A New Middleware for Distributed Data Processing in CUBRID DBMS

Hyeong-Ill Kim, Min Yoon, MunChul Choi, Jae-Woo Chang*

Chonbuk National University, 567 Baekju-daero, deokjin-gu, 561-756 Jeonju City, Korea

Abstract

With the sharply increasing amount of data, studies on bigdata processing based on NoSQL have been actively done. However, NoSQL cannot satisfy the ACID properties of database transactions. Therefore, bigdata processing based on RDBMS has been spotlighted. CUBRID Shard stores data in the distributed CUBRID servers by dividing the database. However, CUBRID Shard cannot process a query when data of a user is distributed on the multiple CUBRID servers. Therefore, in this paper we propose a CUBRID based middleware which supports distributed data processing. Through the performance evaluations, we show that our proposed scheme shows better performance than the existing work in terms of query processing time.

Keywords: Middleware; distributed environment; parallel query processing; CUBRID;

1. Introduction

Due to the development of the SNS (Social Network Service), the amount of data has been rapidly increasing. Therefore, studies on the bigdata processing have been actively performed. With the existing IT technologies, it is very hard to store, process and analyze the bigdata efficiently. The bigdata itself is hard to be used as valuable information because of the immense volume of the bigdata. Therefore, analyzing the bigdata to extract the meaningful information from the bigdata is necessary. To analyze the bigdata, a large scale of computing resources and efficient bigdata management system are required. For this, studies on NoSQL, such as Hadoop, MongoDB, and Cassandra, have been done. However, NoSQL cannot satisfy the ACID properties of database transactions.

* Corresponding author. Tel.: +82-63-270-2414; fax: +82-63-270-2394.
E-mail address: jwchang@jbnu.ac.kr
Especially, the major drawback of NoSQL is that it cannot guarantee data consistency when the NoSQL supports the partition tolerance and availability. Therefore, big data processing based on RDBMS (Relational DataBase Management System) has been spotlighted.

CUBRID Shard\(^4\) is a RDBMS that is designed to process big data. CUBRID Shard stores data in the distributed CUBRID servers by dividing the database. By distributing data into several servers, CUBRID Shard can support parallel query processing. However, parallel query processing of CUBRID Shard is only available when a query of a user requires single CUBRID server to process the query. In other words, if data of the user is distributed on the multiple CUBRID servers, CUBRID Shard cannot process the query of the user. Moreover, CUBRID Shard has a low usability because a user should specify a ‘shard_hint’ in the SQL when requesting the query.

To solve these problems, in this paper we propose a CUBRID based middleware which supports distributed data processing. The proposed middleware runs on the RDBMS which can aid users who are familiar with SQL to conveniently process the big data by using SQL statements. In addition, Besides general SQL queries, the middleware can support the aggregation queries (e.g., min, max, count, sum, average) that have not been handled on the distributed parallel computing environment.

The remainder of this paper is organized as follows. In section 2, we introduce related work concerning big data processing. Section 3 presents overall system architecture and query processing procedure of the proposed scheme. In section 4, we compare the performance of our proposed scheme with that of the existing scheme. Finally, we conclude this paper with future work in section 5.

2. Related work

NoSQL such as Hadoop\(^4\), MongoDB\(^5\), and Cassandra\(^6\) provides a mechanism for storage and retrieval of unstructured data. Motivations for this approach include simplicity of design, horizontal scaling, and finer control over availability. The data structures used by NoSQL (e.g. key-value, graph, or document) differ from those used in relational databases, making some operations faster in NoSQL and others faster in relational databases. However, most NoSQL cannot satisfy the ACID properties of the database transactions. Due to the disadvantages of NoSQL, RDBMS have been recently spotlighted in the field of big data processing. First, CUBRID\(^5\) is an object-oriented relational database developed by NHN (Next Human Networks). CUBRID provides high accurate and predictable automatic fail-over and fail-back features. However, CUBRID cannot run on the distributed system environments because CUBRID is optimized on single machine. So, it is not efficient for dealing with big data. Secondly, to solve the problems of CUBRID, CUBRID Shard\(^8\) is developed. CUBRID Shard can partition the data based on the horizontal partitioning technique. CUBRID Shard allows storing unlimited number of database shards and distributing data based on modulo, DATETIME, or hash/range calculations. With CUBRID Shard, application developers do not need to modify the application logic to divide a database into CUBRID Shards because it is automatically handled by the database system. However, CUBRID Shard cannot process a query when data of a user is distributed on the multiple CUBRID servers. Moreover, CUBRID Shard has a low usability because a user should specify a ‘shard_hint’ in the SQL when requesting the query.

3. Middleware based on the distributed CUBRID

3.1. System architecture and overall query processing procedure

The middleware consists of 4 components; communication component, query analysis component, metadata retrieval component with meta tables, and query result merge component.

First, a communication component is in charge of data transmission with a user or CUBRID servers. SQL query and database connection information such as dbName, ip, port, userID and password are transmitted through the communication component.

Second, a query analysis component performs an SQL parsing sent from a user. By doing so, the component extracts table names in from phrase that are used for retrieving meta tables. In addition, the component distinguishes the query types (e.g., insert, select, aggregation). If the query type is determined as an average type, the component
reconstructs the query by using sum and count to process the query on the distributed environment. This is because it is impossible to draw the final result by using average results sent from CUBRID servers.

Third, metadata retrieval component retrieves meta tables. There are 3 meta tables; 

\textit{MinMaxTable}, \textit{SearchTable}, and \textit{IpPortTable}. \textit{MinMaxTable} stores information for data insertion on the distributed CUBRID servers. The schema of the table is \{dbName, partition, tableName, column, min, max\}. The dbName is the name of the database. The \textit{column} means the name of the column that is used for horizontal partitioning of the tableName table. The \textit{partition} means a CUBRID server which stores records whose values of the \textit{column} are between \textit{min} and \textit{max}. \textit{SearchTable} stores information required for retrieving data stored in the distributed CUBRID servers. The schema of the table is \{userID, dbName, tableName, partition\}. By using the table, we can determine the partitions storing the tableName table that are necessary to process the query of the userID. Meanwhile, the detailed connection information of a partition stored in MinMaxTable and SearchTable are available in the IpPortTable. \textit{IpPortTable} stores connection information of each CURED server. The schema of the table is \{partition, ip, port\}. The \textit{ip} and \textit{port} are the information to connect the partition.

Finally, a query result merge component merges results sent from CUBRID servers. The middleware prepares a buffer for each CUBRID server to receive each query result without any collisions in parallel. In addition, the query result merge component sets an actual query result by eliminating duplicated results and by performing an aggregation if needed. Finally, the query result merge component sends the actual query result to the query issuer.

Meanwhile, the overall query processing procedure with the proposed middleware is as follows. First, a user sends an SQL query to the middleware. Second, by using the query analysis component, the middleware distinguishes a type of the query. Followings are the types of query that the middleware can distinguish and its corresponding role for each query type. i) Insert phrase: Distribute data to multiple servers. ii) Select phrase: Retrieve data in distributed parallel manner. iii) Order by phrase: Sort query results sent from CUBRID servers based on order by conditions. iv) Limit phrase: Extract records as much as the user wants to receive. Third, the middleware reconstructs the SQL query to be processed on the distributed CUBRID servers. Fourth, by using the query analysis component, the middleware extracts table names in from phrase. Fifth, by using metadata retrieval component, the middleware finds a list of CUBRID servers holding the required data to process the query and confirms the \textit{ip} and \textit{port} of the CUBRID servers. Sixth, the middleware generates packets to be sent to each CUBRID server. Seventh, by using the communication component, the middleware sends packets to the selected CUBRID servers. In addition, the middleware prepares a buffer for each CUBRID server to receive query results in parallel. Eighth, each CUBRID server processes the query and returns a query result to the middleware. Ninth, by using the query result merge component, the middleware draws the actual query result based on the query type. For this, the middleware eliminates duplicated results, sorts the query results, or performs an aggregation. Finally, the middleware finishes the query request by sending the actual query result to the client.

3.2. Query processing mechanism based on the query type

3.2.1. Insert phrase

Once the middleware analyzed that a query includes an insert phrase, the middleware stores data into the distributed CUBRID servers. To handle data insertion, information of the designated table should be stored in \textit{MinMaxTable}. So, if the information of the table is not stored in the \textit{MinMaxTable}, a middleware administrator should determine data partition strategy by considering a type of the column and the number of the CUBRID servers. After determining the data partition strategy, the administrator inserts \{dbName, partition, tableName, column, min, max\} data into the MinMaxTable. For example, assume that a partitioning column of the Student table in db01 database is the ID column. In this case, the middleware can generate a MinMaxTable as shown in Table 1.

Otherwise, if the designated table is stored in the MinMaxTable, the middleware performs data insertion. For example, for a given SQL query “Insert into Student(ID, name) values(20, ‘KIM’)”, the middleware can notice that the data should be inserted into the Student table. By referencing the MinMaxTable, the middleware confirms that the Student table is partitioned based on the ID column and the record with the ID value of 20 is related to the partition 1. Then, the middleware retrieves the IpPortTable to find the connection information of the partition 1. Assume that a tuple <partition, ip, port> = <1, 123.456.789.001, 9001> is stored in IpPortTable. The middleware
performs the data insertion by sending the SQL query to the CUBRID server (partition 1) by using the connection information. Through the mechanism, the middleware achieves the distributed data insertion.

<table>
<thead>
<tr>
<th>dbName</th>
<th>partition</th>
<th>TableName</th>
<th>column</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>db01</td>
<td>1</td>
<td>Student</td>
<td>ID</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>db01</td>
<td>2</td>
<td>Student</td>
<td>ID</td>
<td>50</td>
<td>100</td>
</tr>
</tbody>
</table>

3.2.2. Select phrase

If the middleware analyzes that a query includes a select phrase, the middleware retrieves databases in distributed manner. For this, at first, the middleware determines which tables should be retrieved by analyzing the SQL query. Then, the middleware retrieves SearchTable. For example, assume that user01 sends a query like “Select * from Student where age=21”. Assume that a tuple <id, dbName, TableName, Partition> = <user01, db01, Student, {1, 2}> is stored in SearchTable. The middleware can find that Student table of the user01 is distributed in partition 1 and partition 2. Then, the middleware accesses the IpPortTable to retrieve the connection information of the CUBRID servers. By sending the query to these CUBRID servers, data retrieval can be performed in parallel. In addition, the middleware checks whether the order by phrase or limit phrase exists or not. If it is, the middleware confirms the order by conditions, required columns with their data types (e.g., numeric data, character strings) and how many records should be sent to the client. The middleware confirms a type of a column by using CUBRID API.

Meanwhile, when processing the select query type, the middleware should consider following. The query result of each CUBRID server is sorted based on the order by conditions. If there is no order by phrase in the query, the query result of each CUBRID server is sorted based on the key value by default. So, the middleware should re-sort the query result sent from each CUBRID server based on the order by conditions to make the final query result. For this, the middleware performs following steps. i) The middleware finds the order by and limit conditions. ii) The middleware sorts them based on the order by conditions and appends the first record in the sorted result to the final result. v) The middleware sorts another record from the buffer where the record written to the final result was extracted. If there is a duplicated record, the middleware ignores it. These steps are repeated until all query results stored in the buffers are extracted. However, when there is a limit phrase, the middleware terminates the query processing when the middleware writes the required number of records to the final result.

3.2.3. Aggregation phrase

The middleware can support the aggregation queries that have not been handled on the distributed parallel computing environment. By using the query analysis component, the middleware can find what kinds of aggregation operations are included in the query. According to the type of the aggregation operation, the middleware calculates final result in different way. i) If the aggregation type is min or max, the middleware sends the query and receives the query result (minimum value or maximum value) from each CUBRID server. Among them, the middleware sets the smallest or largest value as the final result of the query. ii) If the aggregation type is count, the middleware sends the query and receives the query result (the number of records) from each CUBRID server. The middleware calculates the sum of these values and sets the total value as the final result of the query. iii) If the aggregation type is sum, the middleware sends the query and receives the query result (sum) from each CUBRID server. The middleware calculates the sum of these values and sets the total value as the final result of the query. iv) If the aggregation type is average, the middleware reconstructs the query by using sum and count. Then, the middleware sends the query and receives the query result (count and sum) from each CUBRID server. The middleware calculates the sum of these values respectively and calculates the average value (total sum / total count).

4. Performance evaluation

In this section, we present the extensive experimental results of our middleware. We compare our middleware with the existing CUBRID in terms of query processing time. Because CUBRID does not support parallel query
processing in distributed environments, we perform query processing of CUBRID in a sequential way. We assume that merge time of the intermediate results is same as our middleware. We use one master node and 3 slave nodes for the performance evaluation. We use CUBRID version 2.2.0 and compile the middleware using g++ 4.6.3 running on the Linux 3.5.0-23 with Intel® Core™ i3-3240 3.40Ghz CPU and 8 GB memory. We generate a million data which consists of 16 attributes according to Wisconsin Benchmark10.

Fig. 1 describes query processing time for select operation varying the number of data. The query processing time is increased as the number of data increases. Our proposed middleware shows about 1.8 times better performance than CUBRID. Fig. 2 shows query processing time for average operation varying the number of data. The query processing time is increased as the number of data increases. Our proposed middleware shows about 2.6 times better performance than CUBRID. This is because our middleware supports parallel query processing in a distributed environment.

![Select execution time](image1)

![Average execution time](image2)

5. Conclusion

With the sharply increasing amount of data, studies on bigdata processing based on NoSQL have been actively done. However, NoSQL cannot satisfy the ACID properties of database transactions. Therefore, bigdata processing based on RDBMS has been spotlighted. CUBRID Shard stores data in the distributed CUBRID servers by dividing the database. However, CUBRID Shard cannot process a query when data of a user is distributed on the multiple CUBRID servers. Therefore, in this paper we propose a CUBRID based middleware which supports distributed parallel query processing. Our proposed middleware can support users who are familiar with SQL to conveniently process the bigdata by using SQL statements. In addition, the middleware can support various aggregation operators. Through the performance evaluations, we show that our proposed scheme shows better performance than the existing work in terms of query processing time.

As a future work, we plan to expand our middleware to support various types of join with reasonable efficiency.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(2014065816).

References