Future Generation Computer Systems 76 (2017) 1–17

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A hybrid evolutionary algorithm for task scheduling and data assignment of data-intensive scientific workflows on clouds

Luan Teylo [a], Ubiratam de Paula [b], Yuri Frota [a], Daniel de Oliveira [a,*], Lúcia M.A. Drummond [a]

[a] Institute of Computing, Fluminense Federal University, Niterói, Brazil
[b] UFRRJ – Federal Rural University of Rio de Janeiro, Rio de Janeiro, Brazil

HIGHLIGHTS

- A new workflow model that considers tasks and data.
- The mathematical formulation of Task Scheduling and Data Assignment Problem.
- The design of a Hybrid Evolutionary Algorithm (HEA) for scheduling tasks and data.
- An extensive experimental evaluation, based on synthetic and real executions.

ARTICLE INFO

Article history:
Received 29 April 2016
Received in revised form 25 March 2017
Accepted 10 May 2017
Available online 18 May 2017

Keywords:
Clouds
Combinatorial optimization
Task scheduling
Data assignment
Hybrid evolutionary algorithm

ABSTRACT

A growing number of data- and compute-intensive experiments have been modeled as scientific workflows in the last decade. Meanwhile, clouds have emerged as a prominent environment to execute this type of workflows. In this scenario, the investigation of workflow scheduling strategies, aiming at reducing its execution times, became a top priority and a very popular research field. However, few work consider the problem of data file assignment when solving the task scheduling problem. Usually, a workflow is represented by a graph where nodes represent tasks and the scheduling problem consists in allocating tasks to machines to be executed at a predefined time aiming at reducing the makespan of the whole workflow. In this article, we show that the scheduling of scientific workflows can be improved when both task scheduling and the data file assignment problems are treated together. Thus, we propose a new workflow representation, where nodes of the workflow graph represent either tasks or data files, and define the Task Scheduling and Data Assignment Problem (TaSDAP), considering this new model. We formulated this problem as an integer programming problem. Moreover, a hybrid evolutionary algorithm for solving it, named HEA-TaSDAP, is also introduced. To evaluate our approach we conducted two types of experiments: theoretical and practical ones. At first, we compared HEA-TaSDAP with the solutions produced by the mathematical formulation and by other works from related literature. Then, we considered real executions in Amazon EC2 cloud using a real scientific workflow use case (SciPhy for phylogenetic analyses). In all experiments, HEA-TaSDAP outperformed the other classical approaches from the related literature, such as Min–Min and HEFT.

1. Introduction

The recent advances in computer science have allowed for several different fields of science to benefit from computational simulations in their experiments. These called *in silico* experiments [1,2] are consuming and producing an unprecedented volume of data to be further processed and analyzed, thus being considered as data-intensive experiments. This huge volume of data is found in experiments in many areas, for instance, phylogenetic analysis [3], computational fluid dynamics [4], astronomy [5], *etc.*. Thus, scientists perform their analyses using complex computational simulations and increasing volumes of data in their daily duties.

Most of these experiments are represented as a chaining of scientific programs, where the output of a specific program is the input of another program. In order to manage the execution of

* Corresponding author.
E-mail addresses: luanteylo@ic.uff.br (L. Teylo), upaula@ufrrj.br (U. de Paula), yuri@ic.uff.br (Y. Frota), danielcmo@ic.uff.br (D. de Oliveira), lucia@ic.uff.br (L.M.A. Drummond).

http://dx.doi.org/10.1016/j.future.2017.05.017
0167-739X/

these complex experiments, scientific workflows can be a prominent solution. A scientific workflow is an abstraction that structures the steps of a scientific experiment as a graph of activities (*i.e.* scientific program invocations), in which nodes correspond to data processing activities and edges represent the dataflow among them [1]. Moreover, scientific workflows are commonly managed by complex software named Scientific Workflow Management Systems (SWfMS) that are used to define, execute, and monitor the data-intensive experiments. Well-known SWfMS are Swift/T [6], Pegasus [7], VisTrails [8], Apache Taverna [9] and Kepler [10].

In the same experiment, a scientific workflow is usually re-executed as many times as needed, varying the input dataset or the input parameter values to interpret the quality of the result produced by each execution of the workflow. This situation is well-known in parallel computing as parameter sweep [11] and it occurs when the same workflow (and its activities) is executed using different input data files (or a partition of the input data) and/or different configurations (different parameter values) until the exploration finishes (*i.e.* the analysis of the results is complete). Thus, since workflow activities commonly process big data, to explore data parallelism and parameter sweep we consider that each workflow activity may correspond to several executable tasks. Each task is considered the smallest unit of processing and may execute in parallel by consuming a different portion of the input data. Thus, in the context of this article, a task is the representation of an atomic execution of an activity, which processes a different set of parameter values, a data partition or chunk [12]. In addition, many of the data-intensive workflows are also compute-intensive, since a single task may execute for several hours or even days.

As the complexity of the scientific workflows grows in terms of exploration of thousands of huge datasets or several parameters, the performance requirements for such workflows have been pushing the envelope on the capacity of sequential systems (*e.g.* personal computers with a few of processors) for a while already. If executed sequentially, these data- and compute-intensive workflows could execute for several months, which is error-prone and not desirable due to the competition in science nowadays [13]. Thus, the demand for High Performance Computing (HPC) environments allied to parallelism techniques is extreme for these workflows to produce results in a feasible time for scientists.

Traditional HPC environments such as clusters, supercomputers and computational grids were used over the last decade to execute scientific workflows in parallel. However, in the last decade, Clouds [14] have emerged as a prominent environment to run data- and compute-intensive workflows [15]. Cloud computing is a type of Internet-based computing where virtually unlimited infrastructure, platform, and software are provided on demand and as services (*i.e.*, IaaS, PaaS and SaaS, respectively). Clouds follow a pay-*per*-use model [16,17] where users only pay for resources they actually used and for the time they used those resources. Virtual Machines (VMs) and storage areas are types of resources provided by clouds. Using clouds scientists are not required to acquire expensive infrastructure (such as a cluster) to execute their experiments neither spend much effort to configure a new infrastructure (as in a grid). To enable a data- and compute-intensive workflow execution in a cloud, the execution of each task has to be scheduled to a corresponding VM. Then, the scheduling problem is to decide where to execute all tasks. Scheduling tasks to distributed computing resources is an NP-complete problem [18], even when we consider simple scenarios. However, there are some characteristics of clouds that makes this scheduling process a little bit more complicated.

First of all, in clouds there are several options of VM types to be deployed. Each one with different processing and storage capacity, different bandwidth and financial cost. In addition, some of these VMs may not suitable for HPC (*e.g.* the micro and nano VMs in Amazon EC2 cloud). Thus, when we are executing workflows in parallel in the cloud, the deployed virtual cluster is commonly composed by heterogeneous VMs and this heterogeneity has to be considered in the scheduling approach. The second problem is data locality and transfer. Many of existing workflows consume and produce many GB or even TB of data and this data (or at least a partition of this data) has to be eventually transferred from one VM to another during the workflow execution. These data transfers can produce a huge (and negative) impact in the workflow execution. Let us consider the Montage workflow [5] as example. A simple execution of Montage may produce data files with several GBs. If, during a workflow execution, this data file is transferred several times from one VM to another, a considerable portion of the total workflow execution time will be spent only on data transfer instead of data processing (which is the focus of the experiment). Thus, when we are scheduling tasks of a workflow in the cloud we have to try to avoid unnecessary data transfers, or when the data transfers are necessary, we have to diminish the impact of data transfer in the total execution time of the workflow.

To exemplify this issue, let us consider the case where we have two tasks: $task_1$, which is a short term task (*i.e.* it is not compute-intensive) and $task_2$, which is a long term task (*i.e.* it is a compute-intensive task and requires high processing capacity to finish in a feasible time) in a workflow. In this example, $task_2$ consumes the data produced by $task_1$ (*i.e.* there is a data dependency). Let us also consider that $task_1$ was executed in a *medium* Vm1 of Amazon EC2 cloud and produces a data file with several GBs. To avoid data transfers, we face 2 possible scenarios: (i) to schedule $task_2$ to Vm2 (or Vm3) that have more processing capacity (such as a *2XLarge* VM in Amazon EC2 cloud), which will result in a costly data transfer from Vm1 to Vm2 (or Vm3) and (ii) to execute $task_2$ in Vm1 where $task_1$ was executed. The first scenario will imply into a costly data transfer, but then we can assume that $task_2$ will properly execute in a feasible time. The second scenario does not imply data transfers, but Vm1 may be not suitable to execute $task_2$, so the time needed to process $task_2$ in Vm1 can be huge. The scheduling approach has then to analyze the trade-off between transferring data and executing $task_2$ in Vm2 (or Vm3); or avoiding data transfer and executing $task_2$ in Vm1. In addition, when a data transfer is needed and defined by the scheduling approach, it can be performed by the SWfMS as a independent task of the workflow and executed before the task that will consume the data. This way, when the scheduled time to execute the task comes, all data will be already placed in the correct VM. Thus, it is clear that data distribution and task distribution are not independent problems and have to be analyzed together by the scheduling approach.

Finally, besides the impact of transfer data, we have also to consider a set of data constraints. These constraints usually define that some data cannot (or it is not recommended to) be moved (we call this as static data files), because it is either too big or for proprietary reasons (*e.g.* the Brazilian Internet law defines that all data produced by Brazilian federal universities and research centers should be stored in data-centers located in Brazil. All data cannot be moved or copied beyond the Brazilian frontier — actually this restriction motivated Amazon to create a data-center in the city of São Paulo). For example, if scientists are running phylogenetic analysis workflows [3] they commonly access the RefSeq database (www.ncbi.nlm.nih.gov/refseq/). This database is a static dataset, since it is unfeasible to transfer the entire dataset to the cloud (due to the huge volume of data). Thus, this requires that the scheduling approach "fixes" some tasks to specific VMs that execute as "near" as possible of static data files. Although these constraints "fix" some tasks to specific resources, they do not reduce the complexity of the workflow scheduling.

The aforementioned scenario leads to the development of several solutions for the workflow task scheduling problem in clouds [12,19–24]. However, these solutions do not consider data

distribution and task assignment together in the scheduling. Many of these solutions assume that all data generated is synchronized to all VMs (using Amazon S3, for instance) and just schedule the tasks independently of data placement, which can result in huge overheads. Thus, in this article, in order to reduce the total execution time of the workflow, we propose a scheduling approach that takes into account the variety and heterogeneity of VMs in a cloud virtual cluster (*e.g.* different bandwidths, transfer rates, and processing capacities), the data distribution and data constraints all together within the same solution, *i.e.* tasks and data transfers are scheduled together by the SWfMS. Thus, the scheduler defines the distribution of tasks and data among the several VMs to minimize the total execution time and the unnecessary data transfers while meeting the data constraints of the workflow. In order to validate this approach, we propose a Hybrid Evolutionary Algorithm (HEA) for scheduling tasks and data in the cloud named HEA-TaSDAP.

Our proposal assumes that: (i) tasks and data files of the workflow are allocated and executed exclusively in the cloud environment; (ii) each individual task is executed in a unique virtual machine sequentially (no preemption is admitted); (iii) the static scheduling is followed without changes during the workflow execution; (iv) the only possible communication between tasks is through data files (writing and reading operations); (v) there is enough space to store static and dynamic data files in the cloud environment, i.e., the total storage capacity of the cloud environment is greater or equal to the sum of all data files used by the workflow; and finally, (vi) static data files are previously allocated in virtual machines with enough storage to store them.

In this article we also assume that all VMs are deployed **before** the workflow execution. Although clouds traditionally provide On-Demand VMs, most of the providers also have reserved VMs to be used. Commonly, reserved VMs present a financial discount and capacity reservation. In long-running workflow executions that demand a heavy use of the VMs, reserved VMs can provide financial savings over executing the workflows only On-Demand instances. This way, this may be a common scenario for scientists that run their workflows in the cloud. Although this is an assumption of this article, in future work, we plan to include dynamic provisioning, when the reserved VMs are not enough to execute the workflow.

Thus, the main contributions of this article are:

I A new workflow model represented by a graph in which nodes correspond to data processing activities or data files, and arcs represent reading or writing operations
II The formulation of Task Scheduling and Data Assignment Problem as a mixed integer programming problem, named TaSDAP-IP
III The design of a Hybrid Evolutionary Algorithm (HEA) for scheduling tasks and data in the cloud named HEA-TaSDAP
IV An extensive experimental evaluation, based on synthetic and real executions in Amazon EC2 cloud using a real scientific workflow use case (SciPhy [3], a bioinformatics scientific workflow for phylogenetic analysis) that shows the advantages and benefits of the approach, compared with baseline algorithms.

The rest of the article is organized as follows: Section 2 presents related work. In Section 3, we describe the new proposed model and the task scheduling and data assignment problem. In Section 4, we introduce our HEA-TasDAP, a hybrid evolutionary algorithm for solving the task scheduling and data assignment problem. Section 5 brings the theoretical and practical experimental evaluation of the performance of our heuristic. Finally, Section 6 concludes the article and discusses future work.

2. Related work

Over the last decade, a growing number of data- and compute-intensive experiments has been modeled as scientific workflows. These experiments belong to several scientific fields and present different computational requirements and characteristics, such as total execution times, parallelism level and data transfer volume. In this scenario, the investigation of workflow scheduling strategies, aiming at reducing scientific workflow execution times, became a top priority and a very popular research area [22]. We can find in the literature several papers that propose workflow scheduling algorithms. Many of these algorithms are focused on computational grids, such as [25–29], which makes them not directly applicable to clouds. However, these algorithms inspired several approaches focused on clouds, such as [12,19–24]. Although there are several approaches to schedule workflows in clouds, each one considers different characteristics and requirements, so the development of efficient workflow scheduling algorithms for clouds is still an open, yet important, problem in this research area.

Juve et al. [30] characterized and profiled several scientific workflows from different domains of science. In their characterization they present the time needed to execute each task of the workflow and the total amount of files produced and consumed during a workflow execution. The size of data files varies from 2 GB (SIPHT workflow) to more than 200 TB (Cybershake workflow). It is worth mentioning that the Montage workflow produces data files with several GB and only 30% of CPU time is allocated to computation (70% of the CPU time is consumed by I/O operations). Shibata et al. [31] also analyzed the Montage workflow and concluded that 99% of the CPU time (in some scenarios) is allocated to I/O operations. Shibata et al. claim that scheduling approaches that are aware of data location are able to reduce the data transfer by 96%. Several papers consider the data location and I/O costs when scheduling a scientific workflow in a heterogeneous environment. Following we discuss some of these approaches.

The heterogeneous Earliest-Finish-Time (HEFT) is a well-know scheduling heuristic proposed by Topcuoglu et al. [32]. HEFT is an extension of the classical list scheduling algorithm for homogeneous systems which includes heterogeneous systems [33]. The heuristic is able to schedule applications modeled as a DAG and has two phases: *task prioritizing phase*, that computes the priorities of each task (rank-values); and *machine selection phase*, that selects a task according to its priority, and schedules it on the machine that minimizes its finishing time. In the first phase, tasks are sorted based on their rank-values. The rank-values are computed according to the computation required by each task and its predecessors tasks. After that, in the second phase, HEFT chooses the biggest rank-value task and assigns it to the machine that is able to execute it with the earliest finish time. This step is repeated for each task, until all tasks are scheduled. HEFT presents acceptable run times even in huge instances. However, HEFT does not consider data files during the scheduling process.

The MinMin Task Schedule Heuristic (MinMin-TSH) [34] is a widely-used scheduling algorithm. This is a task-based greedy algorithm that allocates each ready-to-run task to a machine based only on the local information of that task. At each iteration, the finish time of each task is estimated for each machine. The task that presents the minimum execution time over all tasks is chosen to be scheduled first and then it is assigned to the machine whose expected completion time is the earliest. This step is repeated until all tasks are scheduled. It is worth noticing that all data files generated by a task are assigned to the same machine where the task is executed. According to Blythe et al. [34] the idea behind this heuristic is that at each iterative step, the makespan has a small increase, and hopefully resulting in a small makespan for the whole workflow. However, the MinMin-TSH does not consider

the workflow graph (data dependencies) and a specific choice in a specific iteration may impact other tasks in the future.

Pandey et al. [19] presents a Particle Swarm Optimization (PSO) based algorithm, whose objective is minimizing the financial costs of workflow executions in clouds. Similarly to Pandey et al., Rodriguez and Buyya [35] also use the PSO metaheuristic for the same problem. However, differently from Pandey et al. [19], Rodriguez and Buyya propose a global scheduling algorithm that explores the dynamic allocation of VMs, considering possible variations in the Quality-of-Service (QoS) of the cloud environment. Their goal is to minimize the total execution time, respecting the deadline defined by the scientist. Differently from the approach proposed in this article, Pandey et al. and Rodriguez and Buyya do not consider the data assignment in the scheduling. Both approaches consider that a task and all data generated by it, are stored in the same VM. Moreover, they do not consider any limit in the storage capacity of a VM, which could lead to unfeasible solutions in some cases, mainly in data-intensive workflows.

Byun et al. [36] propose the Partitioned balanced time scheduling (PBTS) heuristic, whose objective is to minimize the execution time, respecting the deadline defined by the scientist. The algorithm constructs the scheduling iteratively based on the financial cost of using a certain time quantum of a VM in a commercial cloud. In Amazon EC2, for example, the user pays for time quanta of 60 min. At each interval of time, PBTS defines and allocates a minimum number of computational resources (VMs) that is sufficient to schedule the tasks that are ready to be executed. This algorithm considers the task execution time and data transfer times and assumes that the communication between tasks are exclusively via Amazon S3. In our work we choose a communication based on direct data transfer which is different from communication via the shared storage system; this allows exploring the allocation of files individually on each machine. Besides that, according to Juve et al. [37], the Amazon S3 performed poorly on workflows with a large number of small files, with are common in scientific workflows.

Abrishami, Naghibzadeh and Epema [38] propose the algorithm IC-PCP (IaaS Cloud Partial Critical Paths). This algorithm constructs subsets of tasks based on critical paths of the workflow. Each set is scheduled to one VM, that is chosen based on the financial cost and the capacity of executing the subset, respecting a given deadline provided by scientists. The algorithm prefers to choose VMs that are already in use. However, when it is not possible, the algorithm deploys the cheapest machine capable of executing the subset while also respecting the deadline. The search for critical paths and the whole scheduling process are repeated until there is no remaining task to be scheduled. Although the task grouping in subsets may reduce the data transfer costs, the locality of data files is not explicitly explored in this approach. Actually, most of the work in the related literature neglects the data assignment during the scheduling which is an important and challenging aspect of scientific workflow scheduling. Usually, these workflows use/transfer a massive volume of data, which can cause a significant and negative impact in the total execution time of workflows.

In Szabo et al. [21], the impact of data transfer in the scheduling of data-intensive scientific workflows is discussed. The authors argue that since the volume of data transferred between tasks of workflows have increased a lot, the data and task assignment problems cannot be treated independently. The authors propose an evolutionary algorithm that optimizes the task scheduling aiming at minimizing the data transfer among tasks and the total execution time of the workflow.

The metaheuristic proposed by Szabo et al. uses a storage and data transfer model based on Amazon S3. Moreover, the file allocation is driven only by the task assignment, i.e., the output file is written both in the S3 and in the machine where the task that generated it is assigned. As a result, only tasks that were allocated to the same machine can take advantage of that file allocation. Others tasks have to download the file from the shared storage S3. In our work the allocation of tasks and files considers, among other things, the distribution of all tasks and files in the virtual environment. Although it was our intention to compare the results of Szabo et al. with ours, the description of the approach given in Szabo's paper did not allow us to reproduce the presented results in terms of quality of solutions (the source code was not available either). Thus, in order to present a fair comparison in our paper, we have opted to drop the comparison with Szabo's work and compare with traditional scheduling approaches such as HEFT [32] and Min–Min [34].

Yuan et al. [20] use a clustering technique [39] based on a dependency matrix to identify data files which will be consumed (*i.e.* shared) by the same tasks in the workflow. The main idea is to store these data files in the same data center to reduce data transfer between different data centers. It is worth noticing that such approach assigns a task to the data center in which most of the input data is stored. The scheduling task problem and data assignment are not deeply treated in this approach, that ignores the scheduling and data assignment in VMs within the same data center.

Wang et al. [28] present a solution whose objective is to minimize the data transfer of workflows allocated in multiple data centers. This work defines the initial data locality (*i.e.* static data) by using the *k*-means clustering algorithm [39]. Task replication techniques are used to reduce data transfer, which are generated during the workflow execution (dynamic data), between different data centers. Besides the intrinsic complexity of replication techniques, such as maintenance of data consistency, the approach proposed by Wang et al. does not tackle the data transfer problem within data centers.

Bryk et al. [22] propose a model to execute workflow ensembles in clouds. They present the File Locality-Aware Scheduling Algorithm, which benefits of data locality to increase the performance of workflow executions. The algorithm executes a dynamic task scheduling, in which, at each step, the tasks that are ready to be executed are scheduled to available VMs. The scheduler prioritizes some tasks, considering the data locality, aiming at minimizing data transfer (*i.e.* it tries to choose a VM "near" to data). The execution model considers that both storing and data transfer use a global storage and each VM keeps only data files generated in it during the workflow execution. Thus, although the file locality is not defined by the scheduling algorithm, tasks that consume the same file are preferably allocated in the same VM that contains such a file, avoiding in these cases data transfer between machines.

Çatalyürek et al. [40] presented an algorithm that treats data file and task allocation of workflows executed in clouds. The workflow is modeled as a hypergraph. They propose a partitioning algorithm whose objective is dividing the workflow in k parts (where k is the number of virtual machines), each one to be associated with a different virtual machine, and minimizing the number of file transfers. Besides that, the partition must meet the balancing previously defined by the user. Their work does not consider environment characteristics such as processing and storage capacity and transfer rates. Only the total size of all transferred files are used to evaluate the solution quality.

To the best of author's knowledge, the approach proposed in this article is the first one that considers both task and data as vertices in the workflow graph. This way, we can consider both problems together by proposing a model and an algorithm to minimize the makespan of workflows in cloud environments. Our approach considers different storage capacity of machines, different tasks execution times and data transfer times in these heterogeneous systems. The following section formalizes the problem considered in this article.

3. The Task Scheduling and Data Assignment Problem - TaSDAP

Workflows are abstractions used to model a coherent flow of several scientific applications, which can be executed locally or in distributed systems. Each workflow is composed by multiple steps or activities, which are necessary to produce the outcome of a specific experiment [2]. Each activity may correspond to several executable tasks. Each task is considered the smallest unit of processing and may execute in parallel by consuming a different portion of the input data or parameter values. From the scientist's point of view, the workflow is composed by a few activities (*i.e.* the scientist only considers the abstract representation of the workflow). However, from the execution point of view, each workflow may be composed by several thousands (or even millions) of tasks. Since the approach proposed in this paper is focused in scheduling workflow tasks in clouds, we consider the representation of the workflow from the execution point of view.

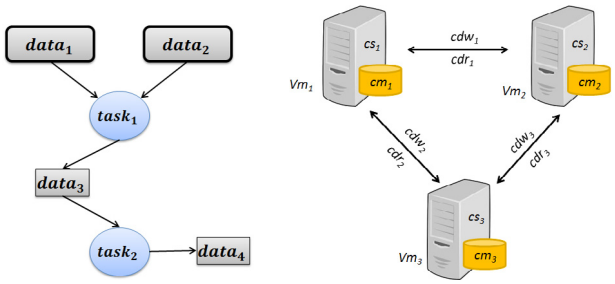
Commonly, a workflow is defined by a Directed Acyclic Graph (DAG) in which each task is represented by a vertex, and each data file or dependency between tasks is represented by an arc between the vertices. In this context, workflow-based scheduling aims at mapping the execution of tasks with dependency constraints on shared resources (*e.g.*, VMs) [41]. Usually, the scheduling methods, proposed in related work, consider that all data files are already placed in some machines or are synchronized among all machines. Differently from these approaches, this work proposes a new workflow model and, based on it, the Task Scheduling and Data Assignment Problem (TaSDAP) is presented. TaSDAP considers that determining the machine where the data file generated during the workflow execution is assigned, is also a crucial step in the workflow scheduling problem.

In the last decade, scientists started migrating their scientific experiments to clouds, where computing resources (infrastructure, platform, and software) are provided to users on demand via the Internet [42]. Motivated by this scenario, we choose to tackle the TaSDAP considering the strength and the potential of cloud environments. The following Sections 3.1 and 3.2 present the application and architecture models for the TaSDAP and the proposed mathematical formulation, respectively.

3.1. Application and architecture models

The Task Scheduling and Data Assignment Problem (TaSDAP) considers a class of parallel applications represented by DAGs (Directed Acyclic Graphs), and denoted by $G = (V, A, a, \omega)$. Differently from the current mainstream, data files are no longer represented as arcs. They are now represented as the set of vertices, similarly to the tasks. Thus, $V = N \cup D$ consists of tasks $i \in N$ and data files $d \in D$; A stands for the set of arcs, which gives the precedence relation between tasks and data files, a_i is the amount of work associated with task $i \in N$, and ω_{mn} represents the cost associated with arc $(m, n) \in A$. The set of immediate predecessor tasks of a task $i \in N$ is defined as $pred(i) = \{j \in N \mid \exists d \in D,$ such that $(j, d) \in A \wedge (d, i) \in A\}$. Similarly, the set of immediate successors is given by $succ(i) = \{j \in N \mid \exists d \in D,$ such that $(i, d) \in A \wedge (d, j) \in A\}$. In the graph, a task is always preceded and succeeded by a data file as illustrated in Fig. 1a, where $task_1$ reads data files $data_1$ and $data_2$ as needed for its execution, and writes $data_3$, which will be read later by $task_2$, which also writes the data file $data_4$.

The architectural model specifies the main features of the target architecture. In order to better reproduce the reality of a large-scale system, let M be the set of all VMs available for execution or storage. Each VM $j \in M$ has a storage capacity cm_j and computational slowdown index cs_j, which is inversely proportional to the computational power cp_j of VM j, according to Silva et al. [43]. In other words, cs_j represents the degree of difference of the processing capacity of the different available VMs and it is an alternative to the representation of the execution time of the tasks through matrices.



(a) Application model. (b) Architectural model.

Fig. 1. Problem definition models.

Moreover, a communication delay index cd_l estimates the latency cost associated with each link l of the system. We propose two types of communication delay here. One for the writing cases, cdw_l, and one for the reading cases, cdr_l. For supporting our model, two communication delay matrices are built, each one concerning one type of delay. Therefore, the execution time of a task $i \in N$ on a VM $j \in M$ is given by $t_{ij} = a_i \times cs_j$. Furthermore, the communication time for a task $i \in N$ executing on VM $j \in M$, to write data $d \in D$ in VM $p \in M$, where j and p are connected by link l, is given by $\overleftarrow{t}_{djp} = \omega_{id} \times cdw_l$. Similarly, the communication time for reading is given by $\overrightarrow{t}_{djp} = \omega_{di} \times cdr_l$. Fig. 1b illustrates an example of the architectural model containing three VMs and its characteristics: (i) the storage capacity cm, (ii) the computational slowdown index cs, (iii) the communication delay index for write operations (cdw) and (iv) the communication delay index for read operations (cdr).

For example, to clarify the proposed models, consider that a task i took one hour to be completely executed on a specific VM j, considered by us as a default VM. This default VM has a computational power, cp_j, expressed in Gflops and we assumed that its slowdown cs_j is equal to 1. Let j' be another VM, with a computational power $cp_{j'}$. The slowdown index $cs_{j'}$ is obtained in relation to the default VM j as: $cs_{j'} = \frac{cp_j}{cp_{j'}}$. So, if the default VM has $cp_j = 30$ Gflops and VM j' has $cp_{j'} = 20$ Gflops, the slowdown index $cs_{j'}$ will be equal to 1.5 and if task i is executed in j' its execution time will be one hour and thirty minutes. Similarly, consider that the communication time to write data d in a specific link l, considered by us as a default link, is one minute. This link l has a certain bandwidth, bw_l expressed in Mbps and we assume that its communication delay index cdw_l is equal to 1. Let be l' another link, with other bandwidth $bw_{l'}$. The communication delay index $cdw_{l'}$ is obtained in relation to the default link l as: $cdw_{l'} = \frac{bw_{l'}}{bw_l}$. Thus, if the default link l has $bw_l = 100$ Mbps and link l' has $bw_{l'} = 200$ Mbps, the $cdw_{l'}$ will be equal to 0.5 and if link l' is used in this write operation the communication time for writing will be thirty seconds. Remark that the communication time for reading is obtained in the same manner, using index $cdr_{l'}$.

Note that when data is read by some VM, we assume that it is held in main (volatile) memory. Thus, read operations do not require available storage capacity to store data. Besides that, since scientific workflows can be modeled using different types of programs and they are black-box ones, we do not know their behavior in relation to file consumption and production. Some programs, for example, have to load all data before processing, while others can start processing before data is totally loaded. Similarly, many programs only write data to the disk when they finish the execution. We have chosen to consider the worst case, where the input file has to be completely available so that the

program starts executing and the output files can only be written when execution finishes. As future work we intend to tackle the data replication problem in our model. In this way, read operations can take advantage of the nearest replica.

3.2. Mathematical formulation

The TaSDAP can be formulated as the mixed integer programming problem, named TaSDAP-IP, as described below. Let us redefine $D = D_s \cup D_d$ as the set of all data, where each data $d \in D$ has a size $W(d)$ and can be either static (D_s), with an origin machine $O(d) \in M$, or dynamic, generated during the workflow execution, (D_d). For each $i \in N$, we consider a set of input data $\Delta_{in}(i) \subseteq D$ needed for their execution and a set of output data $\Delta_{out}(i) \subseteq D_d$ generated by it. Moreover, we define T_M as the maximum execution time for the workflow and $T = \{1...T_M\}$ as the set of feasible periods.

In this vein, the TaSDAP is defined as the problem of scheduling tasks and assigning data on VMs, respecting the available storage capacity and trying to minimize the makespan. Next, we summarize the data and variables used in the proposed mathematical formulation and present the TaSDAP-IP.

Data	***Description***
D_s	Set of static data.
D_d	Set of dynamic data.
$D = D_s \cup D_d$	Set of data.
$O(d)$	Origin machine of static data $d \in D_s$.
$W(d)$	Size of data $d \in D$.
N	Set of tasks.
a_i	Amount of work of task $i \in N$.
M	Set of VMs.
t_{ij}	Processing time of task $i \in N$ in VM $j \in M$.
$\overrightarrow{t}_{djp}$	Time spent by VM $j \in M$ to read the data $d \in D$ stored in VM $p \in M$
$\overleftarrow{t}_{djp}$	Time spent by VM $j \in M$ to write the data $d \in D_d$ stored in VM $p \in M$.
$\Delta_{in}(i) \subseteq D$	Set of data needed for the execution of task $i \in N$.
$\Delta_{out}(i) \subseteq D_d$	Set of data generated by task $i \in N$.
cm_j	Storage capacity of the VM j.
Variables	***Description***
x_{ijt}	Binary variable which indicates whether task $i \in N$ begins its execution in VM $j \in M$ at period $t \in T$ or not.
$\overrightarrow{x}_{idjpt}$	Binary variable which indicates whether task $i \in N$ executing in VM $j \in M$ begins to read data $d \in \Delta_{in}(i)$ stored in VM $p \in M$ at period $t \in T$ or not.
$\overleftarrow{x}_{djpt}$	Binary variable which indicates whether data $d \in D_d$ begins to write from VM $j \in M$ to VM $p \in M$ at period $t \in T$ or not.
y_{djt}	Binary variable which indicates whether data $d \in D$ is stored in VM $j \in M$ at period $t \in T$ or not.
z_T	Continuous variable which indicates the total time to execute the workflow (makespan).

The objective function (1) minimizes the application makespan and it is subject to the following constraints.

$$\min z_T \tag{1}$$

Constraints (2) guarantee that every task must be executed. Constraints (3) and (4) rule that every read and write operations must be accomplished, respectively.

$$\sum_{j\in M}\sum_{t\in T} x_{ijt} = 1, \ \forall i \in N \tag{2}$$

$$\sum_{j,p\in M}\sum_{t\in T} \overrightarrow{x}_{idjpt} = 1, \qquad \forall i \in N, \forall d \in \Delta_{in}(i) \tag{3}$$

$$\sum_{j,p\in M}\sum_{t\in T} \overleftarrow{x}_{djpt} = 1, \qquad \forall d \in D_d. \tag{4}$$

Inequalities (5) guarantee that data $d \in \Delta_{out}(i)$ can only be written, if task i was executed in the correct time. Furthermore, constraints (6) rule that data d cannot be written before the processing time of the task i (responsible for its writing). Note that both sets of constraints ((5) and (6)) work together to guarantee a feasibly time for the writing process.

$$\overleftarrow{x}_{djpt} \leq x_{ij(t-t_{ij})}, \qquad \forall d \in D_d, \forall j, p \in M, \\ \forall t = (t_{ij}+1)\cdots T_M \text{ such } d \in \Delta_{out}(i) \tag{5}$$

$$\overleftarrow{x}_{djpt} = 0 \qquad \forall d \in D_d, \forall j, p \in M, \\ 1 \leq t \leq t_{ij} \text{ such } d \in \Delta_{out}(i). \tag{6}$$

Constraints (7) rule that a task can only be executed, if all necessary reads were concluded in a feasible time.

$$x_{ijt} \leq \sum_{p\in M} \overrightarrow{x}_{idjp(t-\overrightarrow{t}_{djp})}, \qquad \forall i \in N, \forall d \in \Delta_{in}(i), \forall j \in M, \\ \forall t \in T, \text{ such } (t - \overrightarrow{t}_{djp}) \geq 1. \tag{7}$$

Inequalities (8) guarantee that only one action (execution, reading or writing) can be accomplished at each period of time in each VM (e.g. a VM cannot execute a task and write data at the same time).

$$\sum_{i\in N}\sum_{q=\max(1,t-t_{ij}+1)}^{t} x_{ijq} + \sum_{d\in D_d}\sum_{p\in M}\sum_{r=\max(1,t-\overleftarrow{t}_{djp}+1)}^{t} \overleftarrow{x}_{djpr} \\ + \sum_{i\in N}\sum_{d\in\Delta_{in}(i)}\sum_{p\in M}\sum_{r=\max(1,t-\overrightarrow{t}_{djp}+1)}^{t} \overrightarrow{x}_{idjpr} \leq 1, \forall j \in M, \forall t \in T. \tag{8}$$

Constraints (9) establish that there are no dynamic data at starting time. On the other hand, constraints (10) guarantee that all static data are already stored on their origin machines.

$$y_{dj1} = 0, \qquad \forall d \in D_d, \forall j \in M \tag{9}$$

$$y_{djt} = 1, \qquad \forall d \in D_s \mid j \in O(d), \forall t \in T. \tag{10}$$

Constraints (11) and (12) link the storage variable y with the write variable $\overleftarrow{x}$ and the read variable $\overrightarrow{x}$, guaranteeing a feasible write and read process, respectively.

$$y_{dp(t+1)} \leq y_{dpt} + \sum_{j\in M} \overleftarrow{x}_{djp(t-\overleftarrow{t}_{djp})}, \\ \forall d \in D, \forall p \in M, \forall t \in T, \text{ such } (t - \overrightarrow{t}_{djp}) \geq 1 \tag{11}$$

$$\sum_{j\in M} \overrightarrow{x}_{idjpt} \leq y_{dpt}, \qquad \forall i \in N, \forall d \in \Delta_{in}(i), \forall p \in M, \forall t \in T. \tag{12}$$

In detail, constraint (12) ensures that data will only be read if previously stored in a VM, and constraint (11) ensures that data will only be stored in a VM, if it has been already produced (written).

The VMs storage capacity are bounded by constraints (13).

Constraints (14) relate the last write operation with the application execution time (makespan). Note that in our application

model a task always creates data.

$$\sum_{d \in D} y_{djt} W(d) \leq cm_j, \qquad \forall j \in M, \forall t \in T \tag{13}$$

$$\overleftarrow{x}_{djpt} \cdot (t + \overleftarrow{t}_{djp}) \leq z_T, \qquad \forall d \in D_d, \forall j, p \in M, \forall t \in T. \tag{14}$$

Moreover, the following operational constraint (15) must be satisfied: a task i can only begin any reading process if all data $d \in \Delta_{in}(i)$ is already available (*i.e.*, if all data $d \in (\Delta_{in}(i) \cap D_d)$ is written). Finally, the remaining constraints are the integrality and non-negativity constraints.

$$\overrightarrow{x}_{idjpt} \cdot |\Delta_{in}(i) \cap D_d| \leq \sum_{g \in \{\Delta_{in}(i) \cap D_d\}} \sum_{l,o \in M} \sum_{u=1}^{t - \overleftarrow{t}_{glo}} \overleftarrow{x}_{glou},$$
$$\forall i \in N, \forall d \in \Delta_{in}(i), \forall j, p \in M, \forall t \in T. \tag{15}$$

4. HEA-TaSDAP: a hybrid evolutionary algorithm for solving TaSDAP

Evolutionary Algorithms (EA) [44] are optimization methods inspired in biology evolution mechanisms observed in nature. In an EA, each chromosome is an individual of a population which represents a possible solution to the problem. The search for the best solution is guided by a fitness function, which gives the quality of each chromosome. At each iteration of the algorithm, new individuals are generated through a crossover operation and the diversification of the population is obtained through the mutation function. In this article, a Hybrid Evolutionary Algorithm (HEA), called HEA-TaSDAP, which includes EA, local search and a path relinking method [45], was developed. According to Moscato et al. [46], differently from traditional EA, HEA explores the available knowledge on the problem to reach best results.

The HEA-TaSDAP is presented in Algorithms 1 and 2. These algorithms are composed by five main procedures: (i) initial population generation; (ii) crossover; (iii) local search; (iv) mutation; and (v) path relinking. Each one of these steps are explained next.

Algorithm 1: HEA-TasDaP_Main

```
Output: The best solution found
P ← InitialPopulationGeneration()
best ← getBest(P)
Elite_set ← ∅
iteration ← 0
while iteration ≤ MAX_ITERATION do
    if shouldDoLocalSearch() then
        P ← doLocalSearch(P)
    current_best ← getBest(P)
    if fitness(current_best) better than fitness(best) then
        best ← current_best
        if Elite_set ≠ ∅ then
            best_pr ← doPathRelinking(best, Elite_set)
            if fitness(best_pr) better than fitness(best) then
                P ← P \ best ∪ best_pr
                best ← best_pr
        if distance(best, Elite_set) ≥ α then
            Elite_set ← Elite_set ∪ best
            if size(Elite_set) > β then
                removeFirstInChromosome(Elite_set)
    P ← doNextPopulation(P) (Algorithm 2)
    iteration = iteration + 1
return best
```

Algorithm 2: doNextPopulation

```
Input: Current Population P
Output: A New Population P'
Children ← ∅
for i ← 1 to n do
    parent_1 ← tournamentSelection(P)
    parent_2 ← tournamentSelection(P)
    new_chromosome ← crossover(parent_1, parent_2)
    new_chromosome' ← mutate(new_chromosome)
    Children ← Children ∪ new_chromosome'
    ComputeFitness(Children) (Algorithm 3)
AllSolutions ← Children ∪ P
P' ← getBest(AllSolutions)
while size(P') < MAX_POPULATION_SIZE do
    chromosome ← tournamentSelection(AllSolutions)
    P' ← P' ∪ chromosome
    AllSolutions ← AllSolutions \ chromosome
return P'
```

4.1. Representation of a chromosome

In TaSDAP, a feasible solution must respect the precedence order among tasks. A task can only be executed when the input data files are available. It occurs either when all tasks which generated them have already finished their executions or the data files are static ones. For example, in Fig. 1a, $task_2$ depends on the data file generated by $task_1$. So $task_2$ can only be executed after $task_1$.

In this article, a chromosome is composed by two structures, which represent: (i) task and data assignments and (ii) the execution order of tasks. This representation was inspired by the ideas of Szabo et al. [21], and allows both procedures, local search and calculation of the fitness function, to be simplified, as presented following in Sections 4.5 and 4.8, respectively. As can be seen in Fig. 2a, the first structure, that represents the task and data assignments, is a vector where indexes represent tasks and dynamic data files, and each element represents the VM where the task or the data file is assigned.

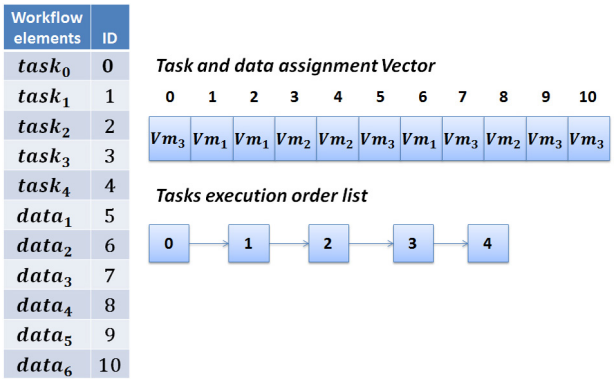
The inclusion of data assignment representation in this vector is an important difference between our work and the others of the related literature. This approach allows for the proposed metaheuristic treating not only the task scheduling problem but the data assignment problem as well, which can reduce the workflow execution time [31].

The task execution order is represented by a linked list, see Fig. 2a for an example. In order to define the task execution order, we associate to each one a height. A task can only execute when all tasks associated with a smaller height have already finished. Tasks with same height execute concurrently. The height value is given by Eqs. (16) and (17), proposed by Tsujimura [47].

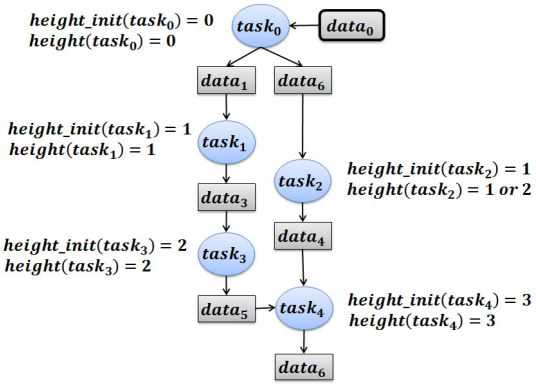
$$height\_init(task_i) = \begin{cases} 0, & \text{if } pred(task_i) = \emptyset \\ 1 + \max\limits_{task_j \in pred(task_i)} height\_init(task_j), & \text{otherwise} \end{cases} \tag{16}$$

$$height(task_i) = \begin{cases} height\_init(task_i), & \text{if } succ(task_i) = \emptyset \\ rand \in [height\_init(task_i), \min\limits_{\forall task_k \in succ(task_j)} \{height(task_k)\} - 1], & \text{otherwise.} \end{cases} \tag{17}$$

At first, Eq. (16) assigns to a task an initial height corresponding to the order in which it can be executed; it can be zero when it has no predecessors, or the maximum of the initial heights among all predecessors plus one. After that, in Eq. (17), a random component is introduced allowing that such tasks can execute in any order

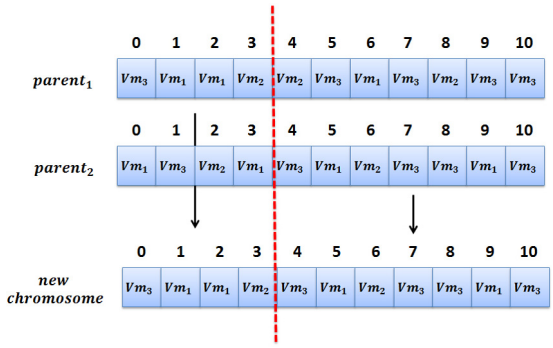


(a) Chromosome representation.

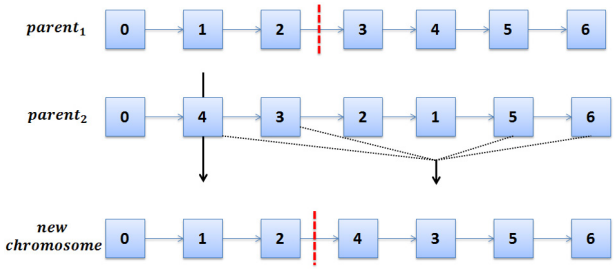


(b) Height of tasks.

Fig. 2. Chromosome encoding.



(a) Crossover operator in the task and data assignments vector.



(b) Crossover operator in the tasks execution order list.

Fig. 3. Crossover operators.
Source: Adapted from [21].

between the previous calculated height (initial height) and the minimum of heights among all its successors less one. For example, in Fig. 2b, $task_2$ can be associated either to height equal 1, its previous calculated height, or to height equal 2, the height of its successor less one. Therefore, $task_2$ can run in parallel either with $task_1$ or $task_2$.

Note that different sequences of tasks can be generated to encode the task execution order list. This feature is exploited in the search procedures as it will be seen in Sections 4.3 and 4.5.

4.2. Initial population generation

To generate the initial population two approaches were applied. The first one uses both MinMin-TSH [34] and HEFT [32] to generate 80% of the initial population. Since these heuristics are deterministic and always produce the same solution to the same input, we applied a random component in each generated solution to ensure that the population is diverse. Firstly, 40% of solutions are generated from the solution produced by MinMin-TSH and we applied a random movement in λ% genes in the task and file assignment vector, where λ ranges from 5% to 100% (increases in 5% at each new solution). The same approach is used to generate the other 40% of chromosomes, but at that point the chromosomes are derived from the solution produced by HEFT. The intuition behind this approach is that it ensures that initial population has a satisfactory diversity and presents a good start point for the search method.

The second approach is used to generate the other 20% of chromosomes. In that approach, each task and dynamic data file is assigned to a VM randomly chosen. The execution order of each task is given by (17), that is calculated for each generated chromosome. In both approaches, it may happen that a dynamic data file is assigned to a VM that does not have enough storage capacity to keep it. When that occurs, a proposed heuristic, named *Move-file*, is executed to re-assign the dynamic files so that restriction is not violated. The heuristic *Move-file* is presented in Section 4.7.

4.3. Crossover operation and selection phase

The crossover operation combines two individuals among the best ones to form an offspring. In the Algorithm 2, to generate a new solution, two chromosomes are chosen from current population P using the tournament selection algorithm [48]. Then, a single-point crossover operator [49] is applied in both structures that compose these chromosomes. Fig. 3 illustrates this function. Initially, two points of cut are defined, one for the task and data assignment vector and other for the list with the order of tasks execution. In the vector, shown in Fig. 3a, the genes on the left-hand side of the cut are copied from $parent_1$ to the corresponding positions of the offspring. The remaining genes, right-hand side of the cut, are copied from $parent_2$.

Concerning the tasks execution order (Fig. 3b), genes from the left-hand side of the cut are copied from $parent_1$ to the genes of the same position of the offspring. The other genes of the offspring are obtained by checking each gene, in the same order they appear in $parent_2$, and copying it to the offspring whenever it has not been inserted in it yet. Thus, the precedence order of tasks is kept in the offspring.

In the selection phase (Algorithm 2, from line 9 to 14), the chromosomes for the next population are selected from the set *AllSolutions*, which is formed by the newly generated chromosomes and the current population P. At first, to ensure that best solution will always be included in the population, an elitism selection is applied in line 10, where the best solution is found and add to next population P'. After that, the remaining solutions are chosen

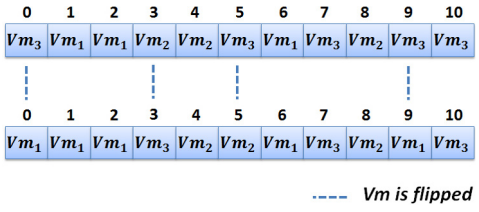


Fig. 4. Mutation operation.
Source: Adapted from [21].

using the tournament selection algorithm and included in *P'* as well. Note that, when a chromosome is selected it is removed from the *AllSolutions* set to prevent that the same chromosome is added more than once to the *P'*.

4.4. Mutation operation

The mutation operation is responsible for the diversification of the population. It enlarges the search space trying to escape from local optima solutions. In this work, the mutation is executed only in the task and data assignment vector. Our tests showed that the use of mutation operation in the structure of the tasks order list does not result in significant gains in terms of quality of solutions, and, additionally, it increases the computational cost of the operation.

The mutate operator is call for each new solution generated by the crossover procedure in the Algorithm 2, on condition that each gene has a 10% probability of being modified. As can be seen in Fig. 4, the VM identification *Vm* is randomly changed. At the end, the fitness of the chromosome is recalculated and the new chromosome replaces the old one.

4.5. Local search procedures

After applying the crossover and mutation operators to build a new population, local search procedures are applied on 15% of the best solutions with a probability of 50%. These values showed the best results and were defined based on empirical tests made to adjust the parameters of the algorithm. How this is done can be seen in Algorithm 1 line 6, for each iteration the local search probability is checked and, in the case of it is true, the local search procedures are executed.

In this work, three local search procedures were defined and are executed in the following order: (i) *swap-vm*: swap two elements in the task and data assignment vector; (ii) *swap-position*: swap two elements of the same height in the task order list; and (iii) *move-element*: move one task or data file to a different machine. Each local search procedure is executed until it obtains an improvement (first-improvement) or until all combinations have been tested.

Fig. 5 presents the three types of local search used in HEA-TaSDAP. Fig. 5a presents the execution of *swap-vm* procedure that, after testing several swaps, exchanges the elements of position 1 and 10 of the task and data assignment vector. Fig. 5b also tests several swaps, until exchanging the order of execution of tasks 4 and 5, which have the same height, in the task execution order list. Finally, in Fig. 5c, a task or data file is moved from VM m_2 to m_3.

4.6. Path relinking

Path relinking is a heuristic capable of generating intermediate solutions between two other ones. It starts with an initial solution and, step by step, inserts elements of a target solution. Its goal is to visit new solutions in the path from one solution to another one [50]. So, during the path relinking execution, the distance between the solutions diminishes gradually.

In the context of TaSDAP, the distance between a chromosome chr_1 and another chr_2 is the number of tasks and data files assigned to different VMs plus the number of swaps necessary to make their task order lists exactly the same. Fig. 6 shows the distance of chromosomes chr_1 and chr_2. The number of tasks or data files placed in different VMs is five and the number of necessary swaps is one, so $dist(chr_1, chr_2) = 6$.

In this work, the path-relinking procedure is applied to an elite set whenever a best solution is found. As can be seen in Algorithm 1 (line 11 and 12), if the elite set was not empty, the path-relinking is applied taking as input the newly found best solution. The elite set is composed of the best solutions whose distances between each other are greater than α, where α is calculated as 25% of the number of tasks and data files. This parameter controls the elite set diversification and, consequently, the search efficiency. Besides that, up to β chromosomes can be added to the set, where β is equal to half of population size. As showed in lines 18 and 19 in Algorithm 1, when this limit is reached a replacement policy First in First out (FIFO) is applied to update the elite set.

The path relinking procedure is applied between the new best solution and each chromosome of the elite set. If during the search a better solution is found, the *best* chromosome is replaced and include in the population, as it might be seen from line 12 to 15.

4.7. Move-file heuristic

When a new chromosome is generated, its feasibility is evaluated. If the data files assigned to a VM would exceed its storage capacity, the VM is considered overloaded and the heuristic *Move-file* is executed.

The heuristic *Move-file* contains the following steps:

1. Select the most overloaded VM, m_s.
2. Select the most underloaded VM, m_d.
3. Move the smallest size data file from VM m_s to m_d.

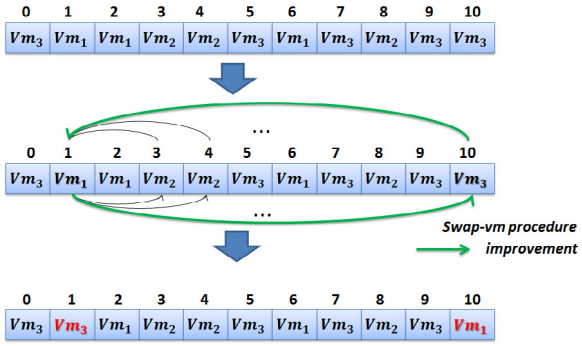
The above steps are repeated until there is no overloaded VM.

Based on the hypothesis that the assignments defined by the metaheuristic correspond to the best ones, it seems interesting not changing the original chromosome very much, since it could impact negatively on the makespan. In order to soften this probable impact, the smallest data files are selected to be moved.

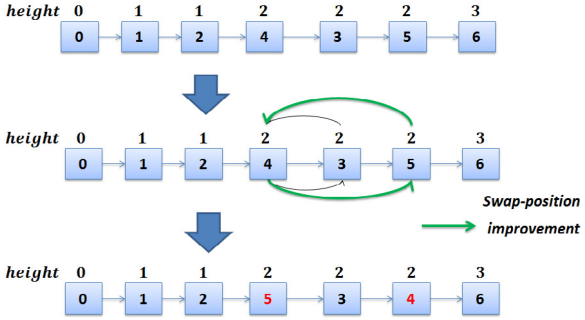
4.8. Fitness function

Let $f : s \rightarrow Z$ be a fitness function, where s is a feasible solution in the search space and Z is an integer number that quantifies the quality of a solution represented by a chromosome. So, the fitness function allows for ordering the solutions in terms of quality and to direct the search towards the best solutions [51].

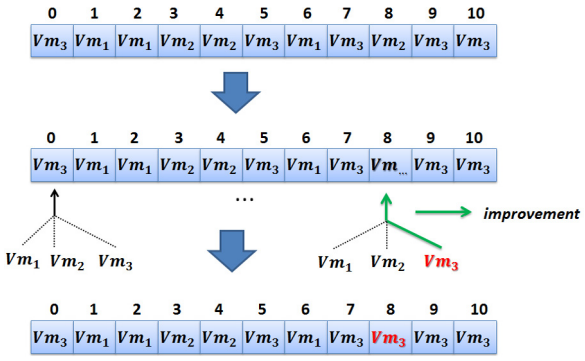
In TaSDAP, the quality of an individual is given by the makespan, *i.e.*, let chr_i and chr_j be two chromosomes, if $makespan(chr_i) < makespan(chr_j)$, then the solution given by chr_i is better than the one given by chr_j. To calculate the makespan, we define the Algorithm 3. As can be seen, the information in the structures *order*



(a) Swap machines in the task and data assignment vector.



(b) Swap tasks in the tasks execution order list.



(c) Moving a task or data file from one machine to another.

Fig. 5. Local search procedures.

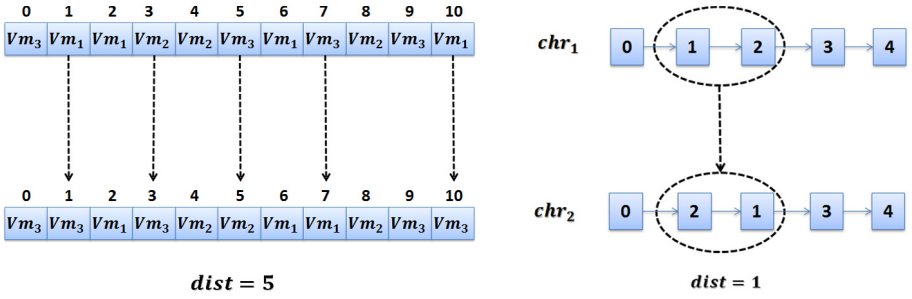


Fig. 6. Distance $dist(chr_1, chr_2) = 6$.

list (sorting by the tasks height values) and *assignment vector* are retrieved to compute the finish time of each task.

Algorithm 3: Compute Fitness

Input: A chromosome *chr*
Output: The value of the makespan

1 $Q \leftarrow 0$
2 $FT \leftarrow 0$
3 **for** $task_i$ ***in*** *chr.order_list* **do**
4 $\quad m_j \leftarrow chr.assigment\_vector[task_i]$
5 $\quad max\_pred\_time \leftarrow maxFinishTimePred(FT, task_i)$
6 $\quad start\_time_i \leftarrow \max(max\_pred\_time, Q[m_j])$
7 $\quad finish\_time_i \leftarrow start\_time_i + runTime(task_i, m_j) + readTime(task_i, m_j) + WriteTime(task_i, m_j)$
8 $\quad Q[vm_j] \leftarrow finish\_time_i$
9 $\quad FT[task_i] \leftarrow finish\_time_i$
10 **return** $\max(FT)$

In Algorithm 3, vectors Q and FT are auxiliary structures used to keep the times computed in every step of the algorithm. Vector Q is indexed by the VM identification (m_j) and contains the finishing time of the last task executed in the corresponding VM. Each element of vector FT corresponds to a $task_i$ and contains the finishing time of the associated task.

The calculations of the start and finishing times of each task follow the model presented in Section 3.1. Firstly, the start time of $task_i$ is computed as the maximum finishing time among all its immediate predecessors (represented by *max_pred_time*) and the finishing time of the last task executed in VM m_j (stored in $Q[m_j]$), as presented in line 6.

Then, in line 7, the finishing time of $task_i$ is computed by summing the following values: (i) the start time of $task_i$ ($start\_time_i$); (ii) the execution time of $task_i$ when executed in VM m_j; (iii) the necessary time for reading all required data files by $task_i$; and (iv) the necessary time for writing all data files generated by $task_i$. Finally, in line 10, the algorithm returns the makespan that is the maximum finishing time among all tasks of the workflow.

5. Experimental results

This section presents experiments conducted with HEA-TaSDAP to evaluate the proposed approach. We conducted two types of experiments: theoretical and practical ones. The central idea of this section is to compare the proposed scheduling approach with existing solutions that can be used to schedule workflows in clouds. It is worth mentioning that all schedules used in this section for experiments and all optimization code are available in a BitBucket Git repository (https://bitbucket.org/danielcmo/hea-tasdap).

5.1. Simulated experiments

We firstly evaluate HEA-TaSDAP, in terms of quality of solution and execution time, by comparing it with: (i) the solutions given by mathematical formulation TaSDAP-IP (solved with CPLEX 12.5.1); (ii) the MinMin-Task Scheduling Heuristic (MinMin-TSH) [34]; and (iii) the Heterogeneous Earliest Finish Time (HEFT) [32]. We have chosen MinMin-TSH and HEFT because these heuristics run in polynomial time, produce efficient schedules and have been used as baselines in related works [52,53,26].

Tests were run in a computer with a processor Intel Core i7-3770 CPU 3.40 GHz with 12Gb memory running Ubuntu 14.04. The HEA-TaSDAP, MinMin-TSH and HEFT were implemented with C/C++, and compiled with g++ version 5.3.0.

We use two types of synthetic instances in these experiments. The first one, used in the comparison between HEA-TasDap and TaSDAP-IP (Section 5.1.1), was randomly generated varying the number of tasks, data files and representations of workflows, as explained later. The second type, used in Section 5.1.2, was generated by the Workflow Generator [54] (https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator). This application generates instances of workflow executions for evaluation of algorithms and systems on a range of workflow sizes. All data within these synthetic workflows is gathered from real executions of scientific workflows on the grid and in the cloud from the Pegasus' team in ISI @ University of Southern California.

The metaheuristic HEA-TaSDAP was executed with a population of 50 individuals and the stopping criterion was satisfied after 100 iterations without improvement. Concerning the other parameters, the algorithm uses the ones defined in Section 4. It is worth mentioning that all parameters were adjusted empirically after several tests.

5.1.1. Comparing HEA-TaSDAP with the exact approach

HEA-TaSDAP and TaSDAP-IP were evaluated using a set of 12 random instances divided into 4 groups according to their sizes. For each group, three different instances were generated by varying the number of tasks, files, and the representation of the workflow. Two computational environments were defined for simulation, with 3 and 5 VMs. File sizes (MB) and the process execution time (seconds) were randomly set between 0.1 and 100. The storage capacity of 1 TB has been set, and the transfer link rates were 5, 10 and 15 MB/s. Finally, the slowdown of the machines were set between 0.01 and 1. Table 1 shows the amounts of tasks and files for each generated instance. In addition, the basic workflow structures found on them are presented according to the classification made by Bharathi et al. [55].

Table 2 summarizes the results obtained by exact approach and the HEA-TaSDAP metaheuristic. The first column identifies the instance. The following two columns present the results obtained by HEA-TaSDAP: makespan and the execution time to obtain the solution. Following, the next 2 columns present the same results for the exact approach. Finally, the last column, shows the gap between the solutions given by HEA-TaSDAP and the exact approach (mathematical formulation). The values shown for HEA-TaSDAP are averages of five executions. The standard deviations were zero for almost all instances, except 15*B_m*3, 15*B_m*5 and 15*A_m*5, which were less than 0.6.

By analyzing Table 2, we can observe that for all instances the HEA-TaSDAP obtains good solutions with small gaps, in average 1.1%. Moreover, it takes significantly less time when compared with the mathematical formulation when solved with the CPLEX, on average 1.5 s against 12, 640.3 s, respectively. Furthermore, CPLEX could not find a feasible solution for instances 10*A_m*5 and 15*C_m*5 within 24 h time limit. Finally, in instance 15*C_m*3, the CPLEX could not prove the optimality of the solution. Although these results are very encouraging, we have also evaluated HEA-TaSDAP with other heuristics and using a real workflow as presented following in this article.

5.1.2. Comparing HEA-TaSDAP with HEFT and MinMin-TSH

In order to compare the performance and the quality of the results of HEA-TaSDAP, HEFT and MinMin-TSH, the aforementioned synthetic workflows produced by workflow generator: Montage, Cybershake, Epigenomics and Inspiral workflows [30] were chosen. Each instance (workflow) has the following corresponding information: (i) a Direct Acyclic Graph representing tasks and data files of the workflow; (ii) execution time of each task in a machine with known processing capacity; and (iii) size of each input and output data file. Table 3 shows the main characteristics of each workflow.

We also considered the characteristics of the VMs from Amazon EC2 cloud environment. So, although we are theoretically evaluating the algorithms, the used data (structure of workflows and VMs characteristics) were acquired from real scenarios. The used cloud environment is composed by 4 VMs: 1 m3.medium (1 vCPU Intel Xeon E5-2670 v2); 1 m3.large (2 vCPU Intel Xeon E5-2670 v2); 1 m3.xlarge (4 vCPU Intel Xeon E5-2670 v2); and 1 m3.2xlarge (8 vCPU Intel Xeon E5-2670 v2). We also considered the following information about each VM: (i) network bandwidth; (ii) storage capacity; and (iii) processing capacity.

It is worth mentioning that the execution time of a task is defined as the product between a basis time and the machine slowdown where it will be executed. A machine slowdown is defined as $\frac{P_B}{P_{m_j}}$, where P_B is the processing capacity of the machine used to calculate the basis time, and P_{m_j} is the processing capacity of the virtual machine of our experiment. So, the slowdown represents the processing capacity of a virtual machine when compared with the machine used to calculate the basis time. In our experiments, the basis time and the processing capacity P_B were obtained from instances of the Workflow Generator, while P_{m_j} is the processing capacity of the virtual machine from Amazon EC2. The transfer-rate parameter was estimated through practical experiments using the tool Iperf, available in https://iperf.fr.

Table 4 presents the makespan (the average of 5 executions), Relative Standard Deviation (RSD) and the execution time of HEA-TaSDAP. It also presents the makespan of MinMin-TSH and HEFT, and since they are deterministic heuristics, the result is always the same for each execution of the algorithm. The execution times of MinMin-TSH and HEFT are not presented, since they are negligible (less than 1 min). Note that, HEA-TaSDAP outperforms with respect to makespan both MinMin-TSH and HEFT algorithms.

In fact, these results were already expected, since variations of solutions given by MinMin-TSH and HEFT are used in initial population. It worth noticing that with our proposal we improved the quality of the makespan in all cases and with a acceptable run time (less than 9 min in the worst case). The HEA-TaSDAP showed an average improvement of 22.72% in relation of MinMin-TSH and 11.15% in relation of HEFT.

Analyzing Table 4 we can state that there is an improvement variation among all workflows. After analyzing all workflow structures and the volume of data transferred with the scheduling plans

Table 1
Description of instances used in the comparison with mathematical formulation.

Workflow	Basic structures	Number of tasks	Number of files
5A; 5B; 5C	Process	2	3
	Process; Data aggregation	2	3
	Data aggregation	1	4
7A; 7B; 7C	Data aggregation; Data redistribution	2	5
	Process	3	4
	Data aggregation; Pipeline	3	4
10A; 10B; 10C	Data aggregation	4	6
	Data redistribution	3	7
	Data distribution; Pipeline	3	6
15A; 15B; 15C	Data aggregation; Data distribution	5	10
	Data distribution; Pipeline	4	11
	Data aggregation; Data distribution	5	10

Table 2
Results of HEA-TaSDAP metaheuristic and the mathematical formulation using CPLEX.

Instance	HEA-TaSDAP		Mathematical formulation		Gap (%)
	Makespan	Time (s)	Makespan	Time (s)	
5A_m3	10.0	0.83	10.0	4.2	0.0
5B_m3	11.0	0.79	11.0	14.0	0.0
5C_m3	13.0	0.71	13.0	2.8	0.0
7A_m3	21.0	0.96	21.0	140.8	0.0
7B_m3	16.0	1.17	16.0	7.3	0.0
7C_m3	14.0	1.11	14.0	105.0	0.0
10A_m3	21.0	1.80	21.0	1644.6	0.0
10B_m3	12.0	1.74	12.0	21.4	0.0
10C_m3	21.0	1.22	21.0	316.7	0.0
15A_m3	16.0	2.47	16.0	2606.6	0.0
15B_m3	11.4	3.01	11.0	600.9	3.6
15C_m3*	19.2	2.42	94.0	86395.5	–
5A_m5	10.0	0.90	10.0	14.4	0.0
5B_m5	9.0	0.50	9.0	462.6	0.0
5C_m5	10.0	0.75	10.0	9.0	0.0
7A_m5	27.0	0.97	27.0	1836.4	0.0
7B_m5	26.0	1.26	26.0	21.0	0.0
7C_m5	16.0	1.09	16.0	2426.9	0.0
10A_m5*	25.0	1.58	–	86489.2	–
10B_m5	14.0	1.75	13.0	103.0	7.6
10C_m5	47.6	1.88	47.0	7353.5	1.2
15A_m5	16.0	2.64	16.0	15783.9	0.0
15B_m5	11.0	2.97	10.0	10764.3	10.0
15C_m5*	20.0	2.49	–	86243.0	–
Average	17.4	1.5	20.2	12640.3	1.1

Table 3
Scientific workflow attributes.[a]
Source: Adapted from [21].

Workflow	Type I/O; memory; CPU	Number of tasks	Number of data files (static and dynamic)
Cybershake	Low; low; medium	30; 50; 100	49; 79; 154
Epigenomics	Low; medium; high	24; 47; 79; 100; 127	38; 71; 119; 152; 192;
Montage	High; low; low	25; 50; 100	38; 53; 93
Inspiral	Low; medium; high	30; 50; 100	47; 77; 151

[a] Columns "Number of tasks" and "Number of data files" show different instance sizes used in our tests.

provided by HEA-TaSDAP, HEFT and MinMin-TSH (Table 5), we can state that those improvements variations are mainly due to the volume of data that is transferred among activities in each workflow. In fact, workflows that produce large files presented significant performance gains when using HEA-TaSDAP since it considers file allocation in its essence. As can be seen in Table 5, Cybershake_100 that presented the best makespan improvement reduced the amount of data transferred in 42.5% and 36.6% when compared with MinMin-TSH and HEFT, respectively. On the other hand, Inspiral_100 that presented the worst makespan improvement reduced the amount of data transferred in 23% and 6.4% when compared with MinMin-TSH and HEFT, respectively. This fact corroborates our analysis since Inspiral_100 has many small files. Even when using MinMin-TSH and HEFT (that do not consider file allocation) the volume of data transferred does not impact the makespan considerably.

5.2. Evaluation of HEA-TaSDAP with a real workflow

This subsection presents the evaluation of the scheduling provided by HEA-TaSDAP using a real workflow from the bioinformatics domain. To achieve that, we have chosen the SWfMS SciCumulus to execute the schedule provided by HEA-TaSDAP. The following sections present the workflow used as case study, a brief description of the SWfMS used and the needed modifications, the environment configurations and the results achieved.

Table 4
Results of HEA-TaSDAP, HEFT and MinMin-TSH.

Instance	MinMin-TSH	HEFT	HEA-TaSDAP		
	Makespan (min)	Makespan (min)	Makespan (min)	RSD	Run time (min)
Cybershake_30	11.98	11.28	10.25	0.0019	0.39
Cybershake_50	14.88	16.65	12.46	0.0264	2.19
Cybershake_100	26.93	28.08	14.52	0.0261	8.11
GENOME.d.3510	530.88	469.38	444.91	0.0109	4.05
GENOME.d.7020	923.46	865.45	833.98	0.0004	6.79
Epigenomics_24	67.36	57.30	55.80	0.0000	0.03
Epigenomics_46	119.95	104.07	99.27	0.0140	0.48
Epigenomics_100	1,004.23	916.73	889.65	0.0001	4.02
Montage_25	1.81	1.16	0.95	0.0242	0.05
Montage_50	3.06	2.08	1.88	0.0036	0.10
Montage_100	5.31	4.66	3.93	0.0038	3.33
Inspiral_30	18.26	14.61	13.95	0.0014	0.17
Inspiral_50	29.96	23.36	22.41	0.0005	0.79
Inspiral_100	44.65	41.80	40.76	0.0036	1.55

Table 5
Volume of data transferred (in MB) with the scheduling plans provided by HEA-TaSDAP, HEFT and MinMin-TSH.

Instance	MinMin-TSH	HEFT	HEA-TaSDAP
Cybershake_30	3,228.59	3,460.30	2,270.97
Cybershake_50	6,025.76	5,549.10	4,833.56
Cybershake_100	14,335.00	12,931.10	8,234.62
GENOME.d.3510	68,511.50	67,893.10	64,897.90
GENOME.d.7020	111,237.00	106,490.00	106,480.00
Epigenomics_24	8,344.20	7,958.83	7956.5
Epigenomics_46	14,489.30	12,871.90	12,607.00
Epigenomics_100	121,654.00	116,368.00	116,355.22
Montage_25	247.10	117.44	80.86
Montage_50	577.60	264.69	240.04
Montage_100	764.21	729.59	593.47
Inspiral_30	812.12	772.32	656.72
Inspiral_50	1,383.53	1,104.66	1,104.58
Inspiral_100	2,603.20	2,139.80	2,002.35

5.2.1. SciPhy: A workflow for phylogenetic analysis

This subsection presents the workflow used as a case study for a real workflow execution in this article. Many types of bioinformatics experiments are based on the outcome of a phylogenetic analysis, such as pharmacophylogenomics experiments, development of new drugs, *etc.* A phylogenetic analysis aims at producing phylogenetic trees, which are structures that show the inferred evolutionary relationships among homologous genes represented in the genomes of divergent species. Managing phylogenetic experiments is far from trivial, since they are compute- and data-intensive. As they are based on a pipeline of scientific programs, computational phylogenetic experiments can be modeled as a scientific workflow.

SciPhy [3] is one of the existing workflows for phylogenetic analysis. The SciPhy workflow is a parameter sweep analysis where the same workflow is executed for each input file in a given large input dataset. It is composed by four main activities: multiple sequence alignment (MSA), sequence conversion, search for the best evolutionary model, and construction of phylogenetic trees, and they respectively execute the following bioinformatics applications: MSA programs (MAFFT, Kalign, ClustalW, Muscle and ProbCons), ReadSeq, ModelGenerator, and RAxML.

The graph representing one execution of SciPhy is presented in Fig. 7. Each one of the circles represents a different activity execution in a VM and the squares represent data produced and consumed. The first activity of SciPhy (with associated tasks t1, t5 and tn) constructs an individual MSA using one of five available MSA programs – ClustalW, Kalign, MAFFT, Muscle, and ProbCons – with default parameters. Each MSA program receives a multi-fasta file as input (containing sequences of DNA, RNA and amino acids — Mf1, Mf2 and Mfm), then producing a MSA as output (Ms1, Ms2 and Msm). Each MSA is then converted to the phylip format in the second activity (with associated tasks t2, t6 and t($n+1$)) to be further processed. After being converted to the phylip format (Rs1, Rs2 and Rsm), it is tested at the third activity to find the best evolutionary model using ModelGenerator (with associated tasks (t3, t7, t($n+2$))), and both of them (individual MSA, converted MSA and evolutionary model) are used in the fourth activity (with associated tasks t4, t8 and t($n+3$)) to generate phylogenetic trees (T1, T2 and Tm) using RAxML with 100 bootstrap replicates [56]. Consequently, we can obtain several different trees for each one of the MSA programs and for several input multi-fasta files. In the experiments executed in this article we have chosen MAFFT as the MSA method.

Since we aim at executing a parameter sweep in SciPhy, each one of the activities is going to be executed for a different input file containing several sequences (multi-fasta file) thus generating several different tasks. Each one of these tasks can be performed in parallel. For more information about SciPhy, please refer to Ocaña et al. [3].

5.2.2. SciCumulus workflow system

This section briefly describes the SciCumulus SWfMS used as the computational infrastructure to execute SciPhy. SciCumulus provides support for two types of parallelism: parameter sweep [11] and data parallelism [57]. SciCumulus aims at distributing, controlling and monitoring parallel execution of scientific workflows in a cloud environment, such as Amazon EC2. SciCumulus is a distributed application as well and it is composed by 4 modules:

I Client Tier: The components of this module dispatches workflows to be executed in the cloud.
II Distribution Tier: It generates and manages the execution of activities in one or more VMs instantiated in one or more cloud environments.
III Execution Tier: It is responsible for executing programs invoked by workflow activities, generating data files and storing provenance data.
IV Data Tier: This tier is the provenance database, which stores all provenance data consumed and generated by the parallel execution of the workflow.

SciCumulus provides the minimal computational infrastructure to support workflow parallelism with provenance management. For the experiments presented in this article the scheduler of SciCumulus had to be modified. In the current and available version of SciCumulus, the scheduler is based on a greedy and dynamic approach. As a VM becomes idle, it requests a task and then the scheduler decides the best task to be executed in that VM at that

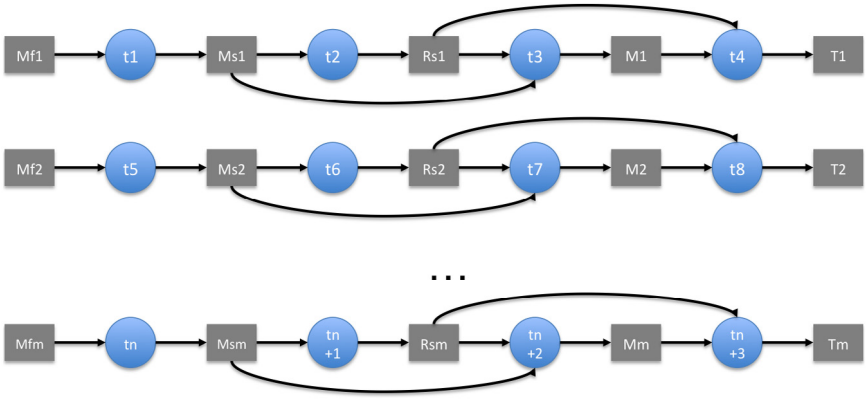


Fig. 7. A graph representing a single execution of SciPhy.

specific moment. This is a simple approach and provides good solutions especially when we face performance variations in the VMs, but since it is based on a greedy algorithm, it is not guaranteed that the optimal solution is provided.

In the version of SciCumulus used in this experiment, the HEA-TaSDAP is invoked before each workflow execution. It is provided with a list of tasks, the estimated execution time, the number of data files to be produced and consumed and the amount of VMs that are part of the virtual cluster in the cloud. All input data for HEA-TaSDAP is queried and retrieved from the provenance database [58]. HEA-TaSDAP then creates a scheduling plan that is returned to SciCumulus. This scheduled plan is loaded into a database table (since SciCumulus is database-oriented) and this table is queried by the scheduler every time a VM is idle. For more information about SciCumulus, please refer to Oliveira et al. [23]. As aforementioned, all scheduling plans provided by HEA-TaSDAP and the executable code will be available on the URL https://bitbucket.org/danielcmo/hea-tasdap.

5.2.3. Environment setup

For the experiments executed in this article, we have deployed the 2 aforementioned versions of SciCumulus on top of Amazon EC2. Amazon EC2 is the most popular commercial cloud and many scientific and commercial applications are being deployed on it. Amazon EC2 provides several different types of VMs for deployment and use. In the experiments presented in this paper we have considered 2 types of VMs: m3.medium (1 CPU and 3.75 GB RAM) and m3.2xlarge (8 virtual CPU and 30 GB RAM). We have instantiated 1 m3.small and 2 m3.2xlarge following the recommendations of GraspCC [59], a GRASP-based approach for dimensioning the cloud environment for scientific workflows.

Each deployed VM presented in this article is based on the Linux Cent OS 7 (64-bit), and was configured with the necessary software and libraries, and the bioinformatics applications. All VMs were configured to be accessed using SSH. Additionally, these VMs are based on a AMI image (ami-7e1a1716) that is also stored in the cloud and SciCumulus creates the virtual cluster to execute the experiment based on this AMI. In terms of software, all VMs, no matter their type, execute the same programs and configurations. All VMs were deployed in the US East - N. Virginia location and follow the pricing rules of that locality. In addition, the executions of SciPhy were performed in a single site of the Amazon EC2 cloud. We did not consider the execution of SciPhy in Multisite clouds neither federated clouds.

5.2.4. Experiment setup

To execute SciPhy, we have used as input a dataset of multi-fasta files of protein sequences extracted from RefSeq release 75 — March 14, 2016. This dataset is formed by 25 amino acid multi-fasta files and each multi-fasta file is constituted by an average of 20 sequences. Once downloaded, each input multi-fast file is stored in one m3.2xlarge VM. In order to generate large data files for the experiment, each multi-fasta file had its size artificially increased. Although this does not produce useful and meaningful results from the biological perspective (since the sequences are replicated several times within the same fasta file), it is suitable to evaluate the performance of HEA-TaSDAP. The following program versions were used in the experiments: MAFFT version 6.857 (Multiple Sequence Aligment), ReadSeq 2.1.22 (Sequence Conversion), ModelGenerator version 0.85 (Search for the best evolutionary model) and RAxML-7.2.8-ALPHA (Construction of the phylogenetic tree). For ModelGenerator, we considered the following evolutionary models: BLOSUM62, CPREV, JTT, WAG, and RtREV. Each execution of SciPhy with these configurations generated 100 activity executions and produced 125 data files.

5.2.5. Experimental results of sciphy

This subsection presents the execution time of SciPhy using a greedy scheduling algorithm [23], MinMin-TSH, HEFT and HEA-TaSDAP scheduling plans in SciCumulus system. The results are presented in Fig. 8. SciPhy was executed 5 times for each approach in order to achieve statistical significance. As presented in Fig. 8, the total execution time of SciPhy when using greedy scheduling was 273.1 min on average, was 224.6 min when using MinMin-TSH, was 214.5 using HEFT and when using HEA-TaSDAP was 198.2 min on average. It represents approximately 27.4% of improvement when using HEA-TaSDAP in comparison with the greedy algorithm of SciCumulus, 11.7% of improvement in comparison with MinMin-TSH and 8.1% of improvement in comparison with HEFT. This difference is due to the data file assignment of HEA-TaSDAP. Most of the bigger data files were placed in VMs with high bandwidth. In case of greedy scheduling and MinMin-TSH, some data files with several GB were placed in VMs with low bandwidth. The difference of the estimated time (presented in Table 6) and the real execution time (presented in Table 7) may be caused by several factors such as the VM performance variations, the provenance data used as input for estimation may be not accurate, deployment time, etc. In fact, in Table 6 we considered the makespan as the period of time between the time that the user started the execution in the SWfMS and the time that the SWfMS informs that the execution finished. Thus, we consider the time needed for deploying VMs in the cloud as well the time needed for undeploying these VMs. The time needed for deployment cannot be negligible since the used SWfMS needs to wait for all VMS to be ready to start execution, so the deployment time is the maximum deployment time of all VMs in the virtual cluster instantiated for the workflow execution.

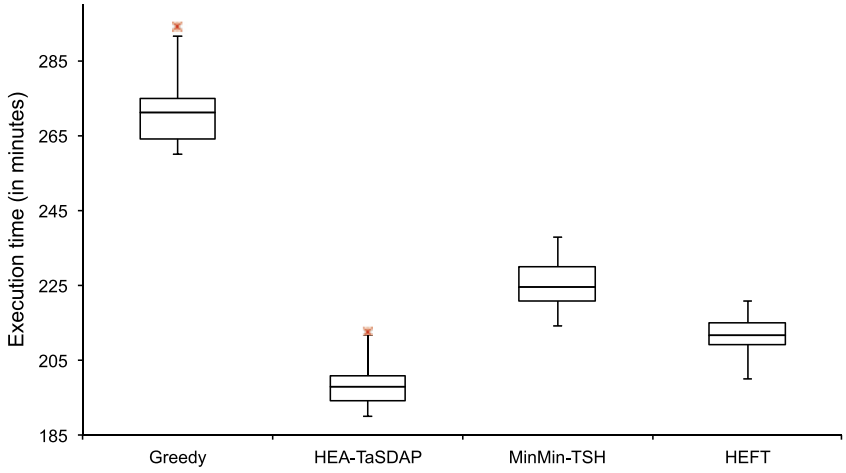


Fig. 8. Boxplot comparing the executions of SciPhy using HEA-TaSDAP, Greedy, HEFT and MinMin-TSH algorithms.

Table 6
Estimated workflow execution time — values in minutes.

Approach	Average
MinMin-TSH	219.31
HEFT	168.10
HEA-TaSDAP	133.36

Table 7
Statistics of the SciPhy execution — values in minutes.

Approach	Average	Median	Min	Max	Q1	Q3
Greedy	273.1	271.0	260.3	295.9	264.1	275.2
HEA-TaSDAP	198.2	198.7	190.3	213.2	194.4	201.8
HEFT	214.5	211.5	198.5	221.7	209.5	215.2
MinMin-TSH	224.6	224.6	214.2	238.0	221.9	230.6

In addition, in the execution with the greedy scheduling approach, all data is synchronized for all 3 VMs using Amazon S3, which implies in a certain overhead for every data file produced. The overall statistics of the execution (median, min, max, Quartile 1, Quartile 3 and Average) are also presented in Table 7. It is worth noticing that there were neither failures nor performance variations in the VMs during the workflow execution. If there were failures or performance variations, the execution using HEA-TaSDAP, HEFT and MinMin-TSH would probably produce worse results since they consider a static set of VMs during the entire execution. Nevertheless, the results are promising and more tests are planned using different workflows such as Montage and Ligo.

6. Concluding remarks

Data- and compute-intensive scientific experiments usually produce a huge volume of data and present a long duration where several executions, using different parameters and input data, are necessary to draw conclusions. These experiments are commonly modeled as scientific workflows, which are composed by a chain of activities and each activity can be further decomposed in a set of tasks. Each task is associated with the execution of an activity for a specific portion of data or set of parameter values. These tasks have to be executed in parallel in HPC environments to produce results in a feasible time with good performance.

However, it is far from trivial to manage the parallel executions of data- and compute-intensive scientific workflows, particularly in clouds. One important issue to consider when executing workflows in parallel in clouds is how to schedule the multiple tasks to a set of heterogeneous VMs. Task scheduling is a well-known NP-complete problem even in its simplest form. But to schedule the tasks to VMs we have to consider several requirements such as the processing capacity of the VM, its storage capacity and the impact of data transfer. In this article we claim that data distribution and task distribution are not independent problems and have to be analyzed together by the scheduling approach. In previous work [23] we have addressed workflow execution in clouds using a greedy scheduling algorithm but it does not consider data placement in the scheduling algorithm, which may result in severe overheads. Thus, to increase the uptake of the cloud paradigm for executing scientific workflows that demand HPC capabilities, new scheduling solutions have to be developed.

In this article, we propose a new model for representing workflows, a mathematical formulation of The Task Scheduling and Data Assignment Problem and a scheduling algorithm based on Hybrid Evolutionary Algorithm (HEA) named HEA-TaSDAP that takes into account the variety and heterogeneity of VMs in a cloud virtual cluster (*e.g.* different bandwidths, transfer rates, and processing capacities), the data distribution and data constraints all together within the same solution, *i.e.* tasks and data files are scheduled together by the SWfMS.

We have evaluated HEA-TaSDAP using synthetic and real workflows in the cloud using SciCumulus. The performance evaluation of the HEA-TasDAP using synthetic workflows in comparison with the exact solution showed that for all instances HEA-TaSDAP obtains good solutions with small gaps, on average 2.6%. Moreover, it takes significantly less time when compared with the mathematical formulation when solved with the CPLEX, in average 1.5 s against 1942.7 s, respectively. In the real execution, we used SciPhy as a case study and compared HEA-TaDASP with the greedy scheduling of SciCumulus, MinMin-TSH and HEFT. The execution time of SciPhy when using the scheduling plan provided by HEA-TaSDAP was about 27.4% smaller than the one using the greedy algorithm, 11.7% smaller than MinMin-TSH and 8.1% smaller than using HEFT. As future work, we plan to use the proposed approach in conjunction with a fault-tolerance and a dimensioning mechanism within the scheduler to distribute backups of tasks and then execute the workflow with more reliability. Another interesting work could involve considering other objectives in our model like minimization of financial costs and power consumption, respecting deadlines. In the future the model can be improved and consider more details regarding the environment, for example, in the reading and writing operations the latencies related to the used storage system could be considered, once it varies a lot depending on the applied technology and their corresponding properties. Finally, we also intend to tackle the data replication problem in our models, as previous commented in Section 3.1.

Acknowledgments

This work was partially funded by CNPq (grants 308966/2015-5 and 305995/2013-8), CAPES (grant 130177/2015-6) and FAPERJ (grant E-26/203.215/2016). The authors also acknowledge the Amazon EC2 for providing HPC and database resources that have contributed to the research results reported within this article.

References

[1] M. Mattoso, C. Werner, G.H. Travassos, V. Braganholo, E. Ogasawara, D.D. Oliveira, S.M. Cruz, W. Martinho, L. Murta, Towards supporting the life cycle of large scale scientific experiments, Int. J. Bus. Process Integr. Manage. 5 (1) (2010) 79+.

[2] I.J. Taylor, E. Deelman, D.B. Gannon, M. Shields, Workflows for e-Science: Scientific Workflows for Grids, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[3] K. Ocaña, D. de Oliveira, E.S. Ogasawara, A.M.R. Dávila, A.A.B. Lima, M. Mattoso, SciPhy: A cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes, in: O.N. de Souza, G.P. Telles, M.J. Palakal (Eds.), BSB, in: Lecture Notes in Computer Science, vol. 6832, Springer, 2011, pp. 66–70.

[4] G.M. Guerra, S. Zio, J.J. Camata, J. Dias, R.N. Elias, M. Mattoso, P.L.B. Paraizo, A.L.G.A. Coutinho, F.A. Rochinha, Uncertainty quantification in numerical simulation of particle-laden flows, Comput. Geosci. 20 (1) (2016) 265–281.

[5] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good, The cost of doing science on the cloud: The montage example, in: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, in: SC'08, IEEE Press, Piscataway, NJ, USA, 2008, pp. 50:1–50:12.

[6] J.M. Wozniak, T.G. Armstrong, M. Wilde, D.S. Katz, E. Lusk, I.T. Foster, Swift/T: Scalable data flow programming for many-task applications, SIGPLAN Not. 48 (8) (2013) 309–310.

[7] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A.C. Laity, J.C. Jacob, D.S. Katz, Pegasus: A framework for mapping complex scientific workflows onto distributed systems, Sci. Program. 13 (3) (2005) 219–237.

[8] S.P. Callahan, J. Freire, E. Santos, C.E. Scheidegger, C.T. Silva, H.T. Vo, VisTrails: Visualization meets data management, in: SIGMOD International Conference on Management of Data, in: SIGMOD'06, ACM, New York, NY, USA, 2006, pp. 745–747.

[9] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M.P. Balcazar Vargas, S. Sufi, C. Goble, The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud, Nucleic Acids Res. 41 (W1) (2013) W557–W561.

[10] I. Altintas, B. Ludaescher, S. Klasky, M.A. Vouk, Introduction to scientific workflow management and the Kepler system, in: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, in: SC'06, ACM, New York, NY, USA, 2006.

[11] E. Walker, C. Guiang, Challenges in executing large parameter sweep studies across widely distributed computing environments, in: Proceedings of the 5th IEEE Workshop on Challenges of Large Applications in Distributed Environments, in: CLADE'07, ACM, New York, NY, USA, 2007, pp. 11–18.

[12] J. Liu, V. Silva, E. Pacitti, P. Valduriez, M. Mattoso, Scientific workflow partitioning in multisite cloud, in: Euro-Par 2014: Parallel Processing Workshops: Euro-Par 2014 International Workshops, Porto, Portugal, August 25–26, 2014. Revised Selected Papers, Part I, in: CLADE'07, ACM, New York, NY, USA, 2014, pp. 105–116.

[13] F. FC, C. A, Competitive science: is competition ruining science? Infect. Immun. 83 (4) (2015) 1229–1233.

[14] L.M. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner, A break in the clouds: Towards a cloud definition, SIGCOMM Comput. Commun. Rev. 39 (1) (2008) 50–55.

[15] J. Liu, E. Pacitti, P. Valduriez, M. Mattoso, A survey of data-intensive scientific workflow management, J. Grid Comput. 13 (4) (2015) 457–493.

[16] L. Youseff, M. Butrico, D.D. Silva, Toward a unified ontology of cloud computing, in: 2008 Grid Computing Environments Workshop, 2008, pp. 1–10.

[17] D. Oliveira, F.A. Baião, M. Mattoso, Towards a taxonomy for cloud computing from an e-Science perspective, in: N. Antonopoulos, L. Gillam (Eds.), Cloud Computing, in: Computer Communications and Networks, Springer London, 2010, pp. 47–62.

[18] J.D. Ullman, Polynomial complete scheduling problems, SIGOPS Oper. Syst. Rev. 7 (4) (1973) 96–101.

[19] S. Pandey, L. Wu, S.M. Guru, R. Buyya, A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments, in: 24th International Conference on Advanced Information Networking and Applications, AINA, IEEE, 2010, pp. 400–407.

[20] D. Yuan, Y. Yang, X. Liu, J. Chen, A data placement strategy in scientific cloud workflows, Future Gener. Comput. Syst. 26 (8) (2010) 1200–1214.

[21] C. Szabo, Q.Z. Sheng, T. Kroeger, Y. Zhang, J. Yu, Science in the cloud: Allocation and execution of data-intensive scientific workflows, J. Grid Comput. 12 (2) (2013) 245–264.

[22] P. Bryk, M. Malawski, G. Juve, E. Deelman, Storage-aware algorithms for scheduling of workflow ensembles in clouds, J. Grid Comput. (2015) 1–20.

[23] D. Oliveira, K.A.C.S. Ocaña, F. Baião, M. Mattoso, A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds, J. Grid Comput. 10 (3) (2012) 521–552.

[24] D.E. de Oliveira, C. Boeres, A. Fausti, F. Porto, Avaliação da localidade de dados intermediários na execução paralela de workflows big data, in: In Proc. of the XXX Brazilian Simposyum on Databases, 2015, pp. 29–40.

[25] J. Yu, R. Buyya, K. Ramamohanarao, Workflow scheduling algorithms for grid computing, in: Metaheuristics for Scheduling in Distributed Computing Environments, Springer, Berlin, Heidelberg, 2008, pp. 173–214.

[26] J. Yu, R. Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms, Sci. Program. 14 (3–4) (2006) 217–230.

[27] W.N. Chen, J. Zhang, An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements, IEEE Trans. Syst. Man Cybern. C 39 (1) (2009) 29–43.

[28] M. Wang, J. Zhang, F. Dong, J. Luo, Data placement and task scheduling optimization for data intensive scientific workflow in multiple data centers environment, in: Advanced Cloud and Big Data (CBD), 2014 Second International Conference on, 2014, pp. 77–84. http://dx.doi.org/10.1109/CBD.2014.19.

[29] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, M. Livny, Pegasus: Mapping scientific workflows onto the grid, in: Grid Computing: Second European AcrossGrids Conference, AxGrids 2004, Nicosia, Cyprus, January 28–30, 2004. Revised Papers, Springer, Berlin, Heidelberg, 2004, pp. 11–20.

[30] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, Future Gener. Comput. Syst. 29 (3) (2013) 682–692. http://dx.doi.org/10.1016/j.future.2012.08.015.

[31] T. Shibata, S. Choi, K. Taura, File-access patterns of data-intensive workflow applications and their implications to distributed filesystems, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, in: HPDC'10, ACM, New York, NY, USA, 2010, pp. 746–755.

[32] H. Topcuouglu, S. Hariri, M.-y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274. http://dx.doi.org/10.1109/71.993206.

[33] H. Zhao, R. Sakellariou, An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm, in: H. Kosch, L. Böszörményi, H. Hellwagner (Eds.), Euro-Par 2003 Parallel Processing: 9th International Euro-Par Conference Klagenfurt, Austria, August 26–29, 2003 Proceedings, Springer, Berlin, Heidelberg, 2003, pp. 189–194. http://dx.doi.org/10.1007/978-3-540-45209-_28.

[34] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, K. Kennedy, Task scheduling strategies for workflow-based applications in grids, in: CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005, vol. 2, 2005, pp. 759–767. http://dx.doi.org/10.1109/CCGRID.2005.1558639.

[35] M.A. Rodriguez, R. Buyya, Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds, IEEE Trans. Cloud Comput. 2 (2) (2014) 222–235.

[36] E.-K. Byun, Y.-S. Kee, J.-S. Kim, S. Maeng, Cost optimized provisioning of elastic resources for application workflows, Future Gener. Comput. Syst. 27 (8) (2011) 1011–1026.

[37] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B.P. Berman, P. Maechling, Data sharing options for scientific workflows on amazon ec2, in: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, in: SC'10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 1–9. http://dx.doi.org/10.1109/SC.2010.17.

[38] S. Abrishami, M. Naghibzadeh, D.H. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Gener. Comput. Syst. 29 (1) (2013) 158–169. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[39] A. Broder, L. Garcia-Pueyo, V. Josifovski, S. Vassilvitskii, S. Venkatesan, Scalable *k*-means by ranked retrieval, in: Proceedings of the 7th ACM International Conference on Web Search and Data Mining, in: WSDM'14, ACM, New York, NY, USA, 2014, pp. 233–242.

[40] U.V. Çatalyürek, K. Kaya, B. Uçar, Integrated data placement and task assignment for scientific workflows in clouds, in: Proceedings of the Fourth International Workshop on Data-Intensive Distributed Computing, in: DIDC'11, ACM, New York, NY, USA, 2011, pp. 45–54. http://dx.doi.org/10.1145/1996014.1996022. URL http://doi.acm.org/10.1145/1996014.1996022.

[41] R.S.C. Navjot Kaur Taranjit Singh Aulakh, Comparison of workflow scheduling algorithms in cloud computing, Int. J. Adv. Comput. Sci. Appl. 2 (10) (2011).

[42] L.M. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner, A break in the clouds: towards a cloud definition, SIGCOMM Computer Commun. Rev. 39 (1) (2008) 50–55.
[43] J.M. Silva, C. Boeres, L.M. Drummond, A.A. Pessoa, Memory aware load balance strategy on a parallel branch-and-bound application, Concurr. Comput. : Pract. Exper. 27 (5) (2015) 1122–1144.
[44] A. Moraglio, Geometry of evolutionary algorithms, in: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, in: GECCO'12, ACM, New York, NY, USA, 2012, pp. 1317–1344.
[45] C.R. Reeves, T. Yamada, Genetic algorithms, path relinking, and the flowshop sequencing problem, Evol. Comput. 6 (1) (1998) 45–60.
[46] P. Moscato, C. Cotta, A modern introduction to memetic algorithms, in: Handbook of Metaheuristics, Springer US, Boston, MA, 2010, pp. 141–183.
[47] Y. Tsujimura, M. Gen, Genetic algorithms for solving multiprocessor scheduling problems, in: Simulated Evolution and Learning: First Asia-Pacific Conference, SEAL'96 Taejon, Korea, November 9–12, 1996. Seclected Papers, Springer, Berlin, Heidelberg, 1997, pp. 106–115.
[48] B.L. Miller, D.E. Goldberg, Genetic algorithms, tournament selection, and the effects of noise, Complex Syst. 9 (3) (1995) 193–212.
[49] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence, MIT Press, Cambridge, MA, USA, 1992.
[50] F. Glover, M. Laguna, R. Marti, Fundamentals of scatter search and path relinking, Control Cybernet. 29 (3) (2000) 653–684.
[51] E.-G. Talbi, Metaheuristics: From Design to Implementation, Wiley Publishing, 2009.
[52] M. Rahman, S. Venugopal, R. Buyya, A dynamic critical path algorithm for scheduling scientific workflow applications on global grids, in: e-Science and Grid Computing, IEEE International Conference on, 2007, pp. 35–42.
[53] M.M. Lopez, E. Heymann, M.A. Senar, Analysis of dynamic heuristics for workflow scheduling on grid systems, in: 2006 Fifth International Symposium on Parallel and Distributed Computing, 2006, pp. 199–207. http://dx.doi.org/10.1109/ISPDC.2006.9.
[54] R.F.d. Silva, W. Chen, G. Juve, K. Vahi, E. Deelman, Community resources for enabling research in distributed scientific workflows, in: e-Science (e-Science), 2014 IEEE 10th International Conference on, vol. 1, 2014, pp. 177–184.
[55] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, K. Vahi, Characterization of scientific workflows, in: Workflows in Support of Large-Scale Science, 2008, WORKS 2008, Third Workshop on, 2008, pp. 1–10.
[56] A. Stamatakis, RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models, Bioinformatics 22 (21) (2006) 2688–2690.
[57] F. Coutinho, E. Ogasawara, D. de Oliveira, V. Braganholo, A.A.B. Lima, A.M.R. Dávila, M. Mattoso, Data parallelism in bioinformatics workflows using hydra, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, in: HPDC'10, ACM, New York, NY, USA, 2010, pp. 507–515.
[58] J. Freire, D. Koop, E. Santos, C.T. Silva, Provenance for computational tasks: A survey, Comput. Sci. Engg. 10 (3) (2008) 11–21.
[59] R. de C. Coutinho, L.M. Drummond, Y. Frota, D. de Oliveira, Optimizing virtual machine allocation for parallel scientific workflows in federated clouds, Future Gener. Comput. Syst. 46 (2015) 51–68.

Luan Teylo is a Ph.D. student at the Institute of Computing at Fluminense Federal University (UFF). He graduated in Computer Science from Mato Grosso Federal University (UFMT) in 2015 and obtained the M.S. degree in Computer Science from UFF (2017). His research interests includes distributed algorithms, cloud computing and optimization problems.

Ubiratam de Paula is a Professor of the Department of Computer Science at Federal Rural University of Rio de Janeiro (UFRRJ) since 2016. He graduated in Computer Science (2009), obtained the M.S. degree (2011) and the D.S. degree (2015) also in Computer Science from Fluminense Federal University (UFF). He has experience in computer science, acting on the following subjects: distributed algorithms, mathematical programming and cloud computing.

Yuri Frota is a Professor of the Department of Computer Science at Fluminense Federal University, where he has been since 2010. He graduated in Computer Science from UECE (1999), obtained an M.S. in Computer Science from UFC (2002) and a Ph.D. also in Computer Science from UFRJ (2008). He has experience in computer science, focusing on algorithms, and engaged in the following subjects: combinatorial optimization and integer programming.

Daniel de Oliveira is a Professor of the Institute of Computing of the Fluminense Federal University (UFF) since 2013. He received the Doctor of Science degree from the Federal University of Rio de Janeiro (UFRJ) in 2012. His current research interests include scientific workflows, provenance, cloud computing, high performance computing and distributed and parallel databases. He participates in research projects in those areas, with funding from several Brazilian government agencies, including CNPq, CAPES and FAPERJ. He is a member of several program committees of national and international conferences and workshops, and is a member of IEEE, ACM and of the Brazilian Computer Society.

Lúcia M.A. Drummond is a full Professor of the Institute of Computing of the Fluminense Federal University (UFF). She received the Doctor of Science degree from the Federal University of Rio de Janeiro (UFRJ) in 1994. Her current research interests are in high performance computing, cloud computing, performance optimization, parallel and distributed algorithms. She has taken part in several research projects in those areas, with funding from Brazilian government agencies and industry, including CNPq, CAPES, FAPERJ and PETROBRAS (Brazilian Oil Company). She has participated of several program and organization committees of national and international conferences and workshops. She has been engaged in the graduation and the graduate program, advising Master's and doctoral students at Fluminense Federal University.