ISSN:0975-9646

A Simplified Data Processing in MapReduce

P. Buddha Reddy
Assistant Professor,
Dept. of CSE,
Vardhaman College of Engineering,
Hyderabad, India

CH Sravan Kumar
Assistant Professor,
Dept. of CSE,
Vardhaman College of Engineering
Hyderabad, India

K. Srinivas
Assistant Professor,
Dept. of CSE,
Vardhaman College of Engineering
Hyderabad, India

***Abstract – F*or processing and generating large data sets we use MapReduce as a programming model and their associated implementations. A *map* function is specified by a user to generate a set of intermediate key/value pairs from processes a key/value pair. The warehousing systems existing based MapReduce are not specially optimized for time-based big data analysis applications. Such applications have two characteristics: 1) A continuously generated data are required to be stored persistently for a long period of time; 2) A processed data is used for applications in some time period, for typical queries. For large Data loading and for a high query execution are Time-based big data analytics is required. The current Mapreduce based existing systems solutions do not solve this problem, because of the two requirements are contradictory**

Keywords - MapReduce; Keys; Data;

I. INTRODUCTION

The computations on the data over the past five years, have been greatly improved and hundreds of special-purpose computations of raw data, such as crawled documents, web request logs are used to compute various kinds of derived data, such as the set of most frequent queries in a given day, inverted indices, the number of pages crawled per host. Such a computations are mostly straightforward. The input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The parallelizable issues of how to parallelize the computation, distribute the data, and handle failures to the original computation with large amounts of complex code to deal with these issues.

The major design issues of Mastiff's are as follows:

- Column-based query execution engine: Mastiff uses a late materialization technique in its column-based execution engine, which can significantly improve query performance with the help of SLC-Store.
- Data storage structure: Mastiff uses an optimized column group store structure, called SLC-Store, which provides fast loading speed and high execution performance for analytical queries.
- Optimized data scan method: Mastiff utilizes the Scan-Map technique to avoid unnecessary data accesses during a table scan by exploiting the information stored in the light-weight helper structures.
- Light-weight helper structure: On the basis of SLCStore. Mastiff uses both segment-level and page-level light-weight helper structures to store properly chosen statistical information, which can achieve the best tradeoff between data loading and query execution.

Hadoop implementation is possible on Mastiff. Mastiff converts all the SQL commands to MapReduce programs and executes them on Hadoop. It is implementation based on Hive, but it has significant extensions to implement the above designs. We have conducted extensive experiments with diverse workloads to compare Mastiff's performance with those of three widely-used systems [3] [4] [5].

II. DESIGN AND IMPLEMENTATION OF MASTIFF

A. Overview:

Mastiff architecture is as shown in Figure 1, implemented over Hadoop, and interacts with Hadoop's components (e.g., HDFS and JobTracker) to accomplish data loading and processing. Mastiff has three kinds of modules as follows:

1) Data loading modules: Data loading in Mastiff can be through either load servers or the Bulk Loader. Load servers receive original data streams, convert them into SLC-Store format, and build the helper structures for each page and each segment1. The Bulk Loader is a tool for loading data in batch
2) Metadata management module: The MetaStore server maintains table schemas and column group information including the number of column groups, column names in each group, and the compression algorithm for each group.
3) Query execution modules: The SQL Translator and the Column Operator Rewriter are responsible for translating an SQL query to one or several Hadoop jobs. Then, the jobs are submitted to the JobTracker, and are executed in the cluster.

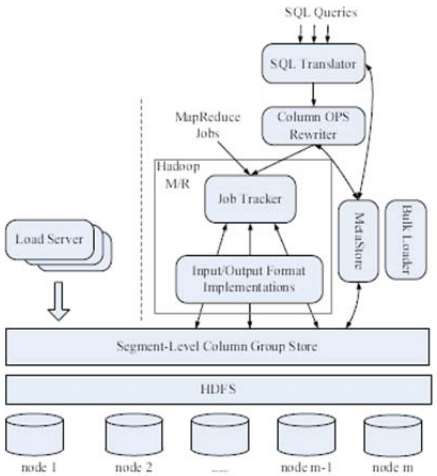

Figure 1. Mastiff Architecture.

B. Data Store and Helper Structure

A designed data store over HDFS called Segment-Level Column Group Store (SLC-Store), integrated with two kinds of light-weight helper structures, to improve both data loading speed and analytical query performance. In Mastiff, first a table is horizontally partitioned into multiple segments. Data is stored in each segment vertically and partitioned into column groups, and column groups consist of one or more columns. A single HDFS physically stores each Mastiff table and an HDFS block is used to store each segment. Segments are stored in column groups one by one. A fixed size of a page is used to divide a segment and each column group in the segment is stored in contiguous pages.

C. Data Loading

The Mastiff data is loaded by a group of *LoadServers*. Until memory buffer is full the data is not loaded into a load server. Data disk will store all the buffered data as a whole segment. Mastiff uses a pair of buffers to improve loading speed, and avoids loading blocked during writing back the buffer. First the buffered data is sorted with a key before written back. The data of each column group are written back to disk page by page. The size of the page is set as 128KB. If the compress option is set during data loading, then each page is compressed before written back. Users can configure the Mastiff, for the compression algorithm used for each column group. Mastiff also provides the Bulk Loader for data warehousing applications.

D. ScanMap

A ScanMap is used built for each column group in the filter expression for a given query. A logical representation of data is maintained that consists of array of entries, one entry for each page in a column group. Each entry is a 3-tuple: (*PageID, RowRange, and ScanMode*) [2].

There are three *ScanModes*:

- *Rough*: Some records in this page satisfy the filter predicate and some not.
- *Negative*: None of the records in this page satisfy the filter predicate. In Mastiff, the execution of a query begins from TableScan Operator in the Map phase. The ScanMaps for a query are generated at the beginning of each Map task before TableScan Operators. With ScanMaps, the pages marked as either *Negative* or *Positive* do not need to be scanned. The difference between the two modes is that the row ID range of each *Positive* page needs to be passed to the subsequent operators in the query plan, while each *Negative* page is simply skipped. The pages marked as *Rough* should be scanned so as to pick out the row IDs that satisfy the filter predicates, and pass them to the subsequent operators.
- *Positive*: All the records in this page satisfy the filter predicate.

In the following subsections, we describe how to build the ScanMap for each query.

1) Multi-Group ScanMaps: Multi-group ScanMaps is built when the attributes in the filter expression of a query are scattered in multiple column groups. First, we split such a multi-group expression into multiple single-group expressions, each as a child expression. Then, we build each child expression's ScanMap by using the aforementioned method of generating single-group ScanMap. Finally, the ScanMaps of all child expressions are *calibrated* into the final ScanMaps. Since the pages in different column groups contain different numbers of rows, the calibration of the ScanMaps of different column groups depends on the relationship of the pages, which must be calculated before calibrating their scan modes. For two column groups, there are four page relationships (*Contain*, *Equal*, *Intersection*, and *Disjoint*), as shown in Figure 3. During calibration, Mastiff traverses the ScanMaps of the two column groups from their first pages, and calibrates the scan mode of each page according to the relationships of each pair of pages[2].

2) Single-Group ScanMap: For a single-group expression where the attributes in the filter expression are all in the same column group, Mastiff builds the ScanMap by scanning the information in the page-level helper structure of this column group. Mastiff computes the scan mode of each page according to the operation type of the filter expression. First, for *comparison operations* (>, <, =, <= and >=), if both of the max and the min values of this page satisfy the filter expression, the scan mode of this page is set to *Positive*; if neither, it is set to *Negative*; otherwise, it is set to *Rough*. Second, for *logical operations* (*AND*, *OR* and *NOT*), recursive computing is needed. After the scan mode of each child expression is computed, the scan mode of this page is computed based on the results of all child expressions. For *AND*, the scan mode of this page is set to *Positive* if the results of all child expressions are *Positive*; it is set to *Negative* if any child expression's result is *Negative*; otherwise, it is set to *Rough*. For *OR*, the scan mode of this page is set to *Positive* if any child expression's result is *Positive*; it is set to *Negative* if the results of all child expressions are *Negative*; otherwise, it is set to *Rough*. For *NOT*, the scan mode of this page is set to *negative* if the child expression's result is *Positive*; it is set to *Positive* if the child expression's result is *Negative*; otherwise, it is set to *Rough*. Third, for *other operations* (e.g., a user defined function), the scan mode is set to *Rough*, which means that this page needs to be scanned [2].

(1) For the case of *PageA Contain PageB* (Figure 3(a)), the scan mode of *PageB* should be set to *Negative*, if the operation is *AND* and the scan mode of *PageA* is *Negative*. The scan mode of *PageB* should be set to *Positive*, if the operation is *OR* and the scan mode of *PageA* is *Positive*. Otherwise, their scan modes need not changing.

(2) For the case of *PageA Equal* to *PageB* (Figure 3(b)), the scan mode of both pages should be set to *Negative*, if the operation is *AND* and the scan mode of either page is *Negative*. The scan mode of both pages should be set to *Positive*, if the operation is *OR* and the scan mode of either page is *Positive*. Otherwise, their scan modes need not changing.

(3) For the cases of *Intersection* and *Disjoint* (Figure 3(c)/(d)), their scan modes need not changing.

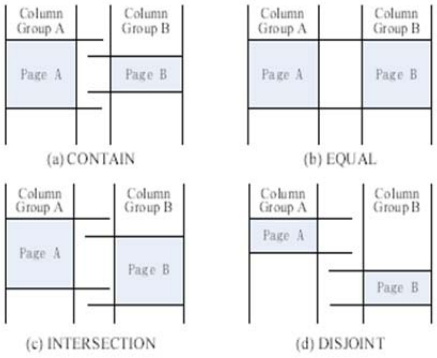

Figure 3. Relationship between pages.

E. Column-Based Query Execution Engine

Hive is a widely-used query execution engine over Hadoop. Although Hive can be configured to use row store or column store (RCFile), its query execution engine can only process data in a *row-based* manner and uses the *early materialization* scheme. As the previous study showed [7], early materialization could be inefficient for column stores when the selectivity is low, because many tuples are needlessly constructed, which results in many unnecessary disk I/O. Since Mastiff uses a column store structure (SLCStore) for on-disk data organization, a column-based query execution engine which uses *late materialization* scheme should be implemented to make full use of the advantage of the column store [2].

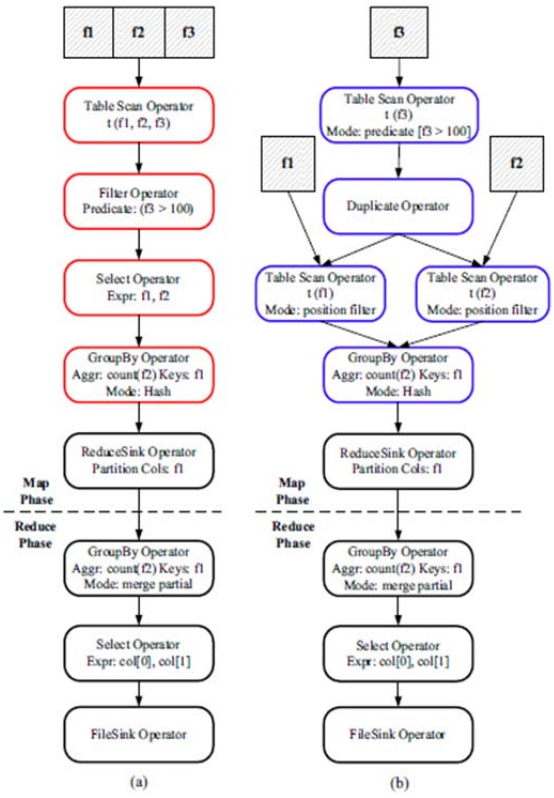

Figure 4. (a) Hive's query plan. (b) Mastiff's query plan.

IV. PERFORMANCE EVALUATION

A. Experiment Setup

1) Workloads

2) Test Platforms and Hadoop Configuration

3) Software Systems

HadoopDB

Mastiff

GridSQL

B. Performance Results of Data Loading

The Performance results of Data loading are as shown in Fig 5 and Fig 6.

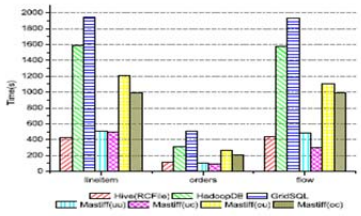

Figure 5. Load performance on 20 nodes.

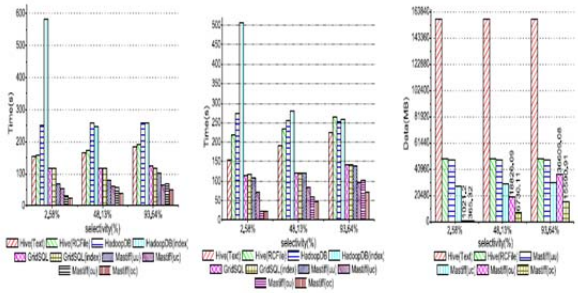

Figure 6. Aggregate Performance

V. CONCLUSION

Mastiff called a Hadoop-based data management system which is optimized for timebased data analytics. Mastiff exploits a systematic combination of an optimized column group store structure to achieve the goals of both high data loading speed and high query execution performance. An optimized data scan method and a column-based query execution engine are the two kinds of light-weight helper structures.

REFERENCES

[1]. Jeffrey, Dean., Sanjay, Ghemawat., "MapReduce: Simplified Data Processing on Large Clusters" in OSDI.

[2]. Sijie, Guo., Jin, Xiong., Weiping, Wang., Rubao Lee., "Mastiff: A MapReduce-based System for Time-based Big Data Analytics", IEEE International Conference on Cluster Computing, 2012.

[3]. http://sourceforge.net/projects/gridsql/.

[4]. He, Y., Lee, R., Huai, Y., Shao, Z., Jain, N., Zhang, X., Xu, Z., "Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems," in *ICDE*, 2011.

[5]. Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D. J., Rasin, A., Silberschatz, A., "Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads," *PVLDB*, 2009.

[6]. http://wiki.apache.org/hadoop/FAQ.

[7]. Abadi, D. J., Myers, D. S., DeWitt, D. J., Madden, S., "Materialization strategies in a column-oriented dbms," in *ICDE*, 2007.