Static-dynamic algorithm for managing asynchronous computations in distributed environments

S A Gorsky[1] and A G Feoktistov[1]

[1]Matrosov Institute for System Dynamics and Control Theory of SB RAS, Lermontov St. 134, Irkutsk, Russia, 664033

gorsky@icc.ru

Abstract. The paper addresses a relevant problem of computation scheduling in scientific applications (distributed applied software packages) executed in distributed environments. Forming an optimal schedule of jobs for executing of applied software (modules) is an NP-hard problem. Therefore, in practice, heuristic methods of scheduling are often used. In this regard, we propose a new static-dynamic algorithm for managing computations in heterogeneous distributed environments. The results of operating the proposed algorithm are simulated in comparison with other scenarios for computing management. They show that applying the algorithm makes it possible to achieve a rational balance between the scheduling time and the computations makespan.

1. Introduction

Applications developed for supporting scientific workflows include a variety of software that implements data processing and analysis, different computational methods, and processes for modeling the systems under study. Typically, such software is based on a modular approach and oriented to high-performance computing [1]. Inputs and outputs of modules determine relations between them. In this case, the computational job describes executed module, its inputs and outputs, features of launching the module, and requirements to a computing environment. Thus, a workflow consists of a set of interconnected jobs. Additional job specification includes information about relations between parent and child jobs.

When a workflow is executed in a parallel or distributed computing environment of the public access, non-trivial problems arise in determining the order of workflow job executions on limited resources of the environment. In general, the formation of the optimal schedule of jobs is NP-hard [2].

We focus on asynchronous computations. Within the framework of such computations, workflow modules are launched when their input data is ready.

In the paper, we consider execution workflows under uncertainties in distributed applied software packages that relate to a special class of scalable scientific applications [3]. The presence of uncertainties determines the use of heuristic approaches to scheduling computations.

Such heuristics should rely on known relations between modules. In addition, considering data placement when executing workflow modules is also essential.

2. Related work

Developers of scientific applications distinguish between two types of workflows: abstract and concrete [4]. Jobs for executing workflows of both types consist of sets of interrelated sub-jobs describing

executable programs (modules) and requirements (number of processors or cores, sizes of RAM and disk memory, etc.) to resources of a computing environment.

In the paper, we consider abstract workflows, jobs and data of which are not related to specific resources of computational environments. In general, such environments can integrate public access resources with additional resources from various grid and cloud platforms.

Well-known Workflow Management Systems (WMSs) implement both the resource allocation and jobs execution of abstract workflows on dedicated resources [5-7]. Often, an abstract workflow is presented in the form of a Directed Acyclic Graph (DAG), which reflects relations between modules or jobs of a workflow [8]. Therefore, many WMSs (for example, Pegasus [9]) use popular tools such as Condor DAGMan [10] or Gridway [11] that support managing similar relation description structures of computing tasks. Often, Condor DAGMan is independently applied for managing workflows.

Within the framework of our study, we assume the presence of uncertainties in the job execution time and the amount of transferred data. In this case, when selecting resources, such tools apply simple heuristics, such, for example, as the allocation of the first available node with suitable characteristics or the node with the least number of jobs [12].

Another popular approach is to split the workflow into a set of micro-workflows, each of which runs on a separate resource [13]. However, such a decomposition may not always be implementable for an arbitrary workflow owing to the possible lack of the necessary structural relations between workflow jobs [14].

In this regard, we propose a new heuristic, which in some cases allows us to improve the allocation of resources by taking into account the relations between jobs in comparison with the Condor DAGMan scheduler. When using this heuristic, the computation planning time is close to the time spent by the compared tool.

3. Static-dynamic algorithm

We have developed an algorithm that includes static and dynamic stages of operation. In the static stage, we assign priorities to workflow jobs according to our proposed heuristic. These priorities can then change dynamically in accordance with the data readiness for the execution of modules.

A. Model Workflow

As an example, let us consider the workflow represented in Figure 1 by a bipartite DAG, which includes two disjoint sets of vertices (a set of the parameters $d_1, d_2, \ldots, d_{11}$ and a set of the modules $m_1, m_2, \ldots, m_{10}$). Arcs reflect the data transfer between modules. Each ith module has a one-to-one correspondence with the ith job.

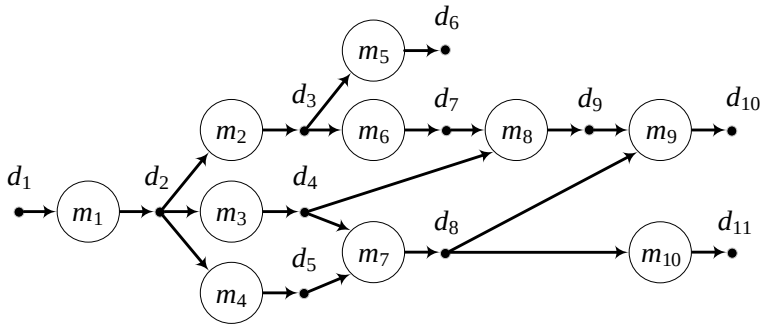

Figure 1. Workflow.

The initial data are represented by the parameter d_1. d_6, d_{10}, and d_{11} are target parameters, whose values are to be calculated. We can see that the workflow is weakly structured in terms of the decomposition into a set of micro-workflows.

B. Heuristic

The proposed heuristic is based on the analysis of causal relations between modules in the static stage of the algorithm operation. The execution of each module causes the further execution of a certain number of modules and the implementation of a certain number of parameter transfers. The analysis is carried out in order to determine the priorities of jobs for the execution of modules.

Assignment of jobs priorities begins with modules that no other module depends on. We apply a lexicographical rule for multi-criteria assigning the job priorities using the following four characteristics with their optimality conditions [15]:

- Number $n_a \to max$ of modules launch in the future caused by the module execution,
- Number $n_{td} \to max$ of parameters, whose transfers in the future are caused by the module execution,
- Sum $n_p \to max$ of priorities of modules, whose launches in the future are caused by the module execution,
- Number $n_{rd} \to min$ of parameters that are received by a module.

In comparison with the lexicographical method mentioned in [15], we use estimates for values of characteristics instead of the values themselves. To obtain such estimates, we apply the original algorithm [16].

The results of assigning priorities to the jobs of the workflow shown in Figure 1 are represented in Table I. The lexicographical rule applied three times to the following groups of modules: $\{m_5, m_9, m_{10}\}$, $\{m_6, m_7\}$, and $\{m_2, m_3\}$.

Table 1. Priorities of jobs.

Module	n_a	n_{td}	n_p	n_{rd}	Job	Job priority
m_1	9	12	45	1	j_1	10
m_2	4	4	14	1	j_2	9
m_3	4	4	12	1	j_3	8
m_4	3	3	8	1	j_4	7
m_5	0	0	0	2	j_5	3
m_6	2	2	5	1	j_6	6
m_7	2	2	3	2	j_7	5
m_8	1	1	1	2	j_8	4
m_9	0	0	0	7	j_9	1
m_{10}	0	0	0	4	j_{10}	2

C. Algorithms for Dynamic Stage

The general scheme for job scheduling is represented by the main algorithm, which is realized using the function *JobScheduling*(). This scheme includes the following operations:

- Finding available resources that can be allocated,
- Determining jobs ready to be run, whose inputs have already been calculated,
- Resources allocation for the ready jobs,
- Preparing data transfer for allocated resources.

These operations are correspondingly implemented by the functions *ResourceMonitoring*(), *JobMonitoring*(), *ResourceAllocation*(), and *DataTransferPreparation*(). In addition, we use auxiliary functions *GetDedicatedResources*(), *ResourseAvailable*(), *JobReady*(), *CheckPriority*(),

PrioritiesChanging(), *ReadyJobAllocation*(), and *JobAllocationPrediction*() which are not discussed in detail in the paper.

The simplified algorithms 1-5 for the key functions for monitoring resources and jobs, data transfer preparation, resource allocation, and job scheduling are given bellow. A monitoring system is represented in [16]. Data transferring preparation for modules is carried out only when they are guaranteed to be launched on the allocated resources in accordance with the priorities of their jobs.

Variables used in the aforementioned algorithms have the following interpretation:

- DAG *wf*,
- Integer vector $r = (r_1, r_2, \ldots, r_k)$ of dedicated resources, where k is a number of resources,
- Boolean vector $v = (v_1, v_2, \ldots, v_k)$ of resource states, where $v_i = 1$ ($v_i = 0$) shows that the ith resource is available (not available),
- Boolean vector $w = (w_1, w_2, \ldots, w_l)$ of job states, where $w_i = 1$ ($w_i = 0$) shows that the ith job is ready (not ready),
- Real vector $p = (p_1, p_2, \ldots, p_l)$ of job priorities,
- Boolean matrix A of the dimensions $l \times k$ shows the result of resource allocation for the ready jobs, where $a_{ij} = 1$ ($a_{ij} = 0$) means that the jth resource is allocated (not allocated) to the ith resource.
- Boolean matrix X of the dimensions $m \times k$ defines the placement of parameters on resources, where $x_{ij} = 1$ ($x_{ij} = 0$) shows that the ith parameter is placed (not placed) on the jth resource, where m is a number of parameters.
- Boolean matrix Y of the dimensions $m \times k$ reflects parameter transfers, where $y_{ij} = 1$ ($y_{ij} = 0$) shows that the ith parameter will be transferred (will not be transferred) to the jth resource.

The algorithms considered above form the basis of the proposed static-dynamic algorithm. This algorithm is implemented in the Orlando Tools framework used to develop distributed applied software packages [17].

Algorithm 1 for resource monitoring

```
1  function ResourceMonitoring()
2  // Getting a list of dedicated resources
3    r ← GetDedicatedResources()
4  // Initializing resource states
5    v ← (0, 0, …, 0)
6  // Determining the current state of a resource
7    for i = 1..k do
8      if ResourseAvailable(r_i) = 1 then
9        v_i = 1
10     else
11       v_i = 0
12     end if
13   end do
14   return v
15 end function
```

Algorithm 2 for job monitoring

```
1  function JobMonitoring()
2  // Initializing job states
3    w ← (0, 0, …, 0)
4  // Determining the job states (ready or not
   ready)
5    for i = 1..l do
6      if JobReady(i) = 1 then
7        w_i = 1
8      end if
9    end do
10   return w
end function
```

Algorithm 3 for data transferring preparation

```
1  function DataTransferPreparation(A, X)
2  // Initializing data transferring
3    Y ← (0, 0, …, 0; 0, 0, …, 0; …; 0, 0, …, 0)
4   for i = 1..l do
5    for j = 1..k do
6  // Analyzing the jth resource allocation for the
   ith module
7      if a_ij = 1 then
8  // Getting inputs of the ith module
9       b ← GetInputs(i)
10      for q = 1..m do
11 // Checking the qth parameter placement on
   the jth resource
12        if b_q = 1 ∧ x_qj = 0 then
13 // Preparing the qth parameter transfer on the
   jth resource
14          y_qi = 1
15        end if
16       end do
17     end if
18    end do
19  end do
20  return Y
21 end function
```

Algorithm 4 for resource allocation

```
1  function ResourceAllocation(wf, v, w, p, X)
2  // Initializing resources allocation
3    A ← (0, 0, …, 0; 0, 0, …, 0; …; 0, 0, …, 0)
4  //  If the priority of the ready job is less than
5  // the priorities of the unready jobs,
6  // then changing the priorities
7    for i = 1..l do
8     if CheckPriority(wf, i, p) then
9       p ← PrioritiesChanging(w, p)
10    end if
11   end do
12 //  Allocation of the ready jobs
13   A ← ReadyJobAllocation(wf, v, w, p, A)
14 // Predicting the allocation for the unready jobs
15 // taking into account the job priorities and
16 // data placement
17   A ← JobAllocationPrediction(wf, v, w, p, A, X)
18   return (p, A)
19 end function
```

Algorithm 5 for job scheduling

```
1  function JobScheduling( wf, p, X)
2  //  Getting current states of resources and jobs
3    v ← ResourceMonitoring()
4    w ← JobMonitoring()
5  //  Checking  available resources and ready jobs
6  if (v1 ∨ v2 ∨…∨ vk = 1) ∧ (w1 ∨ w2 ∨…∨ wl = 1)
   then
7  // Resources allocation
8       (p, A) ← ResourceAllocation(wf, v, w, p, X)
9  end if
10 // Data preparation
11   Y ← DataTransferPreparation(A, X)
12   for i = 1..m do
13     for j = 1..k do
14       x_ij = x_ij ∨ y_ij
15     end do
16   end do
17   return (p, A, X, Y)
18 end function
```

D. Experiment

In the first example, let us suppose that we have to execute flow of workflows in a computing environment consists of a set of Virtual Machines (VMs). Two VM with one core is allocated to running one workflow. All workflows have the structure shown in Figure 1.

This job flow implements parameters sweep computations [18]. Each job is executed with a unique variant of the source data. Therefore, the size of the data, their transfer time, and the execution time may differ for different jobs.

To demonstrate the result in operating our algorithms, we will consider a simple example with model time in applying to one workflow. Let there be the following execution times of jobs $j_1, j_2, \ldots, j_{10}$: $t_1 = 2$, $t_2 = 2$, $t_3 = 3$, $t_4 = 2$, $t_5 = 1$, $t_6 = 2$, $t_7 = 2$, $t_8 = 5$, $t_9 = 2$, and $t_{10} = 1$. The transferring time for any data (d_1, d_2, …, d_{11}) is assumed to be equal to 1. These times are proportional to the average computation results when solving a real problem in the public access Irkutsk Supercomputer Center [19].

Figure 2 and Figure 3 show the job scheduling by the Condor DAGMan algorithm and proposed static-dynamic algorithm. Optimal workflow makespan is equalled to 16. The computation time with Condor DAGMan is 19. The static-dynamic algorithm reduces this time to 17. This is a significant advantage for such an example.

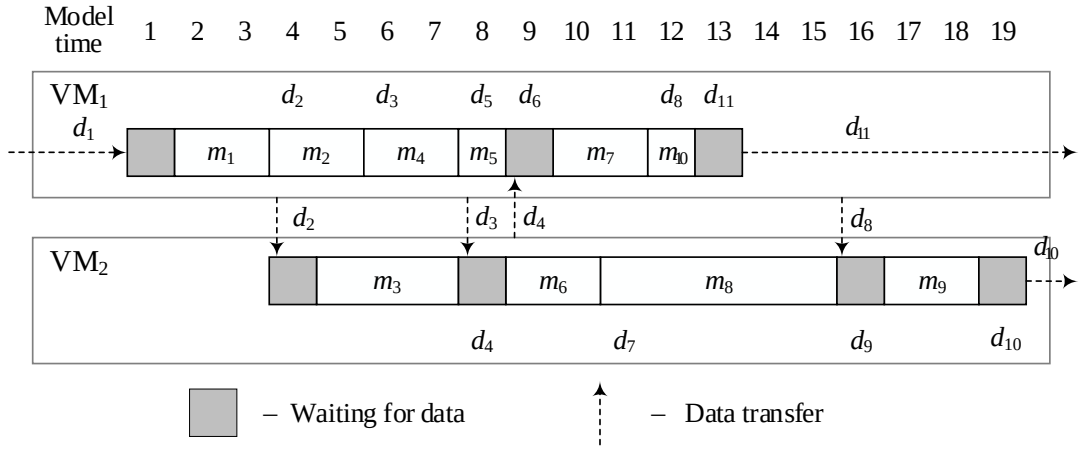

Figure 2. Job scheduling via the Condor DAGMan algorithm.

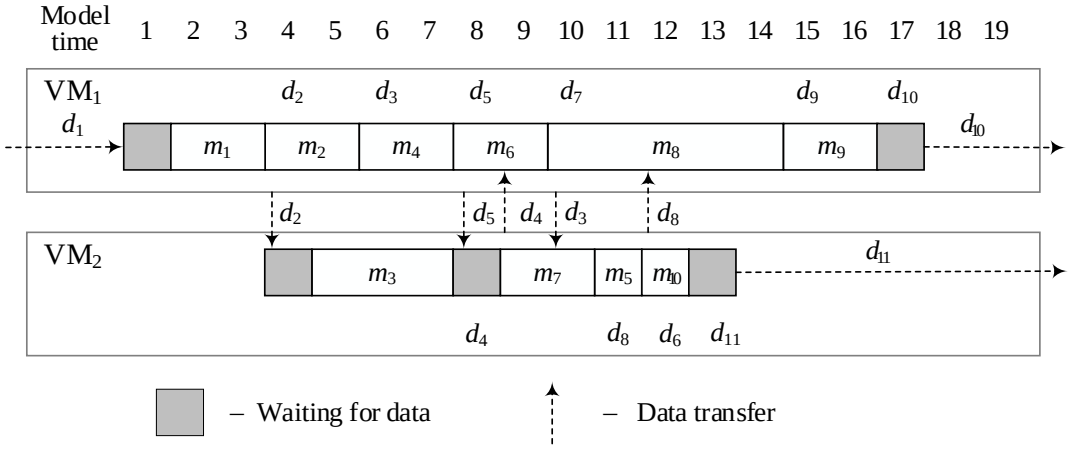

Figure 3. Job scheduling via the proposed algorithm.

We can see that both algorithms start out the same way. The m_1 modules m_1, m_2, and m_4 are consistently launched on VM_1 in accordance with inter-module relations. In parallel, data is being prepared on VM_2 for executing the module m_3.

After that, the operation of the algorithms begins to differ. The Condor DAGMan algorithm launches the m_5 module for execution on VM_1. At the same time, our algorithm selects the m_6 module.

According to the heuristic used by our algorithm, module m_6 has a higher priority than module m_5. Really, more number of modules depends on the m_6 module in further computing within the workflow compared to the m_5 module.

Owing to launching the module m_5, the Condor DAGMan algorithm is forced to delay the execution of the m_6 module because of the need in preparing its input data on VM_2.

In the given example, the reduction of the workflow runtime within our algorithm is also due to the preliminary transfer of the input data for the modules m_8 and m_9. Figures 4 and 5 show these transfers at the corresponding times.

The transfers of the parameters d_4 and d_8 are performed although the modules related to them are not yet ready to run. The parameter d_4 for the module m_8 is transmitted while the module m_6 is being executed (Figure 4). This is due to the fact that the module m_8 will be launched on the VM where the module m_6 is executing.

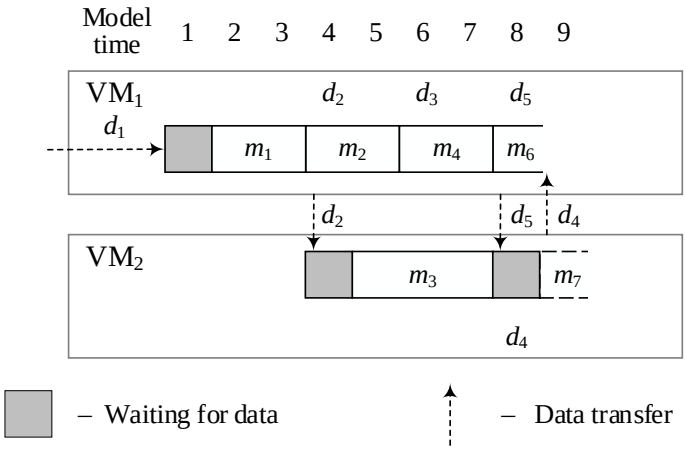

Figure 4. Preliminary data transfer 1.

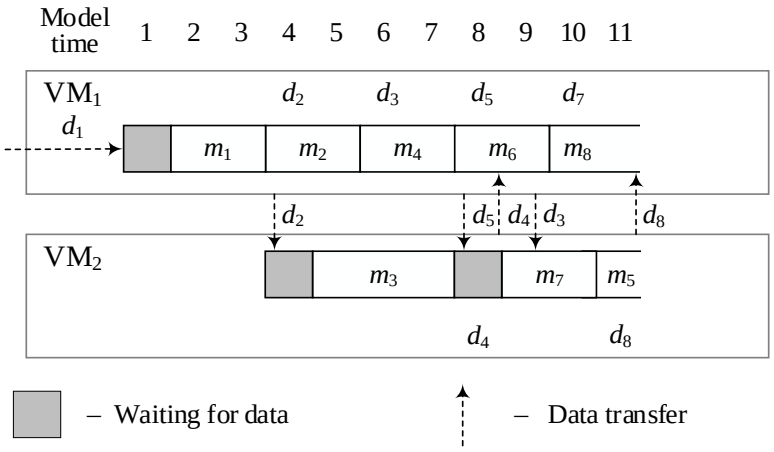

Figure 5. Preliminary data transfer 2.

The decision to transfer the parameter d_8 (Figure 5) is also obvious since the decision to launch the m_{10} module has already been made. In this case, it remains to run the m_9 module. The module m_8 will calculate the parameter d_9 required for the module m_9.

Let us consider the issue of selecting a module for its launch on a resource that is being freed. We have implemented the *JobAllocationPrediction*() function, which selects the module to run on the resource when it becomes free. This check is performed for all resources on which workflow modules are running. Within the check, a set of modules including modules that are not ready to run can exist (for example, by time 8 in Figure 5).

The *JobAllocationPrediction*() function checks how the completion of the module executing and resource freeing will affect the readiness of the remaining unexecuted modules. Moreover, it allows us to assign modules to resources that are not yet ready for launch. This assignment of modules to resources allows us to prepare the necessary input data to the time of their launch.

Within the considered example, there is another background data transfer. The parameter d_3 is transferred to the module m_5. However, this transfer does not affect the workflow runtime.

In the theory, the preparation of the input data for the modules could be performed through overabundant sending the calculated parameters to all available computing resources. However, this solution has significant drawbacks. In practice, data transfers can degrade compute performance and slow down other transfers of the current data. Typically, the negative impact of data over-sending increases with the number of resources used and the number of modules executed. Thus, the overabundant data sending is highly undesirable for scalable workflows focused on a large number of resources.

For the job flow in the whole, the proposed algorithm reduces the total computation time by more than 11%, taking into account the change in the module execution times in different jobs. At the same time, the time spent on job scheduling by the proposed algorithm does not exceed the time of the Condor DAGMan operating.

The next example addresses a workflow with the parallel list of data (Figure 6). This is an important feature of the workflow.

Let there be the following execution times of jobs j_1, j_2, and j_3 for executing the modules m_1, m_2, and m_3: $t_1 = 1$, $t_2 = 2$, and $t_3 = 1$. The transferring time for any data (d_1, d_2, d_3, d_4) in Figure 6 is assumed to be equal to 1.

d_2 and d_3 are the parallel lists of data. Each element d_{2i} of the parallel list d_2 is processed by the ith instance of the module m_2 (Figure 70. The result of the execution of m_{2i} is saved in the element d_{3i} of the parallel data list d_3.

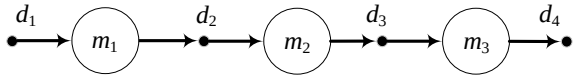

Figure 6. Initial workflow.

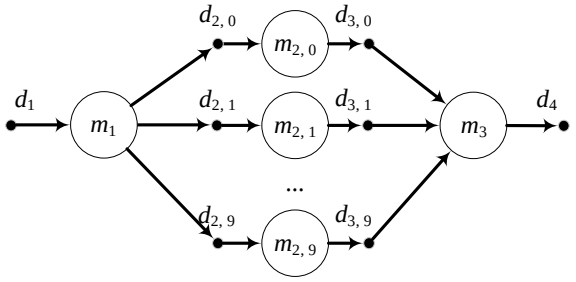

Figure 7. Workflow with parallel list of data.

Figures 8 and 9 show the simulation of the considered algorithms for this example. As in the first example, the presence of tools for assigning preliminary data transfer allows us to reduce the overheads associated with sending input data for modules. In this example, the proposed algorithm provides the reduction by 13% in the workflow makespan.

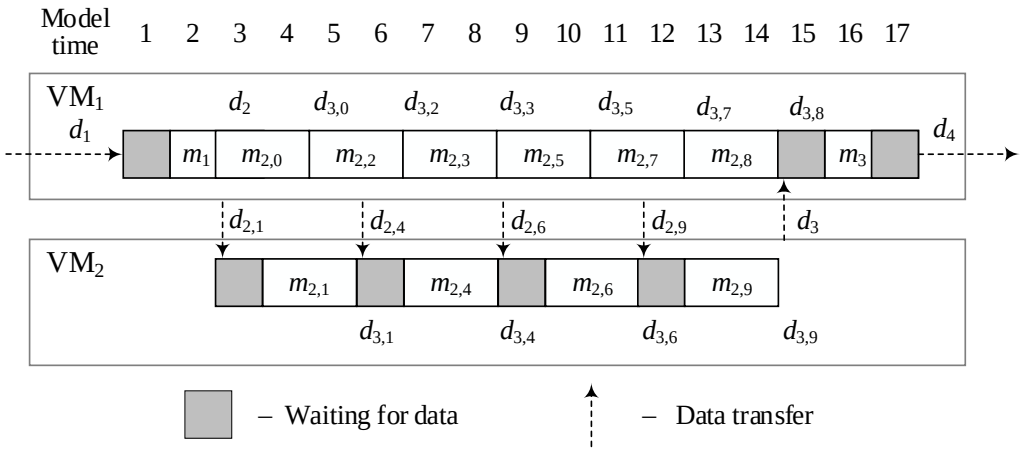

Figure 8. Job scheduling via the Condor DAGMan algorithm for workflow with parallel list of data.

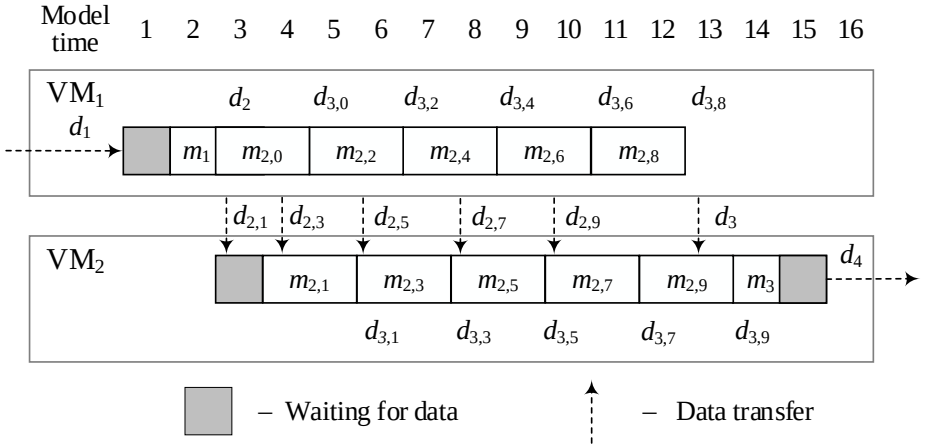

Figure 9. Job scheduling via the proposed algorithm for workflow with parallel list of data.

4. Conclusions

In the paper, we present a new algorithm that implements static-dynamic scheduling asynchronous computations in distributed applied software packages. The algorithm operation is oriented to heterogeneous distributed environments.

Through the experimental results, we demonstrated that the advantages of the proposed algorithm for different workflow types. In conditions of uncertainty over the modules execution time, the algorithm makes it possible to rationally determine the order of modules' launches in comparison with other ways of scheduling.

The advantages of the proposed algorithm for managing asynchronous computations in distributed environments have been demonstrated in specific examples. In the future, we hope to find theoretical estimates supporting such advantages for a larger number of cases.

As additional future work, we plan to use additional heuristics based on analyzing the modules execution time with test data in environment nodes within the framework of continuous integration, delivery, and deployment of package software.

Acknowledgments

The study was supported by the Ministry of Science and Higher Education of the Russian Federation, project no. 121032400051-9 «Technologies for the development and analysis of subject-oriented intelligent group control systems in non-deterministic distributed environments».

References

[1] Hilman M H, Rodriguez M A and Buyya R 2020 Multiple workflows scheduling in multi-tenant distributed systems: A taxonomy and future directions *ACM Comput. Surv*. **53(1)** 1–39

[2] Garey M and Johnson D 1979 Computers and Intractability (San Francisco: W. H. Freeman) p 338

[3] Feoktistov A, Kostromin R, Sidorov I and Gorsky S 2018 Development of distributed subject-oriented applications for cloud computing through the integration of conceptual and modular programming *Proc. of the 41st International Convention on information and communication technology, electronics and microelectronics (MIPRO-2018)* (Riejka: IEEE) pp 256–261

[4] Atkinson M, Gesing S, Montagnat J and Taylor I 2017 Scientific workflows: Past, present and future *Future. Gener. Comp. Sy*.**75** 216–227

[5] Cao J, Jarvis S A, Saini S and Nudd G R 2003 Gridflow: Workflow management for grid computing *Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)* (IEEE Press) pp 198–205

[6] Guler A T and Waaijer C J F, Palmblad M 2016 Scientific workflows for bibliometrics *Scientometrics* **107(2)** 385–398

[7] Monge D and Garino C G 2010 Improving Workflows Execution on DAGMan by a Performance-driven Scheduling Tool *Proc.of the 3rd Symposium on High-Performance Computing in Latin America* **39** pp 3271–3285

[8] 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

[9] Deelman E, Vahi K, Rynge M, Juve G, Mayani R and Silva R F 2016 Pegasus in the cloud: Science automation through workflow technologies *IEEE Internet Computing* **20(1)** 70–76

[10] Directed Acyclic Graph Manager. Available at: https://research.cs.wisc.edu/htcondor/dagman/dagman.html (accessed: 20.11.2020)

[11] Carrión I M, Huedo E and Llorente I M 2015 Interoperating grid infrastructures with the GridWay metascheduler *Concurr. Comp.-Pract. E*. **27(9)** 2278–2290

[12] Kalayci S, Dasgupta G, Fong I, Ezenwoye O and Sadjadi S M 2010 Distributed and adaptive execution of Condor DAGMan workflows *Proc. of the 22nd International Conference on Software Engineering and Knowledge Engineering* pp 587–590

[13] Radchenko G, Alaasam A and Tchernykh A 2018 Micro-workflows: Kafka and kepler fusion to support digital twins of industrial processes *Proc. of the 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)* (IEEE Press) pp 83–88

[14] Bharathi S, Chervenak A, Deelman E, Mehta G, Su M-H and Vahi K 2008 Characterization of scientific workflows *Proc. of the 3rd IEEE Workshop on Workflows in Support of Large-Scale Science* (IEEE Press) pp 1–10

[15] Ho W, Xu X and Dey P K 2010 Multi-criteria decision making approaches for supplier evaluation and selection: A literature review *Eur. J. Oper. Res*. **202(1)** 16–24

[16] Bychkov I V, Oparin G A, Feoktistov A G, Sidorov I A, Bogdanova V G and Gorsky S A 2016 Multiagent control of computational systems on the basis of meta-monitoring and imitational simulation *Optoelectron. Instrum. Data Proces.* **52(2)** 107–112

[17] Feoktistov A, Gorsky S, Sidorov I, Bychkov I, Tchernykh A and Edelev A 2020 Collaborative development and use of scientific applications in Orlando Tools: Integration, delivery, and deployment *Comm. Com. Inf. Sc*.**1087** 18–32

[18] Buyya R, Murshed M, Abramson D and Venugopal S 2005 Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm *Software Pract. Exper.* **35(5)** 491–512

[19] Irkutsk Supercomputer Center. Available at: http://hpc.icc.ru (accessed: 10.05.2020)