A Quantitative Study of Two Matrix Clustering Algorithms

Alexander Slesarev
Saint-Petersburg University, Russia
Saint-Petersburg, Russia
alexander.g.slesarev@gmail.com

Viacheslav Galaktionov[1, 2]
[1] JetBrains Research,
[2] Saint-Petersburg State University, Russia
Saint-Petersburg, Russia
viacheslav.galaktionov@gmail.com

Nikita Bobrov[1, 2]
[1] JetBrains Research,
[2] Saint-Petersburg State University, Russia
Saint-Petersburg, Russia
nikita.v.bobrov@gmail.com

George Chernishev[1, 2]
[1] JetBrains Research,
[2] Saint-Petersburg State University, Russia
Saint-Petersburg, Russia
g.chernyshev@spbu.ru

***Abstract*—Matrix clustering is a technique which permutes rows and columns of a matrix to form densely packed regions. It originated in the 70s and was used for various object grouping problems, such as machine-component grouping. The database community noticed these algorithms and successfully applied them to the vertical partitioning problem. Recently, a resurgence of interest in these algorithms happened. Nowadays they are being considered for dynamic (on-line) vertical partitioning and tuning of multistores.**

In our previous papers we have described our project that aims to study the applicability of the recent matrix clustering algorithms for vertical partitioning problem. We have presented our evaluation approach and reported its results, concerning several of these algorithms. Our idea was to evaluate them directly, using PostgreSQL database. Previous studies found that these algorithms can be of use if they employ attribute replication strategy. In this paper we continue our investigation and consider a novel algorithm of this class. Its distinctive feature is that it performs attribute replication during the branch and bound search. We compare it with the best one of the earlier algorithms using both real and synthetic workloads.

Experiments demonstrated that novel algorithm produces slightly worse configurations (about 10%), but its run times are significantly better and are almost independent of the cohesion parameter.

***Index Terms*—databases, database tuning, vertical partitioning, experimentation, matrix clustering, fragmentation**

I. Introduction

Vertical partitioning is a technique that is used to speed up query processing in databases. The idea is to divide a table into fragments, which will contain only a subset of attributes. In order to ensure that database will not undergo semantic changes, the following rules of vertical partitioning are used [1]: completeness, reconstruction, disjointness. Sometimes the disjointness rule is relaxed. In this case it is said that vertical partitioning is performed with attribute replication.

The speedup comes from the fact that some queries would have to read less data. Indeed, suppose that for a given query all needed attributes are allocated into a single fragment and this fragment contains no extra attributes. In this case, we can roughly estimate that $number\_of\_rows \times extra\_attributes\_lengths$ bytes can be saved during the data reading phase in case of slotted page data layout [2].

However, if there is a query that requires attributes from two or more fragments, then its performance may suffer due to the record reconstruction costs. Data modification operations (inserts, deletes, and updates) complicate things further since they involve all attributes of record, regardless and thus, all fragments should be modified. The impact of the additional disk seeks on a hard drive may be so large, that it can make the whole partitioning impractical.

Due to all these facts, there is still no fully-automatic vertical partitioning support in industrial database systems. Moreover, unlike horizontal, vertical partitioning is not supported in SQL DDL: e.g. in PostgreSQL it is possible to define horizontal fragments using the “PARTITION BY” clause for a “CREATE TABLE” statement.

Nevertheless, there are multiple semi-automatic stand-alone tools (“advisors”, see surveys [3], [4]) for this task. All of them recommend beneficial vertical partitioning schemes for a specified workload (queries) and let the database administrator decide whether to implement them or not.

The reason for the limited success of these tools — the overwhelming majority of them are academic research prototypes, not industrial products — is that finding an optimal solution is an NP-hard problem for many different formulations [5]–[7]. A well-known fact is that the number of different vertical partitioning schemes for a single table is equal to the Nth Bell number, where N is the number of attributes [8]. However, due to the interest of both industrial and academic communities, the development of such advisors continues.

In the core of such system lies an algorithm that traverses the partitioning space and selects a beneficial scheme. There are two classes of algorithms for this task: cost-based and heuristic ones. The former employ some kind of cost-based model to evaluate the quality of a given partitioning scheme in terms of query run times, required space, and other metrics. The latter proposes some kind of procedure to generate a "good" scheme. Usually, some considerations are presented as to why it is likely to generate a beneficial partitioning scheme, but not a strict proof.

The heuristic approach was very popular in the 70–80s, but was later abandoned in favour of the cost-based one. Nowadays there is a resurgence of interest in heuristics due to the appearance of novel application areas: dynamization of vertical partitioning [9]–[12], tuning of multistores [13], big data applications or any other cases featuring constrained resources.

In our previous studies [14]–[16] we have described our project that aims to study the applicability of a recently developed subclass of heuristic algorithms — matrix clustering algorithms. We have constructed a framework for evaluation of such algorithms that uses PostgreSQL. Then we have evaluated a number of these algorithms [17]–[19] using the TPC-H benchmark. In this paper we continue our studies and consider the most recent algorithm of this group [20].

The rest of this paper is organized as follows. In Section II we provide a short introduction into the subject area and describe existing types of heuristic approaches. Next, in Section III we introduce matrix clustering algorithms and provide a description of the considered algorithm. Section IV describes the experimental framework, setup, and the experiments. The results of evaluation are discussed in Section V and Section VII concludes this paper.

II. Related Work

As it was stated in the introduction section, there are two types of approaches to vertical partitioning problem — cost-based and heuristic. Since this problem is almost 40 years old and a lot of results have been accumulated, we will only describe heuristic studies in this section. More broad surveys containing cost-based approaches can be found in references [?], [21]. Heuristic vertical partitioning algorithms can be classified into the following major groups [14], [15]:

- Attribute affinity and matrix clustering approaches [17]–[19], [22]–[24]. Attribute affinity is a measure which shows how frequently two attributes are requested together in a given workload. These approaches use it as follows:
  1) A workload is used to construct an Attribute Usage Matrix (AUM), a special way to represent which attributes are used by each query of a workload.
  2) Attribute affinity is calculated for all pairs of attributes and an Attribute Affinity Matrix (AAM) is constructed.
  3) A special algorithm for row and column permutation is applied to this AAM. Afterwards, "dense" regions are extracted and used to define resulting partitions.

  Th studies belonging to matrix clustering approach, and in particular, the ones considered in our study, permute AUMs, but not AAMs.
- Graph approaches [5], [25]–[28]. Similarly to previous ones, these approaches start with a workload and use it to construct an AAM. However, in this case the AAM is considered as an adjacency matrix of an undirected weighted graph, where the nodes are attributes and the edge weights show the affinity for a given pair of attributes. Finally, this graph is used to search for special structures which will be used to define resulting partitions. There are many approaches, e.g. Kruskal-like algorithms or cutting the Hamiltonian way.
- Data mining approaches [29]–[31]. In this approach association rule mining is used to derive vertical fragments. Workload is considered as a transaction set and the rules use sets of attributes as items. This group of vertical partitioning algorithms is relatively new, so existing algorithms for association rule search are frequently used. For example, a popular choice is to adapt Apriori [32] or FP-Max algorithms.

III. Matrix Clustering Algorithms

A. Basics

The general scheme of this approach is as follows [14], [15]:

- Construct an Attribute Usage Matrix (AUM) from the workload. The matrix is defined 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 in 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 in Figure 2. The first column obstructs the perfect decomposition into several clusters. In this case, the algorithm should produce a decomposition which would

	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

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

minimally harm query processing and would result in an overall performance improvement. Matrix clustering algorithms employ different strategies to select such a decomposition.

A systematic review of matrix clustering algorithms is presented in studies [14], [15]. Here, we will consider only the recently emerged approaches.

B. Recent Advances

In our project we study a series of works by Chun-Hung Cheng et al [17]–[20]. These algorithms employ a branch and bound search that tries find submatrices with a specific conditions. Its input is the threshold (target cohesion) — a ratio of 1 in the resulting matrices.

In this study we are interested in two algorithms — A09 [19] and A11 [20].

The A09 algorithm comes with three different strategies that define the treatment of intersubmatrix attributes — nearest, separate, and replicate. In the first one such attribute goes to the nearest submatrix, in the second all such attributes are assigned into a dedicated submatrix, and the last one replicates attribute into each submatrix requiring it. Note that the strategy is applied after the clustering is done.

The A11 algorithm has a different idea. If during branch and bound traversal the algorithm encounters such attribute, then it replicates it and tries to decompose the matrices further.

IV. Experiments

A. Benchmarking

In our previous works we have developed a special prototype for experimental evaluation of matrix clustering algorithms. The idea of our approach is to directly check whether generated partitioning schemes help to improve query performance. For these purposes we employ the PostgreSQL DBMS and several workloads, both real and synthetic.

The architecture of our prototype is presented in 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.
- 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.

B. Experimental Setup and Evaluation Procedure

In our experiments we have used the following hardware and software setup:

- Inspiron 15 7000 Gaming (0798), 8GiB, Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz, TOSHIBA 1TB MQ02ABD1
- Ubuntu 18.10, PostgreSQL 11.1, gcc 8.2.0

Data for quality-related graphs was obtained by running 10 invocations of the respective algorithm and averaging the result. We deemed sufficient only one run for run time graphs, since even one invocation can require up to two hours.

In order to ensure the maximum quality of experiments, several measures were taken:

1) We have eliminated data caching for both operating system caches and PostgreSQL caches. For this, we have restarted PostgreSQL and dropped the operating system caches before running each query. Operating system caches were dropped by writing “3” to `/proc/sys/vm/drop_caches`.
2) Next, we had manually checked plans for each query and noticed that some queries may have different scan operator implementations, depending on the table. Frequently, query on a partitioned table had not sequential scan, but a parallel one. To handle this,

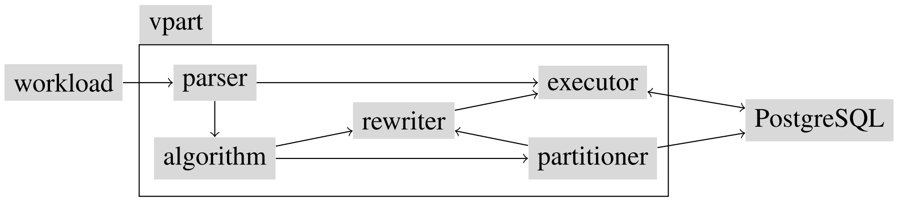


Fig. 3: The architecture of our approach

we have restricted query optimizer to use only sequential scans by issuing the following command `set max_parallel_workers_per_gather to 0;`.

To ensure that no hidden caching or other unaccounted processes happen we have designed the following simple criterion. Suppose that we have a set of queries that involve only a single table and are essentially scans without complex data processing. Initially, we run these queries on the original table and record their run times. Then, for every query we designate a table that will contain all attributes necessary to evaluate it. Thus, no joins are needed. For some queries, their tables will also contain extra attributes, thus it is not a 1:1 mapping. Then we run each query on corresponding table and record its run time. Eventually the following two values should be approximately equal:

1) $\sum_{q_i \in Queries}(size(T)/time(q_i))$
2) $\sum_{q_i \in Queries}(size(table(q_i))/time(q_i))$

In these equations $size(T)$ is the size of a table in bytes. Functions $time(q_i)$ and $table(q_i)$ return the time it took to run a query q_i and a table that corresponds to query q_i.

In other words, we check that workload run times depend solely on the size of the table.

Having applied all the aforementioned measures, we have obtained the difference in these values of about 10-15%. We deemed such result as acceptable and decided to start evaluating algorithms.

Finally, we must note that our matrix clustering algorithms are parallel [16]. However, in this paper we did not consider them and instead employed their sequential versions.

C. Experiments

In our study we have addressed two applicability aspects of matrix clustering algorithms: quality of generated partitioning schemes and algorithm run times. Both of them are important since quality is the primary characteristic of any partitioning algorithm and run times determine its suitability for on-line vertical partitioning.

To evaluate the quality of partitioning we have compared algorithm A11 to the best of the other matrix clustering algorithms (according to our previous studies [14], [15]) — A09. This algorithm has three different strategies, that were described earlier. In our experiments we compare the quality of resulting partitions of all three of them with the ones obtained by A11.

To conduct experiments we have employed the “Star” table of the SDSS (Sloan Digital Sky Survey) dataset. The SDSS is a publicly available astronomical database that contains detailed three-dimensional maps of the Universe. It is frequently used as testing dataset in various data partitioning studies. We employ the following pack: SDSS-IV Data Release 14, 2016. Its “Star” table contains 509 attributes and 492515 records.

To obtain representative workloads we have also used the SDSS dataset. There, it is possible to see what queries users were issuing. For this purpose there exists a special web-site[1]. Through it we have selected 8 queries from the workload that address solely this table.

In our first experiment we have varied cohesion measure for three strategies of A09 and compared it with A11. The results are presented in Figure 5a. In this graph each bar represents the performance of an individual algorithm with the corresponding strategy. There also two horizontal lines: not clustered and pinched not clustered. The first one is the workload run time on the original, unmodified table. The second is the workload run time on cleaned up original table, containing only 30 attributes that are referenced in the workload. In this experiment we varied the cohesion measure parameter.

To evaluate algorithm run times we used both SDSS and synthetic tests. The result of SDSS tests are presented in Figure 5b. Here, we also vary cohesion for the same four algorithms.

In the synthetic tests we have tried to study the scalability of A11 algorithm in terms of run times. For this, we have generated a set of random 0-1 matrices, with different probabilities of having 1 in each position (cohesion). Then, we have studied the dependency of the run time on the size of the matrix. The specified threshold was set to 0.9 in all experiments. If the threshold is more than the used cohesion, then a solution (the original matrix) is found almost immediately. We also set a time limit of 2 hours, after reaching which the algorithm is stopped.

We started with square matrices (see Figure 4a), then separately evaluated the influence of the number of columns (Figure 4b) and the number of rows (Figure 4c) on the algorithm run time. In the last two experiments we fixed one dimension to 20 and increased the other up until the time limit was reached.

Finally, we have looked into the storage requirements of these algorithms (Figure 6). Here, we show the required disk space for each generated configuration. On top of each bar an overall number of fragments is shown. We have also divided each bar into parts representing the sizes of resulting

[1]http://skyserver.sdss.org/log/en/traffic/

fragments. The sizes of original and pinched tables are also shown by horizontal lines.

V. Results and Discussion

- All of the algorithms produced partitioning schemes that provide better performance than the original and pinched tables, regardless of the cohesion value.
- The quality of produced solutions heavily depends on the cohesion value. Starting with the cohesion value of 0.8 results of A11 start to rival the results of the best A09 strategies. However, up to this point, the clear winner is A09 with replication.
- Overall, the best result was produced by a replicating variant of A09 (3.358, cohesion=0.55), with a separate variant of A09 being the fourth (3.500, cohesion=0.8), and A11 being the fifth (3.553, cohesion=0.8).
- It is interesting to note that there is some sort of a global minimum in the 0.7 point. Here the sum of all algorithms is minimal over the whole cohesion range.
- With the SDSS workload Algorithm A11 works almost ten times faster than A09, regardless of the employed strategy. Note that increasing target threshold also increases run times. For A09 run times increased from less than 1 second to almost 140 seconds, while A11 took 0.06 and 0.119 seconds respectively.
- The scalability of A11 is not as good as desired. However, two points should be taken into account. Firstly, run times depend on the number of referenced attributes in the workload, not on the total number. Secondly, in our scalability experiments we used an extremely large threshold of the cohesion – 0.9. Finally, the author [20] noted that it is possible to interrupt the algorithm earlier while still obtaining decent results. Therefore, further studies are needed.
- Increasing the number of attributes impacts run times more than increasing the number of queries. In two hours time it is possible to process either an 20×25 matrix or an 205×20 one.
- The solutions produced by all algorithms require 1.5–2 times more disk space than the pinched table. Increasing target threshold increases the number of fragments and the overall required disk space. Interestingly, for high cohesion values A11 produces more fragments, but does not help to improve performance.

VI. Threats to Validity

In our study we have identified a number of issues that should be kept in mind while discussing our results.

1) First of all, the policy of database restarts after each query may be unfair. In real-life scenarios where these algorithms will hypothetically be used, database caching would be present. However, such scenarios are nearly impossible to simulate since they require hundreds or thousands of real queries and more important, their frequencies and arrival patterns.
2) Next, the SDSS dataset is only a single dataset, on other datasets different results may be obtained. Moreover, it is a scientific dataset used by the astronomy research community and therefore, its queries and data may not match industrial ones. Nevertheless, it is popular in vertical partitioning community (e.g. see [**?**]) since the lack of industrial schema-less benchmarks.
3) There may be errors in our implementation of these algorithms. In order to mitigate this threat we have tested our implementation on example matrices presented in the considered papers and ensured that resulting partitioned matrices are the same. Furthermore, to address this issue we plan to release the source code on GitHub.
4) Contemporary DBMSes are very complex systems in which minimal changes to inputs may drastically affect performance. Therefore, during experimental evaluation performance may change not due to vertical partitioning, but due to other events, such as query optimizer selecting a completely different plan. To counter this we have carefully checked query execution plans to find and eliminate any inconsistencies. We have also developed a criterion that allows to detect such inconsistencies in simple cases.
5) In our evaluation we have considered a relatively simple workload which involves only a single table. Having to perform extra joins in addition to partitioning-induced ones may significantly decrease the overall performance and thus desirability of vertical partitioning. However joins with other tables are extremely rarely considered in literature [3], only a handful studies address them.

VII. Conclusion

In this paper we have presented a quantitative study of two recent matrix clustering algorithms. We have studied their quality of outputs, run times, and storage requirements using synthetic and real datasets.

Evaluation showed that for schema-less data all algorithms can produce beneficial configuration, while a replicating variant of A09 is 10% better than A11. However, A11 is significantly faster and more importantly, less impacted by the target threshold parameter.

References

[1] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems*, 3rd ed. Springer Publishing Company, Incorporated, 2011.

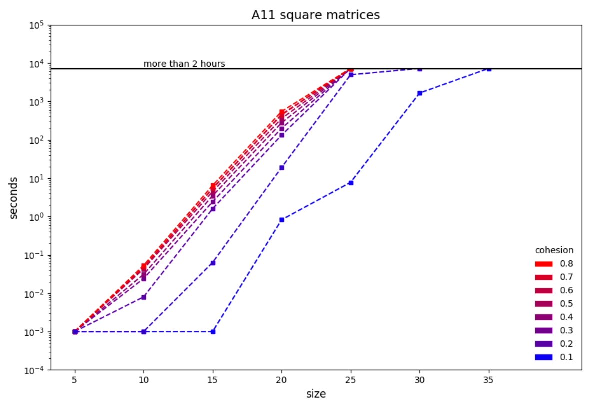
[2] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed. New York, NY, USA: McGraw-Hill, Inc., 2003.

[3] G. Chernishev, "A survey of dbms physical design approaches," *SPIIRAS Proceedings*, vol. 24, pp. 222–276, 2013. [Online]. Available: www.mathnet.ru/trspy580

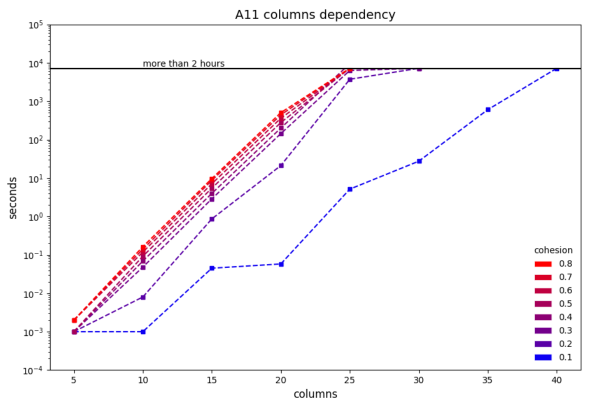
[4] ——, "Vertical Partitioning in Relational DBMS," 30 4 2015, talk at the Moscow ACM SIGMOD chapter meeting; slides and video: http://synthesis.ipi.ac.ru/sigmod/seminar/s20150430.html [Accessed: 2015 11 09].

[5] X. Lin, M. Orlowska, and Y. Zhang, "A graph based cluster approach for vertical partitioning in database design," *Data & Knowledge Engineering*, vol. 11, no. 2, pp. 151–169, 1993.

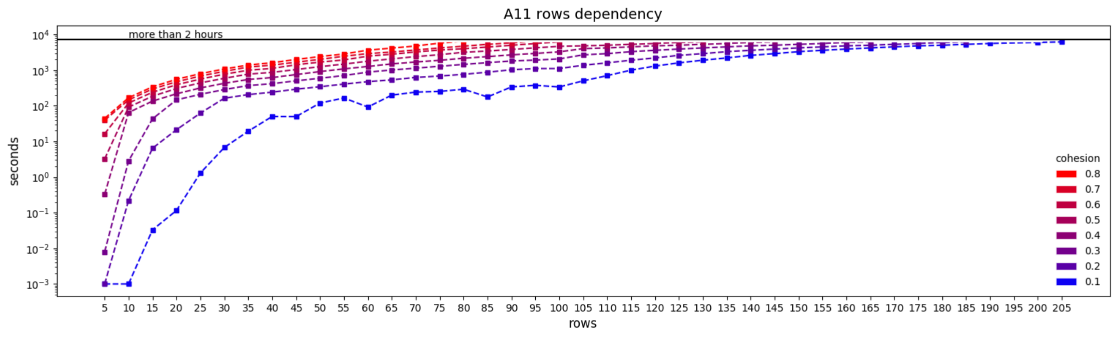
[6] D. Sacca and G. Wiederhold, "Database partitioning in a cluster of processors," *ACM Trans. Database Syst.*, vol. 10, pp. 29–56, 1985.



(a) A11 run times on a square matrix

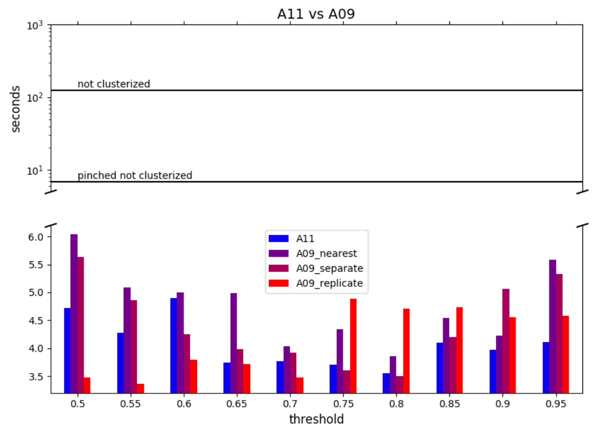


(b) Dependency of A11 run times on matrix width

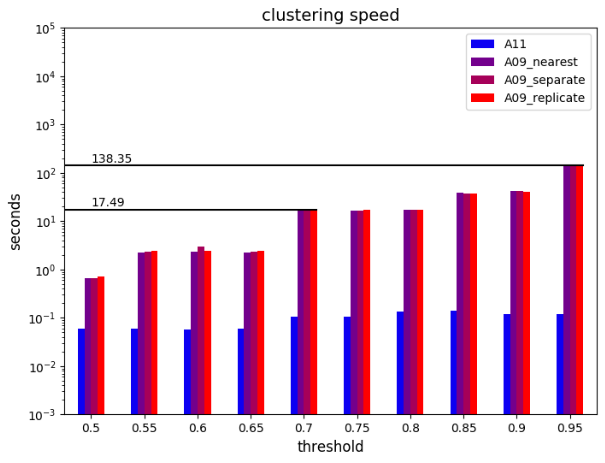


(c) Dependency of A11 run times on matrix height

Fig. 4: Run times of A11 matrix clustering algorithm, synthetic datasets.



(a) Quality of partitioning



(b) Algorithm run times

Fig. 5: Performance of A11 and A09 matrix clustering algorithms, SDSS datasets.

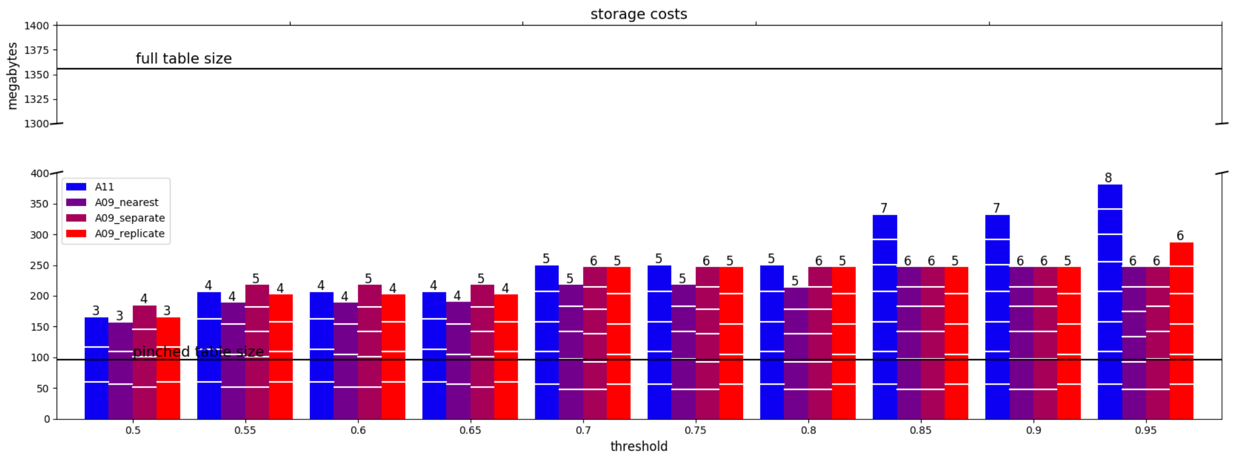


Fig. 6: Storage requirements.

[7] P. M. G. Apers, "Data allocation in distributed database systems," *ACM Trans. Database Syst.*, vol. 13, pp. 263–304, 1988. [Online]. Available: http://doi.acm.org/10.1145/44498.45063

[8] M. Hammer and B. Niamir, "A heuristic approach to attribute partitioning," in *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '79. New York, NY, USA: ACM, 1979, pp. 93–101. [Online]. Available: http://doi.acm.org/10.1145/582095.582110

[9] A. Jindal and J. Dittrich, "Relax and let the database do the partitioning online," ser. Lecture Notes in Business Information Processing, 2012, vol. 126, pp. 65–80.

[10] L. Li and L. Gruenwald, "Self-managing online partitioner for databases (smopd): A vertical database partitioning system with a fully automatic online approach," ser. IDEAS '13, 2013, pp. 168–173.

[11] L. Rodríguez and X. Li, "A dynamic vertical partitioning approach for distributed database system," in *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, 2011, pp. 1853–1858.

[12] L. Rodríguez, X. Li, and P. Mejía-Alvarez, "An active system for dynamic vertical partitioning of relational databases," ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 7095, pp. 273–284.

[13] 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," ser. SIGMOD '14, 2014, pp. 1591–1602. [Online]. Available: http://doi.acm.org/10.1145/2588555.2588568

[14] V. Galaktionov, G. Chernishev, B. Novikov, and D. Grigoriev, "Matrix clustering algorithms for vertical partitioning problem: an initial performance study," in *Selected Papers of the XVIII International Conference on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2016), Ershovo, Moscow Region, Russia, October 11-14, 2016.*, 2016, pp. 24–31. [Online]. Available: http://ceur-ws.org/Vol-1752/paper05.pdf

[15] V. Galaktionov, G. Chernishev, K. Smirnov, B. Novikov, and D. A. Grigoriev, "A study of several matrix-clustering vertical partitioning algorithms in a disk-based environment," in *Data Analytics and Management in Data Intensive Domains*, L. Kalinichenko, S. O. Kuznetsov, and Y. Manolopoulos, Eds. Cham: Springer International Publishing, 2017, pp. 163–177.

[16] V. Galaktionov, "Parallelization of matrix clustering algorithms (accepted)," in *Proceedings of the Sixth International Conference on Informatics Problems (SPISOK 2016)*, 2016. [Online]. Available: http://spisok.math.spbu.ru/2016/txt/SPISOK-2016.pdf

[17] C.-H. Cheng, "A branch and bound clustering algorithm," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 25, no. 5, pp. 895–898, 1995.

[18] C. Cheng, "Algorithms for vertical partitioning in database physical design," *Omega*, vol. 22, no. 3, pp. 291–303, 1994.

[19] C.-H. Cheng and J. Motwani, "An examination of cluster identification-based algorithms for vertical partitions," *Int. J. Bus. Inf. Syst.*, vol. 4, no. 6, pp. 622–638, 2009. [Online]. Available: http://dx.doi.org/10.1504/IJBIS.2009.026695

[20] C.-H. Cheng, K.-F. Wong, and K.-H. Woo, "An improved branch-and-bound clustering approach for data partitioning," *International Transactions in Operational Research*, vol. 18, no. 2, pp. 231–255, 2011. [Online]. Available: http://dx.doi.org/10.1111/j.1475-3995.2010.00781.x

[21] G. Chernishev, "Towards self-management in a distributed column-store system," in *New Trends in Databases and Information Systems*, ser. Communications in Computer and Information Science, T. Morzy, P. Valduriez, and L. Bellatreche, Eds. Springer International Publishing, 2015, vol. 539, pp. 97–107. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-23201-0_12

[22] N. Gorla and W. J. Boe, "Database operating efficiency in fragmented databases in mainframe, mini, and micro system environments," *Data & Knowledge Engineering*, vol. 5, no. 1, pp. 1–19, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0169023X9090030H

[23] J. A. Hoffer and D. G. Severance, "The use of cluster analysis in physical data base design," ser. VLDB '75, 1975, pp. 69–86. [Online]. Available: http://doi.acm.org/10.1145/1282480.1282486

[24] N. Bobrov, G. Chernishev, and B. Novikov, "Workload-independent data-driven vertical partitioning," in *New Trends in Databases and Information Systems*, M. Kirikova, K. Nørvåg, G. A. Papadopoulos, J. Gamper, R. Wrembel, J. Darmont, and S. Rizzi, Eds. Cham: Springer International Publishing, 2017, pp. 275–284.

[25] 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*, vol. 32, no. 3, pp. 215–230, 2002.

[26] S. B. Navathe and M. Ra, "Vertical partitioning for database design: a graphical algorithm," ser. SIGMOD '89, 1989, pp. 440–450.

[27] J. Du, K. Barker, and R. Alhajj, "Attraction — a global affinity measure for database vertical partitioning," in *ICWI*. IADIS, 2003, pp. 538–548.

[28] J. H. Son and M.-H. Kim, "α-partitioning algorithm: Vertical partitioning based on the fuzzy graph," ser. DEXA '01, 2001, pp. 537–546. [Online]. Available: http://dl.acm.org/citation.cfm?id=648314.755837

[29] 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*, march 2012, pp. 273–276.

[30] N. Gorla and B. P. W. Yan, "Vertical fragmentation in databases using data-mining technique," in *Database Technologies: Concepts, Methodologies, Tools, and Applications*, J. Erickson, Ed. IGI Global, 2009, pp. 2543–2563.

[31] L. Rodriguez 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*, oct. 2011, pp. 1–6.

[32] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules in large databases,” ser. VLDB ’94, 1994, pp. 487–499.