Matrix Clustering Algorithms for Vertical Partitioning Problem: an Initial Performance Study

© Viacheslav Galaktionov[1]
viacheslav.galaktionov@gmail.com ,
© Boris Novikov[1]
b.novikov@spbu.ru ,
© George Chernishev[1,2]
g.chernyshev@spbu.ru,
© Dmitry A. Grigoriev[1]
d.a.grigoriev@spbu.ru

[1]Saint-Petersburg State University, Saint-Petersburg, Russia
[2]JetBrains Research, Saint-Petersburg, Russia

Abstract

Matrix clustering algorithms are among the oldest approaches to the vertical partitioning problem. They can be summarized as follows: (1) given a workload, construct an Attribute Usage Matrix (AUM), (2) apply some kind of a row and column permutation algorithm and (3) extract the resulting clusters which define the required fragments.
This naive approach holds some promise for a number of contemporary applications: (1) dynamization of vertical partitioning (2) big data applications and other cases of resource constraints (3) tuning of multistores.
In this paper we examine a number of existing matrix clustering algorithms used for vertical partitioning. We study these algorithms and assess the quality of the solutions. The experiments are run on the TPC-H workload using the PostgreSQL DBMS.

1 Introduction

The vertical partitioning problem [5] 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 [9]: find a configuration (a set of vertical fragments) which would satisfy the given constraints and which will provide the best performance. There are two major classes of approaches to this problem:

- Cost-based approach [3, 16, 21, 33]. 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 [29, 32, 35]. These studies do not use the notion of configuration cost. Instead, they propose some kind of a procedure which will result in a "good" 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 [4, 29, 37], just like many other physical design problems [6, 22, 37].
- 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 because of 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 [24, 28, 34, 36]. 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

Proceedings of the XVIII International Conference «Data Analytics and Management in Data Intensive Domains» (DAMDID/RCDL'2016), Ershovo, Russia, October 11 - 14, 2016

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 queryprocessing stalls.

- Big data applications or any other cases featuring constrained resources.
- Tuning of multistores [27] or any other case when no details or only inaccurate estimates of physical parameters are known. It was already noted in the '80s [31] 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's 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.

In this paper we evaluate a particular subclass of procedural vertical partitioning algorithms – the matrix clustering algorithms.

To the best of our knowledge, this study is the first one to evaluate this class of algorithms using a real DBMS and a real workload [1].

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 the references [14, 15]. 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 [10, 11, 13, 19, 23]. 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 [12, 17. 29, 32, 39]. 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 bound's 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 [8, 20, 35]. 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 [2] 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:

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

- 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:

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;

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

However, not all matrices are fully decomposable. Consider the matrix presented on Figure 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
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	b	c	d	e	f
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

Figure 1 Matrix clustering algorithm

2.3 Matrix Clustering Approach

The first study to introduce matrix clustering to vertical partitioning was the work of Hoffer [23]. 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) [30] is used. It permutes rows and columns to maximize nearest neighbour 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.

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

Figure 2 Non-decomposable matrix

BEA is not the only existing matrix clustering algorithm. Another permutation algorithm was proposed in the reference [38]. 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 [7]. This algorithm is called the matching algorithm and it uses Hamming distance to produce clusters. According to the reference [10], the study [25] 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 [26] 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 [10, 11, 13] 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 [10, 11]:

- 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 [10].

The first study [10] extends the original CI [26] 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 *P*1 [10]: 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.

In order 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

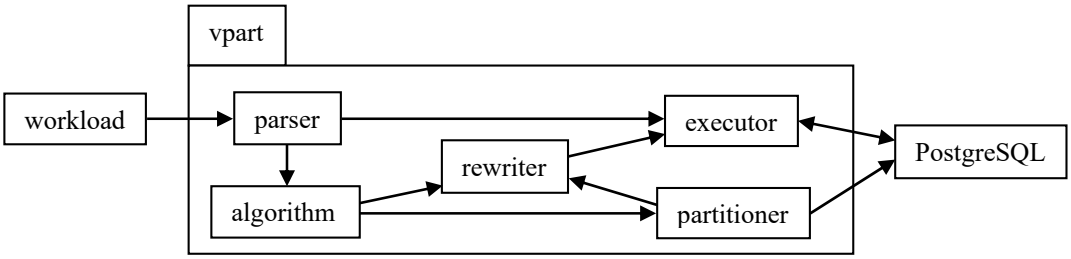

Figure 3 The architecture of our approach

attribute. Next, the columns are ordered by their respective values and the ones with the highest values are checked.

The study [11] also extends the original CI algorithm. The author adopts the same branch and bound approach as in his previous paper [10]. 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 [13] 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 [11] 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 *P*3 is proposed [13]: 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 [13]: minimize the loss of total accesses ($\sum_i \sum_j a_{ij} \times freq_i$) 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 [10, 11, 13].

3 System Architecture

We have developed a program for experimental evaluation of the considered algorithms. Its architecture is presented on Figure 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.

4 Parallelization

Having implemented this system, we noticed unacceptable run times even for relatively small matrices. The author of these algorithms states that this is not a problem because the algorithm finds a good solution quickly and spends the major portion of its time just by checking the rest of the tree.

However, we decided to parallelize all of the algorithms. We managed to achieve this in a generic fashion, i.e. we applied a generic parallelization scheme for all of the branch and bound algorithms. In order to

Type	Q1	Q6	Q14	Q19
Original	11694	1365	1412	1663
partitioned	31558	1602	1379	1797

Figure 4 Scenario 1 – A09, QS1, 0.7

Type	Q6	Q14	Q19
Original	1439	1405	1673
Partitioned	1343	1093	2731

Figure 6 Scenario 2 – A09, QS2, 0.7

	1	2	3	4	5	6	7	8	9	10
Q1	0	1	1	1	1	1	1	1	0	0
Q6	0	1	1	1	0	0	0	1	0	0
Q14	1	0	1	1	0	0	0	1	0	0
Q19	1	1	1	1	0	0	0	0	1	1

(a) Original

	1	3	9	10	2	4	5	6	7	8
Q1	0	*	0	0	1	1	1	1	1	1
Q6	0	*	0	0	1	1	0	0	0	1
Q14	1	1	0	0	0	*	0	0	0	*
Q19	1	1	1	1	*	*	0	0	0	0

(b) Result

Figure 5 Scenario 1 – A09, QS1, 0.7

	1	2	3	4	5	6	7
Q6	0	1	1	1	1	0	0
Q14	1	0	1	1	1	0	0
Q19	1	1	1	1	0	1	1

(a) Original

	1	2	3	4	5	6	7
Q6	0	1	1	1	1	0	0
Q14	1	0	1	1	1	0	0
Q19	*	*	*	*	0	1	1

(b) Result

Figure 7 Scenario 2 – A09, QS2, 0.7

type	Q6	Q14
original	1555	1395
partitioned	1421	1062

Figure 8 Scenario 3 – A09, QS3, 0.7

type	Q6	Q14	Q19
original	1438	1360	1685
partitioned	1405	1148	1704

Figure 9 Scenario 4 – A09, QS2, 0.5

type	Q6	Q14	Q19
original	1385	1409	1648
partitioned	2201	1377	1855

Figure 10 Scenario 5 – A09, QS2, 0.9

Type	Q1	Q6	Q14	Q19
original	11515		1367	1608
partitioned	31173	934	1479	1989

Figure 12 Scenario 6 – A95, QS1, 2

implement it we employed the Threading Building Blocks (TBB)[1].

Using these primitives, our already existing sequential implementation was parallelized with minimal effort. We replaced the explicit stack used in sequential depth-first traversal with TBB constructs[2]. Thus, we kept the node inspection code unchanged.

For the detailed information regarding the parallelization method and the results see the original paper [18].

5 Experiments

We have implemented three recent matrix clustering algorithms [10, 11, 13] (A94, A95, A09) and used PostgreSQL DBMS to evaluate them. Our experiments were conducted using the standard benchmark – TPC-H with scale factor 1. We measured the run times for original and partitioned configurations.

5.1 Hardware and Software Setups

In our experiments the following setup was used:

- PostgreSQL 9.5.2,
- Gentoo Linux (kernel 4.1.12),
- Intel® Core™ i7-3630QM (4 physical cores, hyper-threading enabled)
- 8GB (DDR3) RAM,
- GCC 4.9.3.

The database was placed in the main memory of the machine. In order to accomplish this, the PostgreSQL data directory was put on tmpfs, created with standard GNU/Linux utilities.

To ensure the reproducibility of our results we used sequential versions of algorithms for all comparisons. All of the workloads were executed sequentially.

5.2 Data Setup

For our evaluation we have chosen the LINEITEM and PART tables. 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.

This is the initial series of experiments, so we tried to use simple scenarios. In these experiments we assume uniform distribution of query frequencies.

The author of the studied algorithms indicated that there are three possible strategies for dealing with inter-submatrix attributes: forming a separate cluster for all inter-submatrix attributes, duplicating them to every subrelation and keeping them in the relation which uses them more often. He argues that the decision which

[1] https://www.threadingbuildingblocks.org/
[2] https://www.threadingbuildingblocks.org/docs/help/reference/task_scheduler.htm

strategy to apply is usually left to database administrator. In this study we employ the first strategy.

5.3 Scenario 1

In this experiment we used the most recent algorithm from the reference [13] (A09). The cohesion parameter

	1	2	3	4	5	6	7
Q6	0	1	1	1	1	0	0
Q14	1	0	1	1	1	0	0
Q19	1	1	1	1	0	1	1

(a) Original

Figure 11 Scenario 5 – A09, QS2, 0.9

	1	2	3	4	5	6	7	8	9	10
Q1	0	1	1	1	1	1	1	1	0	0
Q6	0	1	1	1	0	0	0	1	0	0
Q14	1	0	1	1	0	0	0	1	0	0
Q19	1	1	1	1	0	0	0	0	1	1

(a) Original

Figure 13 Scenario 6 – A95, QS1, 2

was set to 0.7 and QS1 was used. Table 4 contains the performance for this scenario.

As we can see, only run time for Q14 improved and the overall time significantly increased. The query Q1 can be characterized by a large number of aggregates and read attributes. This is the possible reason for such performance deterioration. The partitioning scheme is presented in Table 5.

5.4 Scenario 2

This experiment also addresses the A09 algorithm with the same cohesion parameter. However, we decided to discard Q1 from the workload to check whether that would improve the overall performance. The results are presented in Table 6. While Q6 and Q14 performance improved, the Q19 performance has greatly deteriorated. The net gain is also negative in this case. The corresponding partitioning scheme is presented in Table 7.

5.5 Scenario 3

In this scenario we examined A09 on the QS3. The results are shown in Table 8. We do not demonstrate the original and partitioned matrices due to the space constraints and due to the fact that the algorithm returned only one cluster, which was identical to the input one. Thus, overall improvement was achieved via transfer of all of untouched attributes to a separate cluster.

5.6 Scenario 4

In this experiment we again considered A09 on QS2, but lowered the cohesion value to 0.5. Similarly, we obtained a positive net gain (see Table 9). Unfortunately, input and output matrices indicate that the reason for this improvement is the same as in Scenario 3.

5.7 Scenario 5

Here we evaluate the behavior of A09 with QS2 and cohesion value of 0.9. Results are presented in Tables 10 and 11. There is also negative overall gain.

5.8 Scenario 6

	1	2	6	7	3	4	5
Q6	0	*	0	0	1	1	1
Q14	*	0	0	0	1	1	1
Q19	1	1	1	1	*	*	0

(b) Result

	1	9	10	5	6	7	2	3	4	8
Q1	0	0	0	1	1	1	1	1	1	1
Q6	0	0	0	0	0	0	1	1	1	1
Q14	1	0	0	0	0	0	0	1	1	1
Q19	1	1	1	0	0	0	1	1	1	0

(b) Result

In this experiment we tried a different algorithm – A95 on QS1 with the maximum number of rows being 2. The outcome is presented in Tables 12, 13.

5.9 Other Scenarios and Results

We have also conducted a number of other experiments, but unfortunately, we are limited by the space available. Here is a brief summary of our findings:

- If we select a lot of attributes in one query of the workload, these algorithms will perform poorly;
- These algorithms perform well on workloads which have several columns consisting of "0" entirely (containing no accesses in the workload);
- It may be beneficial to set a low cohesion value in order to achieve better performance. This is accomplished by eliminating additional joins;
- Algorithms A95, A94 and Optimal exhibit the similar behavior during our tests;
- There are cases when any of the algorithms (Optimal, A94, A95) can return no solution;
- In order to obtain a non-trivial solution cohesion parameter should be higher than one of the original matrix.

6 Conclusions

In this paper we have studied three newer matrix clustering algorithms [10, 11, 13]. We have implemented these algorithms and used PostgreSQL with TPC-H workload to evaluate them. In our experiments we employed one of several inter-cluster attribute handling strategies. Preliminary results suggest that all of these algorithms perform poorly in this environment, often yielding partitioning schemes worse than the original one.

References

[1] TPC Benchmark H. Decision Support. Version 2.17.1. http://www.tpc.org/tpch [Accessed: 2016 01 08]

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] S. Agrawal, V. Narasayya, and B. Yang. Integrating vertical and horizontal partitioning into automated physical database design. In Proceedings of the 2004 ACM SIGMOD international conference on Management of data, SIGMOD '04, pages 359–370, New York, NY, USA, 2004. ACM.

[4] P. M. G. Apers. Data allocation in distributed database systems. ACM Trans. Database Syst., 13:263–304, 1988.

[5] L. Bellatreche. Optimization and tuning in data warehouses. In L. LIU and M. ÖZSU, editors, Encyclopedia of Database Systems, pages 1995–2003. Springer US, 2009.

[6] L. Bellatreche, K. Boukhalfa, and P. Richard. Data partitioning in data warehouses: Hardness study, heuristics and oracle validation. In I.-Y. Song, J. Eder, and T. Nguyen, editors, Data Warehousing and Knowledge Discovery, volume 5182 of Lecture Notes in Computer Science, pages 87–96. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-85836-2_9.

[7] M. V. Bhat and A. Haupt. An efficient clustering algorithm. Systems, Man and Cybernetics, IEEE Transactions on, SMC-6(1):61–64, 1976.

[8] M. Bouakkaz, Y. Ouinten, and B. Ziani. Vertical fragmentation of data warehouses using the FP-Max algorithm. In Innovations in Information Technology (IIT), 2012 International Conference on, pages 273–276, march 2012.

[9] S. Chaudhuri and G. Weikum. Self-management technology in databases. In L. Liu and M. Özsu, editors, Encyclopedia of Database Systems, pages 2550–2555. Springer US, 2009.

[10] C. Cheng. Algorithms for vertical partitioning in database physical design. Omega, 22(3):291–303, 1994.

[11] C.-H. Cheng. A branch and bound clustering algorithm. Systems, Man and Cybernetics, IEEE Transactions on, 25(5):895–898, 1995.

[12] C.-H. Cheng, W.-K. Lee, and K.-F. Wong. A genetic algorithm-based clustering approach for database partitioning. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 32(3):215–230, 2002.

[13] C.-H. Cheng and J. Motwani. An examination of cluster identification-based algorithms for vertical partitions. Int. J. Bus. Inf. Syst., 4(6):622–638, 2009.

[14] G. Chernishev. Towards self-management in a distributed column-store system. In T. Morzy, P. Valduriez, and L. Bellatreche, editors, New Trends in Databases and Information Systems, volume 539 of Communications in Computer and Information Science, pages 97–107. Springer International Publishing, 2015.

[15] G. Chernishev. Vertical Partitioning in Relational DBMS. Talk at the Moscow ACM SIGMOD chapter meeting; slides and video: http://synthesis.ipi.ac.ru/sigmod/seminar/s20150430 30 4 2015 [Accessed: 2016 01 08].

[16] W. Chu and I. Ieong. A transaction-based approach to vertical partitioning for relational database systems. Software Engineering, IEEE Transactions on, 19(8):804–812, 1993.

[17] J. Du, K. Barker, and R. Alhajj. Attraction — a global affinity measure for database vertical partitioning. In ICWI, pages 538–548. IADIS, 2003.

[18] V. Galaktionov. Parallelization of matrix clustering algorithms (accepted). In Proceedings of the Sixth International Conference on Informatics Problems (SPISOK 2016), 2016.

[19] N. Gorla and W. J. Boe. Database operating efficiency in fragmented databases in mainframe, mini, and micro system environments. Data & Knowledge Engineering, 5(1):1–19, 1990.

[20] N. Gorla and B. P. W. Yan. Vertical fragmentation in databases using data-mining technique. In J. Erickson, editor, Database Technologies: Concepts, Methodologies, Tools, and Applications, pages 2543–2563. IGI Global, 2009.

[21] M. Hammer and B. Niamir. A heuristic approach to attribute partitioning. In Proceedings of the 1979 ACM SIGMOD international conference on Management of data, SIGMOD '79, pages 93–101, New York, NY, USA, 1979. ACM.

[22] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, SIGMOD'96, pages 205–216, New York, NY, USA, 1996. ACM.

[23] J. A. Hoffer and D. G. Severance. The use of cluster analysis in physical data base design. In Proceedings of the 1st International Conference on Very Large Data Bases, VLDB '75, pages 69–86, New York, NY, USA, 1975. ACM.

[24] Jindal and J. Dittrich. Relax and let the database do the partitioning online. In M. Castellanos, U. Dayal, and W. Lehner, editors, Enabling Real-Time Business Intelligence, volume 126 of Lecture Notes in Business Information Processing, pages 65–80. Springer Berlin Heidelberg, 2012.

[25] J.R. King Machine-component grouping in production flow analysis: an approach using a rank order clustering algorithm. Int. J. Prod. Res., 18(2):213–232, 1980.

[26] Kusiak and W. Chow. An efficient cluster identification algorithm. Systems, Man and

Cybernetics, IEEE Transactions on, SMC-17(4):696–699, 1987.

[27] J. LeFevre, J. Sankaranarayanan, H. Hacigumus, J. Tatemura, N. Polyzotis, and M. J. Carey. MISO: Souping up big data query processing with a multistore system. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD '14, pages 1591–1602, New York, NY, USA, 2014. ACM.

[28] L. Li and L. Gruenwald. Self-managing online partitioner for databases (SMOPD): A vertical database partitioning system with a fully automatic online approach. In Proceedings of the 17th International Database Engineering & Applications Symposium, IDEAS '13, pages 168–173, New York, NY, USA, 2013. ACM.

[29] X. Lin, M. Orlowska, and Y. Zhang. A graph based cluster approach for vertical partitioning in database design. Data & Knowledge Engineering, 11(2):151–169, 1993.

[30] W. McCormick, P. Schweitzer, and W. White. Problem decomposition and data reorganization by a clustering technique. Oper. Res., 20(5):993–1009, 1972.

[31] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical partitioning algorithms for database design. ACM Trans. Database Syst., 9:680–710, 1984.

[32] S. B. Navathe and M. Ra. Vertical partitioning for database design: a graphical algorithm. In Proceedings of the 1989 ACM SIGMOD international conference on Management of data, SIGMOD '89, pages 440–450, New York, NY, USA, 1989. ACM.

[33] S. Papadomanolakis and A. Ailamaki. An integer linear programming approach to database design. In Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop, ICDEW '07, pages 442–449, Washington, DC, USA, 2007. IEEE Computer Society.

[34] L. Rodrìguez and X. Li. A dynamic vertical partitioning approach for distributed database system. In Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on, pages 1853–1858, 2011.

[35] L. Rodrìguez and X. Li. A support-based vertical partitioning method for database design. In Electrical Engineering Computing Science and Automatic Control (CCE), 2011 8th International Conference on, pages 1–6, oct. 2011.

[36] L. Rodrìguez, X. Li, and P. Mejìa-Alvarez. An active system for dynamic vertical partitioning of relational databases. In I. Batyrshin and G. Sidorov, editors, Advances in Soft Computing, volume 7095 of Lecture Notes in Computer Science, pages 273–284. Springer Berlin Heidelberg, 2011.

[37] D. Sacca and G. Wiederhold. Database partitioning in a cluster of processors. ACM Trans. Database Syst., 10:29–56, 1985.

[38] J. R. Slagle, C. L. Chang, and S. R. Heller. A clustering and data-reorganizing algorithm. Systems, Man and Cybernetics, IEEE Transactions on, SMC-5(1):125–128, Jan 1975.

[39] J. H. Son and M.-H. Kim. α-partitioning algorithm: Vertical partitioning based on the fuzzy graph. In Proceedings of the 12th International Conference on Database and Expert Systems Applications, DEXA '01, pages 537–546, London, UK, UK, 2001. Springer-Verlag.