A Study of Several Matrix-Clustering Vertical Partitioning Algorithms in a Disk-Based Environment

Viacheslav Galaktionov[1], George Chernishev[1,2(✉)], Kirill Smirnov[1], Boris Novikov[1], and Dmitry A. Grigoriev[1]

[1] Saint-Petersburg State University, Saint-Petersburg, Russia
viacheslav.galaktionov@gmail.com,
{g.chernyshev,k.k.smirnov,b.novikov,d.a.grigoriev}@spbu.ru
[2] JetBrains Research, Saint-Petersburg, Russia

Abstract. In this paper we continue our efforts to evaluate matrix clustering algorithms. In our previous study we presented a test environment and results of preliminary experiments with the "separate" strategy for vertical partitioning. This strategy assigns a separate vertical partition for every cluster found by the algorithm, including inter-submatrix attribute group. In this paper we introduce two other strategies: the "replicate" strategy, which replicates inter-submatrix attributes to every cluster and the "retain" strategy, which assigns inter-submatrix attributes to their original clusters. We experimentally evaluate all strategies in a disk-based environment using the standard TPC-H workload and the PostgreSQL DBMS. We start with the study of record reconstruction methods in the PostgreSQL DBMS. Then, we apply partitioning strategies to three matrix clustering algorithms and evaluate both query performance and storage overhead of the resulting partitions. Finally, we compare the resulting partitioning schemes with the ideal partitioning scenario.

Keywords: Database tuning · Vertical partitioning · Experimentation · Matrix clustering · Fragmentation · TPC-H · PostgreSQL

1 Introduction

The vertical partitioning problem [4] is one of the oldest problems in the database domain. There are dozens or even hundreds of studies available on the subject. It is a subproblem of the general database physical structure selection problem. It can be described as follows [8]: find a configuration (a set of vertical fragments) which would satisfy the given constraints and provide the best performance. There are two major classes of approaches to this problem:

- Cost-based approach [2, 15, 20, 32]. Studies that follow this approach construct a cost model which is used to predict the performance of a workload for any given configuration. Next, an algorithm enumerating the configuration space is used.
- Procedural approach [28, 31, 34]. These studies do not use the notion of configuration cost. Instead, they propose some kind of a procedure which will result in a "good"

L. Kalinichenko et al. (Eds.): DAMDID/RCDL 2016, CCIS 706, pp. 163–177, 2017.
DOI: 10.1007/978-3-319-57135-5_12

configuration. Usually, these studies provide some intuitive explanation why the ensuing configuration would be "good".

The abundance of studies is justified by the following considerations:

- It was proved that the problem of vertical partitioning is an NP-hard problem [3, 28, 36], just like many other physical design problems [5, 21, 36].
- Estimation errors related to both the system parameters and workload parameters. System parameters (hardware and software) in some cases cannot be measured precisely. Workload parameters can also be imprecise, e.g. not all queries are known in advance, or some of the known queries are not run. All these errors can cause the performance of the solution to deteriorate.

The procedural approach was very popular in the '80s and '90s due to the lack of computational resources. Nowadays, the interest for it has largely declined, and the majority of contemporary studies follows the cost-based one. This approach produces more accurate recommendations by incorporating additional information into the selection process. However, procedural approach has a number of promising applications:

- Dynamization of vertical partitioning [23, 27, 33, 35]. All of the previous vertical partitioning studies considered the problem in a static context, i.e. a configuration is selected once. In case of changes in the workload or the data the algorithm has to be re-run. In the new formulation the goal is to adapt the partitioning scheme to a constantly changing workload. The straightforward technique of the repeated re-run of a cost-based algorithm is not applicable due to its formidable costs of operation. Otherwise, its application will result in query processing stalls which should be avoided at all costs in this formulation. However, the procedural approach is not so computationally demanding as the cost-based one. Thus, low-quality solutions are acceptable as long as they provide improvement over the previous configuration and help us avoid query processing stalls.
- Big data applications or any other cases featuring constrained resources.
- Tuning of multistores [26] or any other case when no details or only inaccurate estimates of physical parameters are known. It was already noted in the '80s [30] that the procedural approach is well-suited for such cases. A multistore system is a database system which consists of several distinct data stores, e.g. a Hadoop HDFS and an RDBMS. This kind of a system is a modern example of the case where not every physical parameter of underlying data stores is known.

This paper is an extended version of the paper [17]. In our previous study we presented a test environment and results of preliminary experiments with "separate" strategy for vertical partitioning. This strategy assigns a separate vertical partition for every cluster found by the algorithm, including the inter-submatrix attribute group. The preliminary results showed little to no performance improvement with this strategy for all three algorithms for in-memory environment.

In this paper we continue our study of matrix clustering algorithms. This time we consider a disk-based environment. Firstly, we try to improve record reconstruction times via several approaches. Next, we introduce two strategies:

– A "replicate" strategy, which replicates inter-submatrix attributes to every cluster.
– A "retain" strategy, which assigns inter-submatrix attributes to their original clusters.

We study these strategies and compare them with the "separate" strategy and the unpartitioned case. We evaluate all strategies in terms of query performance and storage overhead. Finally, we compare the resulting partitioning schemes with the ideal partitioning scenario, where each query gets a specially-tuned fragment.

2 Related Work

2.1 Classification

The vertical partitioning problem is one of the oldest problems in the database domain. There are several dozens of studies on this topic, and most of them concern various algorithms. Several surveys can be found in [13, 14]. Vertical partitioning algorithms can be classified into two major groups: cost-based and procedural, where the latter employs three types of approaches:

– Attribute affinity and matrix clustering approaches [9, 10, 12, 18, 22]. In affinity-based approaches, closeness between every two attributes is first calculated, and then it is used to define the borders of the resulting fragments. This closeness is called attribute affinity. At the first step a workload is used to create an AUM, then an Attribute Affinity Matrix (AAM) is constructed using a paper-specific transformation procedure. Finally, a row and column permutation algorithm is applied. Matrix clustering approaches operate on the AUM and start with the permutation part.
– Graph approaches [11, 16, 28, 31, 38]. Most of the graph approaches treat the AAM as an adjacency matrix of an undirected weighted graph. In this graph nodes denote attributes and edges represent a bounds strength. Then a template is sought by various means, e.g. kruskal-like algorithms or hamiltonian way cut. The resulting templates are used to construct partitions.
– Data mining approaches [7, 19, 34]. This is a relatively new vertical partitioning technique that uses association rules to derive vertical fragments. Most of these works mine a workload (a transaction set) for rules which use sets of attributes as items. In these studies, existing algorithms for association rule search are used to uncover relations between attributes. In particular, an adapted Apriori [1] algorithm is a very popular choice.

Let us review the matrix clustering approach in detail.

2.2 Matrix Clustering Approach

The general scheme of this approach is the following:

– Construct an Attribute Usage Matrix (AUM) from the workload. The matrix is constructed as follows:

$$M_{ij} = \begin{cases} 1, & query\ i\ uses\ attribute\ j \\ 0, & otherwise \end{cases}$$

– Cluster the AUM by permuting its rows and columns to obtain a block diagonal matrix.
– Extract these blocks and use them to define the resulting partitions.

Some approaches do not operate on a 0–1 matrix. Instead they modify matrix values to account for additional information like query frequency, attribute size and so on. Let us consider an example. Suppose that we have six queries accessing six attributes:

The next step is the creation of an AUM using this workload. The resulting matrix is shown in Fig. 1a. Having applied a matrix clustering algorithm, we acquire the reordered AUM (Fig. 1b). The resulting fragments are the following: (a, b), (b, f), (d, e).

```
q1: SELECT a FROM T WHERE a > 10;
q2: SELECT b, f FROM T;
q3: SELECT a, c FROM T WHERE a = c;
q4: SELECT a FROM T WHERE a < 10;
q5: SELECT e FROM T;
q6: SELECT d, e FROM T WHERE d + e > 0;
```

	a	b	c	d	e	f
q1	1	0	0	0	0	0
q2	0	1	0	0	0	1
q3	1	0	1	0	0	0
q4	1	0	0	0	0	0
q5	0	0	0	0	1	0
q6	0	0	0	1	1	0

(a) AUM

	a	c	b	f	d	e
q1	1	0	0	0	0	0
q3	1	1	0	0	0	0
q4	1	0	0	0	0	0
q2	0	0	1	1	0	0
q6	0	0	0	0	1	1
q5	0	0	0	0	0	1

(b) Reordered AUM

Fig. 1. Matrix clustering algorithm

However, not all matrices are fully decomposable. Consider the matrix presented in Fig. 2. The first column obstructs the perfect decomposition into several clusters. In this case, the algorithm should produce a decomposition which would minimally harm query processing and would result in an overall performance improvement. Matrix clustering algorithms employ different strategies to select such a decomposition.

a	b	c	d	e	f
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	0	0	1	1	0
1	0	0	1	1	0
1	0	0	0	0	1

Fig. 2. Non-decomposable matrix

2.3 Matrix Clustering Algorithms

The first study to introduce matrix clustering to vertical partitioning was the work of Hoffer [22]. The idea is to store together (in one file) attributes possessing identical retrieval patterns. The patterns are expressed through the notion of attribute cohesion, which shows how attributes in a pair are related to each other. The author proposes a pairwise attribute similarity measure to capture this cohesion.

The proposed measure relies on three parameters: co-access frequency of a pair of attributes, attribute length and relative importance of the query. This measure was designed having the following properties in mind: it is non-decreasing by co-access frequency, non-decreasing by both attribute lengths (individually) and the function is non-increasing in the combined length of attributes.

Finally, having an attribute affinity matrix, an existing clustering algorithm (Bond Energy Algorithm, BEA) [29] is used. It permutes rows and columns to maximize nearest neighbor bond strengths. The author was motivated in his choice by the following: this algorithm is insensitive to the order in which items are presented; it has a low computation time, etc. However, this algorithm has a disadvantage: it requires human attention for cluster selection.

BEA is not the only existing matrix clustering algorithm. Another permutation algorithm was proposed in the reference [37]. Similarly to BEA, it permutes rows and columns, but tries to minimize the spanning path of the graph represented by the original matrix. The improvement of these two algorithms is presented in the reference [6]. This algorithm is called the matching algorithm and it uses Hamming distance to produce clusters. According to [9], the study [24] presents the Rank Order algorithm. Its idea is to sort rows and columns of the original matrix in descending order of their binary weight. The Cluster Identification (CI) algorithm by Kusiak and Chow [25] is an algorithm for clustering 0–1 matrices. The proposed approach is to detect clusters one by one using a special procedure. This procedure resembles the search of a transitive closure for rows and columns. It is an optimal algorithm that can solve the problem when the matrix is perfectly separable, e.g. when clusters do not intersect (there is no attribute sharing).

All of the aforementioned algorithms (except BEA) are generic matrix clustering algorithms. They do not address the vertical partitioning problem and do not even bear any database specifics. The next studies by Chun-Hung Cheng [9, 10, 12] attempt to apply matrix clustering approach to the database domain. Several new vertical partitioning algorithms were developed in his works. Let us consider them.

Chun-Hung Cheng criticizes existing matrix clustering algorithms [9, 10]:

- They do not always produce a solution matrix in a diagonal submatrix structure. Thus, these algorithms may require additional computation to extract them;
- These algorithms may require decision of database administrator to identify inter-submatrix attributes [9].

The first study [9] extends the original CI [25] algorithm to non-decomposable matrices. The proposed approach is to remove columns obstructing the decomposition (inter-submatrix attributes).

The author considered the following problem formulation $P1$ [9]: remove columns to decompose a matrix into separable submatrices with the maximum number of "1" entries retained in submatrices subject to the following constraints:

- C1: A submatrix must contain at least one row;
- C2: The number of rows in a submatrix cannot exceed upper limit, b;
- C3: A submatrix must contain at least one column.

To solve the problem, the branch and bound approach was used. This approach uses an objective function which maximizes the number of "1" entries in the resulting submatrices. During the tree traversal, upper and lower bounds are calculated and used to guide the enumeration process.

However, the basic approach required traversal of too many nodes, so the author augmented it with the following heuristic. A so-called **blocking measure** is calculated for each column. It estimates the likelihood of a column being an obstacle to the further decomposition of the matrix. Basically, it is the number of columns that would be involved in all queries which use the given attribute. Next, the columns are ordered by their respective values and the ones with the highest values are checked.

The study [10] also extends the original CI algorithm. The author adopts the same branch and bound approach as in his previous paper [9]. However, instead of the **blocking measure** a new **void measure** is developed. It has the same purpose, which is the estimation of the likelihood of a column being an inter-submatrix column. Essentially, this measure is the calculated "free space" to the left and to the right of the candidate cluster.

The next study of the author [12] addresses several shortcomings of his previous works:

- The problem of the parameter b. While this parameter helps prevent the formation of the huge clusters, it does not guarantee any quality of the resulting clusters. Also, the problem will have to be reformulated if several clusters of different sizes are needed.
- The dangling transaction problem. Applying the previous algorithm [10] a transaction not belonging to any cluster may be acquired: all of its attributes would be removed. Two examples are presented in the original paper.
- The previous work did not include such an important parameter as the access frequency of the transactions.

Thus, a new formulation $P3$ is proposed [12]: remove a minimal number of "1" entries to decompose a transaction-attribute access matrix into separable submatrices subject to the following constraints:

- C7: Transactions with all "0" entries in a submatrix are not allowed.
- C8: Attributes with all "0" entries in a submatrix are not allowed.
- C9: The **cohesion measure** of a submatrix is more than or equal to a threshold, δ.

Cohesion measure of a submatrix is the ratio of "1" elements to "0" elements. This new measure is used to ensure the quality of a cluster.

The problem is also solved with the branch and bound approach, again, the **void measure** is used to guide the order of node traversal.

Furthermore, in this work the author shows why dangling transactions should be avoided: an example is provided showing a case where it is possible to lose information regarding a cluster. Finally, the author extended his CI framework to consider query frequencies. This *P*4 formulation is the same as *P*3, but features a weighted sum of accesses [12]: minimize the loss of total accesses $\left(\sum_i \sum_j a_{ij} * freq_i\right)$ due to the removal of a_{ij} for decomposing a transaction-attribute matrix into separable submatrices subject to the same constraints C7–C9. In this paper we study the approaches described in the references [9, 10, 12].

3 System Architecture

We have developed a program for experimental evaluation of the considered algorithms. Its architecture is presented on Fig. 3. It consists of the following modules:

- The parser reads the workload from a file. It extracts the queries and passes them to the executor, so that their execution times can be measured. It also constructs the AUM, which serves as input for the selected algorithm.
- The algorithm identifies clusters and passes that information to the partitioner to create corresponding temporary tables.
- The query rewriter also receives this information. It replaces the name of the original table with the ones that were generated by the partitioner. It can handle subqueries; view support is not implemented yet.
- The partitioner generates new names and sends partitioning commands to the database. The exact commands are SELECT INTO and ALTER TABLE. The latter lets it transfer primary keys.
- The executor accepts queries and sends them to PostgreSQL to measure the time of execution.

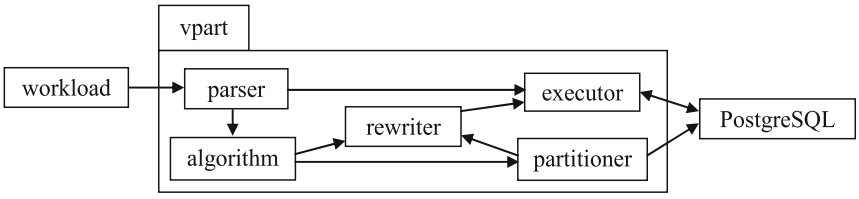

Fig. 3. The architecture of our approach

4 Experiments

4.1 A Brief Summary of Previous Findings

In our previous paper [17] we have implemented three recent matrix clustering algorithms [9, 10, 12] (A94, A95, A09) and used PostgreSQL DBMS to evaluate them. Our experiments were conducted using the standard benchmark—TPC-H with scale factor 1. The database was placed in the main memory of the machine. To accomplish this, the PostgreSQL data directory was put on a tmpfs partition, created with standard GNU/Linux utilities. We have employed a "separate" strategy for vertical partitioning. This strategy assigns a separate vertical partition for every cluster found by the algorithm, including inter-submatrix attribute group.

Experiments showed that all of the considered algorithms perform poorly in this environment, often yielding partitioning schemes worse than the original one.

4.2 Experimental Plan

In this paper we try to analyze the reasons for this outcome. The plan of this study is the following:

1. First of all, we changed the environment from the main memory to the disk-based one. It is used for all of the experiments described in this paper.
2. Then, we performed the evaluation of record reconstruction methods of PostgreSQL. We try different methods to speed-up or to completely avoid record reconstruction expenses. Eventually, we select the best record reconstruction method and use it throughout the rest of the paper.
3. Next, we introduce two new partition generation strategies called "replicate" and "retain". We compare them with each other, with the original non-partitioned configuration and the "separate" strategy employed in our previous work.
4. Finally, we compare our strategies with the ideal partitioning scenario, where each query gets a specially-tuned fragment.

4.3 Data and Query Setup

We decided to stick to the original data setup to support the reproducibility of results. However, we were unable to keep the original data, because we have changed the scale factor from 1 to 10.

Like in our previous work we have chosen the LINEITEM and PART tables for the evaluation. Based on these tables we have formulated the following query setups:

– Query Setup 1 (QS1): Q1, Q6, Q14, Q19;
– Query Setup 2 (QS2): Q6, Q14, Q19;
– Query Setup 3 (QS3): Q6, Q14.

It is important to note that only queries Q14 and Q19 involve join of the LINEITEM and PART tables. Queries Q1 and Q6 contain only filtering predicates and aggregate functions on the LINEITEM table.

4.4 Hardware and Software Setups, Measurements

In our experiments we have used the following setup (almost the same as in our previous study):

- PostgreSQL 9.6.1,
- Funtoo Linux (kernel 4.8.4),
- Intel Core i7-3630QM (4 physical cores, hyper-threading enabled),
- 8 GB (DDR3) RAM,
- GCC 4.9.3.

All numbers presented in this paper are averages over five runs. During evaluation we noticed hot cache effects: the difference between the first and the fifth run may be about 3–5 times. Thus, to negate these effects we restarted PostgreSQL and dropped OS caches for each query. This way we obtained stable, almost equal to each other results for each value in a series of measurements.

4.5 Disk-Based Setup Re-run

All six scenarios from our previous study were re-run in the disk-based environment. It is important to note that we benchmarked the same partitioning schemes, because the algorithm does not depend on the environment-related (physical) parameters.

The benchmarking results were essentially the same, the partitioned scheme was worse than the non-partitioned one in terms of query performance. The overall performance of a query batch degraded 2–3 times, depending on the scenario. Due to the space constraints we will not present all of the scenarios. However, we will closely study scenario 1 in the "New Fragmentation Strategies" section.

4.6 Record Reconstruction Study

During the re-run we found out that the major performance hit was the join cost. Thus, we tried to improve the efficiency of record reconstruction. We have considered the following techniques:

1. Default, where we just rewrite queries with joins.
2. Sequential scan, where we forced sequential scan for reading table data.
3. Index scan, where we made PostgreSQL use the index on join attributes for query processing.

For our experiments we used only query Q1, because it does not have any joins with the PART table in its execution plan. This way we will study the effects of just the reconstruction joins. The results of our experiments are presented in Fig. 10. Here you can see that default and sequential scan techniques are indistinguishable from each other. On the other

hand, index scan performance is significantly worse than both of them. We examined the query plan and noticed that the performance deterioration presents in both scan operators and in the join operator. We also tried to use a clustered index, but performance was worse than with an index scan. Thus, we decided to stick to sequential scan reconstruction technique throughout the paper. Also, note the consistent behavior of these techniques for the "replicate" strategy (we will describe it in the next subsection).

4.7 New Fragmentation Strategies

In our previous paper we studied the "separate" strategy, which assigns a separate vertical partition for every cluster found by the algorithm, including inter-submatrix attribute group. The author of these matrix clustering algorithms indicated several other possible strategies to form vertical partitions. In this paper we are interested in the following:

- A "replicate" strategy, which replicates inter-submatrix attributes to every cluster. This strategy should eliminate all joins by introducing storage overhead. It should also degrade the performance of insert and update queries, because of the need to keep data synchronized.
- A "retain" strategy, which assigns inter-submatrix attributes to their original clusters. This strategy is inherent only to algorithm A09, which removes accesses in the matrix. The columns, access to which has been removed, are considered intersubmatrix and must be handled appropriately. However, the clustering algorithm itself already returns them as parts of some clusters. The point of this strategy is to keep them in place.

In Fig. 4 you can see the performance of the A09 algorithm in the disk-based environment using strategies "separate", "replicate" and "retain" for scenario 1. These graphs are given along with the "original" (unpartitioned) variant. The corresponding partitioning schemes can be found in our previous study.

The general conclusions for this experiment are the following: the configuration generated by the "retain" strategy shows the worst performance, the configuration generated by the "separate" strategy is still worse than the unpartitioned data. On the other hand, the "replicate" strategy's configuration performs significantly better than the unpartitioned one. This behavior is explained by the record reconstruction expenses which have to be performed using a relational join operation. These expenses significantly outweigh the benefits of the vertical partitioning.

However, the "replicate" strategy does not come without a price. Replication of attributes increases the amount of data that needs to be stored. Thus, we decided to assess storage overhead. Figure 5 presents the respective table sizes for all strategies, alongside with the non-partitioned configuration for comparison. Each section of a bar column marks the size of the LINEITEM table partition. Thus, "separate" strategy partitions data into three pieces, while "retain" partitions into two. From this experiment we can see that the "retain" and "replicate" strategies produce very little overhead, while "separate" strategy requires almost 1.5 times more space than the original scheme. Such difference can be justified by the need to copy primary key to every partition (2 vs 3)

and the fact that in LINEORDER the table primary key is a multi-attribute entity, which is rather big. Probably, we could reduce the amount of required space by introducing our own (smaller) primary key, solely for reconstruction.

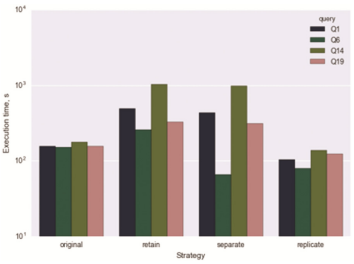

Fig. 4. Strategies comparison, scenario 1

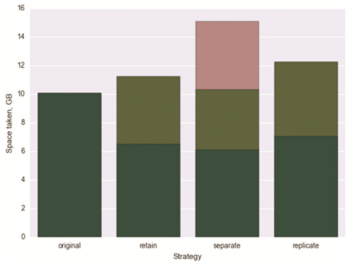

Fig. 5. Disk requirements, scenario 1

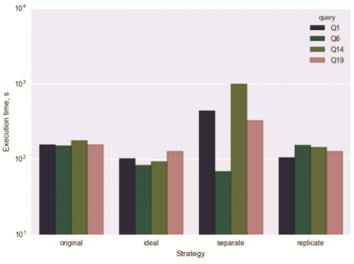

Fig. 6. Strategies comparison, scenario 6

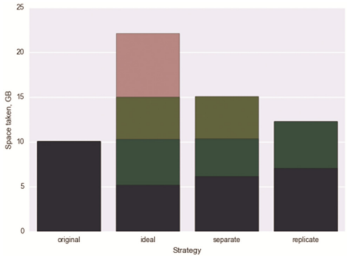

Fig. 7. Disk requirements, scenario 6

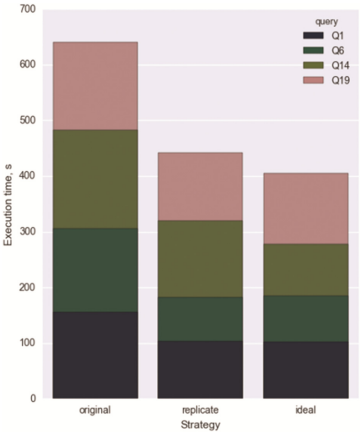

Fig. 8. Ideal Performance

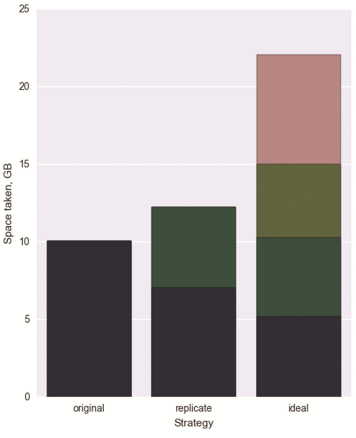

Fig. 9. Ideal disk usage

Figure 6 contains the performance of the A95 algorithm in a disk-based environment using strategies "separate" and "replicate" for scenario 6 (the strategy "ideal" is discussed in the next section). These graphs are given alongside with the original variant,

for the ease of comparison. Here we can also see that the "separate" strategy performs worse than the unpartitioned variant, like in the in-memory case. The "replicate" strategy is the winner in this scenario as well. The "retain" strategy is inapplicable to the A95 algorithm, so it is absent in this graph. Disk space requirements for this scenario are presented in Fig. 7. Similarly to scenario 1, the "replicate" strategy requires less space than the "separate" strategy, but still more than the unpartitioned setup.

We do not describe other scenarios because they do not illustrate anything new compared to the ones considered in this subsection.

4.8 Comparison with Ideal Partitioning Scheme

The next questions of our study were "How good is the 'replicate' scheme? Can we do better?". To answer them we have introduced an ideal partitioning scenario, where each query gets a fragment containing the minimum number of attributes.

We had re-run scenario 1 (algorithm A09) and compared the "replicate" strategy with the ideal partitioning scenario. The results are presented in Figs. 8 and 9. As you can see, the ideal scheme is about 10% better than the "replicate" strategy while consuming almost twice as much space. This improvement comes from the costs of Q14. Thus, "replicate" strategy is quite efficient.

There is a similar picture with scenario 6 (algorithm A95) presented in Figs. 6 and 7. This time the ideal scheme is 24% better than the "replicate" strategy with the same storage overhead. The major sources of improvement are queries Q6 (almost two times) and Q14 (about 1.5 times).

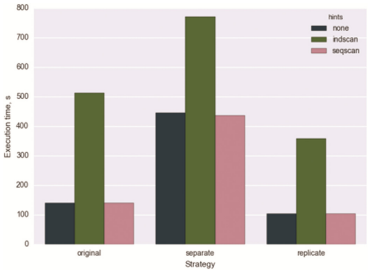

Fig. 10. Record reconstruction

4.9 Conclusions

This is the continuation of our previous study [17] of the matrix clustering algorithms. In this paper we analyze the reasons for the negative outcome of our previous work and explore alternative vertical partition generation strategies. To achieve better understanding of the subject we switched to disk-based configuration, as opposed to in-memory one used in our previous work.

Our experiments showed that PostgreSQL encounters performance problems while reconstructing vertically partitioned records. We had to use relational join operation and we were unable to speed-up it by using regular and clustered indexes.

Next, we found out that the "separate" strategy, which was the only one strategy used in our previous study and which produced partitions that were consistently worse than the unpartitioned schema, behaved the same way in the disk-based environment. Moreover, we demonstrated that in terms of disk space required for the partitioning, this strategy is also very expensive.

On the other hand, the performance of the schemes generated by the new strategy "replicate" were significantly (20%–50% depending on a case) better than the unpartitioned schema. Storage overhead was about 20% that of the unpartitioned case. The other strategy – "retain" generated schemes which performed worse than generated by the "separate" strategy (thus being the worst strategy), while requiring only 10% more storage than in unpartitioned case.

We also compared the "replicate" strategy with the ideal partitioning scheme, where each query receives a specially-tailored minimal partition. To our surprise, the quality of "replicate" strategy schemes was quite close to the ideal one. Meanwhile, the storage overhead of the ideal scheme was almost 2.5 times that of the unpartitioned one and the one generated by "replicate" was only 1.2.

We can conclude with the following:

- The "replicate" strategy generates near ideal schemes, but with the reasonable storage overhead. These schemes provide significant improvement over unpartitioned case for read-only workloads.
- Implementing vertical partitioning one should avoid record reconstruction or should try to implement it without using costly relational join operation.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB 1994 Conference Proceedings, pp. 487–499 (1994)
2. Agrawal, S., Narasayya, V., Yang, B.: Integrating vertical and horizontal partitioning into automated physical database design. In: SIGMOD 2004 Conference Proceedings, pp. 359–370 (2004). doi:10.1145/1007568.1007609
3. Apers, P.M.G.: Data allocation in distributed database systems. ACM Trans. Database Syst. **13**, 263–304 (1988). doi:10.1145/44498.45063
4. Bellatreche, L.: Optimization and tuning in data warehouses. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems, pp. 1995–2003. Springer, New York (2009). doi:10.1007/978-0-387-39940-9_259
5. Bellatreche, L., Boukhalfa, K., Richard, P.: Data partitioning in data warehouses: hardness study, heuristics and Oracle validation. In: DAWAK 2008 Conference Proceedings, pp. 87–96 (2008). doi:10.1007/978-3-540-85836-2_9
6. Bhat, M.V., Haupt, A.: An efficient clustering algorithm. IEEE Trans. Syst. Man Cybern. **6**(1), 61–64 (1976). doi:10.1109/TSMC.1976.5408399
7. Bouakkaz, M., Ouinten, Y., Ziani, B.: Vertical fragmentation of data warehouses using the FP-Max algorithm. In: IIT 2012 Conference Proceedings, pp. 273–276 (2012)

8. Chaudhuri, S., Weikum, G.: Self-management technology in databases. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems, pp. 2550–2555. Springer, New York (2009). doi:10.1007/978-0-387-39940-9_334
9. Cheng, C.: Algorithms for vertical partitioning in database physical design. Omega **22**(3), 291–303 (1994). doi:10.1016/0305-0483(94)90042-6
10. Cheng, C.-H.: A branch and bound clustering algorithm. IEEE Trans. Syst. Man Cybern. **25**(5), 895–898 (1995). doi:10.1109/21.376504
11. Cheng, C.-H., Lee, W.-K., Wong, K.-F.: A genetic algorithm-based clustering approach for database partitioning. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **32**(3), 215–230 (2002). doi:10.1109/TSMCC.2002.804444
12. Cheng, C.-H., Motwani, J.: An examination of cluster identification-based algorithms for vertical partitions. Int. J. Bus. Inf. Syst. **4**(6), 622–638 (2009). doi:10.1504/IJBIS.2009.026695
13. Chernishev, G.: Towards self-management in a distributed column-store system. In: Morzy, T., Valduriez, P., Bellatreche, L. (eds.) ADBIS 2015. CCIS, vol. 539, pp. 97–107. Springer, Cham (2015). doi:10.1007/978-3-319-23201-0_12
14. Chernishev, G.: Vertical partitioning in relational DBMS. In: Talk at the Moscow ACM SIGMOD chapter meeting; slides and video. http://synthesis.ipi.ac.ru/sigmod/seminar/s20150430
15. Chu, W., Ieong, I.: A transaction-based approach to vertical partitioning for relational database systems. IEEE Trans. Softw. Eng. **19**(8), 804–812 (1993)
16. Du, J., Barker, K., Alhajj, R.: Attraction - a global affinity measure for database vertical partitioning. In: IADIS ICWI 2003 Conference Proceedings, pp. 538–548 (2003)
17. Galaktionov, V., Chernishev, G., Novikov, B., Grigoriev, D.: Matrix clustering algorithms for vertical partitioning problem: an initial performance study. In: Selected Papers of the XVIII International Conference on Data Analytics & Management in Data Intensive Domains (DAMDID/RCDL 2016), Ershovo, Moscow Region, Russia, CEUR Workshop Proceedings, vol. 1752, pp. 24–31 (2016)
18. Gorla, N., Boe, W.J.: Database operating efficiency in fragmented databases in mainframe, mini, and micro system environments. Data Knowl. Eng. **5**(1), 1–19 (1990). doi:10.1016/0169-023X(90)90030-H
19. Gorla, N., Yan, B.P.W.: Vertical fragmentation in databases using data-mining technique. In: Erickson, J. (ed.) Database Technologies: Concepts, Methodologies, Tools, & Applications, pp. 2543–2563. IGI Global (2009)
20. Hammer, M., Niamir, B.: A heuristic approach to attribute partitioning. In: SIGMOD 1979 Conference Proceedings, pp. 93–101 (1979). doi:10.1145/582095.582110
21. Harinarayan, V., Rajaraman, A., Ullman, J.D.: Implementing data cubes efficiently. ACM SIGMOD Record **25**(2), 205–216 (1996). doi:10.1145/235968.233333
22. Hoffer, J.A., Severance, D.G.: The use of cluster analysis in physical data base design. In: VLDB 1975 Conference Proceedings, pp. 69–86 (1975). doi:10.1145/1282480.1282486
23. Jindal, A., Dittrich, J.: Relax and let the database do the partitioning online. In: BIRTE 2011 Workshop Proceedings, pp. 65–80 (2012). doi:10.1007/978-3-642-33500-6_5
24. King, J.R.: Machine-component grouping in production flow analysis: an approach using a rank order clustering algorithm. Int. J. Prod. Res. **18**(2), 213–232 (1980)
25. Kusiak, A., Chow, W.: An efficient cluster identification algorithm. IEEE Trans. Syst. Man Cybern. **17**(4), 696–699 (1987)
26. LeFevre, J., Sankaranarayanan, J., Hacigumus, H., Tatemura, J., Polyzotis, N., Carey, M.J.: MISO: souping up big data query processing with a multistore system. In: SIGMOD 2014 Conference Proceedings, pp. 1591–1602 (2014). doi:10.1145/2588555.2588568

27. Li, L., Gruenwald, L.: Self-managing online partitioner for databases (SMOPD): a vertical database partitioning system with a fully automatic online approach. In: IDEAS 2013 Conference Proceedings, pp. 168–173 (2013). doi:10.1145/2513591.2513649
28. Lin, X., Orlowska, M., Zhang, Y.: A graph based cluster approach for vertical partitioning in database design. Data Knowl. Eng. **11**(2), 151–169 (1993)
29. McCormick, W., Schweitzer, P., White, W.: Problem decomposition and data reorganization by a clustering technique. Oper. Res. **20**(5), 993–1009 (1972)
30. Navathe, S., Ceri, S., Wiederhold, G., Dou, J.: Vertical partitioning algorithms for database design. ACM Trans. Database Syst. **9**, 680–710 (1984)
31. Navathe, S.B., Ra, M.: Vertical partitioning for database design: a graphical algorithm. In: SIGMOD 1989 Conference Proceedings, pp. 440–450 (1989). doi:10.1145/67544.66966
32. Papadomanolakis, S., Ailamaki, A.: An integer linear programming approach to database design. In: ICDE 2007 Workshop Proceedings, pp. 442–449 (2007)
33. Rodríguez, L., Li, X.: A dynamic vertical partitioning approach for distributed database system. In: IEEE SMC Conference Proceedings, pp. 1853–1858 (2011). doi:10.1109/ICSMC.2011.6083941
34. Rodríguez, L., Li, X.: A support-based vertical partitioning method for database design. In: CCE 2011 Conference Proceedings, pp. 1–6 (2011). doi:10.1109/ICEEE.2011.6106682
35. Rodríguez, L., Li, X., Mejía-Alvarez, P.: An active system for dynamic vertical partitioning of relational databases. In: MICAI 2011 Conference Proceedings, pp. 273–284 (2011). doi:10.1007/978-3-642-25330-0_24
36. Sacca, D., Wiederhold, G.: Database partitioning in a cluster of processors. ACM Trans. Database Syst. **10**, 29–56 (1985). doi:10.1145/3148.3161
37. Slagle, J.R., Chang, C.L., Heller, S.R.: A clustering and data-reorganizing algorithm. IEEE Trans. Syst. Man Cybern. **5**(1), 125–128 (1975)
38. HyunSon, J., HoKim, M.: α-Partitioning algorithm: vertical partitioning based on the fuzzy graph. In: Mayr, H.C., Lazansky, J., Quirchmayr, G., Vogel, P. (eds.) DEXA 2001. LNCS, vol. 2113, pp. 537–546. Springer, Heidelberg (2001). doi:10.1007/3-540-44759-8_53