An Approach to Data Mining Inside PostgreSQL Based on Parallel Implementation of UDFs

© Timofey Rechkalov © Mikhail Zymbler
South Ural State University,
Chelyabinsk, Russia
trechkalov@yandex.ru mzym@susu.ru

Abstract. Relational DBMSs remain the most popular tool for data processing. However, most of stand-alone data mining packages process flat files outside a DBMS. In-database data mining avoids export-import data/results bottleneck as opposed to use stand-alone mining packages and keeps all the benefits provided by DBMS. The paper describes an approach to data mining inside PostgreSQL based on parallel implementation of user-defined functions (UDFs) for modern Intel many-core platforms. The UDF performs a single mining task on data from the specified table and produces a resulting table. The UDF is organized as a wrapper of an appropriate mining algorithm, which is implemented in C language and is parallelized based on OpenMP technology and thread-level parallelism. The library of such UDFs supports a cache of precomputed mining structures to reduce costs of computations. We compare performance of our approach with *R* data mining package, and experiments show efficiency of the proposed approach.

Keywords: data mining, in-database analytics, PostgreSQL, thread-level parallelism, OpenMP.

1 Introduction

Currently relational DBMSs remain the most popular facility for storing, updating and querying structured data. At the same time, most of data mining algorithms suppose processing of flat file(s) outside a DBMS. However, exporting data sets and importing of mining results impede analysis of large databases outside a DBMS [18]. In addition to avoiding export-import bottleneck, an approach to data mining inside a DBMS provides many benefits for the end-user like query optimization, data consistency and security, etc.

Existing approaches to integrating data mining with relational DBMSs include special data mining languages and SQL extensions, implementation of mining algorithms in plain SQL and user-defined functions (UDFs) implemented in high-level language like C++. The latter approach could serve as a subject of applying parallel processing on modern many-core platforms.

In this paper, we present an approach to data mining inside PostgreSQL open-source DBMS exploiting capabilities of modern Intel MIC (Many Integrated Core) [2] platform. Our approach supposes a library of UDFs where each one of them performs a single mining task on data from the specified table and produces a resulting table. The UDF is organized as a wrapper of an appropriate mining algorithm, which is implemented in C language and is parallelized for Intel MIC platform by OpenMP technology and thread-level parallelism.

The paper is structured as follows. We describe the proposed approach in the Section 2. The results of experimental evaluation of our approach are given in Section 3. Section 4 briefly discusses related works. Section 5 contains summarizing comments and directions for future research.

Proceedings of the XIX International Conference "Data Analytics and Management in Data Intensive Domains" (DAMDID/RCDL'2017), Moscow, Russia, October 10–13, 2017

2 Embedding of data mining functions into PostgreSQL

2.1 Motivation example

Our approach is aimed to provide a database application programmer with the library of data mining functions, which could be run inside DBMS as it shown in Fig. 1.

```
#include <libpq-fe.h> // API of PostgreSQL
#include "pgmining.h" // API of pgMining library

void main(void)
{
  char * inpTable = "points";
  char * outTable = "clusters";
  int dimension = 3;
  int k = 5;
  float Eps = 0.1;
  char * conninfo = "user=postgres port=5432 host=localhost";

  PGconn * conn = PQconnectdb(conninfo);
  pgPAM(conn, inpTable, dimension, k, Eps, outTable);
  PQexec(conn, "SELECT * FROM clusters;");
  PQfinish(conn);
}
```

Figure 1 An example of using data mining function inside PostgreSQL

In this example the mining function performs clustering by Partitioning Around Medoids (PAM) [8] algorithm for the data points from the specified input table and saves results in output table (with respect to the specified number of the input table's columns, number of clusters and accuracy). An application programmer is not obliged to export data to be mined from DBMS and import mining results back. At the same time here PAM

encapsulates parallel implementation [24] based on OpenMP technology and thread-level parallelism.

2.2 Component structure

Fig. 2 depicts the component structure of our approach. The *pgMining* is a library of data mining functions each one of them is to be run inside PostgreSQL. The *mcMining* is a library that exports data mining functions, which are parallelized for modern many-core platforms and are subject of wrapping by the respective functions from *pgMining* library. Implementation of *pgMining* library uses PostgreSQL's SPI (Server Programming Interface), which provides low level functions for data access.

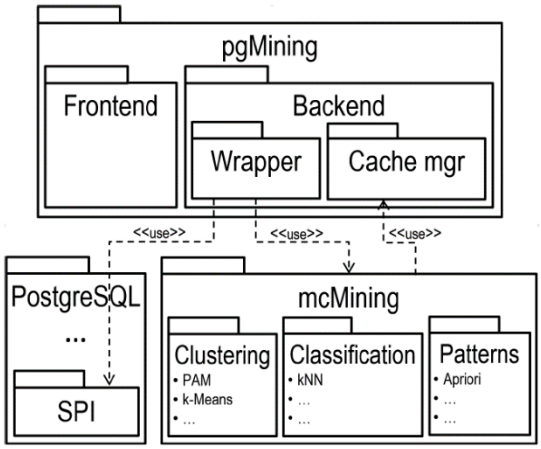


Figure 2 Component structure of the proposed approach

The *pgMining* library consists of two following subsystems, namely *Frontend* and *Backend*, where the former provides presentation layer and the latter – data access layer of concerns for an application programmer.

The *Frontend* provides a set of functions for mining inside PostgreSQL. Each function performs a single mining task (e.g. clustering, classification, search patterns, etc.) and produces a resulting table.

The *Backend* consists of two modules, namely *Wrapper* and *Cache manager*. The *Wrapper* provides functions that serve as envelopes for the respective mining functions from *mcMining* library. The *Cache manager* supports cache of precomputed mining structures to reduce costs of computations.

The *mcMining* library provides a set of functions to solve various data mining tasks in main memory and exploits capabilities of Intel many-core platforms.

2.3 Frontend

An example of *Frontend*'s function is given in Fig. 3. Such a function connects to PostgreSQL, carries out some mining task and returns exit code (0 in case of success, otherwise negative error code). As a side effect, the function creates a table with mining results. The function's mandatory parameters are ID of PostgreSQL connection, name of the input table, name of the output table and number of first left columns in input table containing data to be mined. The rest parameters are specific to the task (e.g. number of clusters, accuracy, etc.).

```
// PAM clustering inside PostgreSQL
// Returns 0 in case of success or negative error code.
int pgPAM(
  PGconn * conn,   // ID of PostgreSQL connection
  char * inpTable,// Name of input table
  int dimension,   // Number of coordinates in data point
  int k,           // Number of clusters
  float Eps,       // Accuracy of computations
  char * outTable)// Name of output table
{
  PQexec(conn, "CREATE OR REPLACE FUNCTION
    wrap_pgPAM(text, integer, integer, real) RETURNS text AS
      'pgmining', 'wrap_pgPAM' LANGUAGE C STRICT;");
  PQexec(conn, "CREATE %s TABLE IF NOT EXISTS (data text)",
    outTable);
  return PQexec(conn, "INSERT INTO %s
    SELECT wrap_pgPAM(%s, %d, %d, %f);",
      outTable, inpTable, dimension, k, Eps);
}
```

Figure 3 Interface and implementation schema of function from *Frontend*

In fact, *Frontend*'s function wraps the respective UDF from *Backend*, which is loaded into PostgreSQL and executed as "`INSERT INTO … SELECT …`" query to save mining results in the specified table.

2.4 Backend

Fig. 4 depicts an example of *Wrapper*'s function. Such a function is an UDF, which wraps a parallelized mining function from *mcMining* and performs as follows. Firstly, the function parses its input to form parameters to call *mcMining* function with. After that, the function checks if input table and/or auxiliary mining structures are in the cache maintained by *Cache manager* and then load them if not. Finally, call of *mcMining* function with appropriate parameters is performed.

```
#include "postgres.h"

// Wrapper for PAM clustering inside PostgreSQL
// Returns 0 in case of success or negative error code.
Datum wrap_pgPAM(PG_FUNCTION_ARGS)
{
  // Extract parameters of the algorithm
  char * inpTable = text_to_cstring(PG_GETARG_TEXT_P(0));
  int dimension = PG_GETARG_INT32(1);
  int k = PG_GETARG_INT32(2);
  float Eps = PG_GETARG_FLOAT4(3);
  int N;
  // Check if mining structure is in the cache
  void * distMatrix = cache_getObject(
    strcat(inpTable, "_distMatrix"));
  if (distMatrix == NULL) {
    // Check if input table is in the cache
    void * inpData = cache_getObject(inpTable);
    if (inpData == NULL) {
      // Allocate memory and load input table to cache
      inpData = (float4 *) palloc(dimension*sizeof(float4));
      wrap_tabRead(inpData, inpTable, dimension, &N);
      cache_putObject(inpTable, inpData, sizeof(inpData));
    }
    distMatrix = mcCalcMatrix(inpData, dimension, N);
    cache_putObject(strcat(inpTable, "_distMatrix"),
      distMatrix, sizeof(distMatrix));
  }
  // Perform clustering
  mcPAM_res * outData = mcPAM_resCreate();
  mcPAM(N, k, Eps, outData, distMatrix);
  // Write results to the output table
  PG_RETURN_TEXT(data2String(outData));
}
```

Figure 4 Interface and implementation schema of function from *Backend*

The *Cache manager* provides buffer pool to store precomputed mining structures. Distance matrix is a

typical example of mining structure to be saved in cache. Indeed, distance matrix $A=(a_{ij})$ stores distances between each pair of a_i and a_j elements in input data set. Being precomputed once, distance matrix could be used many times to perform clustering or kNN-based classification with various parameters (e.g. number of clusters, number of neighbors, accuracy, etc.).

```
// Load an object to cache.
// Returns 0 in case of success or negative error code.
int cache_putObject(
  char * objID, // ID of the object
  void * data, // Pointer to data buffer
  int size);   // Size of data

// Search an object with given ID in cache.
// Returns pointer to object in case of success or NULL.
void * cache_getObject(char * objID);
```

Figure 5 Interface of *Cache manager* module

The *Cache manager* exports the following two basic functions depicted in Fig. 5. The `putObject` function loads a mining structure specified by its ID, buffer pointer and size into cache. The `getObject` searches in cache for an object with the given ID. An ID of mining structure is a string, which is made as concatenation of input table's name and object's informational string (e.g. "`_distMatrix`").

2.5 Library of parallel many-core algorithms

Fig. 6 gives an example of function from *mcMining* library. Such a function encapsulates parallel implementation through OpenMP technology and thread-level parallelism for Intel many-core platforms.

```
// PAM clustering parallelized for Intel many-core platform.
// Returns 0 in case of success or negative error code.
int mcPAM(
    int N,          // Number of data points
    int k,          // Number of clusters
    float Eps,      // Accuracy of computations
    void * outData,// Array of output centroids
    void * distMatrix); // Precomputed distance matrix
```

Figure 6 Interface of function from *mcMining* library

In this example, we use Partition Around Medoids (PAM) [8] clustering algorithm, which is used in a wide spectrum of applications where minimal sensitivity to noise data is required. The PAM provides such a property since it represents cluster centers by points of input data set (*medoids*).

The PAM firstly calculates distance matrix for the given data points. Then in the BUILD phase, an initial clustering is obtained by the successive selection of medoids until the required number of clusters have been found. Next, in the SWAP phase the algorithm attempts to improve clustering in accordance with an objective function. However, for large and high-dimensional datasets PAM's computations are very costly.

In our previous research [24], we parallelize PAM for Intel Xeon CPU and Intel Xeon Phi coprocessor. In order to perform best on Intel many-core platforms the PAM's parallel version exploits modifications of loops to provide vectorization of calculations and chunk-by-chunk data processing to decrease number of cache misses.

3 Experimental evaluation

3.1 Hardware, datasets and goals of experiments

To evaluate the developed approach, we performed experiments on the Tornado SUSU supercomputer [9] whose node provides two different platforms, namely Intel Xeon CPU and Intel Xeon Phi coprocessor (cf. Tab. 1 for the specifications).

Table 1 Specifications of hardware

Specifications	CPU	Coprocessor
Model, Intel Xeon	X5680	Phi SE10X
Cores	2×6	61
Frequency, GHz	3.33	1.1
Threads per core	2	4
Peak performance, TFLOPS	0.371	1.076
Memory, Gb	24	18
Cache, Mb	12	30.5

In the experiments, we used datasets with the characteristics depicted in Tab. 2.

Table 2 Summary of datasets used in experiments

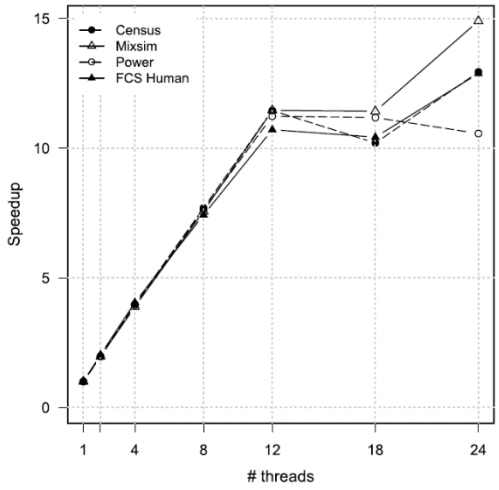
Dataset	dimension	# clusters	# data points, $\times 2^{10}$
FCS Human [3]	423	10	18
MixSim [13]	5	10	35
US Census [12]	67	10	35
Power Consumption [10]	3	10	35

In the experiments, we studied the following aspects of the developed approach. Firstly, we investigated the speedup of *mcPAM* function to understand its scalability on both platforms depending on number of threads employed. Secondly, we evaluated the runtime of *mcPAM* function to understand how the performance on both platforms depends on number of data points and what benefits could we derive from precomputations of the distance matrix. Finally, we compared the performance of *pgPAM* function with implementation of PAM algorithm from *R* data mining package [13].

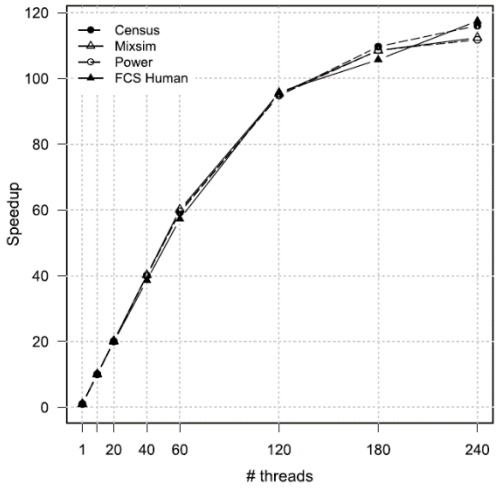
3.2 Results of experiments

The results of the first series of experiments on *mcPAM* speedup are depicted in Fig. 7. On both platforms, *mcPAM*'s speedup is close to linear, when the number of threads matches the number of physical cores the algorithm is running on (i.e. 12 cores for Intel Xeon and 60 cores for Intel Xeon Phi, respectively).

Speedup becomes sub-linear when the algorithm uses more than one thread per physical core. The *mcPAM* achieves up to 15× and 120× speedup on Intel Xeon and Intel Xeon Phi, respectively. Summing up, *mcPAM* demonstrates good scalability on both platforms.

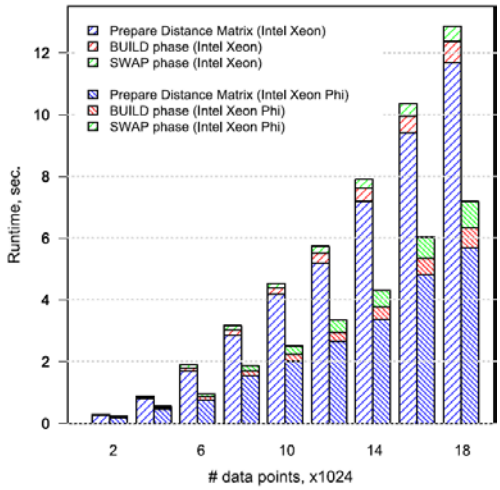


(a) Intel Xeon CPU

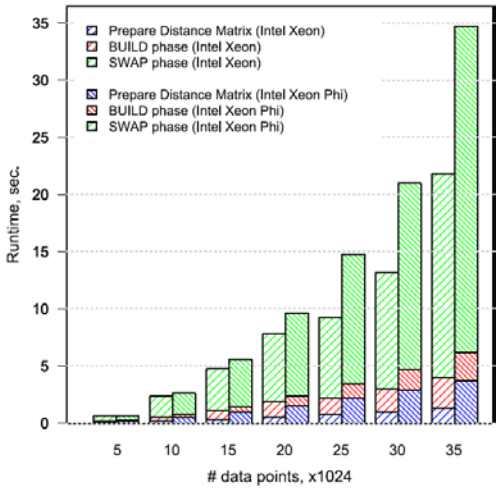


(b) Intel Xeon Phi coprocessor

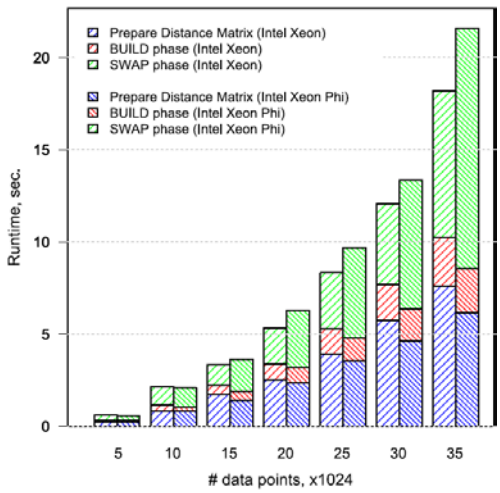
Figure 7 Speedup of the *mcPAM* function



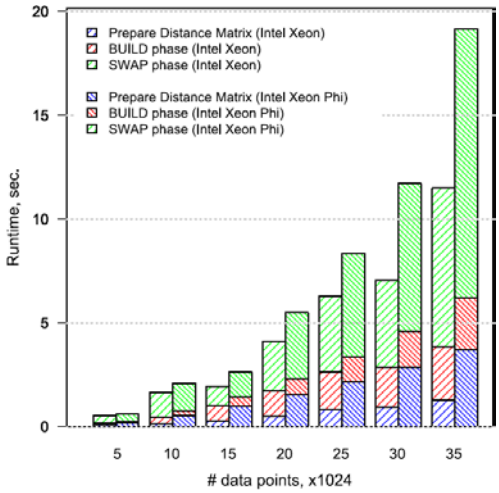
(a) FCS Human dataset



(b) MixSim dataset

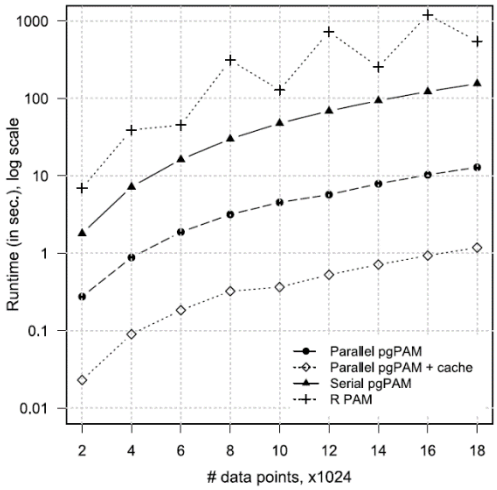


(c) US Census dataset

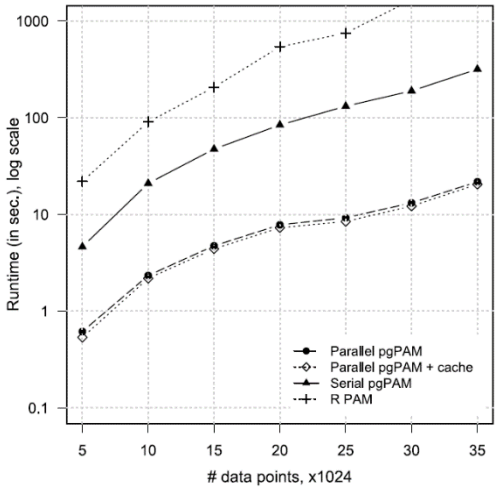


(d) Power Consumption dataset

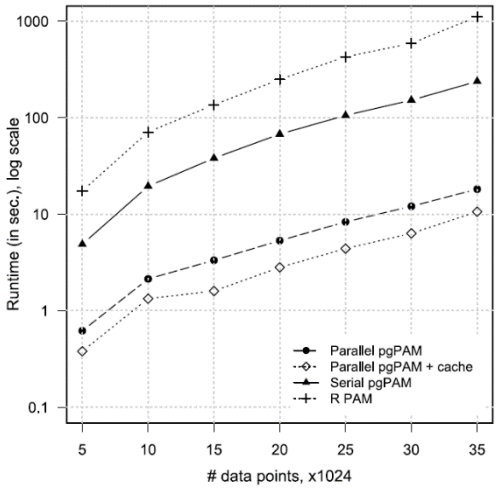
Figure 8 Performance of the *mcPAM* function



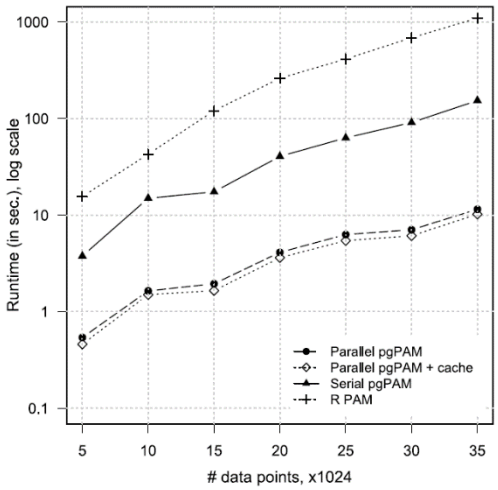
(a) FCS Human dataset



(b) MixSim dataset



(c) US Census dataset



(d) Power Consumption dataset

Figure 9 Performance of the *pgPAM* function (on Intel Xeon)

Fig. 8 shows the results of the second series of experiments on *mcPAM* performance. As was seen, PAM's SWAP phase is performed better on Intel Xeon while BUILD phase performance is equal for both platforms.

Overall performance is better on Intel Xeon Phi than Intel Xeon when the algorithm deals with big dimensionality dataset due to possibility of intensive vectorization in calculations of distance matrix. Since calculations of distance matrix take from 15 to 80 percent of overall runtime, we can derive substantial benefits from caching of the distance matrix.

The results of the third series of experiments on comparison performance of *pgPAM* and PAM from *R* data mining package are illustrated in Fig. 9. We carried out these series of experiments on Intel Xeon platform only due to the following reason. Running PostgreSQL on Intel MIC platform demands Intel Xeon Phi Knights Landing (KNL), which is the next generation product from Intel and is bootable device. However, Intel Xeon Phi KNL is not available yet at Tornado SUSU supercomputer. We plan to perform this study as further research.

We can see that *pgPAM* significantly overtakes *R*'s PAM in both cases when one thread or the maximum number of threads are employed. Caching of distance matrix improves the performance up to 80 percent of overall runtime (in case of high-dimensional dataset).

4 Related work

The problem of integrating data analytics with relational DBMSs has been studied since data mining research originates.

Data mining query languages include DMQL [5], MSQL [7], MINE RULE operator [14] and Microsoft's DMX [28].

There are many *SQL implementations* of data mining algorithms. SQL versions of classical clustering algorithms include K-Means [16], EM [19], Fuzzy C-Means [15]. SQL versions of association rule mining algorithms include K-Way-Join, Three-Way-Join,

Subquery and Two-Group-Bys [25], Set-oriented Apriori [29], Quiver [23], Propad [27]. Classification includes SQL implementations of decision trees [26], kNN [31] and Bayesian classification [21]. SQL is also successfully used in mining applications for data with "non-relational" nature as graphs, for instance in search for frequent graphs [4], detection of cycles in graph [1], graph partitioning [11, 22], etc.

User-defined functions-based approach. Integration of correlation, linear regression, PCA and clustering into the Teradata DBMS based on UDFs is proposed in [17]. There are two sets of UDFs that work in a single table scan, that is an aggregate UDF to compute summary matrices and a set of scalar UDFs to score data sets. Experiments showed that UDFs are faster than SQL queries and UDFs are more efficient than C++, due to long export times. In [20] UDFs implementing common vector operations were presented and it was shown that UDFs are as efficient as automatically generated SQL queries with arithmetic expressions and queries calling scalar UDFs are significantly more efficient than equivalent queries using SQL aggregations.

In-database mining frameworks. The ATLAS [30] is a framework for in-database analytics, which provides SQL-like database language with user-defined aggregates (UDAs) and table functions. The system's language processor translates ATLAS programs into C++ code, which is then compiled and linked with the database storage manager and user-defined external functions. Authors presented ATLAS-based implementations of several data mining algorithms.

The MADlib [6] is an open source library of in-database analytical algorithms for PostgreSQL. The MADlib is implemented by a big team and provides many methods for supervised learning, unsupervised learning and descriptive statistics. The MADlib exploits UDAs, UDFs, and a sparse matrix C library to provide efficient representations on disk and in memory. As many statistical methods are iterative (i.e. they make many passes over a data set), authors wrote a driver UDF in Python to control iteration in such a way that all large data movement is done within the database engine and its buffer pool.

Comparison. In this paper, we suggest an approach to embedding data mining functions into PostgreSQL. As some methods mentioned above our approach exploits UDFs. The difference from the previous works includes the following. Our approach supposes parallelization of UDFs for many-core platform that current DBMS is running on. All the parallelization details are encapsulated in implementation of the UDF and are hided from the DBMS, so our approach could be ported to some other open-source DBMS (with possible non-trivial but mechanical software development effort). In addition, our approach supposes a special module, which provides a cache of precomputed mining structures and lets UDF know to reuse these structures to reduce costs of computations.

5 Conclusion

In this paper, we touch upon the problem of organizing data mining inside a DBMS. We present an approach to implementation of in-database analytical functions for PostgreSQL that exploits capabilities of modern Intel many-core platforms.

Our approach supposes implementation of two libraries, namely *pgMining* and *mcMining*. The *pgMining*is a library of data mining functions each one of them is to be run inside PostgreSQL. The *mcMining* is a library that exports functions to solve various data mining tasks, which are parallelized for Intel MIC platforms.

The *pgMining* consists of *Frontend* and *Backend* subsystems. The *Frontend*'s function loads an UDF from the *Backend* into PostgreSQL and executes it as "`INSERT INTO … SELECT …`" query to save mining results in a table. The *Backend* consists of *Wrapper* and *Cache manager* modules. The *Wrapper* provides functions that serve as envelopes for the respective *mcMining* mining functions. The *Cache manager* supports cache of precomputed mining structures to reduce costs of computations.

Since our approach assumes hiding details of parallel implementation from PostgreSQL, such an approach could be ported to some other open-source DBMS (with possible non-trivial but mechanical software development effort).

We have evaluated our approach on previously implemented parallel clustering algorithm of *mcMining* library and four real datasets. Experiments showed good speedup and performance of the algorithm as well as our approach derive benefits from caching of precomputed mining structures and overtakes *R* data mining package.

As future work, we plan to implement other mining algorithms for*mcMining* library and conduct experiments on Intel Xeon Phi Knights Landing platform.

Acknowledgement

This work was financially supported by the Russian Foundation for BasicResearch (grant No. 17-07-00463), by Act 211 Government of the Russian Federation (contract No. 02.A03.21.0011) and by the Ministry of education and science of Russian Federation (government order 2.7905.2017/8.9).

References

[1] Balachandran, R., Padmanabhan, S., Chakravarthy, S.: Enhanced DB-Subdue: Supporting Subtle Aspects of Graph Mining Using a Relational Approach. In: W.K. Ng, M. Kitsuregawa, J. Li, K. Chang (eds.) Advances in Knowledge Discovery and Data Mining, 10th Pacific-Asia Conf., PAKDD 2006, Singapore, April 9–12, 2006, Proc., Lecture Notes in Computer Science, 3918,

pp. 673-678. Springer (2006). doi:10.1007/11731139 77

[2] Duran, A., Klemm, M.: The Intel Many Integrated Core Architecture. In: W.W. Smari, V. Zeljkovic (eds.) HPCS, pp. 365-366. IEEE (2012)

[3] Engreitz, J.M., Jr., B.J.D., Marshall, J.J., Altman, R.B.: Independent Component Analysis: Mining Microarray Data for Fundamental Human Gene Expression Modules. J. of Biomedical Informatics. 43 (6), pp. 932-944 (2010)

[4] Garcia, W., Ordonez, C., Zhao, K., Chen, P.: Efficient Algorithms Based on Relational Queries to Mine Frequent Graphs. In: A. Nica, A.S. Varde (eds.) Proc. of the Third Ph.D. Workshop on Information and Knowledge Management, PIKM 2010, Toronto, Ontario, Canada, October 30, 2010, pp. 17-24. ACM (2010). doi:10.1145/1871902.1871906

[5] Han, J., Fu, Y., Wang, W., Chiang, J., Gong, W., Koperski, K., Li, D., Lu, Y., Rajan, A., Stefanovic, N., Xia, B., Zaiane, O.R.: Dbminer: A System for Mining Knowledge in Large Relational Databases. In: E. Simoudis, J. Han, U.M. Fayyad (eds.) Proc. of the Second Int. Conf. on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA, pp. 250-255. AAAI Press (1996)

[6] Hellerstein, J.M., Re, C., Schoppmann, F., Wang, D.Z., Fratkin, E., Gorajek, A., Ng, K.S., Welton, C., Feng, X., Li, K., Kumar, A.: The MADlib Analytics Library or MAD Skills, the SQL. PVLDB 5(12), pp. 1700-1711 (2012)

[7] Imielinski, T., Virmani, A.: MSQL: A Query Language for Database Mining. Data Min. Knowl. Discov. 3 (4), pp. 373-408 (1999). doi: 10.1023/A:1009816913055

[8] Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley (1990)

[9] Kostenetskiy, P., Safonov, A.: SUSU Supercomputer Resources. In: L. Sokolinsky, I. Starodubov (eds.) PCT'2016, Int. Scientific Conf. on Parallel Computational Technologies, Arkhangelsk, Russia, March 29–31, 2016, pp. 561-573. CEUR Workshop Proceedings. 1576 (2016)

[10] Lichman, M.: UCI Machine Learning Repository [http://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption]. Irvine, CA: University of California, School of Information and Computer Science (2013)

[11] McCaffrey, J.D.: A Hybrid System for Analyzing Very Large Graphs. In: S. Latifi (ed.) Ninth Int. Conf. on Information Technology: New Generations, ITNG 2012, Las Vegas, Nevada, USA, 16–18 April, 2012, pp. 253-257. IEEE Computer Society (2012). doi:10.1109/ITNG.2012.43

[12] Meek, C., Thiesson, B., Heckerman, D.: The Learning-curve Sampling Method Applied to Model-based Clustering. J. of Machine Learning Research. 2, pp. 397-418 (2002)

[13] Melnykov, V., Chen, W.C., Maitra, R.: Mixsim: An R Package for Simulating Data to Study Performance of Clustering Algorithms. J. of Statistical Software, Articles 51 (12), pp. 1-25 (2012). doi:10.18637/jss.v051.i12

[14] Meo, R., Psaila, G., Ceri, S.: A New SQL-like Operator for Mining Association Rules. In: T.M. Vijayaraman, A.P. Buchmann, C. Mohan, N.L. Sarda (eds.) VLDB'96, Proc. of 22th Int. Conf. on Very Large Data Bases, September 3–6, 1996, Mumbai (Bombay), India, pp. 122-133. Morgan Kaufmann (1996)

[15] Miniakhmetov, R., Zymbler, M.: Integration of Fuzzy c-means Clustering Algorithm with PostgreSQL Database Management System. Numerical Methods and Programming 13 (2(26)), pp. 46-52 (2012) (in Russian)

[16] Ordonez, C.: Integrating k-means Clustering with a Relational DBMS Using SQL. IEEE Trans. Knowl. Data Eng. 18 (2), pp. 188-201 (2006). doi:10.1109/TKDE.2006.31

[17] Ordonez, C.: Building Statistical Models and Scoring with UDFs. In: C.Y. Chan, B.C. Ooi, A. Zhou (eds.) Proc. of the ACM SIGMOD Int. Conf. on Management of Data, Beijing, China, June 12–14, 2007, pp. 1005-1016. ACM (2007). doi:10.1145/1247480.1247599

[18] Ordonez, C.: Statistical Model Computation with UDFs. IEEE Trans. Knowl. Data Eng. 22 (12), pp. 1752-1765 (2010). doi:10.1109/TKDE.2010.44

[19] Ordonez, C., Cereghini, P.: SQLEM: Fast Clustering in SQL Using the EM Algorithm. In: W. Chen, J.F. Naughton, P.A. Bernstein (eds.) Proc. of the 2000 ACM SIGMOD Int. Conf. on Management of Data, May 16–18, 2000, Dallas, Texas, USA, pp. 559-570. ACM (2000). doi: 10.1145/342009.335468

[20] Ordonez, C., Garcia-Garcia, J.: Vector and Matrix Operations Programmed with UDFs in a Relational DBMS. In: P.S. Yu, V.J. Tsotras, E.A. Fox, B. Liu (eds.). Proc. of the 2006 ACM CIKM Int. Conf. on Information and Knowledge Management, Arlington, Virginia, USA, November 6–11, 2006, pp. 503-512. ACM (2006). doi:10.1145/1183614.1183687

[21] Ordonez, C., Pitchaimalai, S.K.: Bayesian Classifiers Programmed in SQL. IEEE Trans. Knowl. Data Eng. 22 (1), pp. 139-144 (2010). doi: 10.1109/TKDE.2009.127

[22] Pan, C., Zymbler, M.: Very Large Graph Partitioning by Means of Parallel DBMS. In: B. Catania, G. Guerrini, J. Pokorny (eds.) Advances in Databases and Information Systems – 17th East European Conf., ADBIS 2013, Genoa, Italy, September 1–4, 2013. Proc., Lecture Notes in Computer Science, 8133, pp. 388-399. Springer (2013). doi: 10.1007/978-3-642-40683-6 29

[23] Rantzau, R.: Frequent Itemset Discovery with SQL Using Universal Quantification. In: R. Meo, P.L. Lanzi, M. Klemettinen (eds.) Database Support for Data Mining Applications: Discovering Knowledge with Inductive Queries, Lecture Notes in Computer Science, 2682, pp. 194-213. Springer (2004). doi: 10.1007/ 978-3-540-44497-8 10

[24] Rechkalov, T., Zymbler, M.: Accelerating Medoids-based Clustering with the Intel Many Integrated Core Architecture. In: 9th Int. Conf. on Application of Information and Communication Technologies, AICT 2015, October 14–16, 2015, Rostov-on-Don, Russia. Proceedings, pp. 413-417 (IEEE, 2015). doi:10.1109/ICAICT.2015. 7338591

[25] Sarawagi, S., Thomas, S., Agrawal, R.: Integrating Association Rule Mining with Relational Database systems: Alternatives and Implications. Data Min. Knowl. Discov. 4 (2/3), pp. 89-125 (2000). doi:10.1023/A:1009887712954

[26] Sattler, K., Dunemann, O.: SQL Database Primitives for Decision Tree Classifiers. In: Proc. of the 2001 ACM CIKM Int. Conf. on Information and Knowledge Management, Atlanta, Georgia, USA, November 5–10, 2001, pp. 379-386. ACM (2001). doi:10.1145/502585.502650

[27] Shang, X., Sattler, K., Geist, I.: SQL Based Frequent Pattern Mining with FPGrowth. In: D. Seipel, M. Hanus, U. Geske, O. Bartenstein (eds.) Applications of Declarative Programming and Knowledge Management, 15th Int. Conf. on Applications of Declarative Programming and Knowledge Management, INAP 2004, and 18th Workshop on Logic Programming, WLP 2004, Potsdam, Germany, March 4–6, 2004, Revised Selected Papers, Lecture Notes in Computer Science, 3392, pp. 32-46. Springer (2004). doi: 10.1007/11415763 3

[28] Tang, Z., Maclennan, J., Kim, P.P.: Building Data Mining Solutions with OLE DB for DM and XML for Analysis. SIGMOD Record, 34 (2), pp. 80-85 (2005). doi:10.1145/1083784.1083805

[29] Thomas, S., Chakravarthy, S.: Performance Evaluation and Optimization of Join Queries for Association Rule Mining. In: M.K. Mohania, A.M. Tjoa (eds.) Data Warehousing and Knowledge Discovery, First Int. Conf., DaWaK '99, Florence, Italy, August 30 – September 1, 1999, Proc., Lecture Notes in Computer Science, 1676, pp. 241-250. Springer (1999). doi:10.1007/ 3-540-48298-9 26

[30] Wang, H., Zaniolo, C., Luo, C.: ATLAS: A Small but Complete SQL Extension for Data Mining and Data Streams. In: VLDB, pp. 1113-1116 (2003)

[31] Yao, B., Li, F., Kumar, P.: K Nearest Neighbor Queries and kNN-joins in Large Relational Databases (almost) for Free. In: F. Li, M.M. Moro, S. Ghandeharizadeh, J.R. Haritsa, G. Weikum, M.J. Carey, F. Casati, E.Y. Chang, I. Manolescu, S. Mehrotra, U. Dayal, V.J. Tsotras (eds.) Proc. of the 26th Int. Conf. on Data Engineering, ICDE 2010, March 1–6, 2010, Long Beach, California, USA, pp. 4-15. IEEE Computer Society (2010). doi:10.1109/ICDE.2010.5447837