the latency.

- **Performance Evaluation**
  - Two write benchmarks -
    - **Sequential write** - 0 to R-1 key partitioned into 10*N clients and clients were chosen by a scheduler that schedules tablets once any tablet server has done processing previous tablet and is free.
    - **Random write** - Same as Sequential write except that row key was hashed to modulo R
  - **Sequential read benchmark** is similar to sequential write and random write except that the value under row is read rather than written.
  - **Scan benchmark** uses support provided by Bigtable for scanning all values in a row range.
  - **Random reads (mem)** similar to random reads satisfied by the in memory table but much faster
  - **Random read** is slower since SSTable is 64KB and single 1000-byte value is retrieved for these type of applications, hence using block size of 8KB can mitigate this effect.
  - Random reads from mem is faster since no need to read the SSTable from the disk.
  - Random and sequential writes are faster since each tablet server appends all incoming writes to a single log and uses group commit to stream them to GFS.
  - Sequential reads are faster than random reads since the block cache stores the nearby SSTables.
  - **Scaling** - aggregate throughput increases as the number of tablet servers increased from 1 – 500 since the bottleneck is individual server.
  - Performance does not increase linearly for some operations like Random read and that is due to the imbalance in the load in multiple servers.

- **Real World Applications of Bigtable**

  - **Google Analytics** – Helps analyze traffic patters of a website via Embedded Javascript code. Two example of tables stored are Raw click table and Summary table that achieves over 14% compression.
  - **Google Earth** – Preprocessing pipelines store the raw images inside the Bigtable and since the images are efficiently compressed, there is no need of Bigtable compression.

- **Personalized Search** – Stores user specific search that is user's data in Bigtable.

Conclusion and Summary

The following paper has carefully analyzed the problems with current relational database and their inability to efficiently store data across a distributed system. Bigtable is able to surpass the expectations of the Implementers by being used by almost all important applications at Google.

Bigtable has sophistication in the way data is represented and stored. It uses intelligent algorithms to allow quick access and dynamically scalable lock service implementations that allow it to be robust in handling concurrent mutations.

The error recovery mechanism is robust enough to handle issues that arise in a distributed environment.

Overall google has gained substantial advantages by developing their own storage solution in a way such that they can remove bottlenecks and inefficiencies.