International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 03, March-2016

Integrating HBase with Berkeley DB (BDB) to Improve the Performance of Write Intensive Workload

Anusha A C; Neha H; Madhuri M; Keerthi P
Department of Information Science and Engineering
East West Institute of Technology
Bangalore, India

Vidhya K
Assistant Professor, B.E, M.tech(PhD)
Department of Information Science and Engineering
East West Institute of Technology
Bangalore, India

***Abstract*— Hbase is a non-relational distributed data-base or NOSQL database that is modeled to deal with big data and is written in Java. It is a layered system that is made of a lower level HDFS (Hadoop distributed file system) and a higher lever layer responsible for managing the data. However this system suffers from overhead due to isolation between the layers.**

To overcome this problem the paper proposes to replace the HDFS layer with a layer called the Berkeley Database (BDB). BDB is made of log-structured binary B+ trees. This paper also puts forth a re-usable plug-in that will save migration time and efforts required when switching to BDB.

The Hbase integrated with BDB is compared with Hbase with HDFS using an inventory database and it is seen that Hbase with BDB outperforms Hbase with HDFS.

Keywords-NoSQL; BDB; HBase; HDFS

I. Introduction

Large-scale, data-intensive computing requires a sophisticated technology to be integrated with distributed file systems to provide clients with efficient and scalable high-performance accesses to stored data. The domain Bigdata is a data analytic which is a huge amount of data used to analyze, process and also gives the technology to access it in the efficient manner. Hadoop is the platform to implement the Bigdata. Hadoop mainly gives us with the two basic components. 1] HDFS is the storage space provided by the hadoop, where all the data will be stored in form of log files. 2] Map Reduce is the special technique provided by the hadoop to process the data stored HDFS. In the current era there are many technologies to process the data. Well known is the RDBMS(Relational database management system), which stores the data in the format of rows and columns. As it stores the data in the form of table, we cannot dynamically add new data . In other words it fails to store data in the form of elastic model. Hence this approach is only suitable for small scale data. Another technology implemented using the hadoop is the Hbase, where the data is stored in the form of log files. Here the data is stored in the distributed form. Hence to retrieve the data it causes a high network load impact and also this technique results in complexity as it stores in the form of log files. Hence this is only suitable for small scale & medium scale data, but not for processing large set of data. In this paper we are going to introduce a new technique by integrating Hbase and Berkeley database where the records are stored in the form of elastic key-value pair. This also uses B+tree implementation for storing the data. This can be used to process the large set of data in the efficient manner.

II. Literature Review on Hadoop HBase and Berkeley DB

. A Performance-Effective and High-Scalable Grid File System paper describes the distributed model and evaluation of buffer size applied for distributed data grid environments. This model is one of the feasible approaches to improve the overall performance in grid communities. Data Consistency Protocol for Distributed File Systems paper presents a distributed locking protocol with which several nodes can simultaneously write to the distinct data portions of a file, while guaranteeing a consistent view of client cached data. The Design & Evaluation of a Distributed Reliable File System DRFS provides high data availability through replication of data and higher fault tolerance through decentralization. Different technique to transfer big data suggests the use the Nice model to handle transfer the data over the network. But this algorithm failed to handle the issues like security and routing problems which occurs when using network to transfer the data. Big data solution for RDBMS problem illustrates the hadoop architecture consisting of name node, data node, edge node, HDFS to handle big data systems. But it also has the issues of data privacy. Mining big data in real time has the capability of extracting useful information from large set of data due to its volume and velocity. Even this approach has issues with data compression and visualization.

III. EXISTING SYSTEM

In a network data is shared among machines using distributed and parallel file systems. The different distributed file systems are NFS (Network File System) , AFS (Andrew File System) and DFS (Distributed File System) . To access remote data through POSIX interfaces these, provide a uniform interface. These file systems are not scalable over a wide area network. NFS requires specific mount points in a logical directory hierarchy. They do not have the concept of virtual organizations Data Grid middleware provides functionality for accessing data on the grid. Dependable, efficient and transparent file sharing is enabled in the Grid file system architecture. The key difference between traditional distributed file systems and Data Grid middleware is that the scientific researcher analysis data require complete abstraction of the logical hierarchy and the physical files. The Grid File System Group of Global Grid Forum specifies the hierarchical structure to share virtualized data by providing the virtual namespace. Rich set of tools available are closest to providing file system services.

When compared to client server systems Peer-to-peer (P2P) systems are fault-tolerant, robust, and scalable. While C/S distributed file systems, such as NFS (Network File System) or SMB (Server Message Block), do not scale with respect to the number of clients and exhibit a single point of failure, P2P file systems have the potential to cope with an increasing number of participants. Thus,we can move to the DRFS (Distributed Reliable File System), a P2P file system. DRFS maintains high performance and low overhead with many read and write operations. DRFS uses random, content-independent identifiers for data storage. Data availability is high due to the dynamic replication mechanism. DRFS has been implemented using the Filesystem in Userspace (FUSE) interface which provides users with transparent read and write operations.

P2P systems

CFS(Cooperative File System)

The Cooperative File System (CFS) is a publisher-based P2P storage system. Data is stored as blocks and spread evenly among the peers. The system contains three layers: a file system client, a distributed hash table (DHT) layer, DHash, and a Chord layer, used for lookup and routing. When the files are accessed by clients blocks are converted to files. CFS allows anyone to publish and update their own file system, and provides read-only access to others. Data expires and is lost after an agreed-upon time interval.

PAST

PAST is a large-scale P2P persistent storage system. It operates on top of the Pastry lookup system. PAST semantics are different from general purpose file system. Because of lack of facilities provided like searching, directory lookup and traversal. The operations provided are insertion, lookup, and reclaim. In PAST, files are not split into fragments. The file is stored along with an unique id which cannot be updated further.

IVY

IVY is a log-based, distributed, and decentralized read/write P2P file system. It supports multiple users concurrently. IVY's main goal is to provide integrity even when participants do not fully trust each other, or the underlying storage system. The changes made to the file systems by each individuals are stored in a log. These logs are stored in the DHash DHT. When a peer issues a lookup, it scans all the logs associated with the item. At this time the state of file system would be composite of all logs. Since every modification can be identified, actions of misbehaving peers can be discarded. Multiple peers are allowed to write simultaneously to a resource since they write in own, separate logs.

IgorFS

IgorFS is a distributed P2P file system built on top of a Chord-like overlay. IgorFS allows transparent access to remotely stored data through FUSE interface. Files stored are split in blocks of a given size. After the splitting, the blocks are processed for encryption. The block data is initially hashed once, with value k as the result of the hash. This value is then used as a key to encrypt the data of the block. Once the block is encrypted, it is hashed again and the result is used as the id, under which the encrypted block will be stored in the DHT. A file is represented as a collection of (offset, ID, k) - tuples, and directories as a collection of file names and their attributes.

DRFS uses a layered architecture. At the network level, UDP is used for messages that fit in a single packet and are not critical if lost, otherwise TCP is used.

Dynamic Ring Online Partitioning (DROP), is a highly scalable and available key-value store, and it provides a simple interface called lookup(key) under put(write) and get(read) operations.

To keep excellent metadata consistency it provides a linearizable consistency mechanism using ZooKeeper .

DROP DESIGN

DROP uses hashing to distribute the metadata across the MDS cluster. However, it still maintains hierarchical directories to support common directory hierarchy.

Goals

1. high scalability of MDS cluster, 2. excellent namespace locality, 3. dynamic load balancing, 4. metadata consistency.

DROP is designed to scale to a large-scale distributed metadata server cluster for EB-scale file systems within a single global namespace.

Hash-Based Mapping

Hash-based mapping applies hash function to a pathname or filename of a file to locate the file's metadata. It helps clients to locate and contact directly to the right metadata server. Client requests can be distributed evenly among a metadata servercluster, eliminating hot-spots consisting of popular directories. Hashing provides a better load balancing across metadata servers and gets rid of hot-spots e.g., popular directories. However, hashing is a random distribution, in which metadata updates may incur huge network overhead.

Subtree Partitioning

Static subtree partitioning provides a simple approach of distributing metadata operations among MDS cluster, which statically partitions the directory hierarchy and assigns each subtree to a particular MDS. It provides better locality of reference and greater MDS independence than hash-based mapping. Its major drawback is that the workload may not be evenly partitioned among MDS cluster, suffering from a system performance bottleneck. Static partitions fail to adapt to the growth or contraction of individual subtrees over time, often requiring intervention of system administrators to repartition or manually rebalance metadata storage load across MDSs. Dynamic subtree partitioning uses dynamic load balancing mechanism to redistribute metadata dynamically among MDS cluster to handle the changing workload.

HBase with HDFS

The HBase plan takes after a layered engineering, in the soul of past frameworks, stacked in two layers. In the base layer, HBase utilizes HDFS(Hadoop Distributed File System) as a capacity back-end. HDFS opens to its customer applications a common namespace and actualizes versatility and adaptation to internal failure systems at the document layer. Having settled those issues in its stockpiling back-end, HBase centers in the rationale and flexibility elements of the database.

Performing key worth rearrangements on top of an affix just appropriated record framework results in high system load and noteworthy compose intensification, affecting read execution. HBase furthermore performs compose ahead logging for strength which advance increases composes.

Layering is frequently in charge of execution punishments because of the absence of joining intrinsic in such plans. HBASE's energy clients have been implying at execution issues under particular workloads .It is not ideal for an extensive variety of workloads that are commanded by arbitrary peruses. It results in high system load and noteworthy compose intensification, affecting read execution.

IV. PROPOSED SYSTEM

In this project, we propose an alternative architecture to HBase, named HBase-BDB, to overcome the aforementioned problems. We show that the replacement of HDFS with a thinner layer of a local key-value store implemented over local volumes benefits performance without requiring a major re-engineering effort.

Since there are several local key value store engines with different properties available, we decided to leverage one of them (Berkeley DB (BDB) Java Edition2) that fits well our design goals. BDB-JE is a robust, efficient, widely deployed integrated database engine. It implements a B+ tree index, known to perform well for random read workloads and provide good support for range queries.

The entire database is implemented as a log avoiding the need for a separate write-ahead (commit) log. Since BDB-JE is available in a replicated high-availability edition we inherit those properties in HBase-BDB. Removing HDFS from the picture takes away several convenient mechanisms that underlie HBase’s elasticity architecture. To make up for this loss we design and implement new efficient elasticity mechanisms suitable for HBase-BDB.

Overall, our key contributions in this project are

- Design and implementation of a distributed key-value store architecture maintaining HBase’s front-end and replacing HDFS with log-structured B+-trees over direct-attached file systems, improving performance and eliminating overheads due to HBase layering
- Novel, efficient elasticity mechanisms for splitting and moving data regions over the direct-attached file systems
- HBase-BDB is designed to overcome HBase’s performance bottlenecks (while retaining compatibility with HBase applications) without losing on elasticity features
- Berkeley DB provides a simple function-call API for a number of programming languages, including C, C++, Java, Perl, Tcl, Python, and PHP. All database operations happen inside the library. Multiple processes, or multiple threads in a single process, can all use the database at the same time as each uses the Berkeley DB library.
- Low-level services like locking, transaction logging, shared buffer management, memory management, and so on are all handled transparently by the library.

Better performance is achieved due to replacement of HDFS with a thinner layer of a local key-value store implemented over local volumes.It also eliminates the overhead due to HBase layering.It implements B+ tree index known to perform well for random read workloads and provide good support for range queries.

ARCHITECTURE OF PROPOSED SYSTEM

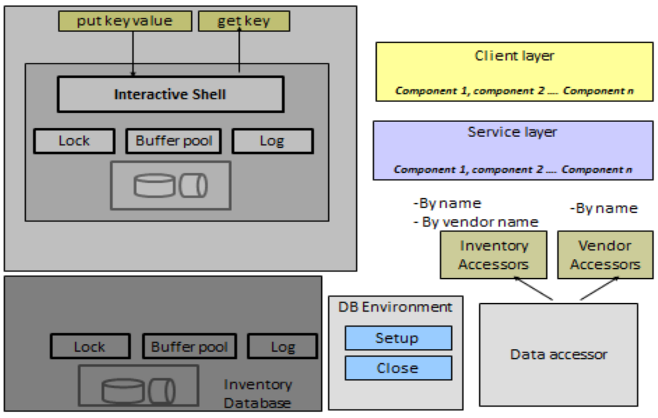

Fig:1 Proposed System Architecture

The architecture of the BDB consists of mainly six modules:

Interactive Shell

This component is designed to provide an user interface for the user to interact with the Berkeley database.

Inventory Database

The inventory database is developed to evaluate the performance of the record accessing operations by the normal Hbase system and the successfully integrated Hbase& Berkeley database system. This system mainly consists of four components as follows

- Lock: This component is designed in such a way that if a user is modifying any record, it locks that particular record so that no othe user can access it till the modification completes.
- Buffer pool: This acts as a cache memory in the system i.e, all the recently accessed record will be stored in this buffer pool
- Log: It performs the role of database administrator, where all recent activity performed are stored.
- Storage space: This component provides the storage space for storing all the records in the form of elastic key value pair.

Database Environment

The database environment provides with the authentication measures where it consists of two components as follows

- Setup: This component is used to setup the connection with the database. It expects the valid user ID and password to setup the connection with the database.
- Close: It is used to terminate the connection with the database.

Data Accessor
The data accessor is used to access the record from the inventory database. It provides with the two major components as follows

- Inventory Accessor: This provides the user with the options to search any product based on its name and also by its respective vendor name.
- Vendor Accessor: This accessor is only provided for the vendors to search any of the records or the products only by its name.

Service Layer
The service layer is mainly composed of backend implementation of Berkeley database which is done using theprovisions such as Java, JEE (Servlet, JDBC, JSP), HBASE, HADOOP, MySQL/Oracle, Wildfly server, Maven as the building tool etc.
Client Layer
The client layer is mainly a front end support provided for the user in the user interface which is implemented using HTML5, CSS3, Skeleton, Foundation,Jquery, Ajax etc.

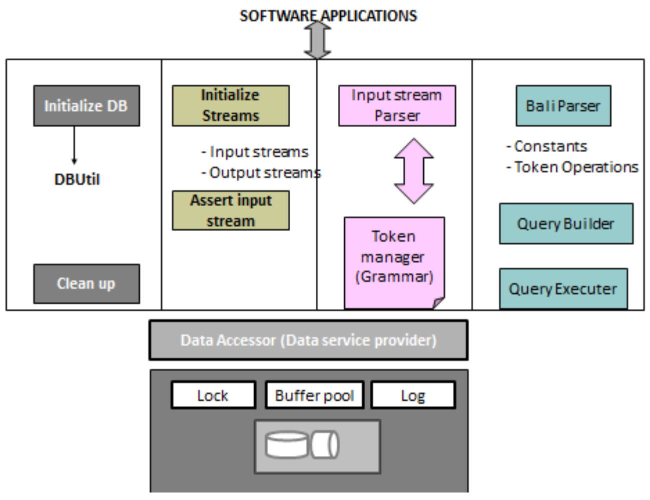

Fig:2 Reusable plugin architecture

The re-usable plugin is mainly used to convert the basic SQL query into BDB compatible commands. This re-usable plugin majorly consists of 4 modules as shown in the fig2.

Module1: The initial module provides the operations like getting the connection with the re-usable plugin in the initial stage and for getting the connection terminated.

- Initialize DB: This component is used for getting the connection with the re-usable plugin. It is built using the tool known as DBUtil.
- Clean up: This component is use for getting the connection terminated from the re-usable plugin.

Module2: This module is used for initializing the particular operation and also consists of a component for checking the syntax of the given SQL queries.

- Initialize Streams: This component is used to select the particular stream given such as input stream & output stream. If the operation is to read the data from the database then the output stream is going to be initialized. If the operation is to write the data to the database then the input stream is going to be initialized.
- Assert Input Stream: This component is used to check the syntax of the given SQL query. If the given query is syntactically correct then its parsed to the next module, else it's going to give the error message for the user.

Module3: This module is designed to collect the parsed SQL query and it will break the query into the form of tokens.

- Input Stream Parser: This component is used to break the SQL query into the form of tokens. All the keywords that are used in the query is going to be stored in the token manager.
- Token manager: This component is used store all the keywords in the query which is tokenized by the input stream parser.

Module4: This module consists of Bali parser which is used to differentiate between the constants and the tokens. It also consists of the query builder and the query executer component.

- Query Builder: This component is designed for building the BDB compatible command using the given appropriate SQL query.
- Query Executer: This component is used for executing the BDB command and to access the data stored the Berkeley database.
- Bali Parser: Bali parser which is used to differentiate between the constants and the tokens.

V. CONCLUSION

In this paper we exhibited HBase-BDB, a conveyed key value store that shares HBase's information model and information dissemination instruments yet leaves from it in the utilization of a log structured B+-tree filed capacity back-end over locally attached files frameworks. With the utilization of a log organized key quality store consolidated with novel versatility systems HBase-BDB can enhance over HBase in irregular read. HBase-BDB lingers behind HBase just in irregular sweeps; these however are just a little share of general operations in well known workloads. Support for flexibility in HBase-BDB is appeared to be viable in yielding comparative accessibility and execution effect to what is achievable with HBase-HDFS.

REFERENCES

[1] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," ACM Transactions on ComputerSystems (TOCS), vol. 2, no. 4, pp. 277–288, 1984.

[2] M. Welsh and D. Culler, "Virtualization considered harmful: OS design directions for well-conditioned services," in Proceedings of the 8th Workshop on Hot Topics in Operating Systems, 2001, pp. 139–144.

[3] Apache HBase, http://www.hbase.org.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," ACM Transactions on Computer Systems (TOCS), vol. 26, no. 2, p. 4, 2008.

[5] E. K. Lee and C. A. Thekkath, "Petal: Distributed virtual disks," in ACM SIGPLAN Notices, vol. 31, no. 9, 1996, pp. 84–92.

Anusha A C (BE)
Information Science and Engineering Department,
East West Institute of Technology,
#63,Magadi Main Road,Vishwaneedam post,
Bangalore 560091 Karnataka.

Madhuri M (BE)
Information Science and Engineering Department,
East West Institute of Technology,
#63,Magadi Main Road,Vishwaneedam post,
Bangalore 560091 Karnataka.

Neha H (BE)
Information Science and Engineering Department,
East West Institute of Technology,
#63,Magadi Main Road,Vishwaneedam post,
Bangalore 560091 Karnataka.

Keerthi Prabhashankar (BE)
Information Science and Engineering Department,
East West Institute of Technology,
#63,Magadi Main Road,Vishwaneedam post,
Bangalore 560091 Karnataka.

Vidhya K
Assistant Professor, B.E, M.tech(Computer Networks),
(Ph.D in Image Processing)
Department of Information Science and Engineering,
East West Institute of Technology,
#63,Magadi Main Road,Vishwaneedam post,
Bangalore 560091 Karnataka.