Integrated Usage between Relational DBs and NoSQL DB

Kousuke Shiromoto
Graduate School of Information Science
Kyoto Institute of Technology
Kyoto, Japan

Teruhisa Hochin
Information and Human Sciences
Kyoto Institute of Technology
Kyoto, Japan
hochin@kit.ac.jp

Hiroki Nomiya
Information and Human Sciences
Kyoto Institute of Technology
Kyoto, Japan
nomiya@kit.ac.jp

***Abstract*—This paper proposes the system realizing integrated use of Relational Databases and a NoSQL Database. A heterogeneous information source integration system has been proposed. It enables equi-join and projection of tables without converting data. With this system, it became possible to handle various databases at the same time by using JDBC, but the databases which can be handled are only relational databases. Therefore, we add functions to the system to handle the NoSQL database which has been increasingly used in recent years. The table join execution time and the memory consumption of the system are experimentally evaluated. It showed that the performance is tolerable in practical use as compared with integrated use of only relational database. We also showed that it is practical to change the join order or to register column families which are unique elements of HBase.**

Keywords—integrated usage; JDBC; NoSQL; HBase;

I. Introduction

With the development and spread of computer technology, increasing the case that the data owned by oneself is registered in the relational database and handled on the computer. For example, relational databases are used in business places and academic studies. In addition, in recent years, the NoSQL database has been increasingly used to process a large variety of large amounts of data at high speed, mainly on online services such as SNS. Thus, there are many database management systems, and various databases are used according to the purpose of the user. These databases are not managed by a unified system, and management methods are diverse.

As a means of integrally using this heterogeneous information source, there is a method of selecting one type of database management system used and converting the format so that all data can be handled on the selected system. However, with this method, it takes time and effort to convert them, and there is a problem that it is difficult to maintain data consistency.

There are other ways of using integrated wrappers and mediators [1]. It converts it to a unified query by wrapper, accesses heterogeneous sources, and integrates heterogeneous sources by intermediary. However, since this method assumes heterogeneous information sources distributed over the network, it is necessary to make a computer be a server. Handling of servers is generally difficult because high-performance equipment and high skills are required.

Based on these backgrounds, a heterogeneous information source integration system that neither convert data and nor make computers be servers has been proposed [2]. This system is realized in the form of extended JDBC (Java Database Connectivity) which added a function to JDBC. There is a function to perform join and set operations on a table acquired from multiple databases. The system can handle four kinds of database management systems, MySQL, PostgreSQL, SQLite, and DB2, and two types of file Excel, and CSV. They can be connected simultaneously in the system. However, it is incompatible with the NoSQL database that has been used frequently in recent years, and has not been able to keep up with the progress of the times.

In this paper, HBase , a NoSQL database, can be handled by this system. By using JDBC for HBase, we realize a system that can integrate and utilize heterogeneous information sources without conversion, as with the handling of relational databases already implemented. In addition, as an evaluation experiment on a real machine, we measured the execution time and memory consumption when HBase was included in the join processing, compared with the case where the joining process was performed only in the relational database, and it is practical performance .

The remaining of the paper is structured as follows: Section 2 describes related matters. Section 3 describes related works. Section 4 describes the previous research. Section 5 shows concrete design. Section 6 describes implementation. Section 7 describes performance evaluation experiment. Section 8 concludes this paper.

II. Preliminary

A. Bigdata

With the development of the information society, we have come to see the term "big data." A video sharing website accepts various videos and anyone can see them. Online shopping sites have large product lists and huge purchase records, and can use them to purchase various products. These are so-called big data[3], [4]. Big data refers not only to huge data groups, but also to data with three major characteristics: volume, speed, and type. Big data is often managed using a database management system. However, if you do not choose an appropriate database management system, dealing with big data is very difficult.

B. *NoSQL Database*

NoSQL is a database management system other than relational database management system. As the name of "Not only SQL," languages and interfaces are not limited to SQL. NoSQL databases are based on BASE (Basically Available, Soft State, and Eventually Consistent) characterized by high availability of data [5], [6], [7]. On the other hand, it is at the expense of consistency compared to relational databases. Along with the processing capability of computers in recent years, most of NoSQL has highly optimized storage and retrieval of data that can ensure processing speed.

In recent years, scenes where handling of big data in relational databases becomes difficult have come up. For increases in the amount of data handled, processing speed, data diversity, and so on are increased. In the relational database, data consistency is maintained by a strict schema definition, but it takes a lot of time and effort for schema migrations. In response to these backgrounds, NoSQL has emerged. It is simple and flexible to change the data structure.

C. *Apache HBase*

HBase [8] is one of the open source software making up the ecosystem of Hadoop [9], which is a distributed database system built on Hadoop's distributed file system (HDFS) . Hadoop is the *de facto* standard processing infrastructure for big data. There are three modes of HBase: standalone mode, pseudo dispersion mode and complete dispersion mode. They are used in learning environments, testing environments, and commercial environments, respectively. Also, the standalone mode handles the local file system.

Because HDFS can read and write large numbers of files with high throughput, it is used for large amounts of data reference and analysis. However, it is a file system. It is not suitable for reading and writing large amounts of small data like a relational database. On the other hand, HBase can process large amounts of small data read/write at high speed. Therefore, HBase complements HDFS.

HBase is a master-slave type configuration, which consists of a master node that manages the entire cluster and a slave node that manages the data. In the slave node, HBase's RegionServer manages the data, requests the request, and caches the data, and the HDFS performs the data persistence. By keeping HBase and HDFS on the same node, data management and storage can be done smoothly.

The data model of HBase is tabular, but the entity of the data is kept in key value form. A value corresponds to the three keys of Row Key, Column (a combination of Column Family and Qualifier), and Timestamp. It is held in a state sorted in that order. Also, the Column Family is used to group the columns. To the Column Family, qualifiers can freely be added later, so data can be added and changed flexibly. Also, in Type information, Put is added to the key value added or overwritten, and Delete type value is added to the deleted key value. Delete key values are ignored and are deleted periodically completely.

D. *Apache Phoenix*

Apache Phoenix [10] is an open source relational database engine that supports HBase. In addition to table manipulation using SQL for HBase, JDBC driver is provided as an interface. In Java, it is possible to connect by specifying "jdbc: phoenix: [host name]" as the argument of getConnection() method. However, care must be taken because Phoenix can not be used when running HBase in standalone mode. Also, when creating a table by using Phoenix, the table name and the column name are unified with uppercase letters.

III. RELATED WORK

Atzeni et al. proposed programming interface called SOS (Save Our Systems) to support application development [11]. Since SOS implicitly specifies a schema, data conversion is unnecessary. Also, VilaÇa et al. implemented a distributed query engine (DQE) with full SQL support and combined with NoSQL to develop a prototype that can execute SQL queries while maintaining NoSQL scalability and schema flexibility[12]. Under the standard TPC-C workload, this prototype scales in proportion to the number of nodes in the system.

The proposal in this paper is to add a function to handle HBase in a previous research system that integratedly uses RDB without converting data or converting a computer into a server. A new table can be obtained as an output by simultaneously connecting to multiple RDB management systems and simultaneously operating multiple tables with one type of query. For example, assume that the user transfers data to RDB in order to hold the data temporarily held in NoSQL permanently.

IV. PREVIOUS RESEARCH

A. *Overview*

Heterogeneous information source integration system is realized in the form of extended JDBC using Java and adding functions to JDBC. The system can handle four kinds of database management systems: MySQL, PostgreSQL, SQLite, and DB2, and two types of files Excel, and CSV. It is possible to perform simultaneous connection to these, and perform equi-join, projections, and set operations on tables, In addition, it is possible to have multiple connections to the same type of database management system.

B. *Processing Overview*

The interface of this system is realized in the form of extended JDBC. Therefore, in addition to JDBC base classes "Connection class," "Statement class," and "ResultSet class," our own "Integrated Usage class" is implemented. This holds connection information. It is used when connecting a plurality of databases. A user can manipulate the corresponding table by

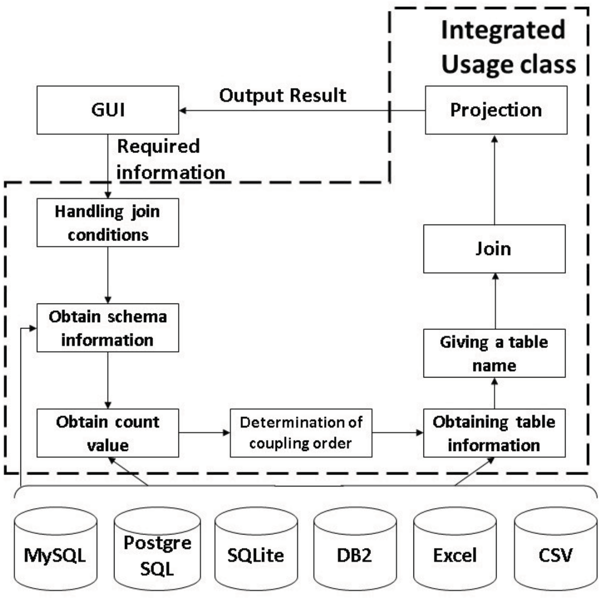

Fig. 1. Flow of heterogeneous information source integrated system[2]

registering the connection information in this class. For connection, use "set_ [DB name]" method, e.g., set_MySQL(),set_postgreSQL(), and set_SQLite(). The processing flow of this system is shown in Fig. 1 [2]. Use of the GUI in the figure is optional. In this system, left-deep tree is used as the join order of tables, and sort merge join is used as the join algorithm. The query statement is described in the form of "<Alias>: <Table Name>. <Column name>" according to SQL. The specification of <Alias> can be set independently by the user. By describing it at the head of the term, it is possible to uniquely identify the registered database.

C. System Issues

Since the database management system handled by JDBC is limited, it can handle only relational databases. Adding NoSQL to the databases that can be handled by the system is strongly required.

V. Design

To the previous system described in Section 3, it is designed to be able to handle equi-join or projection with HBase, which is a NoSQL database. In addition, hereinafter the previous system is described as "old system" for the sake of convenience.

A. Connection to HBase

Connecting to HBase follows the format of the old system. We prepare the "set_HBase()" method. Phoenix, HBase 's JDBC, can connect to the target database simply by giving a host name. In addition, since Phoenix can not handle HBase in the standalone mode, HBase connected needs to be started in the pseudo-distributed mode or the fully distributed mode.

B. Query Syntax

As described in Section 3, in the old system, in order to uniquely identify the registered database, it is possible to describe "<alias>:".

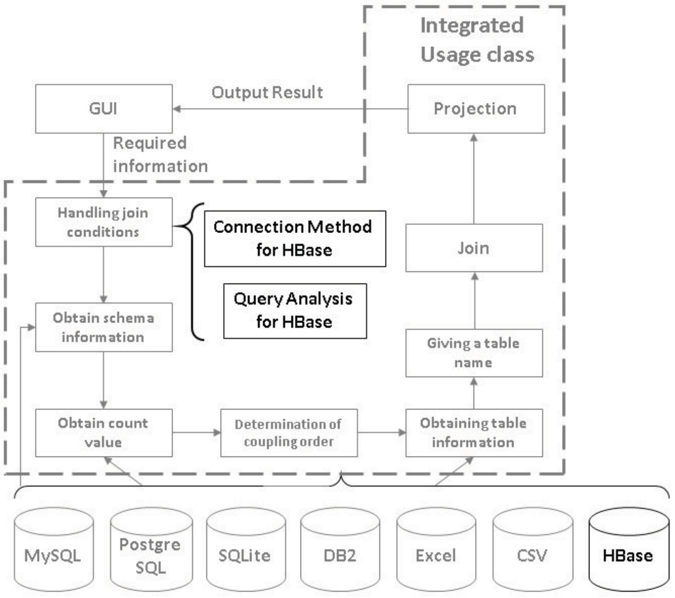

Fig. 2. RDBs and NoSQL DB integrated usage system

Here, in Phoenix, when accessing the table in which the column family in HBase is registered, it is necessary to specify the column name in the form of "<column family name>. <column name>". In addition, since it is possible to nest the column families, it is necessary to describe them continuously by separating them by dots (".") .

Therefore, in this system, different syntax is used for HBase. The syntax is as follows:

1) SELECT clause: "<alias>: <table name>. <column name>, ..." or "<alias>: <table name>. (<column family name>.) * <column name>, ..." in case of HBase.

2) FROM clause: "<alias>: <table name>, ..."

3) WHERE clause: "<CONDITION predicate> AND <CONDITION predicate> AND ..."

4) CONDITION predicate: <CONDITION term> = <CONDITION term>

5) CONDITION term: <constants> or <literal> or "<alias>: <table name>. <column name>" or "<alias>: <table name>. (<column family name>.) * <column name>" in case of HBase.

VI. Implementation

Fig. 2 shows an outline of the developed system. The bold parts in the figure are the newly implemented functions.

A. Create HBase Connection Method

In order to allow the system to connect to HBase, we introduced the set_HBase() method. The flow of outline processing up to connection is as follows:

1) Call set_HBase() method in JDBC class.

2) If the alias passed to set_HBase is unregistered, the host information is stored in the JDBC class.

3) Call the createConnection() method of the JDBC class.

4) Create the URL in the format "jdbc: phoenix: [host name]"

5) Give URL to getConnection() method and get Connection object from Phoenix.

B. Implementation of Query Analysis for HBase Column Family

Query analysis is performed based on the query syntax described in Section 4. The flow of analysis is as follows:

1) Decompose a given query string into SELECT clause, FROM clause, and WHERE clause.

2) Make sure that one of the three clauses contains HBase alias. If it is not included, the processing is passed to the query analysis of the old system.

3) From the CONDITION section of the SELECT clause or WHERE clause, check the number of dots (".") in the part containing HBase alias. If there is only one column, the column having the column family is not the target, so the processing is passed to the query analysis of the old system. If there are two or more dots ("."), it is judged that it is an access to a column having a column family.

4) Check whether the acquired column family is included in the target table.

*5) It holds "<column family name>. * <Column name>" for subsequent table operation and calls this character string each time it handles the corresponding HBase table.*

VII. Performance Evaluation

If the system uses HBase, we will check how much the processing power will be reduced compared with the case is not used, and consider whether it is a practical value.

A. Experimental Method

By changing the input data to the system, the execution time and the influence on the memory consumption by the processing of HBase are evaluated. The effect of registering a column family that is unique to HBase is also evaluated in order to confirm whether HBase can practically be handled.

The specifications of the actual machine used for the experiment are as follows:

- *OS:Ubuntu18.04*
- *Proccesor:IntelCorei7-7700CPU@3.60GHz3.60GHz*
- *Memory:8GB.*

The tables used in the experiment are as follows:

- *The number of tuples is 100,000.*
- *Column has "id," "name," and "loc."*
- *The column "id" contains an integer from 1 to 100,000. Do not allow overlapping, random ordering.*
- *The column "name" contains a random character string of fixed length 20.*
- *The column "loc" contains a random character string of fixed length 40.*

In each database system, an index is not created, and equi-join is performed with the column "id." Execute the system five times for each pattern, and measure the memory consumption consumed by the entire system and the execution time of the join operation.

Experiment1: Prepare four tables, select three tables out of them, and make them equally joined. Four tables prepared are managed with four types of MySQL, PostgreSQL, SQLite, and HBase. Under these conditions, the following four patterns are used as input data. For the sake of convenience, take the initial character of the database name in the order of association of each pattern as its pattern name.

MPS (MySQL, PostgreSQL, SQLite)
MPH (MySQL, PostgreSQL, HBase)
MSH (MySQL, SQLite, HBase)
PSH (PostgreSQL, SQLite, HBase)

Experiment2: It will be shown that MPH pattern is the most effective in the patterns including HBase. In order to clarify the best order of the MPH patterns, we change the join order, perform equi-join, and check the effect. The following six patterns are used as input data.

MPH (MySQL, PostgreSQL, HBase)
MHP (MySQL, HBase, PostgreSQL)
PMH (PostgreSQL, MySQL, HBase)
PHM (PostgreSQL, HBase, MySQL)
HMP (HBase, MySQL, PostgreSQL)
HPM (HBase, PostgreSQL, MySQL)

Experiment3: Confirm the effect of registering a column family. Register the column families in "name," and "loc" for the HBase table of MPH, MSH, and PSH in Experiment 1. The registration pattern of the column family is prepared as follows.

a) Register the same column family in "name" and "loc."

b) Register separate column families for "name" and "loc," respectively.

B. Experiment Result

Experiment1: The average execution time is shown in Fig. 3, and the average consumption memory is shown in Fig. 4 It is understood that patterns other than MPS not including HBase increase in execution time and consumption memory amount. Also, among them, MSH and PSH including SQLite are increasing in both execution time and memory consumption.

Experiment2: The average execution time is shown in Fig. 5, and the average consumption memory is shown in Fig. 6. It is understood that the execution time increases as the join order of HBase comes earlier. On the other hand, regarding the memory consumption, if the connection order of HBase is upper HMP and HPM, it can be understood that there is no difference in the value. However, no rule was found for the other patterns.

Experiment3: Fig. 7 shows the average execution time when column families are registered in "name," and "loc." The average of consumption memory is shown in Fig. 8. In both cases, it can be seen that the execution time and the consumption memory amount are increasing more than

Experiment 1 in which the column family is not registered. In addition, it can be seen from Fig. 7 that execution time is shortened by registering in separate column families, and from Fig. 8 that there is no effect on the amount of memory consumed.

C. *Considerations*

Experiment1: There are differences in measurement results depending on the database used. It is considered that this is the performance difference inherent in each database. In particular, when HBase is involved, it is inferred that it takes time and memory to convert the SQL sentence in Phoenix because it can not be processed by SQL statement originally. Also, compared with the other two relational database systems, processing including SQLite also tends to consume time and memory. This is thought to be caused by SQLite's handy handling, but on the other hand it is not suitable for processing large scale data covering tens of thousands of data. However, since we can process 100,000 records within 2,000 ms, we think that it is sufficiently practical.

Experiment2: As for the execution time, as in Experiment 1, it is considered that the performance difference inherent in each database is affected. The closer the join order comes, the more it is to use ResultSet to refer to the elements and values of that table. ResultSet retrieves data by communicating with the database. In HBase which does not have a JDBC interface originally, data conversion processing and communication are done within Phoenix every time ResultSet is used. Therefore, it is thought that it takes more time than the relational database having the JDBC interface. The flat memory consumption in HMP and HPM is considered to indicate the stability of the operation of Phoenix. However, for other patterns, it is necessary to find the rule in the future by increasing the data size.

Experiment3: From the results, changing the number of row families does not affect the amount of memory consumption, but it shows that there is a difference in execution time. Furthermore, it can be seen that execution time is faster when registering column families separately. This is because the system searches within the column family when acquiring the column name and its column data. When registering different column families this time, since there is only one column belonging to that column family, it is uniquely determined without searching.

VIII. Concluding Remarks

With the progress of informatization of society, every data has been digitized, and it has been increasing to manage them in databases. However, as there are numerous database management systems, and data are managed by various database management systems, integrated use is difficult. In order to integrate heterogeneous information sources, a heterogeneous information source integration system that enables equi-join and projection of tables was proposed. With this system using JDBC, it became possible to handle various databases at the same time. The only databases that can be handled are, however, relational databases. We have the problem of not being able to handle NoSQL which has become used in recent years.

Therefore, in this research, focusing on the NoSQL database, we realized integrated use of HBase and relational databases. Then, we evaluated the performance of table joining execution time and memory consumption of the system on the actual machines. It is shown that the performance is tolerable in practical use as compared with integrated use of only relational database. We also showed that it is practical to change the join order or to register column families which are unique elements of HBase. However, since the size of the data used in the experiment is small and the character string has a fixed length, it is necessary to conduct experiments closer to the practical environment in the future.

References

[1] H. Garcia-Molina, Y. Papakonstaninou, D. Quass, A. Rajaraman, Y.Sagiv, J. Ullman, V.Vassalos and J. Widom, “The TSIMMIS Approach to Mediation: Data Models and Languages”, Journal of Intelligent Information Systems, vol.8, no.2, pp.117-132, 1997.

[2] A. Terakawa, T. Hochin, H. Nomiya, “Integrated Usage of Heterogeneous Databases for Novice Users”, Proc. of International Conference on Advanced Applied Informatics, 2014, pp. 705-710.

[3] S. Sagiroglu and D. Sinanc, “Big Data: A Review,” Proc. of International Conference on Collaboration Technologies and Systems, 2013, pp. 42-47.

[4] D. Agrawal, S. Das, and A. E. Abbadi, “Big Data and Cloud Computing: Current State and Future Opportunities,” Proc. of the 14th International Conference on Extending Database Technology (EDBT/ICDT '11), 2011, pp. 530-533.

[5] D.Pritchett, “BASE: An Acid Alternative.”, ACM Queue, 2008, vol.6, no.3, pp. 48-55.

[6] Cook, J. D.: ACID versus BASE for database transactions, http://www.johndcook.com/blog/2009/07/06/brewer-cap-theorem-base/, 2009.

[7] Massimo Carro, NoSQL Databases, CoRR, 2014. http://arxiv.org/abs/1401.2101.

[8] Apache Software Foundation , HBase, https://hbase.apache.org/ (2019/1/15).

[9] Apache Software Foundation, Hadoop, https://hadoop.apache.org/ (2019/1/15).

[10] Apache Software Foundation, Phoenix, https://phoenix.apache.org/ (2019/1/15).

[11] R. VilaÇa, F. Cruz, J. Pereira and R. Oliveira, “An effective scalable SQL engine for NoSQL databases.”, Proc. of In IFIP International Conference on Distributed Applications and Interoperable Systems, 2013, pp. 155-168.

[12] P. Atzeni, F. Bugiotti and L. Rossi, 2012, June, “Uniform access to non-relational database systems: The SOS platform.”, In Proc. of International Conference on Advanced Information Systems Engineering, pp. 160-174.

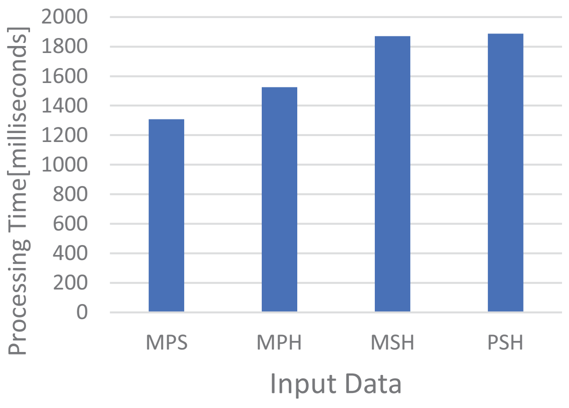

Fig. 3. Experiment 1 Result (Processing Time)

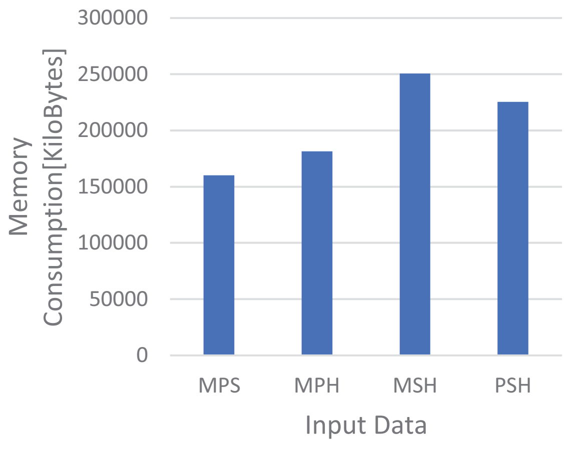

Fig. 4. Experiment 1 Result (Memory Consumption)

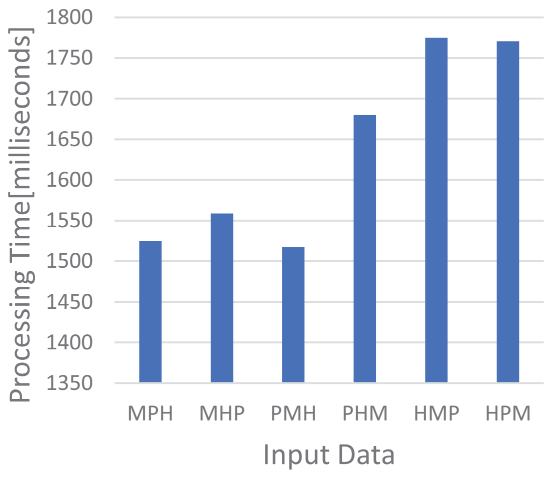

Fig. 5. Experiment 2 Result (Processing Time)

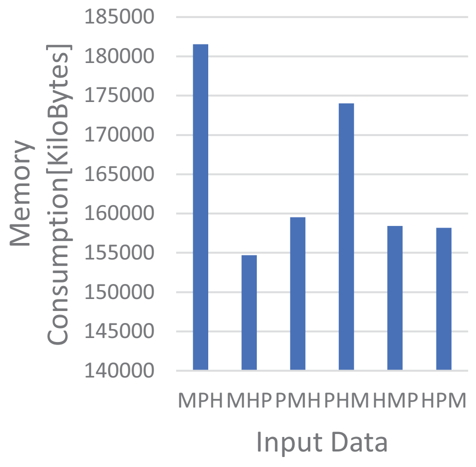

Fig. 6. Experiment 2 Result (Memory Consumption)

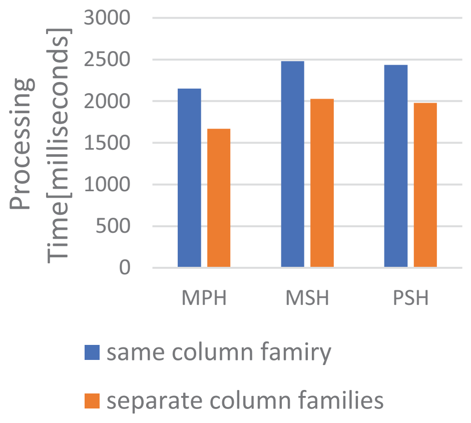

Fig. 7. Experiment 3 Result (Processing Time)

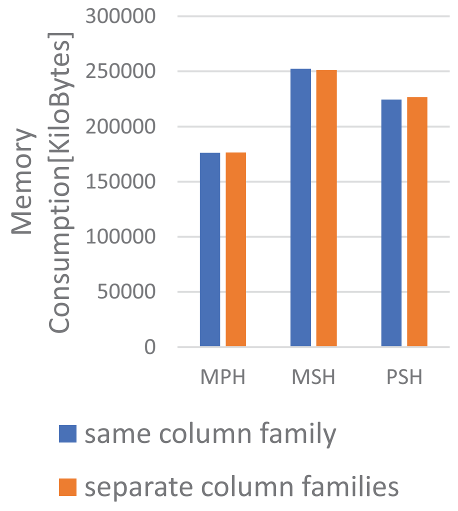

Fig. 8. Experiment 3 Result (Memory Consumption)