Predicting runtime of computational jobs in distributed computing environment

A G Feoktistov[1] **and O Yu Basharina**[2]

[1]Matrosov Institute for System Dynamics and Control Theory of SB RAS, Lermontov St. 134, Irkutsk, Russia, 664033
[2]Irkutsk State University, Karl Marx St. 1, Irkutsk, Russia, 664003

agf@icc.ru

Abstract. The paper addresses a relevant problem of predicting the runtime of jobs for executing problem-solving schemes of large-scale applications in a heterogeneous distributed computing environment. Such an environment includes nodes that have various hardware architectures, different system software, and diverse computational possibilities. We believe that increasing the accuracy in predicting the runtime of jobs can significantly improve the efficiency of problem-solving and rational use of resources in the heterogeneous environment. To this end, we propose new models that make it possible to take into account various estimations of the module runtime for all modules included in the problem-solving scheme. These models were developed using the special computational model of distributed applied software packages (large-scale scientific applications). In addition, we compare the prediction results (jobs runtime and their errors) using different estimations. Among them are the estimations obtained through the modules testing, user's estimations, and estimations based on computational history. These results were obtained in continuous integration, delivery, and deployment of applied and system software of a package for solving warehouse logistics problems. They show that the largest accuracy is achieved by the modules testing.

1. Introduction

Todays, scientific applications focus on carrying out large-scale scientific experiments in a heterogeneous distributed computing environment. They play a significant role in the process of solving important practical problems based on mathematical modeling of complex systems under study [1]. Often, such applications are implemented as distributed applied software packages. The environment heterogeneity means that its nodes (PCs, compute servers, HPC-clusters, and cloud resources) have various hardware architectures, different system software, and diverse computational possibilities. Various local resource managers (LRMs) are hosted in nodes of the environment.

In distributed applied software packages, problem-solving is described by schemes that specify the computing process in terms of the subject domain. We apply methods of the computation and information planning in constructing such problem-solving schemes on the special computational model [2]. Wherein, this model is a special case of the semantic network.

To execute a problem-solving scheme in the heterogeneous distributed computing environment, a computational job is generated. A job enters into the environment. The meta-scheduler selects an environment node suitable for this job. It then submits the job to LRM located on the node. The job falls into the LRM's queue. When the resources of the node are freed, the job is launched.

An improvement in the efficiency of problem-solving and rational use of resources depends a lot on the ability to estimate jobs runtime. In this regard, we propose new models for predicting jobs runtime in the environment. Unlike well-known similar models [3-5], the proposed models make it possible to take into account various estimations of the module runtime for all modules included in the problem-solving scheme. Among them estimates that are obtained on the basis of the methodology proposed by the authors.

This methodology allows us to test the program runtime. It is also used in the process of continuous integration of applied software [6].

The rest of the paper is structured as follows. In Section 2, we briefly review related works on the problem under study. Section 3 provides the models for predicting the jobs runtime. An example of applying the proposed models is considered in Section 4. Section 5 concludes the paper.

2. Related work

When starting jobs in a heterogeneous distributed computing environment, it is necessary to solve the following two problems:

- Forming a rational configuration of heterogeneous resources of the environment,
- Planning a suboptimal schedule of the job execution on the formed configuration of resources.

It's obvious that a qualitative solution to these problems for a large spectrum of practical scientific applications requires an estimation of the execution time of applied programs [7]. For example, such an estimation is used to cluster jobs in the allocation of resources to them [8]. Generally, job runtime estimates are implemented by the user or some runtime prediction procedure [9].

Most of traditional job management systems and many workflow management systems are based on the use of estimates for program execution time that are specified by the user. This is a simple and very flexible approach. However, the errors of such estimates are usually highly large in practice.

There are various methods for predicting program execution time [10]. Among them are static and dynamic methods of program analysis.

The use of methods and tools of static analysis of program code without real program execution in heterogeneous environments is characterized by high overheads to additional programming. Such overheads are owing to the need to simulate operating the processor of the target computing node for executing a large spectrum of programs written in various programming languages.

In practice, the method of frequency characteristics has proven itself well [11]. It is based on the use of special tools for dynamic analysis of programs [12, 13]. Algorithms based on the use of such analysis differ in the sets of studied software and hardware characteristics [14]. Each algorithm determines specific relations between these characteristics. In this regard, the method of profiling the program on some reference node is most applicable in practice [15]. Its results are then extrapolated relative to the characteristics of the target node. Within this method, the accuracy of estimating the program execution time largely depends on the selection of scaling coefficients for computation speedup. These coefficients are determined by the ratios of characteristics for the reference and target nodes.

However, our practical experience in applying continuous integration of applied software represented in [6] allows us to draw the following conclusion. If the target nodes are available, then testing the programs in them can simplify the program runtime prediction in comparison with the dynamic analysis.

3. Models for Predicting Computational Jobs Runtime

The computational model of the environment is determined by the structure

$$\mathbb{M} =< A, Z, F, M, S, J, R, Q, O, MH >,$$

where A, Z, F, M, S, J, R, and Q are, respectively, the sets of applied software packages, parameters, operations, program modules, problem-solving schemes, jobs, resources, and constraints to the job execution and resources use. O is the set of relations between the above-listed objects. The data structure MH represents the computational history that reflects the functioning of modules from M. If necessary, a description of components of the model $\mathbb{M}$ can be found in details in [2].

Operations from F determine computational actions on the set Z of parameters. Parameters can be represented by scalars, vectors, and matrices of various basic data types or arbitrary data structures. Each operation $f_i \in F$ is implemented by the module $m_j \in M$, where $i \in \overline{1, n_f}$, $j \in \overline{1, n_m}$, n_f is the number of operations, and n_m is the number of modules. One module can implement several operations. Each operation f_i has two subsets $Z_i^{in}, Z_i^{out} \subset Z$ of parameters. The subset Z_i^{in} consists of input parameters. Their values must be known in order to calculate values of parameters from Z_i^{out}. Parameters of the subsets Z_i^{in} and Z_i^{out} reflect the purpose and semantics of formal parameters of the module m_j that implements the operation f_i. Parameters are transferred between modules in the form of data files.

Schemes from S represent problems-solving processes in packages. The scheme $s \in S$ that performs operations from F is an analogue of the tiered-parallel form of the algorithm graph. Within the computational model under consideration, a scheme is a connected subgraph of an oriented acyclic bipartite graph. Such a graph represents schematic knowledge about algorithms for studying a subject domain. The example of such a graph is represented in Figure 1. Operations and parameters are depicted on it by filled and unfilled circles. The problem-solving scheme represented by the graph includes 3 tiers. Tier 1 contains the operation f_3. The second consists of the operations f_4, f_5, and f_6. Finally, two operations f_7 and f_8 are placed on tier 3.

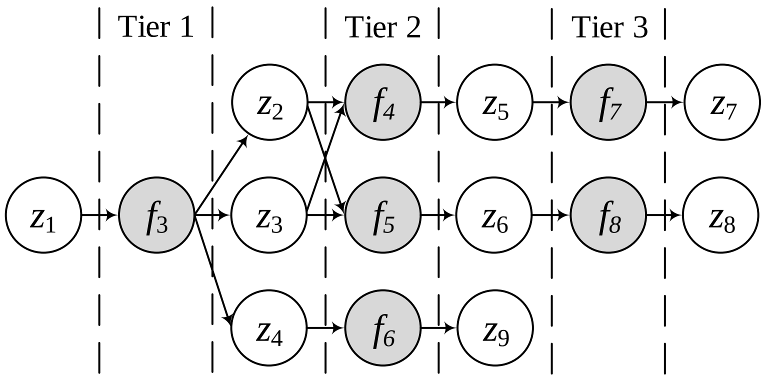

Figure 1. Bipartite directed graph of the problem-solving scheme.

Two special operations f_1 and f_2 are emphasized in the computational model. This allows us to maintain a commonality for computations planning when constructing problem-solving schemes using their formulations [16]. The operations f_1 and f_2 model the conditions of the problem. They respectively define a subset of parameters whose values are known, and a subset of the parameters whose values need to be calculated. Thus, the operation f_1 defines the subsets $Z_1^{in} = \emptyset$ and $Z_1^{out} \neq \emptyset$. The operation f_2 determines the subsets $Z_2^{in} \neq \emptyset$ and $Z_1^{out} = \emptyset$. In the above example, $Z_1^{in} = \emptyset$ and $Z_1^{out} = \{z_1\}$ (Figure 2). At the same time, $Z_2^{in} = \{z_7, z_8, z_9\}$ and $Z_2^{out} = \emptyset$.

The execution of a problem-solving scheme in the environment is specified by a computational job. A job includes a list of modules of a problem-solving scheme. In addition, it contains requirements to the environment that determine the computing resources needed to execute the listed modules. These requirements include the number of processors or cores, sizes of RAM and disk memory, execution time, etc.

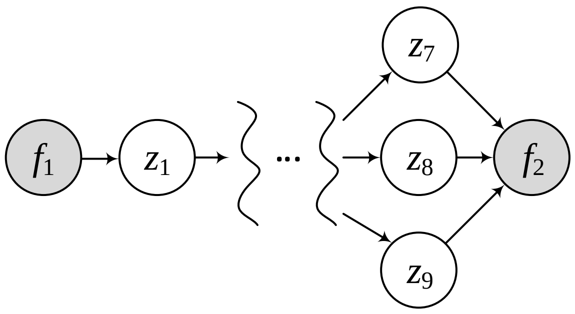

Figure 2. The operations f_1 and f_2.

We propose new models for predicting the job runtime in the heterogeneous environment. In these models, we consider cases of job execution in modes of data readiness and fork/join, resource virtualization, and job restarts.

Let the matrix **P** of dimension $k \times k$ represent information on the precedence of k operations of the problem-solving scheme. The element $p_{ij} = 1$ ($p_{ij} = 0$) of the matrix **P** means that the operation f_j in the problem-solving scheme precedes (does not precede) the operation f_i and $Z_i^{in} \cap Z_j^{out} \neq \emptyset$ ($Z_i^{in} \cap Z_j^{out} = \emptyset$). The matrix **D** of dimension $k \times k$ reflects estimates of the data volumes transmitted between operations. The matrix element $d_{ij} \geq 0$ shows the amount of data transmitted by the operation f_i to the operation f_j. The matrix **W** of dimension $k \times k$ provides information about the interconnect bandwidth between nodes where modules that implement scheme operations are launched. The matrix element $w_{ij} \geq 0$ demonstrates the interconnect bandwidth between nodes, in which the modules implementing the operations f_i and f_j operate.

The estimate E_s of the job runtime in the asynchronous mode based on data availability is defined as follows:

$$E_s = \max_{i=\overline{1,k}} e_i^\tau, \quad e_i^\tau = e_i^q + e_i + \max_{\forall j \in \overline{1,k}: p_{ij}=1, i \neq j} \left(e_j^\tau + u\right), \quad u = \frac{d_{ji}}{c_{ji}(t) w_{ji}}. \tag{1}$$

The variables in the formulas (1) are interpreted as follows:

- e_i^τ (e_j^τ) is the estimate of the period τ elapsed from the beginning of the job execution to the completion of the operation f_i (f_j),
- e_i^q is the estimate of the wait time in a queue for the module that implements the operation f_i (at a time when all the necessary data is ready to execute it),
- e_i is the predictive estimate of the module runtime,
- k is the number of scheme operations,
- $0 < c_{ji}(t) \leq 1$ is the coefficient of decrease in interconnect bandwidth between nodes, in which the modules implementing the operations f_i and f_j operate, at the time t.

Let the matrix **S** of dimension $m \times k$ be a tiered-parallel form describing the scheme execution in the fork/join mode, and m is the number of scheme tiers. The element $s_{li} = 1$ means that the operation f_i must be performed on the *l*th tier. The transition to operations of the (*l*+1)th tier is possible provided that all operations on the *l*th tier are completed. The estimate E_s' of the job runtime in the fork/join mode is defined as follows:

$$E_s' = \sum_{l=1}^{m} e_l^\tau, \quad e_l^\tau = \max_{\forall i \in \overline{1,k}: s_{li}=1} \left(e_i^q + e_i + v\right), \quad v = \sum_{\forall j \in \overline{1,k}: p_{ij}=1, i \neq j} \frac{d_{ji}}{c_{ji}(t) w_{ji}}. \tag{2}$$

We use formulas (1) and (2) to define other estimates. Among them the estimates of the job runtime with virtualized resources, module restarts, user estimates of the module runtime, and estimates adjusted based on the computational history.

In the environment with virtualized resources, the estimate E_{vs} of the job runtime in the asynchronous mode is defined as follows:

$$E_{vs} = \max_{i=\overline{1,k}} e_i^{\tau}, \quad e_i^{\tau} = e_i^q + e_i^{vml} + e_i + e_i^{vmr} + \max_{\forall j \in \overline{1,k}: p_{ij}=1, i \neq j} \left(e_j^{\tau} + u\right).$$

The variables e_i^{vml} and e_i^{vmr} are, respectively, estimates of the time it takes to launch and remove virtual machine with a module that implements the operation f_i. The estimates e_i^{vml} and e_i^{vmr} are determined experimentally for virtual machines of various configurations.

For the same environment, the estimate E'_{vs} of the job runtime in the fork/join mode is defined as follows:

$$E'_{vs} = \sum_{l=1}^{m} e_l^{\tau}, \quad e_l^{\tau} = \max_{\forall i \in \overline{1,k}: s_{li}=1} \left(e_i^q + e_i^{vml} + e_i + e_i^{vmr} + v\right).$$

In the asynchronous mode with restarting modules, the estimate E_{rs} of the job runtime is defined as follows:

$$E_{rs} = \max_{i=\overline{1,k}} e_i^{\tau}, \quad e_i^{\tau} = e_i^q + e_i^f + e_i + e_i^{run} + e_i^{res} + \max_{\forall j \in \overline{1,k}: p_{ij}=1, i \neq j} \left(e_j^{\tau} + u\right).$$

The variables in the formula above are interpreted as follows:

- e_i^f is the estimate of the time it takes for the detection and identification of a failure,
- e_i^{res} is the estimate of the time it takes for the restart of a module that implements operation f_i,
- e_i^{run} is the module runtime to failure.

These variables are required in the case of a hardware-software failure. The estimates e_i^f and e_i^{res} are determined by the average execution time of such processes by a meta-monitoring system for different types of software and hardware failures.

At the same time, in the fork/join mode with restarting modules, the estimate E'_{rs} of the job runtime is defined as follows:

$$E'_{rs} = \sum_{l=1}^{m} e_l^{\tau}, \quad e_l^{\tau} = \max_{\forall i \in \overline{1,k}: s_{li}=1} \left(e_i^q + e_i^f + e_i + e_i^{run} + e_i^{res} + v\right).$$

Let us now look at a case for applying user's estimates. In the asynchronous mode, the estimate E_{us} of the job runtime is defined as follows:

$$E_{us} = \max_{i=\overline{1,k}} e_i^{\tau}, \quad e_i^{\tau} = e_i^q + e'_i + \max_{\forall j \in \overline{1,k}: p_{ij}=1, i \neq j} \left(e_j^{\tau} + u\right),$$

where e'_i is the user’s estimate of the module runtime.

The estimate E'_{us} of the job runtime in the fork/join mode is defined as follows:

$$E'_{us} = \sum_{l=1}^{m} e_l^{\tau}, \quad e_l^{\tau} = \max_{\forall i \in \overline{1,k}: s_{li}=1} \left(e_i^q + e'_i + v\right).$$

In the environment with virtualized resources, the estimate E_{uvs} of the job runtime in the asynchronous mode is defined as follows:

$$E_{uvs} = \max_{i=\overline{1,k}} e_i^{\tau}, \quad e_i^{\tau} = e_i^q + e_i^{vml} + e'_i + e_i^{vmr} + \max_{\forall j \in \overline{1,k}: p_{ij}=1, i \neq j} \left(e_j^{\tau} + u\right).$$

For the same environment, the estimate E'_{uvs} of the job runtime in the fork/join mode is defined as follows:

$$E'_{uvs} = \sum_{l=1}^{m} e_l^{\tau}, \quad e_l^{\tau} = \max_{\forall i \in \overline{1,k}: s_{li}=1} \left(e_i^q + e_i^{vml} + e'_i + e_i^{vmr} + v\right).$$

In the asynchronous mode with restarting the modules, the estimate E_{urs} of the job runtime is defined as follows:

$$E_{urs} = \max_{i=\overline{1,k}} e_i^{\tau}, \quad e_i^{\tau} = e_i^{q} + e_i^{f} + e_i' + e_i^{res} + \max_{\forall j \in \overline{1,k}: p_{ij}=1, i \neq j} \left(e_j^{\tau} + u\right).$$

Meanwhile, the estimate E'_{urs} of the job runtime in the fork/join mode is defined as follows:

$$E'_{urs} = \sum_{l=1}^{m} e_l^{\tau}, \quad e_l^{\tau} = \max_{\forall i \in \overline{1,k}: s_{li}=1} \left(e_i^{q} + e_i^{f} + e_i' + e_i^{res} + v\right).$$

Finally, let us consider the use of estimates based on computational history. In the asynchronous mode, the estimate E_{uss} of the job runtime is defined as follows:

$$E_{uss} = \max_{i=\overline{1,k}} e_i^{\tau}, \quad e_i^{\tau} = e_i^{q} + b_i e_i'' + \max_{\forall j \in \overline{1,k}: p_{ij}=1, i \neq j} \left(e_j^{\tau} + u\right).$$

The variable e_i'' is the estimate adjusted based on the computational history. Its correction coefficient $b_i = h(i, MH, T_h) > 0$ is calculated on the basis of the computational history. The function $h(i, MH, T_h)$ calculates b_i using the average or median values from the sample time of the execution of the module that implements the operation f_i for the period T_h.

The estimate E'_{uss} of the job runtime in the fork/join mode is defined as follows:

$$E'_{uss} = \sum_{l=1}^{m} e_l^{\tau}, e_l^{\tau} = \max_{\forall i \in \overline{1,k}: s_{li}=1} \left(e_i^{q} + b_i e_i'' + v\right).$$

In the environment with virtualized resources, the estimate E_{usvs} of the job runtime in the asynchronous mode is defined as follows:

$$E_{usvs} = \max_{i=\overline{1,k}} e_i^{\tau}, e_i^{\tau} = e_i^{q} + e_i^{vml} + b_i e_i'' + e_i^{vmr} + \max_{\forall j \in \overline{1,k}: p_{ij}=1, i \neq j} \left(e_j^{\tau} + u\right).$$

For the same environment, the estimate E'_{usvs} of the job runtime in the fork/join mode is defined as follows:

$$E'_{usvs} = \sum_{l=1}^{m} e_l^{\tau}, e_l^{\tau} = \max_{\forall i \in \overline{1,k}: s_{li}=1} \left(e_i^{q} + e_i^{vml} + b_i e_i'' + e_i^{vmr} + v\right).$$

In the asynchronous mode with restarting modules, the estimate E_{usrs} of the job runtime is defined as follows:

$$E_{usrs} = \max_{i=\overline{1,k}} e_i^{\tau}, e_i^{\tau} = e_i^{q} + e_i^{f} + b_i e_i'' + e_i^{run} + e_i^{res} + \max_{\forall j \in \overline{1,k}: p_{ij}=1, i \neq j} \left(e_j^{\tau} + u\right).$$

At the same time, the estimate E'_{usrs} of the job runtime in the fork/join mode is defined as follows:

$$E'_{usrs} = \sum_{l=1}^{m} e_l^{\tau}, e_l^{\tau} = \max_{\forall i \in \overline{1,k}: s_{li}=1} \left(e_i^{q} + e_i^{f} + b_i e_i'' + e_i^{run} + e_i^{res} + v\right).$$

Estimates of the job runtime for virtual machines can be adapted to the container application case.

4. Example

As an example, we consider the problem of improving the processes of loading and unloading of goods in a warehouse through their simulation. To solve this problem, a distributed applied software package has been developed using the Orlando Tools framework [17]. This package is a parameter sweep application [18]. Simulations models (modules) are created using a special toolkit [19].

The heterogeneous distributed computing environment, in which this package is applied, was created on the basis of the resources of the public access Irkutsk Supercomputer Center SB RAS [20]. We compare the prediction results (jobs runtime and their errors) using different estimations.

The experiments were carried out in the continuous integration process of package modules. Such integration includes the modification, version control, build, testing, delivery, and deployment of applied and system software on different nodes of the heterogeneous environment. The experiments were carried out on two different clusters: PC-cluster and HPC-cluster. The characteristics of both clusters are provided in [17]. Figure 3 and Figure 4 show the actual and predicted jobs runtime on the PC-cluster and HPC-cluster, respectively. The jobs runtime are shown for different data size that is determined by the number of problem parameter variants. This time was predicted using the three methods discussed above. Obviously, the most accurate prediction in both cases was formed on the basis of the modules testing.

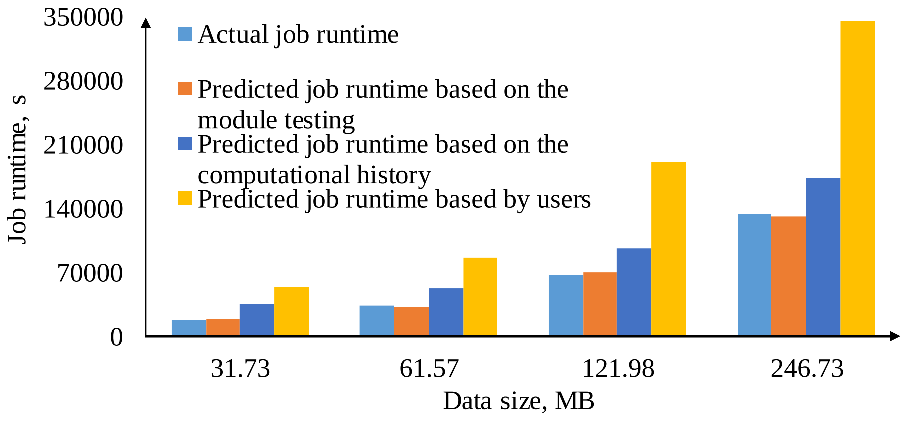

Figure 3. Predictive and actual results for PC-cluster.

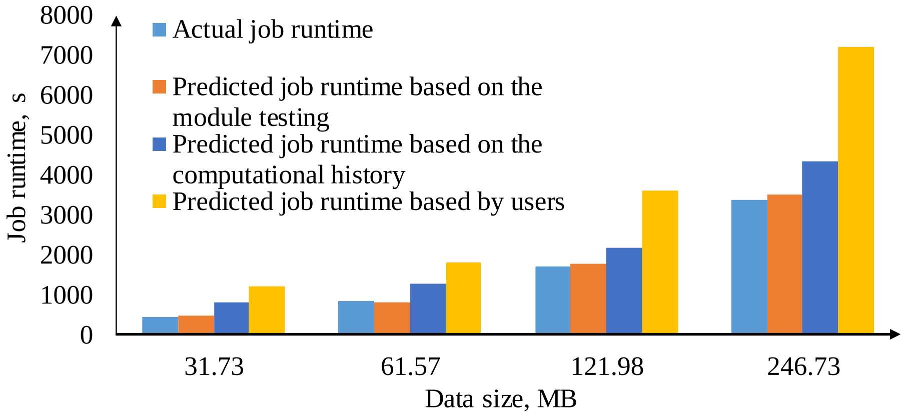

Figure 4. Predictive and actual results for HPC-cluster.

Figure 5 and Figure 6 demonstrate the runtime prediction errors obtained through the various methods for estimating the module runtime. The errors are shown in percentages relative to the actual job runtime on the PC-cluster and HPC-cluster, respectively.

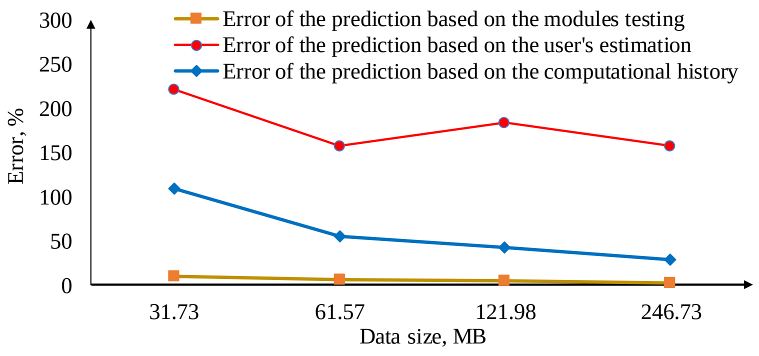

Figure 5. Runtime prediction error for the PC-cluster.

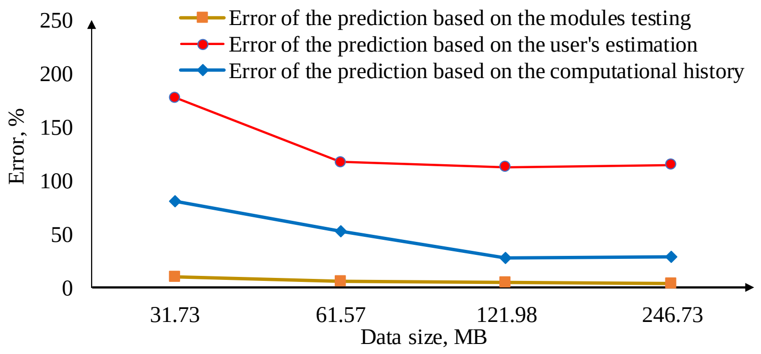

Figure 6. Runtime prediction error for the HPC-cluster.

These results show that the error in the job runtime predicted based on the module testing decreases with increasing the data size in the both cases. In the example, it does not exceed 10%. At the same time, a change of the runtime prediction errors obtained owing to user's estimates or computational history can be non-monotonous. In practice, the job runtime predicted based on a user's estimates is usually overstated. The use of computational history can slightly reduce such errors.

5. Conclusions

The rational allocation of resources in solving large problems in a heterogeneous distributed computing environment depends on the effectiveness of job maintenance schedules planned by LRMs. Usually, these schedules are based on estimates of program execution time.

In the paper, we propose new models for predicting the job runtime using a variety of estimates. Such job specifies the execution of a problem-solving scheme of a distributed applied software package (large-scale scientific application) in the environment. Within these models, we take into account job execution in modes of data readiness and fork/join, resource virtualization, and job restarts.

The practical significance of the study lies in improving the quality of planning computations and resource allocation in heterogeneous environments. Such an improvement is due to reducing the error in estimating the job runtime through the application of the proposed models.

Acknowledgments

The study is supported by the Russian Foundation of Basic Research, project no. 19-07-00097. The development of the heterogeneous distributed computing environment was supported in part by the Basic Research Program of SB RAS, project no. IV.38.1.1.

References

[1] Deelman E, Peterka T, Altintas I, Carothers C D, van Dam K K, Moreland K, Parashar M, Ramakrishnan L, Taufer M and Vetter J 2018 The future of scientific workflows *Int. J. High Perform. C.* **32(1)** 159–175

[2] Bychkov I, Oparin G, Tchernykh A, Feoktistov A, Bogdanova V and Gorsky S 2017 Conceptual Model of Problem-Oriented Heterogeneous Distributed Computing Environment with Multi-Agent Management *Procedia Comput. Sci.***103** 162–167

[3] Cao F, Zhu M M and Ding D 2013 Distributed workflow scheduling under throughput and budget constraints in grid environments *Lect. Notes Comput. Sci.* **8429** 62–80

[4] Qin J and Fahringer T 2012 Semantic-based scientific workflow composition *Scientific Workflows* (Berlin and Heidelberg: Springer) 115–134

[5] Deelman E, Vahia K, Juvea G, Ryngea M, Callaghan S Maechling P J, Mayani R, Chen W, da Silva R F, Livny M and Wenger K 2015 Pegasus, a workflow management system for science automation *Future Gener. Comp. Sy.* **46** 17–35

[6] Feoktistov A, Gorsky S, Sidorov I and Tchernykh A 2019 Continuous Integration in Distributed

Applied Software Packages *Proc. of the 42th Int. Convention on information and communication technology, electronics and microelectronics* (Riejka: IEEE) pp 1775–1780

[7] Voevodin V V 2007 The solution of large problems in distributed computational media. *Automat. Rem. Contr.+* **68(5)** 773–786

[8] Pegasus WMS – Automate, recover, and debug scientific computations. Available at: https://pegasus.isi.edu/ (accessed: 02.03.2020)

[9] da Silva R F, Juve G, Deelman E, Glatard T, Desprez F, Thain D, Tovar B and Livny M 2013 Toward fine-grained online taskcharac-teristics estimation in scientific workflows: *Proc. of the 8thWorkshop on Workflows in Support of Large-Scale Science* pp 58–67

[10] Wilhelm R et al. 2008 The worst-case execution-time problem – overview of methods and survey of tools *ACM T. Embed. Comput. S.* **7(3)** 1–52

[11] Wang W, Wang W, Guan X, Zhang X and Yang L 2006 Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data *Comput. Secur.* **25(7)** 539–550

[12] Intel® VTune™ Profiler (formerly Intel® VTune™ Amplifier). Available at: https://software.intel.com/en-us/vtune/ (accessed: 02.03.2020)

[13] OProfile – A System Profiler for Linux. Available at: https://oprofile.sourceforge.io/news/ (accessed: 02.03.2020)

[14] Adhianto L, Banerjee S, Fagan M, Krentel M, Marin G, Mellor-Crummey J and Tallent N R 2010 HPCToolkit: Tools for performance analysis of optimized parallel programs *Concurr. Comp.-Pract. E.* **22(6)** 685–701

[15] Ivannikov V P, Gaisaryan S S, Avetisyan A I and Padaryan V A 2006 Estimation of dynamical characteristics of a parallel program on a model *Program. Comput. Soft.+* **32(4)** 203–214

[16] Novopashin A P and Oparin G A 2004 Boolean Modeling of Action Planning in Distributed Computing Systems *J. Comput. Sys. Sc. Int.+* **43(5)** 763–766

[17] Tchernykh A, Feoktistov A, Gorsky S, Sidorov I, Kostromin R, Bychkov I, Basharina O, Alexandrov A and Rivera-Rodriguez R 2019 Orlando Tools: Development, Training, and Use of Scalable Applications in Heterogeneous Distributed Computing Environments *Comm. Com. Inf. Sc.* **979** 265–279

[18] Bychkov I, Oparin G, Tchernykh A, Feoktistov A, Bogdanova V, Dyadkin Yu, Andrukhova V and Basharina O 2017 Simulation Modeling in Heterogeneous Distributed Computing Environments to Support Decisions Making in Warehouse Logistics *Procedia Eng.* **201** 524–533

[19] Feoktistov A G, Kostromin R O, Fereferov E S, Tchernykh A, Basharina O Yu, Dmitriev V I and Kurzybova Ya V 2019 Toolkit for simulation modeling of queue systems in Grid *Proc. of the 1st International Workshop on Information, Computation, and Control Systems for Distributed Environments (ICCS-DE)* (CEUR-WS Proceedings) **2430** pp 51–59

[20] Public access center Irkutsk Supercomputer Center. Available at: http://hpc.icc.ru (accessed: 02.03.2020)