EasyChair Preprint

№ 3978

New approach to MPI program execution time prediction

Andrey Chupakhin, Alexey Kolosov, Ruslan Smeliansky,
Vitaly Antonenko and Gleb Ishelev

July 30, 2020

New approach to MPI program execution time prediction

Andrey Chupakhin
Lomonosov Moscow State University
Moscow, Russian Federation
achupakhin@arccn.ru

Alexey Kolosov
Lomonosov Moscow State University
Moscow, Russian Federation
akolosov@cs.msu.ru

Ruslan Smeliansky
Applied Research Center for Computer Networks (ARCCN)
Lomonosov Moscow State University
Moscow, Russian Federation
smel@cs.msu.ru

Vitaly Antonenko
Applied Research Center for Computer Networks (ARCCN)
Lomonosov Moscow State University
Moscow, Russian Federation
anvial@lvk.cs.msu.su

Gleb Ishelev
Lomonosov Moscow State University
Moscow, Russian Federation
Gleb.Ishelev@skoltech.ru

***Abstract*—The problem of MPI programs execution time prediction on a certain set of computer installations is considered. This problem emerges with orchestration and provisioning a virtual infrastructure in a cloud computing environment over a heterogeneous network of computer installations: supercomputers or clusters of servers (e.g. mini data centers). One of the key criteria for the effectiveness of the cloud computing environment is the time staying by the program inside the environment. This time consists of the waiting time in the queue and the execution time on the selected physical computer installation, to which the computational resource of the virtual infrastructure is dynamically mapped. One of the components of this problem is the estimation of the MPI programs execution time on a certain set of computer installations. This is necessary to determine a proper choice of order and place for program execution. The article proposes two new approaches to the program execution time prediction problem. The first one is based on computer installations grouping based on the Pearson correlation coefficient. The second one is based on vector representations of computer installations and MPI programs, so-called embeddings. The embedding technique is actively used in recommendation systems, such as for goods (Amazon), for articles (Arxiv.org), for videos (YouTube, Netflix). The article shows how the embeddings technique helps to predict the execution time of a MPI program on a certain set of computer installations.**

***Index Terms*—Pearson correlation coefficient, matrix decomposition, embeddings, MPI, execution time prediction, ensemble**

I. Introduction

The idea of building a virtual infrastructure in a cloud computing environment over a heterogeneous network of computer installations (CI) is not new [1], [3]. To do this several administrative authorities called federates, allocate some their computational resources for the cloud computing environment operation. Such unions, called federations, have already been established [1], [4], some are only planning to be created [2], [5]. Each federate can consists of several CIs with different computer powers, such as high-performance computing cluster, data center or supercomputer. Federate provides resources of its computing environment [1] for creating a virtual infrastructure in accordance with the access policy set by the federate administration. Further in this article, the term program will be understood as MPI Program.

One of the main criteria for the effectiveness of such cloud computing environment is the time spending by a program in this environment. This time consists of the time spent by program in the queue (waiting time) and the program execution time (execution time). This value depends on the resources allocation algorithms of the cloud environment (CE) (mapping virtual CIs to physical ones) and the queue discipline, taking into account the heterogeneity of physical CIs.

The focus of this article is how to choose the most effective CI that is a part of the federation computing environment for the received program based on the minimum execution time criterion. The article offers algorithms for program execution time prediction on a certain set of CIs based on the execution history of the program on different CIs.

The program execution time on the specific CI, as well as the waiting time in the queue, can be predicted based on the history of its executions on this CI [14], [15]. To do this, various extrapolation algorithms can be used [16], regression [6], or more complex algorithms are used, such as the ensemble of decision trees (e.g. Random Forest). The main

This work is supported by Russian Ministry of Science and Higher Education, grant N 05.613.21.0088, unique ID RFMEFI61318X0088 and the National Key R&D Program of China (2017YFE0123600)

[1]Further the computing environment will be understood as a heterogeneous network of CIs

disadvantage of these algorithms is that they applicable only to a single certain CI. Of course, the algorithms that were mentioned above can be used to get estimates for programs execution time on the several CIs. However, this requires a history of running of this program on each CI from the set. As a rule, this information isn't available.

As noted above, in order to allocate programs between several CIs, a history of program executions on each of the CIs is required.

As demonstrated in this paper this requirement can be relaxed: to predict the program execution time on several CIs, some running histories of this program on them are sufficient. Moreover, it is not necessary that each program starts on each CI. The accuracy of the prediction depends on the number of program running histories on CIs from a certain set. To do this, the article proposes a new approaches to estimate program execution time, which allows predicting the program execution time on a certain set of CIs. The algorithms presented in this article for implementing these approaches allow one to predict the execution time of a program on a certain CI, even there is no execution history the program on it.

In the proposed approaches, the problem of the program execution time prediction is reduced to filling empty entries in the matrix "Programs-Computers" built for a given set of programs and a given set of CIs, in the entries which is an execution time a certain program with certain sets of arguments on a specific CI. For more information about the mathematical statement of this problem see Section II.

In this article proposed two approaches to the program execution time prediction problem on a certain set of CIs. The first one is based on CIs grouping based on the Pearson correlation coefficient [19]. It is shown that this method should be used for dense "Programs-Computers" matrix (at least 95% of entries are filled). The second one is based on decomposition of "Programs-Computers" matrix into vector representations of CIs and programs, so-called embeddings. Here embedding is treated as a structured type of unstructured data. For example, the form of a vector implies the structure that is important here in contrast to hashing, which simply converts a potentially infinite object into a finite one. In the paper it is shown how to use embedding of program and embedding of CI to predict the program execution time on specific CI. To calculate embeddings, it's proposed to use the technique [20] decomposition of the matrix "Programs-Computers". In Section VI it is shown by experimental studies that the second method works well with sparse matrices. Histories of runnings of MPI and OpenMP benchmarks [21], [22] on a dozens of different CIs were used for experiments. This data is described in Section IV.

It's important to emphases that, as it will be shown below, the proposed decomposition technique results embeddings of dimension 1, that allows to place a total order relation on a set of CIs and programs.

The rest part of the article is structured as follows. The Section V contains a description of the developed algorithms for programs execution time prediction. Section VII describes the application of the developed algorithms to solve the problem of "minimization of the time spent by the program inside the cloud environment". In the last Section VIII, conclusions are presented and goals for further research are formulated.

II. Problem Description

We consider the case when almost nothing is known about the program and the CI. The program source code and binary code are not available, the architecture of the CI is unknown. Only the program execution time, its arguments (see point 1 in Section 1) and the numbers of used resources of CI are known.

Further, the program will be considered in the form of job which has the following characteristics: the time when job is submitted to the queue, the start time and end time, resource utilization (cpu, ram, disk), input data, arguments of the program and arguments of the computing environment in which this job is started (e.g., the number of MPI processes, the number of threads for OpenMP, and the amount of resources needed to complete the job).

It is assumed that only the total amount of resources on the CI is known: the number of nodes/CPUs/cores, the amount of RAM and the storage space.

We will use the following notations to formulate the problem:

1) P_i – unique identifier of program;
2) $\{P_1, P_2, \ldots, P_N\} = \{P_i\}$ – a set of N unique identifiers of programs; Arg – arguments of program: program's arguments (for the executable file) and arguments of the computing environment where the program is running (the number of MPI processes, the amount of requested computing nodes /CPUs /cores);
3) $\{Arg_1, Arg_2, \ldots, Arg_{A_i}\} = \{Arg_j\}$ – here each Arg_i is the set of arguments of i-th program. A total amount of different sets of arguments – A_i;
4) $[(P_i, Arg_1), (P_i, Arg_2), \ldots, (P_i, Arg_{A_i}), (P_i, Arg_1)]$ – a history of P_i program running, arguments can be the same in some pairs;
5) C_i – unique identifier of CI;
6) $\{C_1, C_2, \ldots, C_M\} = \{C_i\}$ – a set of M unique identifiers of CIs;
7) Matrix PC (P – "program", C – "computer") – the matrix where rows correspond to history of running of programs, columns correspond to the CIs. This matrix is divided into groups of rows. There are N groups in total (by the number of the programs). The i-th group corresponds to the history of running of the i-th program. The group consists of A_i rows, where each matrix entry display the execution time of the program P_i with the corresponding arguments Arg_i on the corresponding CI. The matrix entry is empty if the program was never run on this CI.

The PC matrix shows the history of finished program executions. It can be built from log files of the CIs – $C_1, ..., C_M$.

The terms of the notations introduced above the problem of program execution time prediction on a set of CIs can be formulated as following:

- *Given*
  - A set of N unique identifiers of programs – $\{P_i\}$;
  - For each program P_i a set with A_i sets of arguments – $\{Arg_j\}$, $|\{Arg_j\}| = A_i$;
  - A set of M unique identifiers of CIs – $\{C_i\}$;
  - A PC matrix where entries contain the execution times of programs P (with the corresponding arguments – Arg) on CI C;
- *To find*
  - Fill in empty entries of the PC matrix, or in other words predict the program execution times on the CIs.

To evaluate the program execution time prediction, the prediction error is calculated as follows:

$$PredictionError(P_i, Arg_j) = \frac{|predict - target|}{target} \quad (1)$$

where *predict* is predicted time of program P_i with arguments Arg_j, *target* is a true execution time of program P_i with arguments Arg_j.

The *total prediction error* is calculated as an average of the errors calculated using the equation (1) for all programs. This allows you to compare algorithms among themselves.

III. Related Works

The problem of the execution time prediction has been studied for a very long time. For example, for real-time systems, the problem of estimation the worst-case execution time (WCET) is still one of the main problems, since WCET programs are used to build a schedule of tasks in real-time systems. In [8] a model of the program and the computer are built in order to estimate the program execution time on the computer. In [9] the execution time of programs on Grid systems are predicted. For time prediction purposes are used Analytical approaches [10], [11], statistical approaches [12], [13] approaches based on historical data [14], [15], time series prediction [16], neural networks are also used [17]. Execution time can be predicted from test runs [18].

Their main drawback of the algorithms mentioned above is that these algorithms only work when the history of program executions is known on the CI, i.e. to predict the execution time on a certain CI, you need more than one run of the program on this CI because the history of executions have to consists of at least two executions. Besides above, in order to build a program model or make test runs, you need the source code and executable files, and this information is often not available. In addition, almost all articles that consider the prediction of the program execution time often considered only for one CI. However, in the point of question of this paper it has to be able to estimate the program execution time on a set of CIs in that case when there is a minimum amount of information about CIs and programs, actually available.

IV. Data for Prediction Model

Information about programs (benchmarks) executions on many CIs from the website *spec.org* was used for the experiments presented in this article. This information was used to form the matrix "Programs-Computers" PC, described in Section II above. Three data sets with the execution results of programs on different CIs are presented in [21], [22]:

1) MPIL2007 – 12 programs, 163 CIs;
2) MPIM2007 – 13 programs, 396 CIs.
3) ACCEL_OMP – 15 programs, 25 CIs.

It should be taken into account that the data on the website *spec.org* is constantly updated. The data are used in the presented research dated *April 8, 2020 12:13*. The history of changes can be viewed in [23]. The execution results of OpenMP benchmarks running [22] were also used as a data to form the PC matrix.

The problem formulated in Section II implies that the program can be run with different arguments. However for the simplicity reason we will assume that each program was run only with one set of arguments, i.e. one program corresponds to one row in the PC matrix. Note that this assumption does not limit the generality of the proposed approaches.

Each entry in [21] and [22] was considered as a separate CI. The used data was uploaded on Github [24].

V. Proposed Solutions

To solve the problem of the programs execution time prediction (see Section II) the following algorithm was developed (for brevity, only the algorithm scheme is presented):

1) *Data preparation*. Form a PC matrix of dimension N by M;
2) *Execution time prediction*. Applying one of the algorithms described below:
   a) *Ridge*. Using ridge regression [6] for each row of the PC matrix. In other words the unknown execution time on some CI was predicted from the known execution times of the particular program on all other CIs;
   b) *Cliques*. Grouping CIs using the Pearson correlation coefficient;
   c) *Decomposition*. Apply matrix [20] or tensor decomposition [32] of the PC matrix in order to fill in empty entries of the PC matrix;
   d) Apply an ensemble of algorithms: *Ridge* + *Cliques* + *Decomposition*;
3) *Evaluation of prediction quality*. The prediction of program execution times is evaluated as follows:
   a) For each program a prediction error is calculated by the equation (1) specified in Section II;
   b) The error average is a general prediction error.

As it was mentioned above each program was run only with one set of arguments, i.e. one program corresponds to one row in the PC matrix. For more information about the case when a single program corresponds to multiple rows in the PC matrix, see Section VII.

A. Computer installations grouping based on the Pearson correlation

The Pearson correlation coefficient [19] characterizes the presence of a linear relationship between two sets of numbers. If the Pearson correlation between columns C_i and C_j is close to 1 in modulus, then there is a linear relationship between the execution times of programs on C_i and C_j CIs. Therefore, a column with program execution times on the C_i CI can be obtained by multiplying by a certain constant the column with program execution times on the C_j CI.

The grouping procedure for CIs consists of the following steps:

1) Calculate Pearson correlation for each pair of columns of the matrix PC;
2) Build the graph of CIs. Each vertex represents one CI. An edge between the C_i and C_j CIs exists if the Pearson correlation between the corresponding columns in the PC matrix is modulo greater than some threshold. The value of threshold is an algorithm parameter;
3) Find all cliques [25] in the graph of CIs. Cliques searching is NP-complete problem. To solve this problem you can use the algorithm from [26]. To speed up the experiments, a simplified cliques search algorithm was implemented [24];
4) Each clique is a group of CIs with execution times of each program are linear related, i.e. each pair of CIs in such a group has execution times related by a specific scalar coefficient. Pay attention that the same CI can fall into several groups;
5) The resulting cliques are groups of CIs.

To search for cliques, the simplified algorithm was developed with complexity no bigger than N^3 [24], which is significantly less than the complexity of the algorithm for searching for all maximum size cliques [26] – $3^{\frac{N}{3}}$. It finds at least one clique for each vertex in the graph. The disadvantage of the proposed algorithm is that it doesn't find all the maximum cliques. However, if threshold for the value of Pearson correlation is close to 1, then further prediction algorithm described below is pretty good even some vertexes in the cliques would be missed.

Based on CIs grouping, the procedure for predicting program execution time on the CI can be described as follows:

1) Select P – the program for execution time prediction;
2) Select C – the CI where P program will be run;
3) Determine the group that C belongs to;
4) If the CI doesn't belong to any group of CIs, then make a prediction using the Ridge Regression (See Section V, point 2.a). Regression is applied for the entire row in PC matrix correspond to the P program;
5) If the CI C belongs to a certain group, then there is a clique in the graph of CIs, in which all the CIs, including C, are connected to each other. This means that Pearson correlation is greater than some given threshold for each pair *(CI C; some other CI from the clique – C')*. Therefore, it's possible to calculate the coupling coefficient between the programs execution times for two CIs from a pair;
6) Then multiply the P program execution time on the CI C' on this coefficient. As result we get the estimate of P program execution time on the considered CI C;
7) This procedure should be applied to each CI from the clique, and then calculate the average of the execution times. The average time is the estimation of the P program execution time on the C;
8) If it isn't possible to calculate Pearson correlation (e.g, the considered program P hasn't been run on any of the CIs from clique), but corresponding row in PC matrix for P program is non-empty then one need to use Ridge Regression for the prediction. See *Step 4*. If the row for P program is empty then method described in Section VII is used.

The error of prediction for the algorithm presented above can be estimated by the equation (1) from Section II.

B. Matrix decomposition and Tensor decomposition

The problem of filling empty entries in recommendation systems is solved by matrix decomposition method [20]. We propose to use the matrix decomposition technique to solve the problem of programs execution time prediction.

The result of application matrix decomposition techniques to some matrix is two or more matrices, the product of which gives a matrix that approximates the original one. For empty entries of the original matrix, i.e. for unknown values, the product of the matrices gives values that estimate the unknown values.

PC matrix decomposition allows one to get a vector representations of programs and CIs, which have a remarkable property: the scalar product of the vector representation of the program and the vector representation of the CI is the program execution time on the CI. Thus, the program execution time can be divided into two components: one is related to the program itself, the other to the CI.

The vector representations of programs and CIs is embeddings of programs and CIs. Embedding techniques and methods of applying embeddings are very well known in such areas as NLP [27], topic modeling [28], recommendation systems [29].

The following matrix decomposition algorithms were used to solve the problem of programs execution time prediction: Singular Value Decomposition (SVD) [30], Alternating Least Square (ALS) [31]. It was also used the tensor decomposition algorithm [32], based on adaptation of SVD decomposition for tensors.

As a result of experimental evaluation, the ALS algorithm was selected as one with the minimum error (see equation (1) from Section II.).

C. Ensemble of algorithms

In Sections 5.A and 5.B, algorithms for the program execution time prediction on a certain set of CIs were proposed. These algorithms can be combined into an ensemble of

algorithms to improve the accuracy of the prediction [33]. In the article averaging ensemble [33] is used, where the estimation for execution time calculated for each program, as the average value of the execution time estimations of the following algorithms:

1) Ridge Regression – *Ridge*;
2) CIs grouping based on the Pearson correlation – *Cliques*;
3) Matrix decomposition by the ALS algorithm – *Decomposition*.

VI. Experiments

In this section the results of experimental studies of the algorithms proposed in Section V are presented.

The purposes of the experiments are analysis of the quality of prediction results:

1) based on grouping CIs by the Pearson correlation;
2) based on ALS matrix decomposition algorithm. Selecting the parameter K – the number of components in the vector representation of programs and CIs (see Section V.B);
3) by the ensemble of algorithms;
4) by the proposed algorithms with outliers in the source data.

Data

The data sets described in Section IV were used in experiments. These data sets were used to form the PC matrix.

The experimental technique

Implemented algorithms [24] were run with different data sets during experimentation.

A. Analysis of the quality of prediction based on grouping computer installations by Pearson correlation

Data set MPIM2007 (see Section IV) with 13 programs and 396 CIs were used for the experiments.

The algorithm based on grouping CIs by Pearson correlation is very sensitive to the presence of outliers in the data, as well as to what extend the PC matrix is low-density. In order to analyze the quality of prediction by this algorithm for each entry of the PC matrix, an execution time prediction was made using information from the remaining entries of this matrix. Thus, only one value was removed from the matrix, and then the algorithm described in Section V.A was applied. Results are presented in the Table I.

TABLE I
Results of the prediction based on grouping computer installations by Pearson correlation

Exp.	Corr.	Groups	Groups with size 1	Prediction	Average Error
1	-	-	-	Regression	0.25
2	0.97	46	27	In groups	0.068
3	0.97	46	27	In groups + Regression	0.115

3 experiments (Exp.) were made by following steps:

1) *Correlation* (Corr.) is the threshold for Pearson correlation coefficient module. It's necessary to build a graph of CIs, see Section V.A;
2) *Number of groups* (Groups) is the number of cliques in graph of CIs;
3) *Number of groups with size 1* (Groups with size 1) is the number of cliques only with one CI;
4) *Prediction* – "Regression" – prediction a value in removed entry by applying of Ridge Regression to remaining elements in row; "In groups" – prediction in groups according to the algorithm described in Section V.A, groups with only one CI aren't considered in this case; "In groups + Regression" – prediction execution time in groups with size more than 2, to other CIs used Ridge Regression;
5) *Average error* – after each removing a value from the matrix entry, a prediction is made, and the prediction error is calculated by equation (1) from Section II. After that for all entries the arithmetic mean of prediction errors for all entries of the matrix is calculated.

According to the results presented in Table I, the algorithm for programs execution time prediction based on grouping of CI by Pearson correlation gives a prediction error on dense matrices of 11.5% (only single empty entry in PC matrix). Also the results in Table I shows that the number of groups with size 1 is quite large, therefore the prediction error with and without them differs almost twice: 11.5% and 6.8% respectively.

B. Analysis of the quality of prediction based on ALS matrix decomposition algorithm

Data set MPIM2007 (see Section IV) with 13 programs and 396 CIs used for the experiments.

To study the quality of prediction based on ALS matrix decomposition algorithm 4 experiments were made: ALS matrix decomposition of PC matrix with $K = 1, 2, 3, 4$. K is ALS algorithm's parameter: the length of vector representation of program and CI. The results of the decomposition were compared with each other, as well as with Ridge regression algorithm that was chosen as the basic prediction algorithm.

The Fig. 1 contains graphs of the relationship between the prediction error and the percentage of empty entry in the PC matrix for Ridge regression and ALS algorithm with different $K = 1, 2, 3, 4$. Prediction error for the algorithms was calculated using the equation (1) from Section II. In Fig. 1 X-axis is the percentage of empty entries in the PC matrix (which randomly was removed from it), Y-axis is the prediction error. In the following figures the axises has the same meaning. Some values from X-axis were removed for readability.

According to the Fig. 1 using the matrix decomposition technique (by comparison with Ridge and Cliques) with $K = 1$ gives the best results for sparse matrices in which the number of empty entries is more than 15%. Graph for matrix decomposition with $K = 1$ is the lowest in the Fig. 1. Even if 80% of the entries are removed from the PC matrix,

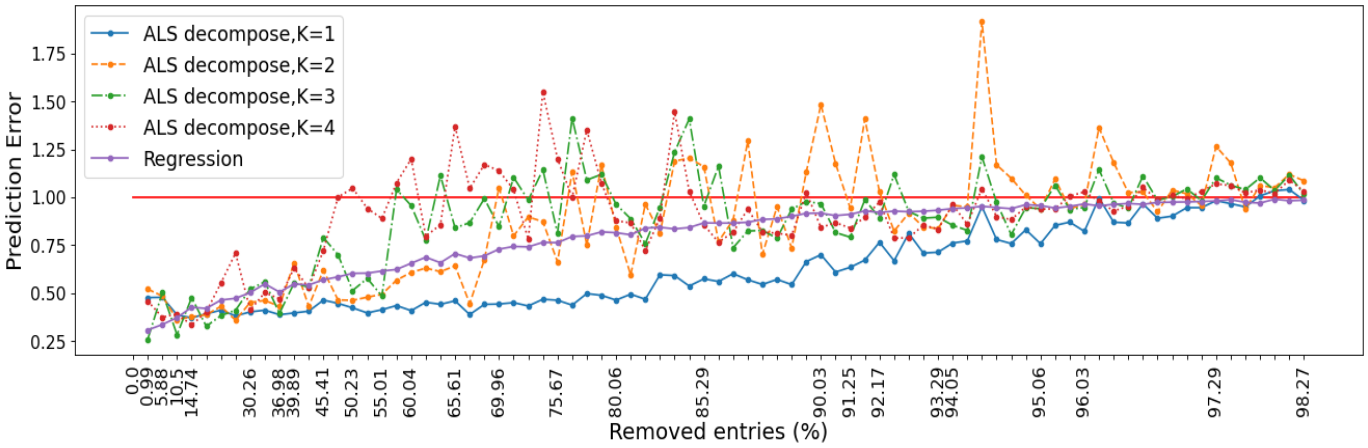

Fig. 1. Ridge regression and ALS matrix decomposition with K=1,2,3,4.

it can be filled in so that the execution time will deviate by no more than 60% from its true value.

Thus, as a result of experiments, we can conclude that the technique of matrix decomposition with the parameter $K = 1$ gives the best solutions when the percentage of empty entries in the PC matrix is more than 15%.

An important advantage of usage the matrix decomposition technique is the following. The result of applying this technique is vector representations (embeddings) of programs and CIs. These embeddings have dimension 1! This means that you can enter total ordering on the performance on a set of CIs. The less embedding a particular CI has, the less time the program will run on it. This follows from the fact that for $K = 1$, the program execution time on a specific CI is calculated as the product of two numbers: the program embedding and the embedding of the CI.

C. Analysis of the prediction quality of an ensemble of algorithms

The ensemble of algorithms is described in Section V.C. The ensemble includes the following algorithms: *Ridge*, *Cliques* and *ALS*. The simple arithmetic mean of the prediction results of the *Ridge*, *Cliques* and *ALS* algorithms was considered. All three data sets – MPIL2007, MPIM2007 and ACCEL_OMP (see Section IV) were used for the experiments.

The Fig. 2 shows that the ALS algorithm gives the best results in all cases when percentage of empty entries in PC matrix is more than 14%. The Fig. 3 shows that ensemble of algorithms gives the best results after 14% up to 93%.

To test the ensemble of algorithms another experiments was conducted on the data set ACCEL_OMP with 15 programs and 25 CIs. Due to small size of PC matrix – 15 programs, 25 CIs – different algorithms give better results for different percentage of empty entries in PC matrix. It's difficult to distinguish the dominance of any one algorithm. The Fig. 4 shows that first the best result is given by ALS (up to 19%), then ensemble (from 19% to 56%), then ALS (from 56% to 83%), then the ensemble again (from 83% to 92%).

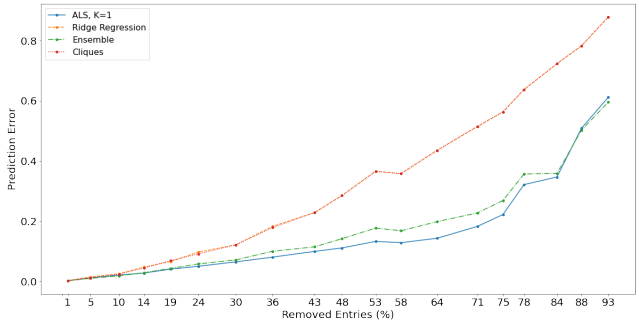

Fig. 2. Results of Ridge, Cliques, Decomposition and an ensemble of algorithms on MPIL2007 (1%-93%).

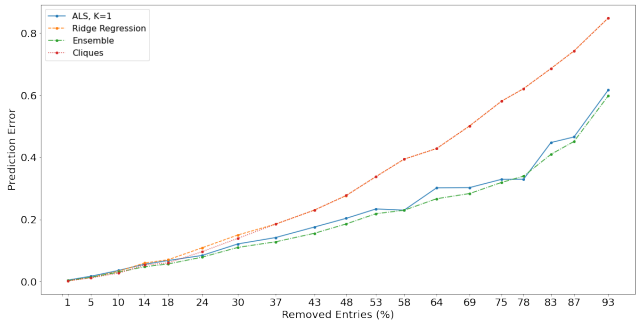

Fig. 3. Results of Ridge, Cliques, Decomposition and an ensemble of algorithms on MPIM2007 (1%-93%).

D. Analysis of the prediction quality of the proposed algorithms in the presence of outliers in the source data

One more experiment was conducted to test the stability of each algorithm to outliers. For the experiments, the PC matrix was formed, after that 10% of its entries were randomly selected and multiplied on a random number in the interval $(0, 10)$.

The Fig. 5 shows the results of a prediction based on data with outliers. As you can see from the Fig. 5 the ensemble of algorithms is almost always better than all other algorithms.

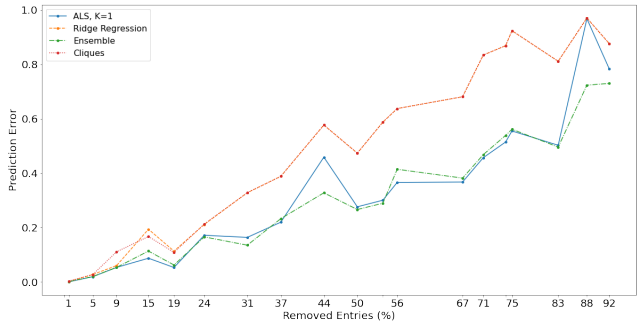

Fig. 4. Results of Ridge, Cliques, Decomposition and an ensemble of algorithms on ACCEL_OMP (1%-92%).

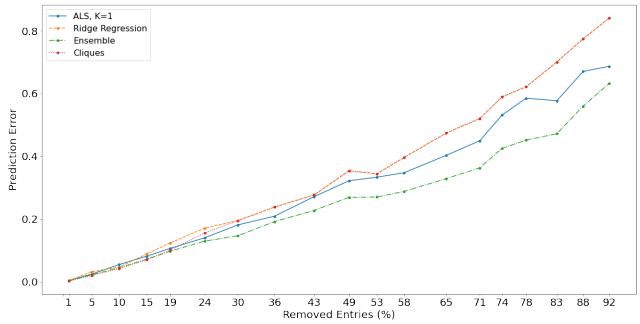

Fig. 5. Results of Ridge, Cliques, Decomposition and an ensemble of algorithms on MPIM2007 (1%-92%), 10% of outliers, coefficient from 0 to 10.

The *Cliques* algorithm also gives a good result up to 19% of the percentage of empty entries.

E. Conclusions from the experiments

Experiments showed that an ensemble of algorithms with a small percentage of empty entries in "Program-Computer" matrix makes a better prediction compared to all other algorithms. Also, the ensemble and ALS show good results even in the presence of outliers in the source data sets. Thus, for dense matrices, it is better to use an ensemble of algorithms, for sparse ones – matrix decomposition, in particular the ALS algorithm.

VII. Execution time prediction application

Based on the research presented above the following algorithm was constructed for solving the problem of efficient resource allocation in a cloud computing environment over a heterogeneous network of computers:

1) Create a PC matrix based on the history of finished program executions;
2) Fill in empty entries in PC matrix using one of the algorithms described in the article (see Section V);
3) Knowing the estimated programs execution times on CIs apply one of the algorithms below:
   a) *Greedy algorithm*. Send the current program to the CI where it has the minimum execution time. If this program hasn't been executed anywhere else, it can be placed on the most powerful CI. Performance can be determined by the embedding of the CI (see Section V);
   b) *Build a schedule*. If one plan the calculation for multiple programs, one can create a schedule for running Programs that minimizes the total execution time [7].

In the experiments presented in Section VI, each program had only one set of arguments and only one row in the PC matrix corresponded to it. Such data was considered to for simplicity of studying the quality of the created algorithms. However, the developed algorithms can be easily applied for the case when each program was run several times.

There are at least two ways to apply the described algorithms in this case. First – form a PC matrix as described in Section II, then apply the algorithms proposed in this article in Section V. The second method is as follows. First make predictions based on the history of the program executions on a single CI [14], [15]. This way, you can fill in the empty entries in the columns. Then the algorithms proposed in this article in Section V are applied to fill the remaining empty entries.

VIII. Conclusion

This article describes algorithms for predicting the programs execution time on the federation's computer installations that meet the requirements of the virtual infrastructure. Existing algorithms for predicting program execution time use the history of executions to make a prediction. The main drawback of existing algorithms is that the algorithms work only within a single computer installation, i.e. to predict the execution time on a certain computer installation, you need more than one start of the program on this computer installation – that is, you can't do without the history.

In this paper, a new approach that allows to predict the program execution time on a certain computer installation was proposed, even when it was not executed on it. Two algorithms were constructed and analyzed: an algorithm based on computer installations grouping based on the Pearson correlation coefficient (for cases when the program was executed on almost all federates in the federation) and an algorithm based on matrix decomposition technique, which allows to obtain vector representations (embeddings) of the program and computer installations (for cases when the Program was executed only on a small subset of federates). Matrix decomposition, which was used to predict the programs execution times, is actively used in the field of recommendation systems, for example, SVD decomposition [30], ALS [31] and tensor decomposition [32]. In the article, for the first time, it is proposed to use this technique for the problem of predicting the programs execution time on an set of computer installations.

In addition, an ensemble of algorithms consisting of Ridge regression, grouping computer installations based on Pearson correlation coefficient, and matrix decomposition was proposed. It is shown that the ensemble of algorithms is more

resistant to outliers than other algorithms and gives the best results on dense matrices.

Embeddings of programs and computer installations were obtained as a result of matrix decomposition of the matrix "Programs-Computers", where value in each entry is the program execution time on the computer installation. This approach to obtaining embeddings is widely used in many areas, for example, in recommendation systems [29], in thematic modeling [28]. This approach was successfully applied in solving the problem. The approach proved to be universal, as similar results were obtained for MPI benchmarks and OpenMP benchmarks (see experiments in Section VI).

It is important to emphasize that the proposed approach to predicting program execution time requires a minimal set of data about the program, which is usually available on all modern computer installations. Another important advantage of the proposed approach is that a result of matrix decomposition is the embeddings of programs and computer installations of dimension 1. This fact allows one set up the total order as on a set computer installations as on a set of programs what significantly help to properly select the computer installation with effective execution time.

The final remark is as a hypothesis we state that execution time prediction techniques proposed in this article one can apply not only to MPI programs.

REFERENCES

[1] Hwang, T. (2017, March). NSF GENI cloud enabled architecture for distributed scientific computing. In 2017 IEEE Aerospace Conference (pp. 1-8). IEEE

[2] Baldin, Ilya and Nikolich, Anita and Griffioen, James and Monga, Indermohan and Wang, Kuang-Ching and Lehman, Tom and Ruth, Paul. "FABRIC: A National-ScaleProgrammable ExperimentalNetwork Infrastructure," IEEE Internet Computing, v.23, 2020

[3] V. Antonenko, R. Smeliansky, I. Baldin, Y. Izhvanov and Y. Gugel, "Towards SDI-bases Infrastructure for supporting science in Russia," 2014 International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), Moscow, 2014, pp. 1-7, doi: 10.1109/MoNeTeC.2014.6995576.

[4] European Commission, "Federation for FIRE, projects from FP7 programme" [Online]. Available: https://cordis.europa.eu/project/rcn/105823/ [Accessed: 29-Apr-2019]

[5] Antonenko, V., Petrov, I., Smeliansky, R.L., Huang, Z., Chen, M., Cao, D., & Chen, X. (2019). MC2E: META-CLOUD COMPUTING ENVIRONMENT FOR HPC

[6] Ridge Regression[Online] Available: https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf [Accessed: 21-Jun-2020]

[7] MR Gary and DS Johnson, Computers and Intractability: A Guide to the Theory of NP-CompletenessJanuary 1979

[8] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia and A. Purkayastha, "A Framework for Performance Modeling and Prediction," SC '02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, Baltimore, MD, USA, 2002, pp. 21-21, doi: 10.1109/SC.2002.10004

[9] Nadeem, F., Alghazzawi, D., Mashat, A., Fakeeh, K., Almalaise, A., & Hagras, H. (2017). Modeling and predicting execution time of scientific workflows in the Grid using radial basis function neural network. Cluster Computing, 20(3), 2805–2819. doi:10.1007/s10586-017-1018-x

[10] Bacigalupo, D.A., et al.: An investigation into the application of different performance prediction methods to distributed enterprise applications. J. Supercomput. 34(2), 93–111 (2005)

[11] Yero, E.J.H., Henriques, M.A.A.: Contention-sensitive static performance prediction for parallel distributed applications. Perform. Eval. 63(4), 265–277 (2006)

[12] Barnes, B.J., Rountree, B., Lowenthal, D.K., Reeves, J., de Supinski, B., Schulz, M.: A regression-based approach to scalability prediction. In: Proceedings of the 22nd Annual International Conference on Supercomputing, ICS '08, pp. 368–377. ACM (2008)

[13] Wu, Q., Datla, V.V.: On performance modeling and prediction in support of scientific workflow optimization. In: Proceedings of the 2011 IEEE World Congress on Services, SERVICES '11, pp. 161– 168. IEEE Computer Society (2011)

[14] Li, H., Groep, D., Templon, J., Wolters, L.: Predicting job start times on clusters. In: CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid, pp. 301–308 (2004)

[15] Mohr, B., Wolf, F.: Kojak—a tool set for automatic performance analysis of parallel programs. In: Euro-Par 2003 Parallel Processing, pp. 1301–1304. Springer (2003)

[16] Liu, X., Chen, J., Liu, K., Yang, Y.: Forecasting duration intervals of scientific workflow activities based on time-series patterns. In: Proceedings of the IEEE Fourth International Conference on eScience, 2008, eScience '08, pp. 23–30 (2008)

[17] Lee, B.C., Brooks, D.M., de Supinski, B.R., Schulz, M., Singh, K., McKee, S.A.: Methods of inference and learning for performance modeling of parallel applications. In: Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '07, pp. 249–258. ACM (2007)

[18] Yang, L.T., Ma, X., Mueller, F.: Cross-platform performance prediction of parallel applications using partial execution. In: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, SC '05, pp. 40–44. IEEE Computer Society (2005)

[19] Pearson's Correlation Coefficient [Online] Available: https://link.springer.com/referenceworkentry/10.1007%2F978-1-4020-5614-7_2569 [Accessed: 21-Jun-2020]

[20] Cheng CM., Jin XQ. (2018) Matrix Decomposition. In: Alhajj R., Rokne J. (eds) Encyclopedia of Social Network Analysis and Mining. Springer, New York, NY

[21] MPI2007 datasets [Online] Available: https://spec.org/mpi2007/results/mpi2007.html [Accessed: 24-Jun-2020]

[22] Accel OpenMP dataset [Online] Available: https://spec.org/accel/results/accel.html [Accessed: 24-Jun-2020]

[23] Update history for MPI2007 [Online] Available: https://spec.org/mpi2007/results/ [Accessed: 24-Jun-2020]

[24] MPI program execution time prediction algorithms from the article. https://github.com/andxeg/CloudCom_2020_Embeddings

[25] Skiena, S. S. "Clique and Independent Set" and "Clique." §6.2.3 and 8.5.1 in The Algorithm Design Manual. New York: Springer-Verlag, pp. 144 and 312-314, 1997

[26] Johnston, H.C. Cliques of a graph-variations on the Bron-Kerbosch algorithm. International Journal of Computer and Information Sciences 5, 209–238 (1976). https://doi.org/10.1007/BF00991836

[27] Li Y., Yang T. (2018) Word Embedding for Understanding Natural Language: A Survey. In: Srinivasan S. (eds) Guide to Big Data Applications. Studies in Big Data, vol 26. Springer, Cham

[28] Aggarwal C.C. (2018) Matrix Factorization and Topic Modeling. In: Machine Learning for Text. Springer, Cham

[29] Recommender system. [Online] Available: https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-30164-8_705 [Accessed: 24-Jun-2020]

[30] Helmke U., Moore J.B. (1994) Singular Value Decomposition. In: Optimization and Dynamical Systems. Communications and Control Engineering. Springer, London

[31] Gábor Takács and Domonkos Tikk. 2012. Alternating least squares for personalized ranking. In Proceedings of the sixth ACM conference on Recommender systems (RecSys '12). Association for Computing Machinery, New York, NY, USA, 83–90. DOI:https://doi.org/10.1145/2365952.2365972

[32] Evgeny Frolov and Ivan Oseledets. 2016. Fifty Shades of Ratings: How to Benefit from a Negative Feedback in Top-N Recommendations Tasks. In Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16). Association for Computing Machinery, New York, NY, USA, 91–98. DOI:https://doi.org/10.1145/2959100.2959170

[33] Džeroski S., Panov P., Ženko B. (2009) Machine Learning, Ensemble Methods in. In: Meyers R. (eds) Encyclopedia of Complexity and Systems Science. Springer, New York, NY .RIS

[34] MPI benchmarks [Online] Available: https://spec.org/mpi2007/benchmarks/ [Accessed: 21-Jun-2020]