GLORY-DB: A Distributed Data Management System for Large Scale High-Dimensional Data

Hyun Hwa Choi, Hun Soon Lee, Kyeong Hyeon Park, Mi Young Lee
Database Research Team, Electronics and Telecommunications Research Institute
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-700, Korea
E-mail: {hyunwha, hunsoon, hareton, mylee}@etri.re.kr

Abstract: Recently, the proliferation of the web and digital photography has resulted in the need of a distributed storage system for managing large scale data and an indexing technique for supporting efficient nearest neighbor search on high-dimensional data. One of the most challenging areas in the fields of a distributed data managing and image processing is scalability of data and machines. Especially, for a large scale image clustering problem, which can not fit on a single machine, the traditional nearest neighbor search can not be applied. This paper presents the design of a distributed data management system, highly available and scalable storage system which provides contents-based retrieval using a hybrid spill tree with local signature files. We describe our scalable index structure and how it can be used to find the nearest neighbors in the cluster environments.

1. GLORY-DB

1. 1 Data Model and Index Mechanism

GLORY-DB supports the relational logical data model, where a database consists of a collection of named tables, each with a named collection of columns. But GLORY-DB differs significantly from traditional relational databases in its data model like Figure 1.

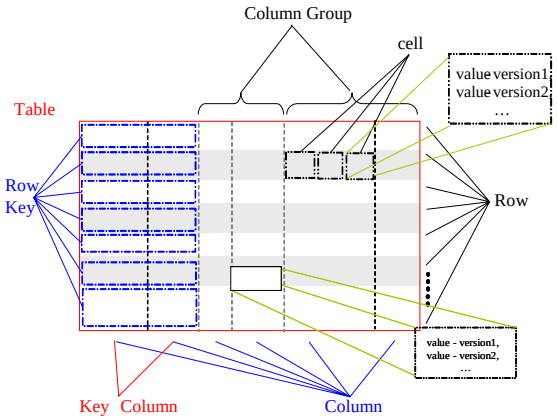


Figure 1. Data Model

Columns in GLORY-DB tables have one or more cells. A cell stores data with a key that is represented as arbitrary strings. Each cell can also contain multiple versions of the same data like Bigtable [1]. Columns in a GLORY-DB table can form unique row keys that are used to specify any row in the table. There are restrictions that key columns can have only a cell with the last version and not be deleted as long as there is the table containing them. On the other hand, GLORY-DB implements column groups to store physically tables. A column group consists of the columns which are typically accessed together in a table. Especially columns in a column group can not belong to any other column groups. Some useful tuning parameters can be specified on a column group basis. For example, a column group can be declared to be in-memory, to be compressed by user-specified compression format and to create a Bloom filter [2] that allows us to ask whether the column group might contain any data for a specified row/column pair.

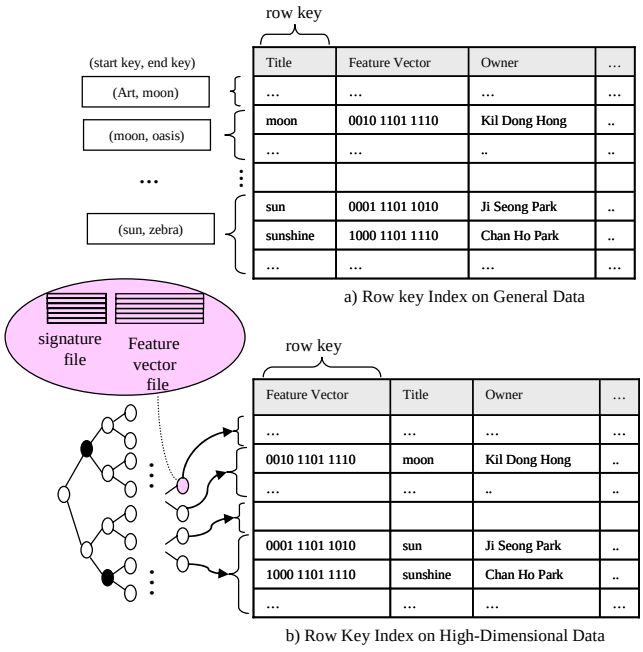


Figure 2. Index Mechanisms

GLORY-DB provides two index mechanisms based on row keys as shown in Figure 2. One is to sort row keys of tables in lexicographical order. It is similar to a two-level hierarchy of a METADATA table in Bigtable [1]. The first level, called as *root table*, contains the location of all partitions in a special *meta table*, and is treated specially – it is never split. Each partition of a *meta table* contains the location of a set of user table partitions. The location information of a partition in *root* and *meta tables* is stored under a row key that is an encoding of the partition's table name and its start row. Partitions are the sorted data sets

* This work was supported by the IT R&D program of MIC/IITA. [2007-S-016-01, The Development of Global Resource Management System for Future Internet Service]

with a predefined size as a management unit and are serviced by multiple nodes. The other is to use a hybrid spill-tree with local signature files [3], which is to combine a hybrid spill tree with signature files, in order to search the nearest neighbors on large scale high-dimensional data. A signature file is a small abstraction of feature vectors from multimedia data, which is typically encoded as bit sequences.

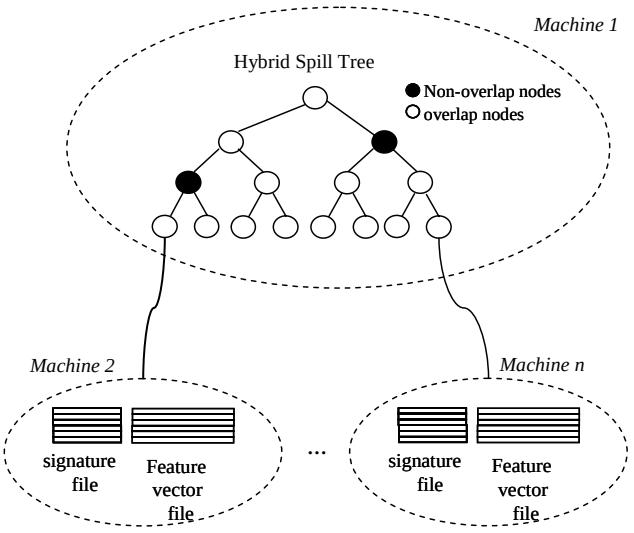


Figure 3. Hybrid Spill Tree with Local Signature Files

In Figure 3, we build a hybrid spill tree based on the sample vector data that are extracted from the original feature vector files. The size of a random sampling data has to be determined by the capacity of a machine that accommodates the hybrid spill tree, because it is difficult that a tree structure is operated on multiple nodes and is traversed in parallel. Each leaf node in the constructed hybrid spill tree defines a partition, where we store a signature file and feature vector file of the corresponding range of the hybrid spill tree on a separate machine. In essence, our index structure can be viewed as a single hybrid spill tree, spanning a large number of machines.

Due to construction of a hybrid spill tree [4], which can have either a spill-tree [4] partition with the overlapping buffer or a metric-tree [5] partition without overlapping as child node, the approximate NN (Nearest Neighbor) search on a hybrid spill tree also becomes a hybrid of the MT-DFS search and the defeatist search. Namely, we do defeatist search on overlapping nodes and MT-DFS search on non-overlapping nodes. The candidate nodes determined from traversing a hybrid spill tree find the nearest neighbors about given query point concurrently. At this point, each candidate node processes K-NN search by sequentially scanning a local signature file with a signature extracted from query point. Then it returns as the result of query the feature vectors corresponding to the candidates of the local signature file.

Our hybrid spill tree with local signature files is an appropriate index method for high-dimensional data in cluster environments, because it provides index scalability and runs K-NN search on the distributed computing nodes in parallel.

1. 2 System Architecture

GLORY-DB consists of a master server and multiple partition servers as Figure 4 shown below.

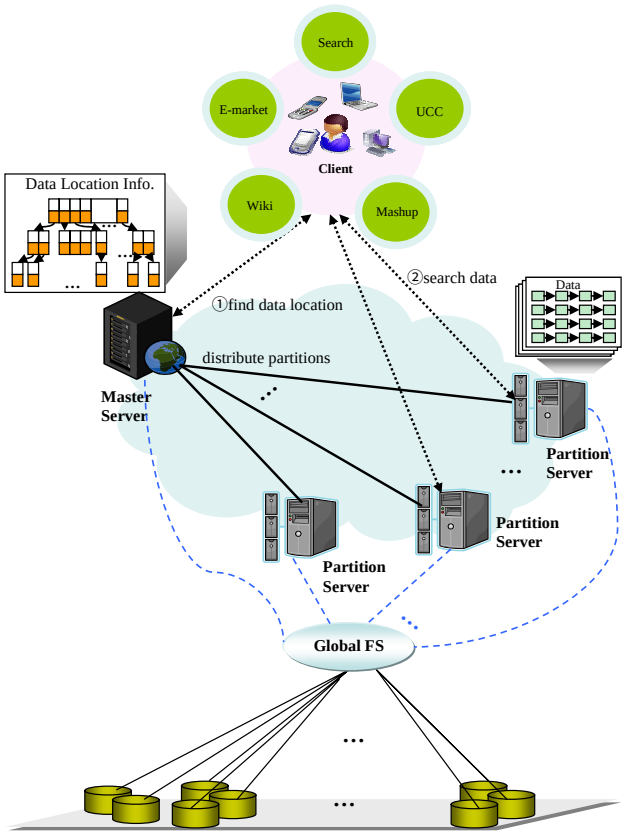


Figure 4. GLORY-DB Architecture

The master server manages catalogs of tables, users and privileges. In addition the master server manages information about both partition servers and partitions, and automatically assigns partitions to partition servers on the distributive law by load of partition servers in monitoring. If the master server detects a partition server faulty or beyond its capacity, the master server reassigns all or some partitions of it to other partition servers. It means that GLORY-DB provides data services despite failure of partition servers. Partition servers perform data operations on partitions assigned to themself. If data size of a partition reaches a threshold after insert operations, a partition server for the partition splits it into two smaller partitions and notifies the master server. Then the master server asks one partition server to load the partition that has split. Two existing partitions may be merged to form one larger partition in GLORY-DB. In order to provide an efficient data search, partition servers have writable and readable buffers, manage data on column group basis and use Bloom filters.

These functions of GLORY-DB are archived on the distributed file system, such as GFS (Google File System [6] and HDFS (Hadoop Distributed File System) [7]. They

are file systems that efficiently cluster storage servers in a manner that scalability and availability are maintained.

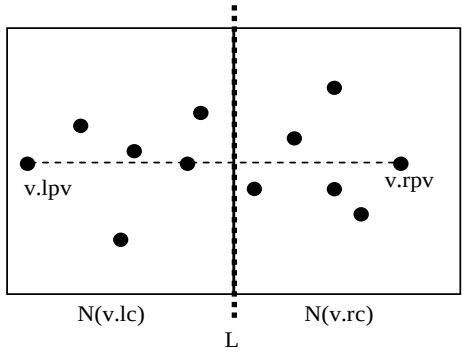
A row-key based search of GLORY-DB is performed differently according to index mechanisms of row keys. With two-level hierarchy index, the GLORY-DB behaves like Bigtable. Whereas, the user queries with feature vector are processed through similarity search on a hybrid spill tree with local signature files as the followings: (1) Given a feature vector, we first find a partition server assigned a hybrid spill tree related it. (2) We traverse the hybrid spill tree on the partition server in order to determine which leaf nodes can process the given feature vector, and transfer it to the determined candidate partition servers concurrently. (3) Then, each candidate partition server makes the signature of the query's feature vector, performs K-NN search on a local signature file with the generated query's signature, and returns k-feature vectors as results of the similarity search. (4) We decide final k-feature vectors among the feature vectors obtained from the candidate partition servers, and obtain data from partition servers containing them.

2. Related Work

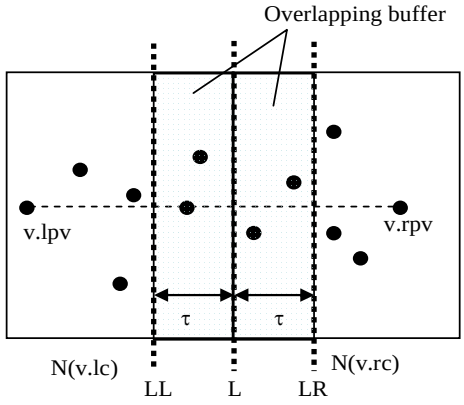
The proliferation of digital photography has increased the importance of the high quality internet service based on the moving picture like user generated contents (UGC). For this, global internet service providers need the large-scale efficient internet service environment. Traditional database vendors have developed databases that can store large volumes of data. Oracle's Real Application Cluster database [8] uses shared disks and IBM's DB2 Parallel Edition [9] is based on a shared-nothing architecture. Both products provide a complete relational model with transactions and require many professionals to manage them.

GLORY-DB shares many characteristics with Bigtable[1] and C-Store [10][11]. In order to store large scale data, the three systems consist of thousands of commodity servers and use a shared-nothing architecture. GLORY-DB and Bigtable distribute subsets of rows in a table to servers, whereas C-Store does subsets of columns in a table. The three systems have a column storage structure as a read-optimized relational DBMS. GLORY-DB stores data on column groups, Bigtable on locality groups, and C-Store on projections. Especially, in C-Store, a column data can repeatedly be stored in multiple projects on servers. For speed-up of data search, three systems had two different data structure, one for recent writes, and one for storing long-lived data, with moving data from one to the other by periods. In addition three systems use compression algorithms and lightweight transaction.

Recent internet search engine may provide content-based retrieval on billions of multimedia data, because the keyword-based search on large scale images and video collections is too expensive and requires much manual intervention. Herein we define a content-based retrieval as identifying the images or video data most similar to a given query image or video clip. A content-based retrieval performs similarity search using features such as color, shape and so on, extracted from the original image or video data. Typically, a feature is represented by a point in a high-dimensional data space. Bigtable and C-Store are designed to efficiently manage and search text data, whereas GLORY-DB considers how to manage and search both text and multimedia data. GLORY-DB provides two index mechanisms. One is a two level hierarchy of root and meta tables for text data. The other is a hybrid spill-tree with local signature files for high-dimensional data. A hybrid spill tree with local signature files is an appropriate index method for large scale high-dimensional data in cluster environments due to it's scalability and parallel processing. We let the user determine an index mechanism for data in creating table. GLORY-DB provides simple keyword search APIs with row key and similarity search APIs with feature vector data as input for content-based retrieval.



a) Partitioning in a metric tree



b) Partitioning in a spill tree

Figure 5. Metric Tree and Spill Tree

Over the years, techniques for solving the exact and approximate k nearest neighbor (K-NN) problem have evolved. The metric tree [5] in Figure 5 shown above organizes a set of points in a spatial hierarchical manner. A search on a metric-tree is MT-DFS that is very efficient for NN search but starts to slow down as the dimension of the dataset increases. Moreover it spends up to 95% of the time verifying that it is in fact the true NN. A spill-tree [4] is a variant of metric-trees in which the children of a node can contain shared datapoints. A defeatist search on a spill-tree is very fast than a MT-DFS on metric-tree because it descends the metric tree using the decision boundaries at each level without backtracking. A hybrid spill tree [4] is a hybrid of both a metric-tree and a spill-tree. It can have

either a spill-tree partition with the overlapping buffer or a metric-tree partition without overlapping as child node.

Unfortunately, these methods are all designed to run on a single machine. For large scale multimedia data clustering problem, which can not fit on a single machine, the traditional algorithms simply cannot be applied. To solve this problem, Google introduced a hybrid spill tree in parallel [12]. It builds a metric tree as a top tree for a random sample of data small enough to fit on a single machine. Each of the leaf nodes in this top tree then defines a partition, for which a hybrid spill tree can be built on a separate machine.

3. Conclusion

In this paper, we have described GLORY-DB, a distributed data management system for storing large scale of text data or high-dimensional data.

GLORY-DB supports the multi-dimensional data model based on row, column and cell. Especially GLORY-DB manages multiple versions of cell data. In addition, GLORY-DB has a physical data model, column group, to store tables. A column group consists of the columns which are typically accessed together in a table. For efficient data search, GLORY-DB has two index methods: a two-level hierarchy of row keys in lexicographical order and a hybrid spill tree with local signature files. These index mechanisms are appropriate for scalability of data and machines in cluster environments. GLORY-DB provides keyword search of text data and content-based retrieval using similarity searching of multimedia data.

GLORY-DB consists of thousands of commodity servers and uses a shared-nothing architecture. A master server in GLORY-DB monitors partition servers periodically and assigns partitions to partition servers automatically. Though this operating scenario, GLORY-DB provides availability of data services despite failures of partition servers.

References

[1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A Distributed Storage System for Structured Data”, *OSDI 2006*

[2] R. H. Bloom, “Space/Time Trade-offs in Hash Coding with Allowable Errors”, CACM 1970, pp.422-426

[3] K. Lee, H. Lee, M. Lee, M. Kim, “A Scalable Index Mechanism for High-Dimensional Data in Cluster File Systems”, *IMECS 2008*

[4] T. Liu, A. W. Moore, A. Gray, and K. Yang, “An Investigation of Practical Approximate Nearest Neighbor Algorithms”, *NIPS 2004*

[5] P. Ciaccia, M. Patella and P. Zezula, ”M-tree: An Efficient Access Method for Similarity Search in Metric Spaces”, *VLDB 1997*

[6] S. Ghemawat, H. Gobioff, and Shun-Tak Leung, “The Google File System”, SOSP 2003

[7] APACHE.ORG http://hadoop.apache.org/core/docs/current/hdfs_design.html Project page

[8] ORACLE.COM www.oracle.com/techonology/products/database/clustering/index.html Product page

[9] C. K. Baru, G. Fecteau, A. Goyal, H. Hsiao, A. Jhingran, S. Padmanabhan, G. P. Copeland, and W. G. Wilson , ”DB2 Parallel Edition”, *IBM Systems Journal* 34, 2 (1995), pp. 292-322

[10] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O’Neil, P. O’Neil, A. Rasin, N. Tran and S. Zdonik, ”C-Store: A Column-oriented DBMS”, *VLDB 2005*, pp. 553-564

[11] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. R. Madden, ”Materialization Strategies in a Column-Oriented DBMS”, *ICDE 2007*, pp. 466-475

[12] T. Liu, C. Rosenberg and H. A. Rowley, ”Clustering Billions of Images with Large Scale Nearest Neighbor Search”, *WACV 2007*