An Approach to Fuzzy Clustering of Big Data Inside a Parallel Relational DBMS

Mikhail Zymbler(✉), Yana Kraeva, Alexander Grents, Anastasiya Perkova, and Sachin Kumar

South Ural State University, Chelyabinsk, Russia
{mzym,kraevaya,grentsav,perkovaai}@susu.ru,
sachinagnihotri16@gmail.com

Abstract. Currently, despite the widespread use of numerous NoSQL systems, relational DBMSs remain the basic tool for data processing in various subject domains. Integration of data mining methods with relational DBMS is a topical issue since such an approach avoids export-import bottleneck and provides the end-user with all the built-in DBMS services. Proprietary parallel DBMSs could be a subject for integration of data mining methods but they are expensive and oriented to custom hardware that is difficult to expand. At the same time, open-source DBMSs are now being a reliable alternative to commercial DBMSs and could be seen as a subject to encapsulate parallelism. In this study, we present an approach to fuzzy clustering of very large data sets inside a PDBMS. Such a PDBMS is obtained by small-scale modifications of the original source code of an open-source serial DBMS to encapsulate partitioned parallelism. The experimental evaluation shows that the proposed approach overtakes parallel out-of-DBMS solutions with respect to export-import overhead.

Keywords: Big Data · Parallel DBMS · PostgreSQL · Fuzzy clustering

1 Introduction

Currently, despite the widespread use of numerous NoSQL systems, relational DBMSs remain the basic tool for data processing in various subject domains. To make sure of this fact, let us take a look to statistics from the DB-Engines.com portal. At the end of 2019, relational DBMSs kept the last place by popularity among the data management systems (by their mentions in news feeds, social and professional networks, etc.) yielding all the NoSQL systems. At the same time, ranking of the systems above by database model showed that relational DBMSs take three quarters of the market.

In addition to OLTP and OLAP scenarios, current data intensive applications should provide data mining abilities. This fact makes the integration of data mining methods with relational DBMS a topical issue [18]. Indeed, if we

A. Elizarov et al. (Eds.): DAMDID/RCDL 2019, CCIS 1223, pp. 211–223, 2020.
https://doi.org/10.1007/978-3-030-51913-1_14

consider DBMS only as a fast and reliable data repository, we get significant overhead for export large data volumes outside a DBMS, changing data format, and import results of analysis back into a DBMS. Moreover, such an integration provides the end-user with all the built-in DBMS services (query optimization, data consistency and security, etc.).

Nowadays, Big Data phenomenon demands parallel DBMSs (PDBMSs) to processing very large databases. Proprietary PDBMSs could be a subject for integration of data mining methods but they are expensive and oriented to custom hardware that is difficult to expand. Open-source DBMSs are now being a reliable alternative to commercial DBMSs but there is a lack of open-source parallel DBMSs since the development of such a complex software system is rather expensive and takes a lot of time.

This paper is a revised and extended version of the invited talk [31]. In the paper, we present an approach to fuzzy clustering of very large data sets inside a PDBMS. Such a PDBMS is obtained by small-scale modifications of the original source code of an open-source serial DBMS to encapsulate partitioned parallelism. The paper is structured as follows. Section 2 contains a short overview of related work. Section 3 describes a method of encapsulation of partitioned parallelism into serial DBMS. In Sect. 4, we present fuzzy clustering algorithm for PDBMS described above. In Sect. 5, we give the results of the experimental evaluation of our approach. Finally, Sect. 6 summarizes the results obtained and suggests directions for further research.

2 Related Work

Research on the integration of data analytic methods with relational DBMS started as far as data mining became full-fledged scientific discipline. First investigators proposed mining query languages [7,10] and SQL extensions for data mining [15,29]. There are SQL implementations of algorithms to solve data mining basic problems, namely association rules [26,27], classification [19,25], and clustering [16,17], as well as graph mining problems [14,21].

The MADlib library [8] provides many data mining methods inside PostgreSQL. The MADlib exploits user-defined aggregates (UDAs), user-defined functions (UDFs), and a sparse matrix C library to provide efficient data processing on disk and in memory.

The Bismarck system [5] exploits UDFs as a convenient interface for in-DBMS data mining. The Bismarck supports logistic regression, support vector machines and other mining methods, which are based on incremental gradient descent technique.

The DAnA system [13] automatically maps a high-level specification of in-DBMS analytic queries to the FPGA accelerator. The accelerator implementation is generated from an UDF, expressed as part of a SQL query in a Python-embedded Domain-Specific Language. The DAnA supports striders, a special hardware structures that directly interface with the buffer pool of the DBMS to extract and clean the data tuples, and pass the data to FPGA.

This paper extends our previous research as follows. In [23,24], we presented an approach to encapsulation parallel mining algorithms for many-core accelerators into an open-source DBMS (with PostgreSQL as an example) for small-scale data sets. In [20,22], we developed a method for encapsulation parallelism into an open-source DBMS (with PostgreSQL as an example). In this paper, we apply the resulting PDBMS to fuzzy clustering very large data sets, extending our previous in-PostgreSQL fuzzy clustering algorithm [16].

3 Encapsulation of Partitioned Parallelism into a Serial DBMS

3.1 Basic Ideas

Our development is based on the concept of the partitioned parallelism [4], which assumes the following. Let us consider a computer cluster as a set of alike computers (nodes) connected with high-speed network where each node is equipped with its own main memory and disk, and has an instance of PDBMS installed.

Each database table is fragmented into a set of horizontal partitions according to the number of nodes in the cluster, and partitions are distributed across nodes. The way of partitioning is defined by a fragmentation function (a table is associated with its own fragmentation function), which takes a tuple of the table as an input and returns a node where the tuple should be stored. One of the table attributes is declared as a partitioning attribute to be an argument of the fragmentation function.

One of the PDBMS instances is declared as a coordinator. A retrieve query is executed independently by all the PDBMS instances where each instance processes its own database partition and generates a partial query result, and then partial results are merged by the coordinator into the resulting table.

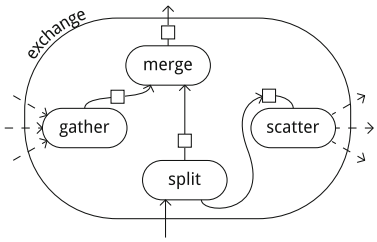


Fig. 1. Structure of the EXCHANGE operator

To implement the schema described above, PDBMS engine provides the EXCHANGE operator [28]. Such an operator is inserted into a serial query plan and encapsulates all the details related to parallelism. EXCHANGE is a composite operator (see Fig. 1) with two attributes, namely port and distribution function.

Port is serial number of the Exchange operator in the query plan to provide concordance of all the query plans of PDBMS instances. The distribution function takes a tuple as an input and returns a number of the instance where the tuple should be processed.

The Split operator computes the distribution function for an input tuple. If the tuple is to be processed by the current instance, it passed to the Merge operator. Otherwise, the tuple is passed to the Scatter operator to be sent to the respective instance. The Gather operator receives tuples from other instances and passes them to Merge. The Merge operator alternately combines the tuple streams from Split and Gather.

Exchange provides data transfers in case of queries where tables are joined, and partitioning attribute of the table(s) does not match to join attribute. Also, Exchange with distribution function identical to the coordinator number being inserted into the root of a query plan, provides merging partial results of the query into the resulting table.

3.2 Implementation with PostgreSQL

Open-source DBMSs are now being a reliable alternative to commercial DBMSs. In this regard, an idea of obtaining a PDBMS by small-scale modifications of the original source code of an open-source serial DBMS to encapsulate partitioned parallelism looks promising. PargreSQL [20,22] is an example of PDBMS implemented on top of PostgreSQL in an above-mentioned way.

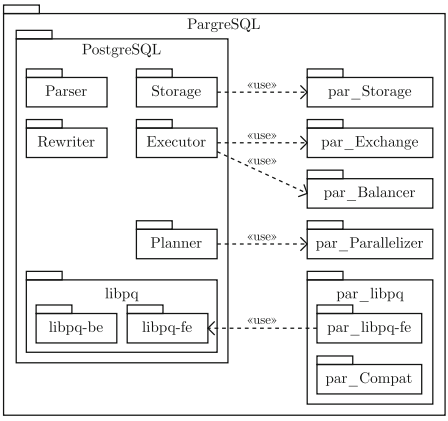


Fig. 2. Structure of PargreSQL

Figure 2 depicts module structure of PargreSQL. The original open-source DBMS is considered as one of PDBMS subsystems. Novel subsystems extend PargreSQL as follows.

The *par_Storage* subsystem provides metadata on partitioning in PDBMS dictionary. In PargreSQL, each table should contain at least one integer column to be a partitioning attribute. Thus, each `CREATE TABLE` command should be ended by the `(WITH FRAGATTR=`*pa*`)` clause to provide the table's fragmentation function as *pa mod P* where *pa* is partitioning attribute and P is the number of PDBMS instances.

The *par_Exchange* subsystem implements the EXCHANGE operator described above. The *par_Parallelizer* subsystem inserts EXCHANGEs into the appropriate places of a query plan given from PostgreSQL.

The *par_libpq* subsystem is a modified version of the PostgreSQL `libpq` library. Being a wrapper over the original PostgreSQL `libpq-fe` front-end, the *par_libpq-fe* provides connection of a client to all the PDBMS instances and replicates a query to each PargreSQL engine.

The *par_Compat* subsystem is a set of C preprocessor macros that change the original PostgreSQL API calls into the PargreSQL API calls to provide transparent migration of PostgreSQL applications to PargreSQL.

In the end, resulting parallel DBMS is obtained by modifications that took less than one per cent of whole source code of PostgreSQL.

4 Fuzzy Clustering Inside a Parallel DBMS

Clustering could be seen as a task of grouping a finite set of objects from finite-dimensional metric space in such a way that objects in the same group (called a cluster) are closer (with respect to a chosen distance function) to each other than to those in other groups (clusters). Hard clustering implies that each object must belong to a cluster or not. In fuzzy clustering, each object belongs to each cluster to a certain membership degree. Fuzzy C-means (FCM) [3] is the one of the most widely used fuzzy clustering algorithm.

Further, in Sect. 4.1, we give notations and definitions regarding FCM, and in Sect. 4.2, we show in-PDBMS implementation of FCM.

4.1 Notations and Definitions

Let $X = \{x_1, \ldots, x_n\}$ is a set of objects to be clustered where an object $x_i \in \mathbb{R}^d$. Let $k \in \mathbb{N}$ is the number of clusters where each cluster is identified by a number from 1 to k. Then $C \in \mathbb{R}^{k \times d}$ is the *matrix of centroids* where $c_j \in \mathbb{R}^d$ is center of a j-th cluster.

Let $U \in \mathbb{R}^{n \times k}$ is the *matrix of memberships* where $u_{ij} \in \mathbb{R}$ reflects membership of an object x_i to a centroid c_j and the following holds:

$$\forall\, i, j \quad u_{ij} \in [0; 1], \quad \forall\, i \quad \sum_{j=1}^{k} u_{ij} = 1. \tag{1}$$

The *objective function* J_{FCM} is defined as follows:

$$J_{FCM}(X,k,m) = \sum_{i=1}^{n}\sum_{j=1}^{k} u_{ij}^{m}\rho^2(x_i,c_j) \tag{2}$$

where $\rho : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+ \cup 0$ is the distance function to compute proximity of an object x_i to a centroid c_j, and $m \in \mathbb{R}$ $(m > 1)$ is the fuzzyfication degree of J_{FCM} (the algorithm's parameter, which is usually taken as $m = 2$). Without loss of generality, we may use the Euclidean distance, which is defined as follows:

$$\rho^2(x_i,c_j) = \sum_{\ell=1}^{d}(x_{i\ell} - c_{j\ell})^2. \tag{3}$$

FCM iteratively minimizes J_{FCM} according to the following formulas:

$$\forall j,\ell \quad c_{j\ell} = \frac{\sum\limits_{i=1}^{n} u_{ij}^{m}\cdot x_{i\ell}}{\sum\limits_{i=1}^{n} u_{ij}^{m}} \tag{4}$$

$$u_{ij} = \sum_{t=1}^{k}\left(\frac{\rho(x_i,c_j)}{\rho(x_i,c_t)}\right)^{\frac{2}{1-m}}. \tag{5}$$

Finally, Algorithm 1 depicts basic FCM.

Alg. 1. FCM(IN X, m, ε, k; OUT U)

1: $U^{(0)} \leftarrow random(0..1)$; $s \leftarrow 0$
2: **repeat**
3: Compute $C^{(s)}$ by (4)
4: Compute $U^{(s)}$ and $U^{(s+1)}$ by (5)
5: $s \leftarrow s + 1$
6: **until** $\max_{ij}\{|u_{ij}^{(s+1)} - u_{ij}^{(s)}|\} \geq \varepsilon$
7: **return** U

4.2 The PgFCM Algorithm

The pgFCM algorithm is an in-PDBMS implementation of FCM. Our algorithm provides a database partitioned among the disks of computer cluster nodes, and performs fuzzy clustering by SQL queries where each database partition is processed independently by the respective instance of the PargreSQL DBMS described above.

Table 1 depicts design of pgFCM database. Underlined column name specifies primary key, and double underlined column name specifies partitioning attribute of the respective table. To improve efficiency of query execution, we provide index

Table 1. Database scheme of the pgFCM algorithm

Table	Columns	Indexed column(s)	Meaning
SH	$\underline{i}, x_1, x_2, \ldots, x_d$	i	Set of objects X, horizontal representation
SV	$\underline{i}, \underline{\ell}, val$	i, ℓ, (i, ℓ)	Set of objects X, vertical representation
C	$\underline{j}, \underline{\ell}, val$	ℓ, (j, ℓ)	Matrix of centroids C, vertical representation
SD	$\underline{i}, \underline{j}, dist$	i, (i, j)	Distances between objects x_i and centroids c_j
U	$\underline{i}, \underline{j}, val$	i, (i, j)	Matrix of memberships U at the s-th step, vertical representation
UT	$\underline{i}, \underline{j}, val$	(i, j)	Matrix of memberships U at the $(s+1)$-th step, vertical representation

file where indices by the primary key of the tables are automatically created, and the rest indices are created manually.

FCM computations (2)–(5) require aggregations over columns of the input table, which are directly impossible in SQL. To overcome this, in addition to horizontal representation of the algorithm's key data, namely set of objects and the matrix of memberships, we provide their vertical representation. Such a technique allows for using SQL aggregation functions `SUM` and `MAX` while implementing computations in FCM as a set of SQL queries.

Alg. 2. pgFCM(IN table SH, m, ε, k; OUT table U)

1: Initialize table U, table SV
2: **repeat**
3: Compute centroids by modifying table C
4: Compute distances by modifying table SD
5: Compute memberships by modifying table UT
6: **TRUNCATE** U; **INSERT INTO** U **SELECT * FROM** UT
7: $\delta \leftarrow$ **SELECT max**(**abs**(UT.val - U.val)) **FROM** UT, U **WHERE** UT.i=U.i **AND** UT.j=U.j
8: **until** $\delta \geq \varepsilon$
9: **SELECT * FROM** U

Algorithm 2 depicts implementation schema of pgFCM. The algorithm is implemented as an application in C language, which connects to all PargreSQL instances and performs computations by SQL queries over tables described in Table 1.

```
-- Initialization of table SV
for each cnt ∈ 1..d do
  INSERT INTO SV
    SELECT SH.i, cnt, x_cnt FROM SH
-- Initialization of table U
for each i ∈ 1..n do
  for each j ∈ 1..k do
    INSERT INTO U VALUES (i,j,random(0..1))
UPDATE U SET val=val/U1.tmp
  FROM (SELECT i, sum(val) AS tmp FROM U GROUP BY i) AS U1
  WHERE U1.i=U.i
```

Fig. 3. Implementation of initialization steps of pgFCM

Figure 3 shows how to form vertical representation of the SH table, and initialize the U table according to (1).

```
-- Computing centroids by modifying table C
INSERT INTO C
  SELECT R.j, SV.ℓ, sum(R.s * SV.val)/sum(R.s) AS val
  FROM (SELECT i, j, U.val^m AS s FROM U) AS R, SV
  WHERE R.i=SV.i
  GROUP BY j, ℓ;
-- Computing distances by modifying table SD
INSERT INTO SD
  SELECT i, j, sqrt(sum((SV.val-C.val)^2)) AS dist
  FROM SV, C
  WHERE SV.ℓ=C.ℓ;
  GROUP BY i, j;
-- Computing memberships by modifying table UT
INSERT INTO UT
  SELECT i, j, SD.dist^(2^(1-m))*SD1.den AS val
  FROM (SELECT i, 1/sum(dist^(2^(m-1))) AS den FROM SD
    GROUP BY i) AS SD1, SD
  WHERE SD.i=SD1.i;
```

Fig. 4. Implementation of computing steps of pgFCM

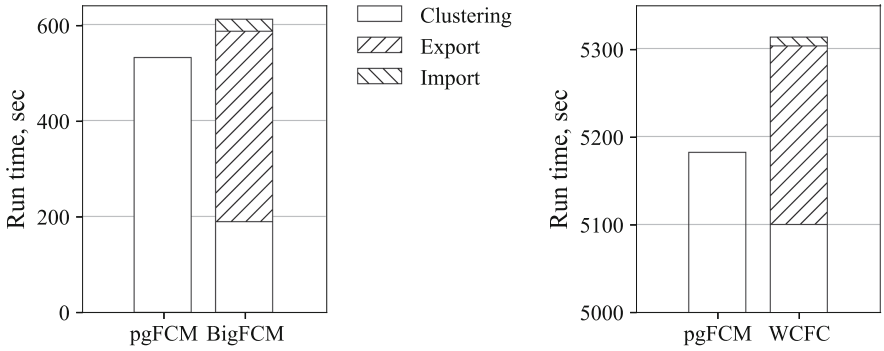
Figure 4 depicts computational steps of the pgFCM algorithm. While modifying the UT table, we exploit the following version of (5), which is more convenient for computations in SQL:

$$u_{ij} = \rho^{\frac{2}{1-m}}(x_i, c_j) \cdot \left(\sum_{t=1}^{k} \rho^{\frac{2}{m-1}}(x_i, c_t) \right)^{-1}. \quad (6)$$

5 The Experiments

We evaluated the proposed algorithm in experiments conducted on the Tornado SUSU supercomputer [11]. In the experiments, we compared the performance of pgFCM with the following analogs.

In [6], Ghadiri *et al.* presented BigFCM, the MapReduce-based fuzzy clustering algorithm. BigFCM was evaluated on a computer cluster of one master and 8 slave nodes over the HIGGS dataset [2] consisting of $1.1 \cdot 10^7$ the 29-dimensional objects. Hidri *et al.* [9] proposed the parallel WCFC (Weighted Consensus Fuzzy Clustering) algorithm. In the experiments, WCFC was evaluated on a computer cluster of 20 nodes over the KDD99 dataset [1] consisting of $4.9 \cdot 10^6$ the 41-dimensional objects. According to the experimental evaluation performed by the authors of the above-mentioned algorithms, both BigFCM and WCFC overtake other out-of-DBMS parallel implementations of FCM, namely MR-FCM [12] and Mahout FKM [30].



(a) pgFCM vs BigFCM over the HIGGS dataset ($n = 1.1 \cdot 10^7$, $d = 29$, $k = 2$)

(b) pgFCM vs WCFC over the KDD99 dataset ($n = 4.9 \cdot 10^6$, $m = 41$, $k = 2$)

Fig. 5. Comparison of pgFCM with analogs

We ran pgFCM on Tornado SUSU with a reduced number of nodes to make the peak performance of our system approximately equal to that of the system on which the corresponding competitor was evaluated. Throughout the experiments, we used the same datasets that were employed for the evaluation of the competitors. For comparison purposes, we used the run times reported in the respective papers [6,9].

For out-of-DBMS analogs, we measured the run time needed to export the initial dataset from the PostgreSQL DBMS (as conversion of a table to CSV file) and import the clustering results back into PostgreSQL (as loading of a CSV file into DBMS as a table), and added these overhead costs to the clustering running time of analogs. We assume the typical scenario when the data to be clustered are stored in a DBMS, and it is necessary to export the data outside the DBMS before clustering and import clustering results back into the DBMS after clustering.

Experimental results are depicted in Fig. 5. As can be seen, pgFCM is inferior to analogs in the performance of clustering. However, unlike the analogs, pgFCM performs clustering inside a PDBMS and does not need to export data and import results, and we can see that the proposed algorithm outruns analogs with respect to the overhead costs on export-import data.

6 Conclusions

In this paper, we addressed the task of mining in very large data sets inside a relational DBMSs, which remain the basic tool for data processing in various subject domains. Integration of data mining methods with relational DBMS avoids export-import bottleneck and provides the end-user with all the built-in DBMS services (query optimization, data consistency and security, etc.).

To effectively process very large databases, we encapsulate parallelism into the PostgreSQL open-source DBMS. Resulting parallel DBMS (called PargreSQL) is obtained by small-scale modifications of the original source code of PostgreSQL. Such modifications took less than one per cent of whole source code of PostgreSQL.

In this study, we implement Fuzzy C-Means clustering algorithm inside PargreSQL. The algorithms is implemented as an application in C language, which utilizes PargreSQL API. We design the algorithm's database in a way that allows for employing SQL row aggregation functions. We carry out experiments on computer cluster system using referenced data sets and compare our approach with parallel out-of-DBMS solutions. The experimental evaluation shows that the proposed approach overtakes analogs with respect to overhead on export data outside a DBMS and import results of analysis back into a DBMS.

In further studies, we plan to apply PargreSQL to other data mining problems over very large databases, e.g. association rules and classification.

Acknowledgments. The study was financially supported by the Ministry of Science and Higher Education of the Russian Federation within the framework of the Russian Federal Program for the Development of Russian Science and Technology from 2014 to 2020; project identifier: RFMEFI57818X0265 (contract no. 075-15-2019-1339 (14.578.21.0265)).

References

1. KDD Cup 1999 Data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Accessed 01 July 2019

2. Baldi, P., Sadowski, P., Whiteson, D.: Searching for exotic particles in high-energy physics with deep learning. Nat. Commun. **4**, 4308 (2014). https://doi.org/10.1038/ncomms5308
3. Bezdek, J.C.: Pattern Recognition with Fuzzy Objective Function Algorithms. Springer, New York (1981). https://doi.org/10.1007/978-1-4757-0450-1
4. DeWitt, D.J., Gray, J.: Parallel database systems: the future of high performance database systems. Commun. ACM **35**(6), 85–98 (1992). https://doi.org/10.1145/129888.129894
5. Feng, X., Kumar, A., Recht, B., Ré, C.: Towards a unified architecture for in-RDBMS analytics. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, 20–24 May 2012, pp. 325–336 (2012). https://doi.org/10.1145/2213836.2213874
6. Ghadiri, N., Ghaffari, M., Nikbakht, M.A.: BigFCM: fast, precise and scalable FCM on hadoop. Future Gener. Comput. Syst. **77**, 29–39 (2017). https://doi.org/10.1016/j.future.2017.06.010
7. Han, J., et al.: DBMiner: a system for mining knowledge in large relational databases. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD 1996), pp. 250–255, Portland (1996). http://www.aaai.org/Library/KDD/1996/kdd96-041.php
8. Hellerstein, J.M., et al.: The MADlib analytics library or MAD skills, the SQL. PVLDB **5**(12), 1700–1711 (2012). https://doi.org/10.14778/2367502.2367510
9. Hidri, M.S., Zoghlami, M.A., Ayed, R.B.: Speeding up the large-scale consensus fuzzy clustering for handling big data. Fuzzy Sets Syst. **348**, 50–74 (2018). https://doi.org/10.1016/j.fss.2017.11.003
10. Imielinski, T., Virmani, A.: MSQL: a query language for database mining. Data Min. Knowl. Discov. **3**(4), 373–408 (1999). https://doi.org/10.1023/A:1009816913055
11. Kostenetskiy, P., Semenikhina, P.: SUSU supercomputer resources for industry and fundamental science. In: 2018 Global Smart Industry Conference (GloSIC), Chelyabinsk, Russia, 13–15 November 2018, p. 8570068 (2018). https://doi.org/10.1109/GloSIC.2018.8570068
12. Ludwig, S.A.: MapReduce-based fuzzy c-means clustering algorithm: implementation and scalability. Int. J. Mach. Learn. Cybern. **6**(6), 923–934 (2015). https://doi.org/10.1007/s13042-015-0367-0
13. Mahajan, D., Kim, J.K., Sacks, J., Ardalan, A., Kumar, A., Esmaeilzadeh, H.: In-RDBMS hardware acceleration of advanced analytics. PVLDB **11**(11), 1317–1331 (2018). https://doi.org/10.14778/3236187.3236188
14. McCaffrey, J.D.: A hybrid system for analyzing very large graphs. In: 9th International Conference on Information Technology: New Generations, ITNG 2012, Las Vegas, Nevada, USA, 16–18 April 2012, pp. 253–257 (2012). https://doi.org/10.1109/ITNG.2012.43
15. Meo, R., Psaila, G., Ceri, S.: A new SQL-like operator for mining association rules. In: Proceedings of 22th International Conference on Very Large Data Bases, VLDB 1996, 3–6 September 1996, Mumbai, India, pp. 122–133 (1996). http://www.vldb.org/conf/1996/P122.PDF
16. Miniakhmetov, R., Zymbler, M.: Integration of the fuzzy c-means algorithm into PostgreSQL. Numer. Methods Program. **13**, 46–52 (2012). https://num-meth.srcc.msu.ru/english/zhurnal/tom_2012/v13r207.html
17. Ordonez, C.: Integrating k-means clustering with a relational DBMS using SQL. IEEE Trans. Knowl. Data Eng. **18**(2), 188–201 (2006) https://doi.org/10.1109/TKDE.2006.31

18. Ordonez, C.: Can we analyze big data inside a DBMS? In: Proceedings of the 16th International Workshop on Data warehousing and OLAP, DOLAP 2013, San Francisco, CA, USA, 28 October 2013, pp. 85–92 (2013). https://doi.org/10.1145/2513190.2513198
19. Ordonez, C., Pitchaimalai, S.K.: Bayesian classifiers programmed in SQL. IEEE Trans. Knowl. Data Eng. **22**(1), 139–144 (2010). https://doi.org/10.1109/TKDE.2009.127
20. Pan, C.S., Zymbler, M.L.: Taming elephants, or how to embed parallelism into PostgreSQL. In: Decker, H., Lhotská, L., Link, S., Basl, J., Tjoa, A.M. (eds.) DEXA 2013. LNCS, vol. 8055, pp. 153–164. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40285-2_15
21. Pan, C.S., Zymbler, M.L.: Very large graph partitioning by means of parallel DBMS. In: Catania, B., Guerrini, G., Pokorný, J. (eds.) ADBIS 2013. LNCS, vol. 8133, pp. 388–399. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40683-6_29
22. Pan, C.S., Zymbler, M.L.: Encapsulation of partitioned parallelism into open-source database management systems. Program. Comput. Softw. **41**(6), 350–360 (2015). https://doi.org/10.1134/S0361768815060067
23. Rechkalov, T., Zymbler, M.: An approach to data mining inside PostgreSQL based on parallel implementation of UDFs. In: Selected Papers of the XIX International Conference on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2017), Moscow, Russia, 9–13 October 2017, vol. 2022, pp. 114–121 (2017). http://ceur-ws.org/Vol-2022/paper20.pdf
24. Rechkalov, T., Zymbler, M.: Integrating DBMS and parallel data mining algorithms for modern many-core processors. In: Kalinichenko, L., Manolopoulos, Y., Malkov, O., Skvortsov, N., Stupnikov, S., Sukhomlin, V. (eds.) DAMDID/RCDL 2017. CCIS, vol. 822, pp. 230–245. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96553-6_17
25. Sattler, K., Dunemann, O.: SQL database primitives for decision tree classifiers. In: Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management, Atlanta, Georgia, USA, 5–10 November 2001, pp. 379–386 (2001). https://doi.org/10.1145/502585.502650
26. Shang, X., Sattler, K.-U., Geist, I.: SQL based frequent pattern mining with FP-growth. In: Seipel, D., Hanus, M., Geske, U., Bartenstein, O. (eds.) INAP/WLP -2004. LNCS (LNAI), vol. 3392, pp. 32–46. Springer, Heidelberg (2005). https://doi.org/10.1007/11415763_3
27. Sidló, C.I., Lukács, A.: Shaping SQL-based frequent pattern mining algorithms. In: Bonchi, F., Boulicaut, J.-F. (eds.) KDID 2005. LNCS, vol. 3933, pp. 188–201. Springer, Heidelberg (2006). https://doi.org/10.1007/11733492_11
28. Sokolinsky, L.B.: Organization of parallel query processing in multiprocessor database machines with hierarchical architecture. Program. Comput. Softw. **27**(6), 297–308 (2001). https://doi.org/10.1023/A:1012706401123
29. Sun, P., Huang, Y., Zhang, C.: Cluster-by: an efficient clustering operator in emergency management database systems. In: Gao, Y., et al. (eds.) WAIM 2013. LNCS, vol. 7901, pp. 152–164. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39527-7_17

30. Xhafa, F., Bogza, A., Caballé, S., Barolli, L.: Apache Mahout's k-Means vs Fuzzy k-Means performance evaluation. In: 2016 International Conference on Intelligent Networking and Collaborative Systems, INCoS 2016, Ostrawva, Czech Republic, 7–9 September 2016, pp. 110–116 (2016). https://doi.org/10.1109/INCoS.2016.103
31. Zymbler, M., Kumar, S., Kraeva, Y., Grents, A., Perkova, A.: Big data processing and analytics inside DBMS. In: Selected Papers of the XXI International Conference on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2019), Kazan, Russia, 15–18 October 2019, p. 21 (2019). http://ceur-ws.org/Vol-2523/invited04.pdf