Scheduling in Workflow-as-a-Service Model with Pre-Parameterized DAG using Inaccurate Estimates

Victor Toporkov [0000−0002−1484−2255], Dmitry Yemelyanov [0000−0002−9359−8245] ,

and Artem Bulkhak

National Research University “MPEI”, Russia
ToporkovVV@mpei.ru, YemelyanovDM@mpei.ru,
BulkhakAN@mpei.ru

Abstract. Workflow is currently the most common execution model for composite applications across multiple disciplines: seismology (CyberShake), bioinformatics (Epigenomics, SIPHT), astrophysics (Montage), gravitational wave physics (LIGO), hydro and aerodynamics, quantum chemistry, nanotechnology, hydrometeorology, modeling of social systems and transport infrastructure. The workflow is usually a collection of interrelated tasks within the directed acyclic graph (DAG) model parameterized by a priori, usually inaccurate, user estimates for tasks (relative computational or data transfer volumes, execution durations etc.). In this work, we propose an approach for scheduling science-intensive applications within the framework of the concept of Workflow-as-a-Service (WaaS). The proposed scheduling model is built based on the critical jobs’ method, which allows us to obtain the deadlines for completing each of the workflow tasks under given efficiency criteria and inaccurate user estimates. This schedule must consider the actual dynamics of the WaaS resources’ utilization and lifecycle of virtual machines (VMs). To solve this problem, we propose a novel procedure to group and assign workflow tasks to VMs instances provided by the Infrastructure as a Service (IaaS) provider.

Keywords: Cloud Computing, Scientific Workflow, Scheduling, Inaccurate Estimates, Critical Job, Task, Batch, Hungarian algorithm.

1 Introduction

Many well-known scientific-intensive projects, such as Montage, CyberShake, Epigenomics, SIPHT, and LIGO are implemented as workflows [1-16]. IaaS allows a Workflow Management System (WMS) to access a practically unlimited pool of virtualized resources on a “pay-per-use” basis. To date, there are a huge number of workflow management systems [17]. They include ASKALON, Galaxy, HyperFlow, Kepler, Pegasus, Taverna, CloudBus and several others. The paradigm of WaaS makes it possible to implement effective mechanisms for managing continuous flows of diverse types of jobs in cloud computing. However, this raises a few fundamental problems associated

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

with organizing the scheduling of the heterogeneous job-flows and composite applications.

The main purpose of the presented solution is the implementation of WaaS platforms for monitoring, processing requests, scheduling, and managing heterogeneous cloud resources, and in particular, the dynamic creation and removal of VMs and containers for their efficient assignment to various workflows tasks.

The proposed solution intends to take into account a number of important aspects: 1) the presence of multiple IaaS providers and different types of resources; 2) geographic distribution of data centers; 3) heterogeneity of workflows; 4) the need to implement the "pay-per-use" model for a specific system user; 5) finally, solving the problem of an efficient deploying of VMs and providing many containers ready for multi-threaded environment on physical servers.

The rest of this paper is organized as follows. Section 2 reviews work that is related to our discussion. Section 3 presents the critical jobs' method (CJM) method for scheduling science-intensive applications and its implementation. Sections 4 and 5 introduce strategies and a general optimization scheme for VMs allocation. Sections 6 and 7 contain software implementation details and workflows scheduling results obtained for the considered algorithms. Section 8 summarizes the paper.

2 Related Works

The development of cloud technologies and WMS has given new impetus to research in the field of workflow scheduling in various applications. In particular, one of the areas of research is related to the active development of a paradigm in cloud computing WaaS [1-16].

As a rule, in known scheduling algorithms, the total cost of executing a workflow is used as one of the optimization criteria or restrictions [1-6, 13]. The work [1] proposes an approach to scheduling workflows in a container cloud environment for the WaaS model. The paper [2] introduces the use of a workflow broker based on a combination of on-demand and spot resource instances to minimize flow execution costs while meeting deadline constraints. The authors of [4] propose a time- and budget-aware dynamic workflow scheduling (DDBWS) algorithm designed specifically for WaaS environments. DDBWS schedules workflows by solving the problem of packing multiple resources. Unlike existing algorithms, it simultaneously considers the processor and memory demands of tasks. The proposed algorithm can significantly reduce the total number of rented VMs. In [6], the results of expanding the functionality of WMS CloudBus to process multiple workflows are presented and a prototype of a WaaS cloud platform for applications in the field of bioinformatics is proposed. A budget-constrained resource scheduling algorithm for multiple workflows (EBPSM) is implemented. In [13], a scheduling method is proposed that can reduce monetary costs and complete the work process within the minimum execution time. To analyze the performance of the proposed algorithm, an experiment is carried out in the WorkflowSim environment, and the results are compared with existing well-known algorithms - HEFT and DHEFT.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

One of the main challenges of workflow scheduling is to make it energy efficient for cloud providers [3, 14, 16]. To date, formal formulations of workflow scheduling problems in scientific applications with several criteria are known [7, 8]. Artificial intelligence methods are increasingly being used in workflow scheduling tasks [5, 9, 10]. An important issue is representing the workflow model as a DAG. In a number of applications, loops are present naturally. In some known WMS (Pegasus. Apache Airflow, Taverna, Kepler) palliative techniques are used. This results in increased planning time.

In this work, in contrast to the studies discussed above, we propose innovative technologies and tools for scheduling and managing workflows of varying complexity and structure, considering many factors affecting the efficiency of using cloud platform resources, and the simultaneous passage of workflows on WaaS platforms.

3 Workflow Scheduling with Critical Jobs' Method

3.1 Critical Jobs' Method

The core of WaaS system is obviously the algorithm for processing and scheduling the workflows. Although there are many approaches to this problem, including classical ones and presented in the above section, we begin our consideration with the Critical Jobs' Method (CJM). The main important feature of CJM is the possibility to prepare ***a reference scheduling plan*** and define execution ***deadlines*** for each task of the workflow using ***a priori, usually inaccurate, user estimates***.

More formally, the CJM algorithm solves the following problem. The workflow of data-dependent tasks can be represented with DAG, the vertices of which correspond to tasks and data transfers (Fig. 1). The processing of the flow of independent tasks is implemented in groups, in which the tasks are ordered by priorities. A ***job*** is a sequence of tasks (a path on the DAG). Let G be a parameterized workflow graph with various levels of parallelism of its partially ordered tasks. The partial order relation on the set $T = J \cup D$ of tasks and data transfers is defined with a DAG, where a subset J correspond to computational tasks, and a subset D - to the data transfers between the tasks. The set of oriented edges of the graph represents information and logical connections and dependencies. The graph is parameterized by a priori estimates of relative computational or data transfer volumes v_{ik}, execution durations t_{ik}^0 for tasks $j_i \in J, i = 1, \dots, n$, on the corresponding resource type $k \in K, K$ is a number of resource types, and n is a number of tasks. Examples of these parameters for graph from Fig. 1 on four types of resources are given in Table 1.

We define the distribution r of resources between tasks in J over a period of time $[0, t^*]$ as follows:

$$r = \{(a_i, b_i, t_i), i = 1, .., n,\ a_i = k \vee k^\circ, k = 1, .., K, k^\circ \in \{1, .., K\}, b_i \in [0, t^*)\ \}, \quad (1)$$

where a_i is a parameter determining the assignment of a task $j_i \in J$ to the corresponding resource; b_i and t_i are, respectively, the start time and execution duration of the task $j_i \in J$ on the resource, the type of which is determined by the assignment a_i.

In (1), $a_i = k$ if a task $j_i \in J$ is tied to a so-called base resource, the level (for example, the number of processors) of which is limited and depends on the capabilities of

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

the task parallelization system, the cost of using the resource of the type k, and a number of other factors.

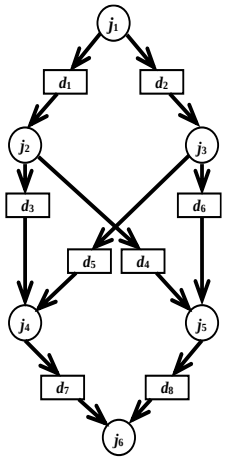

Fig. 1. Example of workflow DAG

In the event of a conflict between parallel tasks from J, competing for the same resource of type k, taking into account the scalability of the computing environment, a resource of type $k° \in \{1,..,K\}$ is introduced that is not inferior in its characteristics to the basic one, and at the same time $a_i = k°$. This may be, for example, an additional processor node of the same type k or an unused base node of the type $k° \neq k$.

Table 1. Examples of parameterization for tasks $j_1, \ldots, j_6$

Parameters	j_1	j_2	j_3	j_4	j_5	j_6
$\boldsymbol{t^0_{i1}}$	2	3	1	2	1	2
$\boldsymbol{t^0_{i2}}$	4	6	2	4	2	4
$\boldsymbol{t^0_{i3}}$	6	9	3	6	3	6
$\boldsymbol{t^0_{i4}}$	8	12	4	8	4	8
$\boldsymbol{v_{ik}}$	20	30	10	20	10	20

Let us assume that restrictions (deadlines) are configured for completion times of individual tasks and jobs, i.e. sequences $j_{i_1},..,j_{i_L}$ of informationally or logically related tasks $1 \le i_1 \le \cdots \le i_L \le n$ that make up the job:

$$t^*_g - t_g \ge 0, t^*_h - \sum_h t_h \ge 0, g, h \in \{1,..,n\}, \quad (2)$$

where t_g, t_h - completion times of tasks $j_g, j_h \in J$, and t^*_g, t^*_h - deadlines for task j_g and the job containing the task j_h.

One example of a scheduling criterion is the function of a workflow completion cost:

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

$$CF = \sum_{i=1}^{n} \left\lceil \frac{v_{ik}}{t_{ik}} \right\rceil, t_{ik} \geq t_{ik}^{0}, \quad (3)$$

where v_{ik} - is the relative computational volume of task j_i; t_{ik} - is the time allocated for executing a task j_i on a processor of type k; n - is the number of tasks; $\lceil \cdot \rceil$ denotes the nearest integer that is not less than the value $\frac{v_{ik}}{t_{ik}}$ of the cost of executing the i-th task.

Resource allocation (1) is admissible if constraints (2) are met and the corresponding optimality criterion is defined, for example (3).

A ***critical job*** is a sequence of tasks (containing unassigned tasks) with the largest sum of specified prior execution estimates using ***the best combination of resources***. Let us assume the durations of all data transfers $d_1, \ldots, d_8$ in the workflow model G (see Fig. 1 and Table 1) are equal to one unit of time. Let $t^* = 20$ be the deadline for completing the workflow. Let us rank critical jobs according to the values of their a priori maximin duration: $(j_1, d_1, j_2, d_3, j_4, d_7, j_6)$; $(j_1, d_1, j_2, d_4, j_5, d_8, j_6)$; $(j_1, d_2, j_3, d_5, j_4, d_7, j_6)$; $(j_1, d_2, j_3, d_6, j_5, d_8, j_6)$. For the above-mentioned works, this indicator is respectively equal to 12, 11, 10 and 9 units of time (Fig. 2).

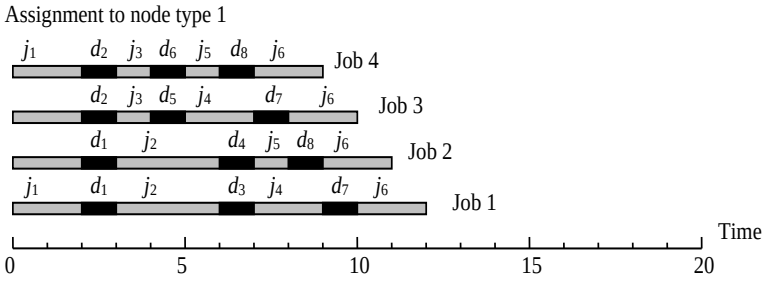

Fig. 2. Ranking of critical jobs when assigned to the first node

Then the first critical job in terms of duration corresponds to the path $(j_1, d_1, j_2, d_3, j_4, d_7, j_6)$. It is a priori maximin duration is 12 time units (see Table 1). After the assignment of tasks to this sequence, the next considered job is $(j_1, d_1, j_2, d_4, j_5, d_8, j_6)$ since tasks d_4, j_5, d_8 have not been assigned yet and the a priori duration of the job is 11 time units. The resource allocation for the subsequence (d_4, j_5, d_8) must account for the assignment results of the previous critical job. More detailed information on CJM scheduling, collision resolution and formalization based on dynamic programming schemes is presented in the paper [18].

The iterative application of this procedure with conflict resolution between parallel tasks competing for the same resource, in accordance with (1), is the essence of the ***critical jobs' method***. Based on this method, it is possible to construct a scheme for the sequential formation of reference (optimal) and suboptimal schedules for a given efficiency criterion. Let the criterion for the efficiency of resource use be given as a cost function (3). The CF takes the closest to t_{ik} duration estimate t_{ik}^0 that determines the type k of resource being used (slot set or node). The Gantt chart shown in Fig. 3, represents the result of workflow scheduling for graph from Fig. 1 on four types of resources are given in Table 1.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

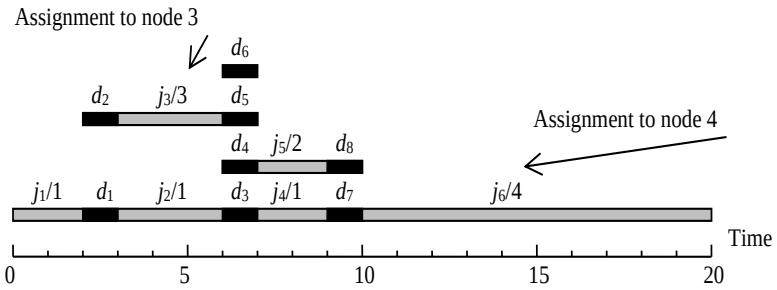

Fig. 3. Gantt chart of workflow scheduling

3.2 Critical Jobs' Method Modification

Firstly, we list major features and limitations of the original CJM:

- the method is designed to schedule a single static workflow;
- the point of the base method is to use a multiphases procedure, which identifies the next critical job and resolves possible conflicts (collisions) with previously assigned tasks over a shared resource;
- it allows to obtain optimal and close to optimal plans for specified restrictions on the total cost or task execution time for a single workflow.

Considering the dynamic workflow processing environment, the basic CJM algorithm is available to process incoming workflow jobs individually at the moment of the arrival to the platform. The main problem that arises is the dynamic component of the system and compliance with the relevant requirements for the quality of service of each workflow. In the base implementation of CJM at the conflict resolution stage, it is proposed to reduce the bipartite graph of a "multilayer" collision to a bipartite graph of a "two-layer" collision by sequentially viewing adjacent pairs of the graph vertices (tasks) and selecting vertices with the required weight. Thus, each scheduling of the next critical job may cause a conflict with the previous allocation. Further it can lead to collisions between parallel tasks and jobs of multiple simultaneously processing workflows.

Thus, in our dynamic model, for all arriving workflow jobs we implement only the first CJM stage of planning time ranges for performing critical job tasks according to budget and time user restrictions for the whole workflow. These calculations can be made simultaneously for multiple different workflows without any conflicts for the resources. The resulting time ranges represent execution recommendations and deadlines for each task which should be further handled by the cloud resource assignment module.

While developing the method, several statements have been made.

1) Since only resource types are considered at CJM planning stage, and not their specific instances, there are no conflicts within a job, a workflow, or many workflows. Therefore, the conflict resolution process is not included in the modified CJM but is transferred to the stage of assigning tasks to specific instances of VMs.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

2) The modified CJM builds a workflow execution plan based on a priori time estimates of task execution from given resource types and the data transfer time between workflow tasks. Information about the amount of data transmitted and the time of each data transaction is recorded in the result of the CJM operation and passed to the assignment module.

3) At the stage of scheduling a critical job and calculating the additive separable criterion, in the case of several identical values of the criterion, we settle on the first one. At this stage it is important to select the optimal value, regardless of the specific assignment for resource types.

In the result, the modified CJM is applied for each new workflow that arrives on the WaaS platform, and as the output it provides the following: time ranges of the workflow tasks execution; tasks' volume and execution requirements; data volumes and the data transfer dependencies between the tasks of the workflow.

4 The General Optimization Scheme of the Virtual Resources Allocation

4.1 Virtual Resources Allocation Strategies

One of the most important problems arose when scheduling and executing many computational tasks from the workflows is the effective allocation and management of VMs. By managing VMs, we mean determining the moments of start (creation) and finish (stopping, releasing) of individual VMs and containers, as well as the assignment and the execution order of ready-to-run computational tasks. We propose several strategies for managing VMs to execute workflow tasks.

Firstly, one can create a new dedicated VM specifically to execute each individual task and release it when the task is completed. This ***greedy*** but flexible strategy allows us to strictly match VM types and lifetime with the tasks' requirements, especially in conditions when the time required to create and shut down VM is much less compared to the average task execution time. However, the VM creation and preparation time includes time to configure the necessary software environment and the time to copy and receive the required input data. The shutdown/cleanup time may increase due to the need to save and copy the calculation results to the next task of the global data storage (e.g., Amazon 3s).

Secondly, one can maintain some dynamically changing pool of constantly active VMs and distribute ready-to-run tasks between them (the so-called ***control strategy***). The pool dynamics implies a decrease and an increase in the number of active machines, depending on the computing needs at a certain time. With this approach, it may be possible to schedule and assign tasks more efficiently by matching tasks' requirements with already active VMs and sometimes skip the data transfer routines. For example, when two consecutive (data-dependent) tasks are executed on the same VM, then the operation of copying and transferring data is not required. On the other hand, due to the specifics and variety of workflow structures, as well as their variable number, it is not always possible to ensure full and constant loading of the entire pool of active VMs.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

Thus, some of the VMs will be idle from time to time, thereby reducing the usefulness and cost-effectiveness of this approach. In addition, a certain difficulty lies in designing an algorithm for efficiently assigning tasks to available VMs.

Thirdly, there is a ***mixed strategy***, when some basic minimum pool of active VMs is maintained during the execution of workflows, but additional dedicated VMs can be created to account for all unassigned ready-to-run tasks.

Since the implementation of the first greedy approach is trivial but does not provide clear mechanisms to optimize the use of virtual resources, a control strategy is further considered in this paper.

4.2 High-level Optimization Scheme

To implement the control strategy, we propose the following general algorithm scheme.

1) As input data the algorithm receives execution time plans for the individual tasks, data transfer volumes between them, as well as available types of VMs. The task execution plan is the time interval expected for its actual execution (i.e. the earliest start time and the latest completion time). It is assumed that these time ranges are passed from modified CJM implementation and maintain the relationship of continuity and sequence of execution in the initial workflows.

2) At the first stage, the input task flow is divided into parallel execution batches. The main requirement for dividing is that all tasks in one batch must be independent (there must be no data dependencies between them) and can be performed in parallel, taking into account the execution plan. Thus, the entire set of tasks is divided into many consecutive groups-batches. Tasks with data dependency should be in different, sequential (although not necessarily adjacent) parallel execution batches. The dividing into batches can be performed dynamically, taking into account the constantly incoming tasks. It is enough to operate with two parallel execution bacthes to implement the general optimization scheme.

3) At the second stage, the algorithm performs sequential scheduling and task assignment of each batch to the virtual resources. To achieve this, the problem of the minimum perfect matching (assignment problem) is solved using the Hungarian (Kuhn-Munkres) algorithm [19].

5 Algorithms to Group Tasks into Parallel Batches

5.1 Generalization and Input Data

The important initial step for efficient VM allocation and processing is to determine the number of simultaneously required VMs at any given time. This number depends on the structure of workflows, the relationships between tasks, the time of their execution, the history of assignment to VMs, etc. Thus, it should be determined dynamically in the runtime. For this purpose, we propose and study two different algorithms of grouping workflow tasks into batches. Each batch should contain tasks which can be executed in parallel without breaking data dependencies and local deadlines.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

Firstly, we proposed and implemented the algorithm Follow The Leader (FTL) [19]. The main idea behind FTL is to preserve the following invariant *given the batch start time and tasks' deadlines, no two tasks in the batch can be executed sequentially even on the fastest VM*. Thus, FTL determines the minimum required parallelism level of the incoming task flow at any given time. The result of its work can be further used for predicting and dynamic management of multiple VMs.

FTL algorithm receives a set of computational tasks t_i and a list of available VM types Vm_t^j. Every task t_i contains the following information: its own computational volume Vp^i (which is necessary to predict the time of its execution Te^i on each type of VM), the amount of input Vin^i and output $Vout^i$ data, and the relationship of precedence with parent and children tasks. In addition, an expected execution interval is defined for each task (the earliest start time tr^i and the deadline for completion Td^i) at the stage of preprocessing by the CJM module. Based on the early start time values tr^i, the execution time Te^i on the given virtual machine and the deadline for completion Td^i, two additional characteristics can be computed separately: the earliest completion time $mint_f^i = tr^i + Te^i$ and the latest start time $maxt_s^i = Td^i - Te^i$.

5.2 ASAP Algorithm

However, despite the implementation of the above-described invariant, FTL algorithm tends to save and minimize average demand for VMs and to postpone the execution of all tasks closer to their deadlines. These features may lead to an increase in the required VM performance and may result in parent and child tasks not being included in adjacent packages. In turn, this may prevent the reuse of execution data when performing parent and child tasks on the same VM instance. Thus, as an alternative to FTL we considered the more straightforward algorithm described in [6]. The basic idea of this algorithm is to execute each task as soon as possible when all parent tasks are finished. Each task is placed into the batch following the batch containing its last parent task. We call this algorithm ASAP. Thus, in contrast to FTL, ASAP strives to execute all tasks immediately when ready, without considering local deadlines. However, looking ahead, this policy of placing tasks into batches as early as possible turns out to be more suitable and flexible for the VM assignment stage.

Another important numerical metric of the batch grouping quality is the number N_{pc} of parent and child tasks in adjacent batches. Generally, such groupings may allow us to save time on data transfers due to reuse of VMs, their internal data storage and configurations. Table 2 shows N_{pc} provided by FTL and ASAP algorithms for different workflows. The results in Table 2 demonstrate the general advantage of the ASAP algorithm over FTL in terms of N_{pc} even for small workflows consisting of 50 tasks. On the other hand, the batch grouping result strongly depends on the workflow structure and the given task deadlines. The greatest advantage is demonstrated in heterogeneous workflows with tasks of varying duration. However, for MONTAGE workflow consisting of 1000 tasks, ASAP and FTL algorithms provided identical batch groupings with $N_{pc} = 834$. As a result, ASAP algorithm generally demonstrates more efficient

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

results because it groups tasks into batches more tightly and allows for resource reuse during the workflow execution.

Table 2. Number N_{pc} provided by FTL and ASAP algorithms for different workflows

Algorithm	LIGO50	GENOME50	CYBERSHAKE50	MONTAGE1000
FTL	31	15	29	834
ASAP	38	48	45	834

5.3 Dynamic VM Allocation

Next, we discuss the problem of efficient resources allocation and assignment for each batch of parallel execution retrieved with FTL algorithm. It should be noticed that this assignment operation implicitly assumes the possibility of disabling unnecessary VMs.

The VMA (VM Allocation) algorithm is proposed to solve this problem, based on the Hungarian Kuhn-Munkres algorithm (Kuhn, H.W., 1955) to find a perfect matching in a bipartite graph $G = (T, R, E)$, where T is the set of batch B tasks, R is the set of available resources, and E is the set of edges between T and R. The edge between the task from T and the resource from R means that the task can be executed on the corresponding VM in compliance with all requirements. The edge weight is a target optimization criterion of this assignment and scheduling in general. By these means we can optimize VMs total usage cost, tasks' execution runtimes, data transfers time, etc. [19].

6 Software Implementation and Analysis

In developing the software implementation of the proposed algorithms, the following basic assumptions were made.

Firstly, in the current implementation, it is assumed that the expected task execution time on some instance of VM can be calculated as the ratio of the task computational volume to a given performance characteristic of the VM. This assumption is optional and is made for greater clarity of the model, input data and calculations. In future versions of the program, it is assumed to provide a more flexible calculation of task execution time, for example, based on a given matrix of correspondence between tasks and types of available VMs.

The second assumption is there is global data storage with sufficient volume for simultaneous storage of all intermediate data necessary for transfer between the tasks. At the same time, the calculated data transfer rate remains the same in parallel copying of output data from a set of completed tasks. This assumption is necessary to simplify calculations in cases where the sequential tasks are executed with some acceptable delay, and it is more beneficial to copy output data to a centralized storage, rather than keep it on a VM until the next task is started. This assumption can be justified in scenarios where the volume of transferred data does not exceed a certain critical value, and centralized, possibly cloud storage implements effective balancing of requests between several nodes.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

Thirdly, the developed program solution does not implement a single best-case processing scenario for all possible workflows but provides a set of configuration parameters to tailor processing for a specific scheduling scenario. These parameters are primarily aimed at minimizing the schedule calculation time, which can grow cubically relative to the set of elements in the parallel batch.
An important stage is the preparation of data for the assignment problem and Hungarian algorithm: calculating the edges' cost of the considered bipartite graph. Thus, it is necessary to precalculate the expected result and parameters (including the target criterion) for each pair between the tasks T and VMs R.

Each pair of a task t_i and a VM VM^j can be considered independently. The execution plan of each pair may include VM waiting (idle) time, preparation time, actual task execution runtime (based on the task computational volume Vp^i and VM performance P^j) and release time. VM preparation time includes the time to create a new machine and the time to copy the necessary data from the previous related task. Data can be copied either directly from the VM on which the previous task was executed, or from global storage. If the current task is scheduled to be performed on the same instance of VM, then data transfer is not required. Based on the preparation time of the VM and the task execution time, the required usage time of the VM and the corresponding economic cost are calculated.

7 Workflow Scheduling Optimization Results

7.1 Optimization Results

To demonstrate the optimization capabilities of the presented algorithm, we conducted a series of scheduling experiments on many real workflows, including GENOME, LIGO, CYBERSHAKE, SIPHT, MONTAGE and their intersecting combinations.

Table 3 shows the main execution characteristics of LIGO workflows in different optimization scenarios. The results were obtained with developed software in Python 3 environment, CPU Core i5, and 8 GB RAM. The presented results show significant optimization potential realized by selecting the required criterion for in the implemented algorithm.

Table 3. LIGO Workflow Optimization Results

Optimization	Total VM Cost	Total Runtime, sec	Total VM Time, sec
Cost minimization	**12740**	4260	4328
Cost maximization	**13057**	4576	4754
Runtime minimization	12929	**4180**	4310
Runtime maximization	12769	**4757**	4840
VM time minimization	12743	4200	**4269**
VM time maximization	12952	4823	**4980**

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

7.2 Comparison of VM Allocation Strategies

In this experiment series we study VM allocation efficiency provided by the following strategies and algorithms.

1. **Greedy strategy** creates a new dedicated VM for each workflow task. VM type is selected to optimize the global scheduling criterion while meeting the execution time deadline. VM usage may require additional uptime to receive data transfers required for the task execution.
2. **Control strategy** is represented with the proposed **VMA** algorithm: VM assignment procedure optimizes global criteria given the deadline constraints for workflow tasks grouped in batches.

Testing was carried out based on a workload consisting of 100 independent workflows. The workflow instances were built based on real scientific applications (Montage, Cybershake, Genome, LIGO, SIPHT) and contain 50 vertices each. The execution deadline for each individual workflow was generated randomly between the fastest (using highest performance VMs) and the slowest (using the least performance VMs) possible execution times. The deadline limit determines the base execution plan and therefore affects the allocation variants and strategies. These workflows differ in their structure, required computational and data transfers volumes. For example, the average execution time of Montage workflow is 100 seconds, while Cybershake on average requires 30000 seconds. So, workflows with larger computational volumes may have a greater impact on the simulation results.

The following environment configuration parameters were studied:

- the arrival rate of workflows to the WaaS platform (quantity per minute);
- time intervals required to create and initialize and to release the VM.

It is worth to mention CJM scheduling step allows us to specify a specialized optimization criterion for each received workflow. This optimization will generally affect deadlines for the tasks of the workflow.

However, only one common optimization criterion can be used in VMA during the actual VM allocation. To support specific optimization criteria for input workflows it is possible to run several instances of VMA algorithms, each processing workflow matching one particular global criterion.

Table 4 contains total results of the workload execution depending on the workflow arrival rate (from 0.5 to 100 workflows per minute). Firstly, we note that Greedy execution results do not depend on the workflow arrival rate: all VMs are created and tailored for specific tasks, so the absolute start time of the task does not affect the choice of the VM type. On the other hand, VMA scheduling directly depends on the density of the incoming tasks. So, based on the workflow arrival rate and the composition of the parallel batches, VMA allocated 5-20% less VM instances by reusing them to execute several consecutive tasks. This strategy resulted in nearly 5% advantage over the Greedy strategy by the total VM cost criterion. Total tasks' execution time remained nearly constant as it depends on the pre-configured deadlines specific to each workflow and independent from the arrival rate.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

Table 4. Algorithms' comparison depending on workflow arrival rate

Workflow Arrival Rate (per minute)	Algorithm	Total Task Execution Time, sec	Total VM Cost	# of Created VMs
0.5	VMA	409280	13226	3912
1	VMA	409486	13277	4116
2	VMA	409602	13316	4334
6	VMA	409601	13279	4503
12	VMA	409586	13257	4574
60	VMA	409578	13168	4619
100	VMA	409596	13168	4633
*****	**Greedy**	**409650**	**13906**	**4955**

Table 5 contains total results of the workload execution depending on the time required to create and initialize and to release (destroy) VM. This parameter affects the result of both VMA and Greedy algorithms, as total cost directly depends on total VM usage time, including periods of VM initialization and release.

Based on the results obtained, VMA allocated 16% less VM instances to execute the same amount of tasks, resulting in an up to 10% advantage over Greedy algorithms by the total cost criterion. Sa expected, the advantage increases with increasing VM creation and initialization time.

Table 5. Algorithms comparison depending on VM initialization and release time

VM Init/Release Time	Algorithm	Total Task Execution Time, sec	Total Cost	# of Created VMs
0/0	VMA	391361	10878	4175
0/0	Greedy	391559	10885	4955
10/1	VMA	391297	11009	4127
10/1	Greedy	391559	11053	4955
100/10	VMA	391449	12144	4125
100/10	Greedy	391559	12557	4955
300/30	VMA	391453	14576	4134
300/30	Greedy	391559	15899	4955
500/50	VMA	391413	17076	4118
500/50	Greedy	391559	19242	4955

Overall, the following main conclusions can be drawn from the comparison results:

- both VMA and Greedy algorithms meet workflow deadlines in 100% of simulation experiments; total cost optimization implies approximate equality in terms of total time criterion;

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

- VMA exploits the possibility of reusing VMs, minimizing VMs initialization time and data transfer times between data-dependent tasks; in this way VMA at average allocates 10-15% less VM instances, resulting in 5% less total VM usage cost.

8 Conclusion and Future Work

In this work, a multi-module procedure was proposed and implemented for scheduling and executing multiple independent workflows. For this purpose, several modifications of the CJM were implemented, including the possibility of time modeling to receive and schedule a set of workflows that are spaced in time.

The resource assignment stage allows us to optimize many global characteristics of cloud resource usage. The proposed solution manages the pool of active VMs by defining for each instance the creation, preparation, utilization, data transfer and shutdown intervals. The developed solution has been evaluated on several examples of real-world workflows. It is worth emphasizing once again that in many respects the results of this work were obtained based on a study of both classical and highly specialized optimization algorithms.

The main limiting factor is the high (cubic) computational complexity of the solution relative to the parallelism degree of the incoming task flow [19]. Thus, there is a natural limitation in the size of workflows, the scheduling of which can be completed in a feasible time. Future work will concern problems of scheduling algorithms complexity in scalable WaaS platforms.

Acknowledgments. This work was supported by the Russian Science Foundation (project no. 22-21-00372, https://rscf.ru/en/project/22-21-00372/).

References

1. Karmakar, K., Tarafdar, A., Das, R.K. et al. Cost-efficient Workflow as a Service using Containers. J Grid Computing 22, 40 (2024). https://doi.org/10.1007/s10723-024-09745-7
2. Taghavi, B., Zolfaghari, B. & Abrishami, S. A Cost-Efficient Workflow as a Service Broker Using On-demand and Spot Instances. J Grid Computing 21, 40 (2023). https://doi.org/10.1007/s10723-023-09676-9
3. Tarafdar, A., Karmakar, K., Khatua, S., Das, R.K.: Energy-efficient scheduling of deadline-sensitive and budget-constrained workflows in the cloud. In: International conference on distributed computing and internet technology. Springer, pp. 65–80 (2021)
4. Saeedizade, E., Ashtiani, M.: Ddbws: A dynamic deadline and budget-aware workflow scheduling algorithm in workflow-as-a-service environments. J. Supercomput. 77(12), 14525–14564 (2021)
5. Qin, Y., Wang, H., Yi, S., Li, X., Zhai, L.: An energy-aware scheduling algorithm for budget-constrained scientific workflows based on multi-objective reinforcement learning. J. Supercomput. 76(1), 455–480 (2020)
6. Muhammad H. Hilman, Maria A. Rodriguez, and Rajkumar Buyya. Workflow-as-a-Service Cloud Platform and Deployment of Bioinformatics Workflow Applications. Preprint. June

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12

2020. 30 p. https://www.researchgate.net/scientific-contributions/Maria-A-Rodriguez-2114894132
7. Anurina Tarafdar, Kamalesh Karmakar, Rajib K Das, Sunirmal Khatua. Multi-criteria scheduling of scientific workflows in the Workflow as a Service platform // Computers and Electrical Engineering Volume 105, January 2023, 108458
8. Vincenzo De Maio a, Dragi Kimovski . Multi-objective scheduling of extreme data scientific workflows in Fog // Future Generation Computer Systems. Volume 106, May 2020, Pages 171-184.
9. Huifang Li a, Yizhu Wang a, Jingwei Huang a, Yushun Fan. Mutation and dynamic objective-based farmland fertility algorithm for workflow scheduling in the cloud // Journal of Parallel and Distributed Computing. Volume 164, June 2022, Pages 69-82. https://doi.org/10.1016/j.jpdc.2022.02.005
10. Yang Gu, Jian Cao, Shiyou Qian, Nengjun Zhu, Wei Guan. MANSOR: A module alignment method based on neighbor information for scientific workflow // Concurrency and Computation: Practice and Experience Volume36, Issue10, May 2024 e7736. https://doi.org/10.1002/cpe.7736
11. Ahmad, Z., Nazir, B., Umer, A.: A fault-tolerant workflow management system with quality-of-service-aware scheduling for scientific workflows in cloud computing. Int. J. Commun. Syst. 34(1), 4649 (2021)
12. B Burkat, K., Pawlik, M., Balis, B., Malawski, M., Vahi, K., Rynge, M., da Silva, R.F., Deelman, E.: Serverless Containers–rising viable approach to Scientific Workflows. In: 17th International Conference on eScience (eScience). IEEE, pp. 40-49 (2021)
13. Karmakar, K., Das, R.K., Khatua, S.: Resource scheduling for tasks of a workflow in cloud environment. In: International conference on distributed computing and internet technology. Springer, pp. 214–226 (2020)
14. Ranjan, R., Thakur, I.S., Aujla, G.S., Kumar, N., Zomaya, A.Y.: Energy-efficient workflow scheduling using container-based virtualization in software-defined data centers. IEEE Trans. Ind. Inform. 16(12), 7646–7657 (2020)
15. Silva, R.F., Pottier, L., Coleman, T., Deelman, E., Casanova, H.: Workflowhub: community framework for enabling scientific workflow research and development. In: 2020 IEEE/ACM workflows in support of large-scale science (WORKS). IEEE, pp. 49–56 (2020)
16. Medara, R., Singh, R.S. A Review on Energy-Aware Scheduling Techniques for Workflows in IaaS Clouds. Wireless Pers Commun 125, 1545–1584 (2022). https://doi.org/10.1007/s11277-022-09621-1
17. Peter Amstutz, Maxim Mikheev, Michael R. Crusoe, Nebojša Tijanić, Samuel Lampa, et al. (2024): Existing Workflow systems. Common Workflow Language wiki, GitHub.https://s.apache.org/existing-workflow-systems updated 2024-08-18, accessed 2024-08-18.
18. Victor Toporkov and Dmitry Yemelyanov (2021). Micro-scheduling for Dependable Resources Allocation // In: Performance Evaluation Models for Distributed Service Networks. Studies in Systems, Decision and Control. Vol. 343. Editors: Bocewicz, Grzegorz, Pempera, Jarosław, Toporkov, Victor. Springer International Publishing., pp. 81-105.
19. Toporkov, V., Yemelyanov, D., Bulkhak, A., Pirogova, M. (2024). Job Batch Scheduling in Workflow-as-a-Service Platforms. In: Sokolinsky, L., Zymbler, M., Voevodin, V., Dongarra, J. (eds) Parallel Computational Technologies. PCT 2024. Communications in Computer and Information Science, Springer, Cham. vol 2241. Pp. 65–79.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_12