ASAWA: An Automatic Partition Key Selection Strategy

Xiaoyan Wang[1,2,3], Jinchuan Chen[3], and Xiaoyong Du[1,3,*]

[1] School of Information, Renmin University of China
[2] School of Information and Electrical Engineering, Ludong University
[3] Key Laboratory of Data Engineering and Knowledge Engineering, MOE
{wxy,jcchen,duyong}@ruc.edu.cn

Abstract. With the rapid increase of data volume, more and more applications have to be implemented in a distributed environment. In order to obtain high performance, we need to carefully divide the whole dataset into multiple partitions and put them into distributed data nodes. During this process, the selection of partition key would greatly affect the overall performance. Nevertheless, there are few works addressing this topic. Most previous projects on data partitioning either utilize a simple strategy, or rely on a commercial database system, to choose partition keys. In this work, we present an automatic partition key selection strategy called ASAWA. It chooses partition keys according to the analysis on both dataset and workload schemas. In this way, intimate tuples, i.e. co-appearing in queries frequently, would be probably put into the same partition. Hence the cross-node joins could be greatly reduced and the system performance could be improved. We conduct a series of experiments over the TPC-H datasets to illustrate the effectiveness of the ASAWA strategy.

Keywords: partition key selection, data partitioning.

1 Introduction

Nowadays, more and more applications need to manage huge volume of data and serve large amount of requests concurrently. As it is too hard for a single machine to support the whole system operations, these applications are mostly installed on clusters containing multiple data nodes. Thus, the whole data in an application needs to be partitioned and then stored in different machines. The goal of a partitioning algorithm is to achieve low response time and high throughput by minimizing inter-node communications.

During data partitioning, partition key selection, which would select attributes for tables as the baseline in partitioning, is the most important decision made regarding the scalability of database [1]. Data in different partitions are serviced by different processing nodes, and concurrent Read and Write operations over multiple partitions will be a bottleneck for applications. Poor partition key selection can then greatly affect query performance [2]. For example, an application, which has partitioned data

* Corresponding author.

Y. Ishikawa et al. (Eds.): APWeb 2013, LNCS 7808, pp. 609–620, 2013.

by the *Location* attribute but accesses data with queries containing many joins on the *Date* column, would undoubtedly has an expensive communication cost.

Previously, there have been many efforts on the problem of data partitioning [1, 2, 5-8]. However, few of them address the problem of partition key selection. Instead, most of them utilize some simple strategy, while some others rely on a commercial database system for this selection.

In this work, we discuss how to select partition keys automatically for tables in applications and propose ASAWA (**A**utomatic **S**election based on **A**ttribute **W**eight **A**nalysis) strategy. Its general idea is based on the observation that if an attribute is found to be accessed in many queries, it would probably act as keys in join operations. Thus it would be better to partition datasets according to the attributes in search condition defined in query statements. The basic process of ASAWA would first compute an aggregated score for each attribute by identifying the data dependency among tables and finding out associated attributes, and then select partition keys with top scores evaluated by grouping features in workload schema. Combined both data and workload information predefined, it is expected to reduce communication cost among data nodes and improve system performance.

The contributions of this paper are summarized as follows:

1. We propose an automatic partition key selection strategy.
2. We implement the strategy within a novel algorithm for choosing partition keys based on data and workload information.
3. We conduct a series of experiments on TPC-H [3] to show both the efficiency and effectiveness of our proposed approaches.

The remainder of this paper is organized as follows. Section 2 presents related works in data partitioning with particular focus on partition key selection. Next, Section 3 illustrates the automatic partition key selection method ASAWA. Using three kinds of partition key selection strategies, Section 4 describes the performance comparison to show the efficiency of ASAWA. Finally, we have briefly summarized the contributions in this work and discussed works in the future in Section 5.

2 Related Works

Data partitioning [4] is known as one of the most famous technologies for its efficiency in large-scale application scalability. Traditionally, there are 3 kinds of partitioning strategy: horizontal partitioning [5], vertical partitioning [6] and hybrid approaches [7], any of which needs to select a partition key for each relation as the basis of partitioning algorithms. In recent years, there have emerged some new researches, which do not need to select partition keys. Schism [8] uses Metis [9] to divide mapping graph of relations in data into fine-grained partitioning results. In [2], it has proposed a method to have a skew-aware partitioning solution based on large-scale neighborhood search. It would like to select the most frequently accessed column in the workload as the partition key for each table originally, and change it when a better one

founded during the process of relaxation. As followed by the partitioning strategies, when there are too many columns in a large amount of tables, the search space would be fairly huge. Designed in a bottom-up view, both of them need to be recalculated and partitioned if there is incremental data. Consequently, this kind of solutions is much more suitable for the applications with fixed data and workloads.

The ideal solution is supposed to partition data without manual assignment, while the performance is as well as the best-known. In most research works, they assume [10] that the proper partitioning keys have been chosen, and concentrate on partitioning algorithms with less aware on partition key selection. Although many traditional commercial database systems (e.g. Oracle [11], DB2 [12], SQL Server [13], etc.) have mentioned the importance of partition key selection and have their own mechanisms for recommendation, most of them are not open sources and combined deeply with underlying database optimizers. Therefore, they could not be applied on others directly. Based on some guidelines [14, 15], it is mostly operated manually in cases that the data management architecture is new or open source.

Basically, there are four kinds of partition key selection strategies:

- **Primary Key Selection:** This is the most widely used partition key selection strategy as default in commercial database systems. One of its variations is to use foreign key or the first column in a table as its partition key.
- **Random Selection:** This occurs when partition keys are specified without sufficient database management experiences or by systems without any constrains.
- **Exhaustive Selection:** This would like to try all possible partition key groups to have the one with best performance. It needs to use a sample data and workload set to simulate as in the real world, which is hard to predict the cost correctly.
- **Analytical Selection:** This is based on data and workload schema analysis, which is usually applied manually, or recommended by optimizing advisors integrated within expensive commercial database systems.

As data amount increasing rapidly and workload requests variously, this work would concentrate on automatically partition key selection from a top-down view that suits the requirements from data and workload. It prompts ASAWA method for automatic partition key selection, which not only takes the data and workload schemas into consideration, but also combined with design and execution information.

3 Partition Key Selection

To illustrate the problem of partition key selection, consider the following motivating scenario: *Given a data center with many query workloads, it needs to build a data management configuration that consists of many storage nodes and to partition data to suitable nodes so that the workloads could be executed in parallel. As the first step in data partitioning, a critical problem is how to select partition keys for relationships in the data center to minimize communication costs as much as possible.*

3.1 Problem Description

For a well-designed application, the basic information of the system, like data schema, workload pattern and their utility features, is given before data partitioning. In this work, we focus on applications with well defined data schema and static workload pattern which could be obtained with access frequencies for queries in it.

The partition key selection problem can be generally formulated as follows:

Definition. **Partition Key Selection**: Given a database $\boldsymbol{D} = \{R_1,R_2,\ldots,R_d\}$, a query workload $\boldsymbol{W} = \{Q_1,Q_2,\ldots,Q_w\}$, and a storage bound $\boldsymbol{S} = \{B_1,B_2,\ldots,B_s\}$, select an attribute PKS_x $(1 \leq x \leq d)$ for each table as the partition key. Here,

- R_x $(1 \leq x \leq d) = <A_{x1}, A_{x2},\ldots, A_{xt}>$ is a predefined table schema with a scale factor coefficient C_x in database D. A_{xi} $(1 \leq i \leq t_x)$ is one of the attributes in R_x.
- Q_y $(1 \leq y \leq w)$ is a predefined query pattern with an access frequency AF_y in workload W,
- B_z $(1 \leq z \leq s)$ is the storage bound of node z.

The total execution time T_{total} consists of two parts: data loading time T_{load} and the whole workload execution time T_{exec}. In this work, we assume all the data nodes are the same in physical features and use the widely used Hash function $h1:S \rightarrow [0,1]$ to distribute datasets into data nodes uniformly. The best partition key selection solution should partition data to nodes and complete queries within the least response time.

3.2 ASAWA Method

The basic process of ASAWA would first compute an aggregated score for each attribute based on identifying the dependency among tables and finding out the corresponding associated attributes, and then select partition keys with top scores optimized by grouping features in workload schema.

Database Analysis

Data Schema

As mentioned in Section 3.1, we assume that the data schema is well defined to guarantee data integrity for the database and all of the basic information (including attribute name, data type, constraints in the table, etc.) could be obtained.

To guarantee the efficiency of ASAWA, we would like to select the attributes with data integrity declarations as partition key candidates, including *Primary Key*, *Foreign Key* and columns declared *Unique* or *Not Null*. The weight of affect in dimension of data schema for an attribute could then be assigned equally as the values of $w_{DS}(PK)$, $w_{DS}(FK)$, $w_{DS}(U)$ and $w_{DS}(NN)$ respectively for each of declarations mentioned above. The values of weights could be the same or different, which depend on specific applications. The existences of the 4 kinds of statements are present as $e_{DS}(PK)$, $e_{DS}(FK)$, $e_{DS}(U)$ and $e_{DS}(NN)$ respectively, whose value is 1 or 0 based on the integrity statement on the relative attribute.

For a given attribute in the data schema, its weight in dimension of data schema is shown as $w_{DS}(GA)$ in Formula (1), which is the product of the existences that have defined integrity statements on it and the values of weights in a relative sequence:

$$w_{DS}(GA) = \sum(w_{DS}(PK) \times e_{DS}(GA_{PK}) + w_{DS}(FK) \times e_{DS}(GA_{FK}) + w_{DS}(U) \times e_{DS}(GA_{U}) + w_{DS}(NN)) \times e_{DS}(GA_{NN})) \quad (1)$$

Scale Factor.

A scale factor is the number used as a multiplier in scaling. This is mainly used in data generation. In ASAWA, the weight of affect for a given attribute in dimension of scale factor is as in Formula (2):

$$w_{SF}(GA) = 1 + log\,(\,SF_x\ /\ SF_{MIN}\,) \quad (2)$$

in which $SF_{MIN} = min\{\,SF_1, SF_2, \ldots, SF_d\,\}$.Obviously, $min(w_{SF}(GA))=1$.

In some applications, each table's volume might increase in the same growth rate, which means that the scale factor for each of them is 1. Thus, w_{SF} of each table is *1*.In most of real world applications, tables' volumes are not increased in the same pace. For example, in TPC-H, the scale factor coefficient of table Lineitem is 6000000, while Orders's is 15000000 and Supplier's is 10000. According to Formula (2), $w_{SF}(A_{MIN}) = w_{SF}(A_{SUPPLIER}) = 1$, $w_{SF}(A_{LINEITEM}) \approx 3.78$, and $w_{SF}(A_{ORDER}) \approx 4.18$.

An extreme case is that, there are tables that too tiny to be worth partitioning. For example, in an OLAP application with Star Schema, the volume of fact tables increases fast while dimension tables always stay the same. These tiny tables are considered do not have scale factor. Instead of partitioning, they should be replicated for considerations on system performance.

Thus, the total weight of a given attribute in database is present in Formula (3):

$$w_{DATABASE}(GA) = w_{DS}(GA) \times w_{SF}(GA) \quad (3)$$

Workload Analysis

Query Pattern

Query pattern is a piece of code that will make a reasonably normal query expression compile. As mentioned in Section 3.1, we assume that each query would be executed using the same structure with variable parameters. This ensures that the attributes touched in each query is the same in every execution of it. As mentioned in [10], The most relevant operations that present the attributes touched in SQL statements include equi-joins, Group By operations, duplicate elimination, and selections. The weights of them in dimension of query pattern are as $w_{QP}(Join)$, $w_{QP}(GB)$, $w_{QP}(DR)$, $w_{QP}(SelectConstant)$ and $w_{QP}(SelectHostVariable)$ relatively, while the existences as $e_{QP}(Join)$, $e_{QP}(GB)$, $e_{QP}(DR)$, $e_{QP}(SelectConstant)$ and $e_{QP}(SelectHostVariable)$ respectively, which are the counts of their appearance in the query.

For a given attribute in a given query, the weight of it in dimension of data schema is shown as $w_{QP}(GA_{GQ})$ in Formula (4):

$$w_{QP}(GA_{GQ}) = \sum(w_{QP}(Join) \times e_{QP}(Join) + w_{QP}(GB) \times e_{QP}(GB) + w_{QP}(DR) \times e_{QP}(DR) + w_{QP}(SelectConstant) \times e_{QP}(SelectConstant) + w_{QP}(SelectHostVariable) \times e_{QP}(SelectHostVariable)) \quad (4)$$

Access Frequency

Access frequency represents how many times an application runs in a given period. Its measurement is decided by the designer, e.g. an hour, a day, et al. In ASAWA, we assume the access frequency is steady for queries in the workload and define that the weight of affect in dimension of access frequency is as in Formula (5):

$$w_{AF}(Q_y) = 1 + \log(AF_y / AF_{MIN}) \quad (5)$$

in which $AF_{MIN} = min\{AF_1, AF_2, \ldots, AF_w\}$. Obviously, $min(w_{AF}(Q_y))=1$.

For example, we assume that the access frequency of Q_1 and Q_{11} is 7000 and 150 respectively during an hour in TPC-H, while the minimized access frequency is 3 on Q_{21}. According to Formula (6), $w_{AF}(Q_{MIN}) = w_{AF}(Q_{21}) = 1$, $w_{AF}(Q_1) \approx 3.37$ and $w_{AF}(Q_{11}) \approx 2.70$.

Thus, the total weight of a given attribute in workload is present in Formula (6):

$$w_{WORKLOAD}(GA) = \sum_{y=1}^{w}(w_{QP}(GA_{Qy}) \times w_{AF}(Q_y)) \quad (6)$$

According to the above analysis, the general weight of a given attribute w(a) could be calculated as presented in Formula (7):

$$w(a) = w_{DATABASE}(a) \times w_{WORKLOAD}(a) = w_{DS}(a) \times w_{SF}(a) \times \sum_{y=1}^{w}(w_{QP}(a_{Qy}) \times w_{AF}(Q_y)) \quad (7)$$

ASAWA Algorithm

Although there have been many partition key selection strategies for manual operation tips, how to combine these tips for automatic partition key selection is a problem. Here we propose ASAWA (**A**utomatic **S**election based on **A**ttribute **W**eight **A**nalysis) method to solve this issue.

The whole ASAWA algorithm is listed in Fig.1. Combining both data and workload information predefined, it takes attributes with data integrity constrains into consideration to reduce the complexity in practice and is expected to reduce communication cost among data nodes and benefit for system performance.

4 Experimental Study

In this section, we have shown results of our experiments using different partition key selection strategies and compared with the results with analysis.

Input:

1. Data schema of the designed database
2. Scale Factors of tables defined in the data schema
3. Query Pattern of the designed workload
4. Access Frequency of each query in the workload

Output:

Solution $PKSD = \{PKS_1, PKS_2, \ldots, PKS_x\}$ $(1 \le x \leqslant d)$, in which PKS_j $(1 \le j \le x)$ is an attribute in table $T_x \in D$.

Procedure ASAWA:

1. Eliminate attributes without impact and obtain partition key candidates with its weight $w(a)$ for each.
   a. Extract the attributes with data integrity constrains from the data schema.
   b. Assign the weight of each attribute in dimension of data schema, scale factor, Query Pattern and access frequency as $w(a)$ using Formulate (8).
2. Exhaustively consider all combinations of attributes that are not eliminated:
   a. Call optimizer with candidate set and grouping to estimate execution times of queries.
   b. Estimate aggregate Response time of all queries, and update {best partition key, relation grouping} to the current one to choose the best.
3. Output $PKSD = \{ PKS_1, PKS_2, \ldots, PKS_x\}$ $(1 \bullet x \bullet d)$ over D as the partition key for the given data schema.

Fig. 1. The ASAWA Algorithm

4.1 Environment and Configuration

In the experiment, we have setup 3 data nodes, each of which is equipped with 2.66GHz * 8 cores, more than 300G disk, and CentOS 6.1 x86-64 with Open SSH 5.2 inside. As the scenario mentioned in Section 3, we select Greenplum Database (GPDB for short) 4.2.0.0 [26] as the platform. The dataset and queries used are generated from TPC-H 2.14.2, which has well-defined data schema and query patters to illustrate a system that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. The node with 32G main memories is chosen as the master node, while the other two nodes have 16G main memory and are set as slave nodes. Each of them has 8 segments.

4.2 Experimental Procedure

A complete experimental procedure is shown in Fig. 2. It would be executed in every round of each strategy on a generated dataset with a specific scale factor. To be fair, this procedure should be repeated in each round of every case.

```
Input:
   1. A specific scale of datasets generated from TPC-H
   2. Queries defined in the benchmark
Output:
   1. Data partitioning time
   2. Executing time of each query
   3. Total time of the whole process
Procedure:
   1. Analysis the partition key for each tables based on
   applied strategy within the round.
   2. Generate a new TPC-H dataset with the specific
   scale factor
   3. Partition data to data nodes based on keys se-
   lected, use Hash function h1:S•[0,1] to distribute it
   to nodes and record this time period as data loading
   time T_load.
   4. Execute specific query set for this round and
   record this time period as query execution time T_exec.
   5. Record the whole process executing time as T_total
   when the last query finishes its execution.
```

Fig. 2. One Round of Experimental Procedure in Each Case

It needs to select partition keys based on different strategies in the first step. In this work, we implement three partition key selection strategies for comparison, which are based on Primary Key, Random Assigned and ASAWA respectively. The detail descriptions of the first two strategies could be found in Section 2. For ASAWA, we set the weights in dimension of query pattern as: $w_{QP}(Join) = 1.0$, $w_{QP}(GB) = 0.1$, $w_{QP}(DR) = 0.08$, $w_{QP}(SelectConstant) = -0.05$ and $w_{QP}(SelectHostVariable) = 0.05$ as in [10]. In order to present the underlying structure information in the schema, we assign $w_{DS}(PK) = w_{DS}(FK) = 1$ and $w_{DS}(U) = w_{DS}(NN) = 0.5$ respectively for $w_{DS}(GA)$ in this work. Then we could run Procedure ASAWA shown in Fig. 1 for partition key selection for case ASAWA with the specific scale factor in this round.

In the TPC-H Schema, there are two tables named Nation and Region respectively that are not affected by scale factors. For the size for both of them are fixed and quite limited, instead of partitioned, both of them are copied to each data node in every selected strategy case for performance consideration.

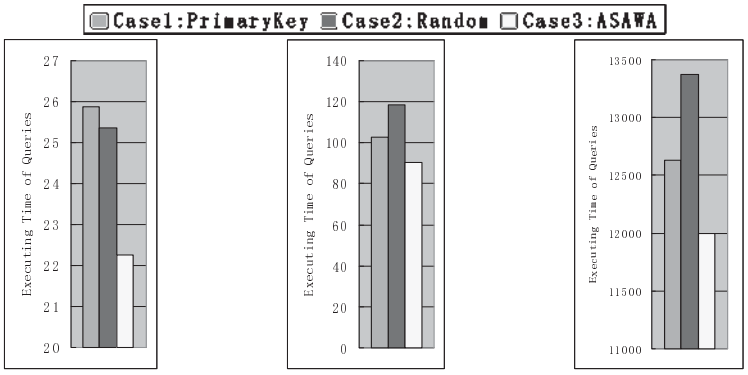
4.3 Results and Analysis

As a matter of fact that the servers may have many different networks and application status, the running result is most probably different from each other, more or less.

In order to get a steady result and have a much more reliable result analysis, each strategy with relative dataset and workload in a specific scale should be executed multi round in the same physical environment. Due to the limitation of current hardware environment, the scale factor of dataset is set to be 1, 10 and 100 respectively. We run query set 5 times within the newly partitioned dataset in every round.

Overall Results

As a decision support oriented benchmark, it is hard to predict the access frequency of queries in TPC-H. We first all of the 22 queries in the data set of scale factor =1, 10 and 100 respectively. The results are shown in Figure 3.



(a) scale factor = 1 (b) scale factor = 10 (c) scale factor = 100

Fig. 3. Executing Time of 22 Queries

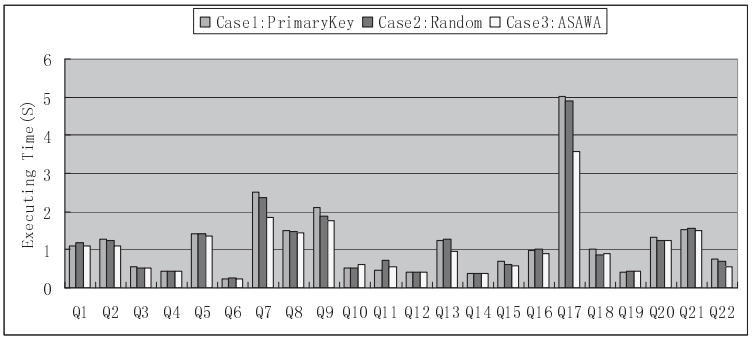
From the series figures shown in Fig. 3, we could see that in each scale factor round, Case 2: Random Strategy always runs the longest time and shows the worst performance, Case 1: Primary Key Strategy is faster than Case 2 and shows fine performance, while Case 3: ASAWA is even better than Case 1 and always shows the best performance in the 3 scales of datasets. Besides the results shown in Figure 3, we have also recorded data loading time in each round of every case. In these records, Case 2 shows obviously longer data loading time than the others.

The result verifies again that the performance in queries is affected by the key selection switch. Also, we found that compared with Case 1, the advantages of Case 3 in Figure 3(c) are not so distinct than it in Figure 3(a) and 3(b). This is because GPDB has implemented a mechanism named Motion to do live migration and eliminate partition differences during running time, which means the long time running, the less distance. But this would add extra workload to data nodes and are not common implemented in every product.

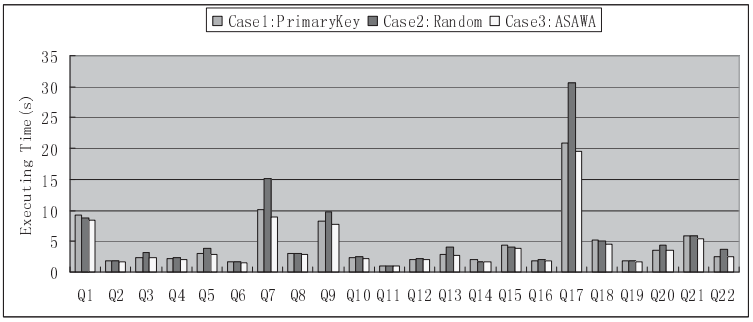
Specific Query Results

The series of figures shown in Fig. 4 indicate the average executing time of each query in the data set of scale factor =1, 10 and 100 respectively. From these figures, we could see that for most TPC-H queries in each scale factor round, Case 2 runs the

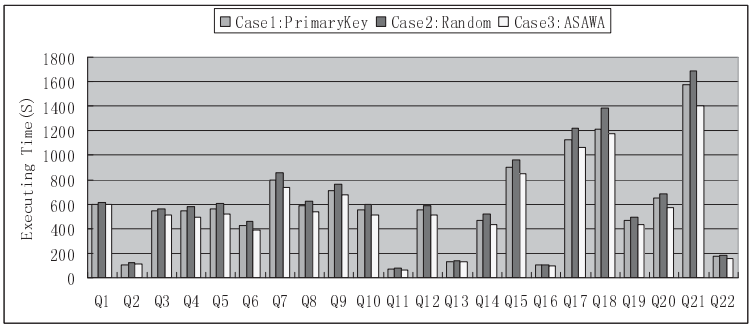
longest time and shows the worst performance, Case 1 shows fine performance than Case 2 especially in large scales, while Case 3 is shows the best. This indicates that the partition strategy should take the actual workloads of applications into consideration to get much better performance. Due to identify tuples specifically but having not considered the business semantical information, Primary Keys strategy always obtains reasonable but limited performance. For improper partition keys selection, Random strategy leads to bad system performance. Having taken data and workload features into consideration, ASAWA shows the best performance in all of these 3 strategies.



(a) scale factor = 1



(b) scale factor = 10



(c) scale factor = 100

Fig. 4. Executing Time of Each Query

Besides, we have intentionally selected Q1, Q5 and Q13 from the 22 queries of TPC-H, in which Q1 is represented for queries on a single large table, Q5 for queries on multi tables and Q13 for queries only on small tables, to take a continuous running for 7 rounds respectively on the dataset with the sale factor set as 100. The result shown in Fig.5 indicates that, for the queries that data distributed originally and memory could not cached, the executing time does not change too much, like Q1 and Q5, while for the queries that memory cacheable as Q13, the executing time is the most at the first time, less in the second time, and lest and maintaining in the following 5 times. It also shows the downtrend slightly in Q1 and Q5. Due to the automatic motion operation (like live migration) in GPDB, Round 3-7 show steady results. Obviously, if similar queries could be scheduled closer, which our strategy want to achieve, executing performance would be improved a lot.

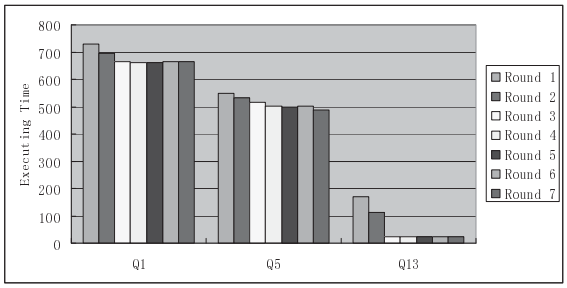


Fig. 5. Continuous Running of Single Query (scale factor = 100)

5 Conclusion

As more and more applications need to manage data in huge volume and response large amount of requests concurrently, the whole dataset for an application need to be partitioned into multiple parts and put into different machines. During this process, the selection of partition key would greatly affect the partitioning results. Nevertheless, there are few works addressing this topic. In this work, we present an automatic partition key selection strategy, called ASAWA. It would first compute an aggregated score for each attribute based on identifying the dependency among tables and finding out the corresponding associated attributes, and then select partition keys with top scores optimized by grouping features in workload schema. In this way, close tuples, i.e. co-appearing in queries frequently, would be probably put into the same partition. Hence the inter-node joins could be greatly reduced and the system performance could be improved. We then conduct a series of experiments over TPC-H queries with datasets generated in different scale factors. The experiment results illustrate the effectiveness of ASAWA method.

For most applications with huge data volume and great concurrent requests, partitioning strategy should be well planed before data storage and would be difficult to change once it adopted. If it could be adjusted automatically according to the actual applications, this would be undoubtedly more meaningful and useful for the widely use of database system products and release the burden of database administrators or developers. Partition key selection is the first step in data partitioning, and we only

use empirical weight values assigned in ASAWA in the experiments. In the future work, we would like to adjust the values by learning methods to get a more practical weight set. Of course, there must be some differences among application features and visions. We would like to explore an automatic partitioning method combined with both ASAWA and proper partitioning algorithms in further research.

Acknowledgements. This work is supported by State Key Laboratory of Software Development Environment Open Fund under Grant No.SKLSDE-2012KF-09, the National Science Foundation of China under Grant No. 61003086, and the Graduate Student Scientific Research Foundation of Renmin University of China under Grant No. 42306176. The authors would like to thank the anonymous reviewers for their helpful comments and valuable suggestions.

References

1. Zilio, D.C.: Physical Database Design Decision Algorithms and Concurrent Reorganization for Parallel Database Systems. PhD Thesis, Department of Computer Science, University of Toronto (1998)
2. Pavlo, A., Curino, C., Zdonik, S.: Skew-aware Automatic Database Partitioning in Shared-Nothing Parallel OLTP Systems. In: Proc. of the ACM SIGMOD, pp. 61–72 (2012)
3. TPC Benchmark™ H, http://www.tpc.org/tpch/
4. Stonebraker, M., Cattell, R.: 10 Rules for Scalable Performance in 'Simple Operation' Datastores. Communications of the ACM 54, 72–80 (2011)
5. Ceri, S., Negri, M., Pelagatti, G.: Horizontal Data Partitioning in Database Design. In: Proc. of the ACM SIGMOD, pp. 128–136 (1982)
6. Navathe, S., Ceri, G., Wiederhold, G., Dou, J.: Vertical Partitioning Algorithms for Database Systems. ACM Transactions on Database Systems 9(4), 680–710 (1984)
7. Agrawal, S., Narasayya, V., Yang, B.: Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design. In: Proc. of the ACM SIGMOD, pp. 359–370 (2004)
8. Curino, C., Jones, E., Zhang, Y., Madden, S.: Schism: a Workload-Driven Approach to Database Replication and Partitioning. Proc. of the VLDB Endowment 3, 48–57 (2010)
9. Metis, http://glaros.dtc.umn.edu/gkhome/views/metis/index.html
10. Zilio, D.C., Jhingran, A., Padmanabhan, S.: Partition Key Selection for a Shared-nothing Parallel Database System. Technical Report RC 19820(87739) 11/10/94, IBM T. J. Watson Research Center (1994)
11. Eadon, G., Chong, E.I., Shankar, S., Raghavan, A., Srinivasan, J., Das, S.: Supporting Table Partitioning by Reference in Oracle. In: Proc. of the ACM SIGMOD, pp. 1111–1122 (2008)
12. Zilio, D.C., Rao, J., Lightstone, S., Lohman, G., et al.: DB2 Design Advisor: Integrated Automatic Physical Database Design. In: Proceedings of the VLDB, pp. 1087–1097 (2004)
13. Nehme, R., Bruno, N.: Automated Partitioning Design in Parallel Database Systems. In: Proc. of the ACM SIGMOD, pp. 1137–1148 (2011)
14. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 3rd edn. Springer, New York (2011)
15. Rahimi, S., Haug, F.S.: Distributed Database Management Systems: A Practical Approach. IEEE Computer Society, Hoboken (2010)
16. Greenplum Database, http://www.greenplum.com/products/greenplum-database