

PSO-DS: a scheduling engine for scientific workflow managers

Israel Casas^{1,4} · Javid Taheri² · Rajiv Ranjan^{3,4} ·
Albert Y. Zomaya¹

Published online: 3 March 2017

© Springer Science+Business Media New York 2017

Abstract Cloud computing, an important source of computing power for the scientific community, requires enhanced tools for an efficient use of resources. Current solutions for workflows execution lack frameworks to deeply analyze applications and consider realistic execution times as well as computation costs. In this study, we propose cloud user–provider affiliation (CUPA) to guide workflow’s owners in identifying the required tools to have his/her application running. Additionally, we develop PSO-DS, a specialized scheduling algorithm based on particle swarm optimization. CUPA encompasses the interaction of cloud resources, workflow manager system and scheduling algorithm. Its featured scheduler PSO-DS is capable of converging strategic tasks distribution among resources to efficiently optimize makespan and monetary cost. We compared PSO-DS performance against four well-known scientific workflow schedulers. In a test bed based on VMware vSphere, schedulers mapped five up-to-date benchmarks representing different scientific areas. PSO-DS proved its efficiency by reducing makespan and monetary cost of tested workflows by 75 and 78%,

✉ Israel Casas
icas8033@uni.sydney.edu.au; israel_fime@hotmail.com

Javid Taheri
javid.taheri@kau.se

Rajiv Ranjan
raj.ranjan@ncl.ac.uk

Albert Y. Zomaya
albert.zomaya@sydney.edu.au

¹ School of Information Technologies, The University of Sydney, Sydney, Australia

² Department of Mathematics and Computer Science, Karlstad University, Karlstad, Sweden

³ Newcastle University, Newcastle upon Tyne, UK

⁴ Data61, CSIRO, Canberra, Australia

respectively, when compared with other algorithms. CUPA, with the featured PSO-DS, opens the path to develop a full system in which scientific cloud users can run their computationally expensive experiments.

Keywords Cloud computing · Scientific workflow · Scheduling algorithms · Particle Swarm Optimization · Virtual machine

1 Introduction

Cloud computing (CC) is a revolutionary solution for science and business communities demanding powerful computing services. In its *Infrastructure as a Service (IaaS)* deployment model, CC offers a virtualized hardware for those applications where customers require stronger control over computing services. A virtual machine (VM), the main unit of computation in *IaaS*, is almost identical to desktop machines. This flexibility allows users to build a network of VMs to create a strong tool of computing power. These tools have a great impact over the scientific community which has a strong dependency on computing tools to solve a wide range of problems, including scientific workflows.

A workflow is a group of simple processes to solve a complex problem. Scientific-oriented workflows cover a wide number of areas such as astronomy, geology, biology, cosmic analysis and biotechnology [1] as well as image processing [2,3]. These applications demand high computing power and large storage capacity. Current computing systems have enough capacity to execute these applications; CC possesses an attractive environment to run the scientific workflows due to (1) its extraordinary computing power and immense data storage capacity, (2) that contrary to grid computing systems, any person or institution can access cloud resources, (3) that users do not need to invest in costly systems as with supercomputers, (4) that as opposed to cluster computing systems, customers can scale up/down the numbers of resources and (5) that customers have immediate access to resources in contrast to supercomputing systems where a waiting list of weeks are common.

Workflow owners must first obtain the required tools in order to run their experiments on the cloud resources. To begin, the application owner must define which variables need to be optimized based on his/her needs. Secondly, the user must divide the application into subgroups and upload them into the available resources. Additionally, the user must set computer communication protocols to establish a connection between computers in order to execute parallel tasks. At the moment, existing tools offer solutions to the aforementioned issues. For instance, WMS (workflow manager systems) [4,5] produce a link between workflow owner and cloud resources. WMS deal with the necessary protocols in order to establish communication among computer resources to trigger experimentation. Even so, most WMS do not offer a critical analysis of application. To overcome this issue, researchers have developed different scheduling algorithms that analyze users' application in order to optimize selected optimization variables. However, specialized scheduling algorithms have not reached full computer efficiency. Additionally, cloud users need assistance in building a complete system with the aforementioned tools.

To date, different studies exist to guide cloud customers to execute their workflows on cloud environments; however, these approaches fail in at least one of the following ways: (1) no complete framework with time and cost estimations to execute workflow tasks; (2) do not present different options to the user to run his/her experiments; and (3) do not provide a specialized scheduler that considers inter-dependencies and/or large data files in their workflows [6–11]. CUPA, our proposed framework in this article, encompasses the aspects of workflow execution on cloud environments; it acts as a layer on top of a WMS. In contrast to common WMS, CUPA provides a specialized scheduler capable of analyzing an application prior to execution in order to discover file and task dependencies for an efficient distribution of resources.

For the aforementioned reasons, the objective of this study is to provide architecture for the execution of scientific workflows on cloud environments. In summary, this study outlines the following contributions: (1) a cloud architecture with focus on the relation of provider–customer in executing scientific workflows (i) execution time and cost estimation, (ii) feedback to user with the available options to execute his/her application and (iii) present the user all the stages he/she needs to follow in order to execute his/her application; and (2) a workflow scheduler based on the PSO metaheuristic able to analyze both computing- and data-intensive applications.

2 Related work

The background of this study has three foundations: workflow manager systems (WMS), schedulers and architectures for the execution of applications on cloud systems.

2.1 Workflow Manager Systems for Cloud Systems

The main objective of a WMS is to act as a link between cloud resources and users' workflows as expressed in Fig. 1. WMS produce the necessary configuration files to execute the workflows computer tasks over the available computer resources. Consequently, users do not need to deal with low-level specification such as data transfers and task dependencies. WMS offer different configuration options to distribute and execute dedicated workflows over computer systems. Numerous WMS were originally designed for grid systems. Even though cloud computing and grids systems run on similar technologies, for this reason, most of these WMS can be used on cloud systems with minor revision or even without any adjustments. Following is a list of famous WMS in use in the scientific community. The Pegasus system [4] is able to handle heterogeneous resources and manage applications ranging from a few tasks up to one million. This tool has given proof of its functionality on the Amazon EC2 as well as on the Nimbus cloud platform. HTCondor [5], a specialized system to manage computing-intensive tasks, has its main feature in executing tasks on dedicated and non-dedicated machines, i.e., computer resources being used by a person or other process which can be used by HTCondor whenever they become idle. Kepler [12], a workflow manager based on web services, provides a graphical interface to the user in visualizing the flow of job execution. It simplifies the required effort to create an

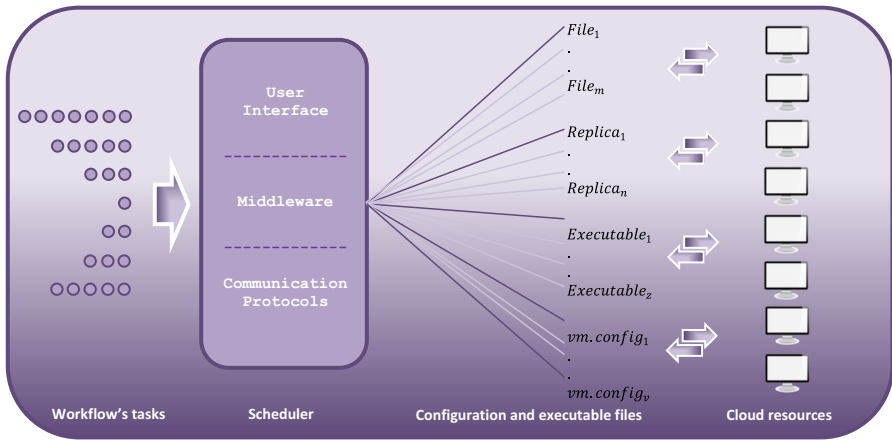


Fig. 1 Representation of the input and output of a workflow manager system. It receives workflow tasks and produces the required files that the systems need to interpret to execute workflows tasks

executable model by using a visual representation of a particular process. The Taverna [13] provides a friendly interface to the user to build their workflows. It also provides the ability to monitor previous and current workflows as well as intermediate and final results while executing the application. Similarly, SwinDeW-G [14] is a peer-to-peer manager system. It accepts workflow submission on any peer, and then it distributes it among near peers with enough capabilities. However, this tool needs further adaptation for use on cloud systems.

2.2 Schedulers

A scheduler analyzes an application and distributes its tasks among available resources in order to optimize particular objectives as represented in Fig. 2. It shows the representation of a workflow scheduler. A scheduler receives a workflow description, and then it defines a number of queues and distributes tasks to efficiently execute the application based on user demands. Workflows can be sent to resources following simple approaches such as queues or using more complex techniques. Whenever a user requires an efficient usage of cloud resources, then he/she requires a specialized scheduler focusing on his/her particular objectives. The following works present applicable scheduling algorithms for cloud environments. The HEFT (Heterogeneous Earliest-Finish-Time) [15] is a widely recognized scheduler in accomplishing efficient performance in computing systems. HEFT's objective is to execute tasks onto heterogeneous machines in the earliest completion time. An important feature of this scheduler is its low scheduling overhead time. It first lists tasks based on their computational demands, and then it starts assigning each task to the VM that executes it in the earliest finishing time. Authors in [16] developed a scheduler which scales up and down the number of VMs founded on provenance information obtained through execution. Provenance, as we will denote this algorithm, optimizes execution time, monetary cost and reliability. This algorithm starts grouping tasks with similar file dependencies; then, it chooses a set of VMs to execute the given tasks. Authors in [17] present a

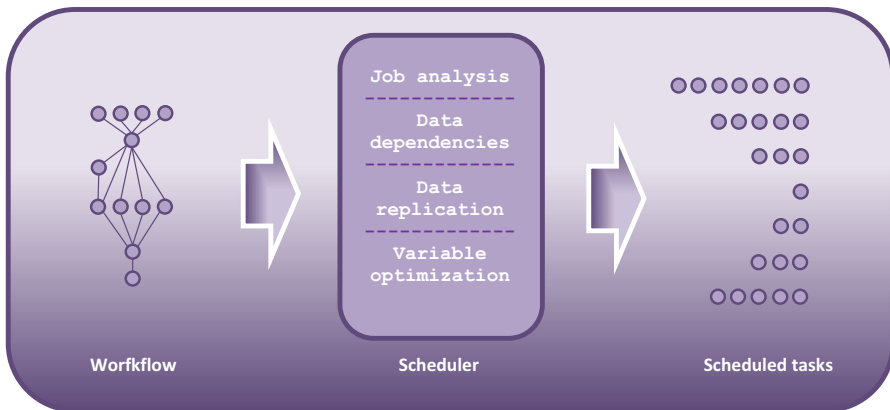


Fig. 2 Representation of a workflow scheduler. A scheduler receives a workflow description, and then it defines a number of queues and distributes tasks to efficiently execute the application based on user demands

scheduler with a flexible selection of the number of VMs balancing cloud revenue and customer budget. Flexible, as we will call this scheduler, measures the number of VMs needed to execute a given application based on its computing requirements. With its base in microeconomics, Flexible finds equilibrium between cloud user and provider needs. On the one side, a customer aims to execute his/her workflow within a budget. On the other side, the cloud provider's objective is to maximize revenue. Equilibrium is reached when both parties fulfill their needs. In [18], the authors developed a load forecasting algorithm, an approach to efficiently manage web applications on cloud computing environments. This algorithm predicts loads on computer machines based on historical and current data. The load forecasting algorithm produced outstanding results regarding its decision making in selecting a target resource. BaRRS [19] is a scheduler with the objective of balancing task queues to execute the scientific application in cloud environments optimizing runtime and monetary cost. GA-ETI [20] develops a powerful scheduler tool to map workflow application to cloud resources. It modified the genetic algorithm providing specific alteration to the crossover and mutation operators with the objective to reduce randomness, an inherent characteristic of genetic algorithms. After a deep examination of the mentioned scheduling approaches, we observe most of them (1) do not provide different options to execute their application with different execution times, budget limits and number of resources; (2) tend to spend user budget without analyzing different scheduling options that may end up with higher efficiency; (3) do not analyze the complete application prior to execution resulting in limited scheduling decisions and a poor performance; or (4) cause an excessive scheduling overhead time.

2.3 Architectures for execution of application of cloud systems

Architecture, in this context, is a framework with the required components involved in the execution of applications on cloud environments. Cloud computing only provides the hardware to run computer experimentation, but additional components are

needed in order to build a complete system for the execution of applications such as a scheduler, application receiver, workflow editor/engine, application monitor and a discovery manager. Following is a list of architectures with a similar objective as this study. The broker-based framework in [7] defines architecture with three main components: a workflow engine, service broker and cloud providers. Its objective is to exploit economic benefits from the use of cloud environments. Based on the service level of agreements chosen by the user, this tool selects adequate resources using a cloud service broker. The Workflow Engine for Clouds from [9] proposed a versatile architecture for the business perspective of cloud computing. It has a featured market-orientated component acting as an intermediary between user and computing resources. The multi-agent architecture presented in [6] runs applications using one or more cloud providers targeting concurrent and parallel execution. The work in [8] introduces the SciCumulus middleware that exploits parallelism with two techniques: parameter sweep and data fragmentation. Nevertheless, these approaches fail in providing a feedback to users with the options to run his/her application in the cloud.

As previous paragraphs exhibited, each of these foundations has its own objective. The purpose of this study is to embrace these roots to find a strong architecture to execute workflows on cloud environments. Firstly, current WMS are well-designed tools that only require specialized scheduling policies to be added, for this reason, there is no necessity to develop a new WMS, and instead this study selects one of them to act as a link between cloud resources and our scheduler. Secondly, we proposed a scheduler capable of overcoming the negative aspects of the aforementioned scheduling algorithms. Finally, every component is enclosed into the architecture to exhibit the user the main components he/she needs to retrieve and understand in order to have the application executed.

3 Cloud user–provider affiliation: architecture to conduct scientific experiments on cloud systems

CUPA is a mechanism to guide researchers in executing their workflows on cloud environments acting as a bridge between workflows and cloud resources. CUPA's main features are: (1) to specify and provide guidance on the steps a user must follow to get his/her workflow executed on the cloud resources; (2) to use history data or use a program profiler to retrieve tasks running time for those cases where information is unknown; (3) to accept scientific applications demanding high computing power and/or intense file transferring; and (4) to develop a complete scheduler based on the PSO to make an efficient use of cloud resources.

The architecture of CUPA is presented in Fig. 4. Stage (1) is a process completed by user; he/she develops a workflow $W = \{t_1, \dots, t_n\}$ to solve a problem in a particular scientific area. Each task t_i from W has a set of parents t_i^{parents} linked by a set of files with total size f^{size} . Examples of scientific workflows are presented in Fig. 3. At stage (2), CUPA requests execution time \hat{t}_i^{exe} for each task. CUPA gives two options for obtaining this information, (i) using a Program Profiler and (ii) retrieve historical data from previous executions. CUPA recommends that a user employs historical data; in cases where these data are not available, different profilers such as SimpleScalar [21]

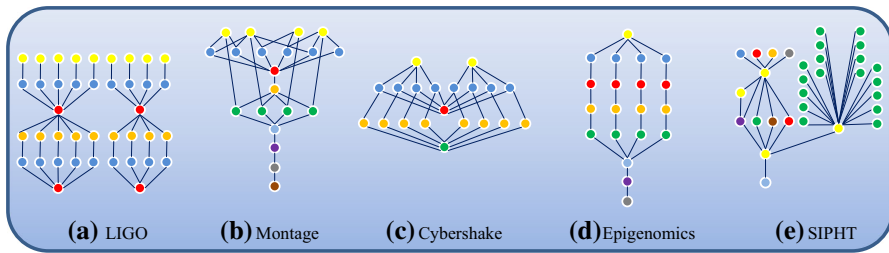


Fig. 3 Example of five well-known scientific applications from different research areas including astronomy, geology, biology, cosmic analysis and biotechnology. The CUPA objective is to guide the user in analyzing, scheduling and executing these types of workflows

may be used. A user collects the calculated execution time and attaches it to each task. Then at stage (3) a user selects the optimization level for each objective, i.e., monetary cost and makespan. It is important to highlight user selects a percentage and not a budget or time value limit since CUPA's internal scheduler provides an analysis and then provides different options based on user selection. Then at stage (4) the PSO-DS receives a workflow with its information attached including optimization levels. Then a user receives feedback with the possible scheduling configuration with a different number of VMs, monetary cost $MCst$ and makespan $MSpn$ and then he/she selects the one that is best for his situation. Finally, at stages (5–6), CUPA requests the required VMs and submits tasks to each resource based on the selected scheduling configuration and triggers execution.

3.1 Profiler description

This study proposes the usage of a profiler to obtain task execution time whenever this information is not available to the user. The objective of a profiler is to examine a computer program to discern memory usage, instruction count and time complexity. The profiling process is realized over program code or binary executable files. Different profilers are currently available; as an example, SimpleScalar LLC [21], a popular profiler in the research on computer architecture [22–29], is a system to model a program's performance and provide a detailed examination of processor architecture as well as hardware and software co-verification as exemplified in Fig. 5. SimpleScalar is able to output the execution time, number of instructions and the required CPU cycles to execute a program on a particular architecture processor. The use of profilers such as SimpleScalar is recommended for workflows in which the same task/program is executed with different file inputs; SimpleScalar needs to run the task/program itself to produce the aforementioned stats.

3.2 The scheduling problem

Scheduler responsibility is to organize tasks into a set of vm_j^{queue} to be executed by a given vm_j with the objective of minimizing total makespan $MSpn$ (Eq. 1) and

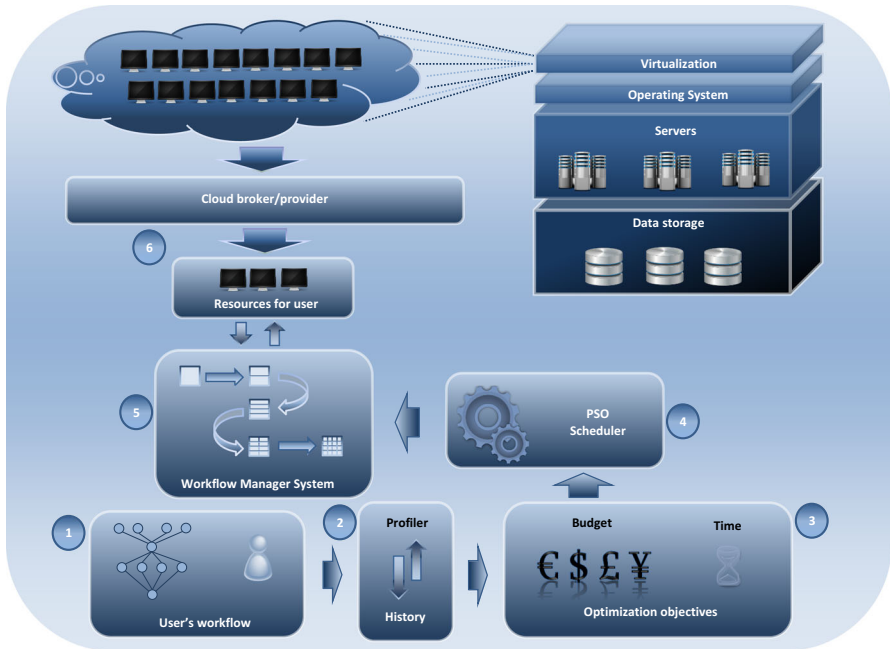


Fig. 4 Cloud user–provider affiliation. Cloud affiliation consists of the steps a user’s workflow must take to be analyzed before a scheduler presents executing options in the cloud system

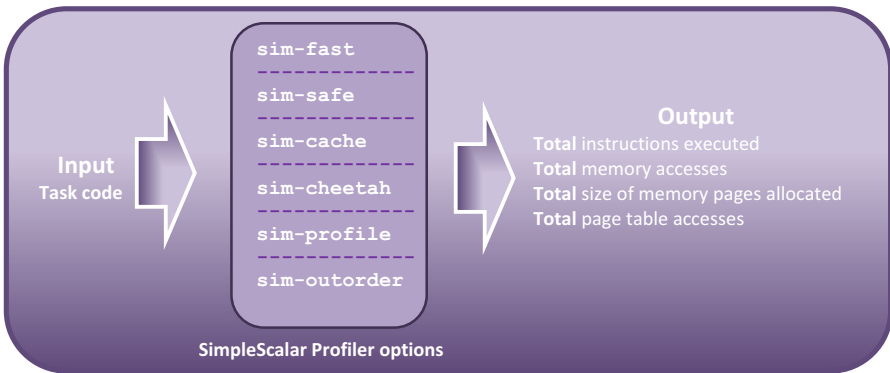


Fig. 5 SimpleScalar basic functions to analyze applications

monetary cost MC_{st} (Eq. 2). To accomplish this task, a scheduler needs to define the size of pool of resources $VM = \{vm_1, \dots, vm_v\}$ based on cost to hire each resource vm_j^{cost} and the potential time to execute each set of tasks vm_j^{time} from a given vm_j . Makespan is the latest finishing time (LFT) to execute all vm_j^{queue} . Equation 3 defines vm_j^{time} as the LFT from all tasks assigned to vm_j . Equation 4 expresses \hat{t}_i^{total} as the time to execute t_i , while Eq. 5 is the time to execute all its parent tasks. Finally, Eq. 6 is employed to calculate the file transfer between vm_p and vm_i . Figure 6 presents an

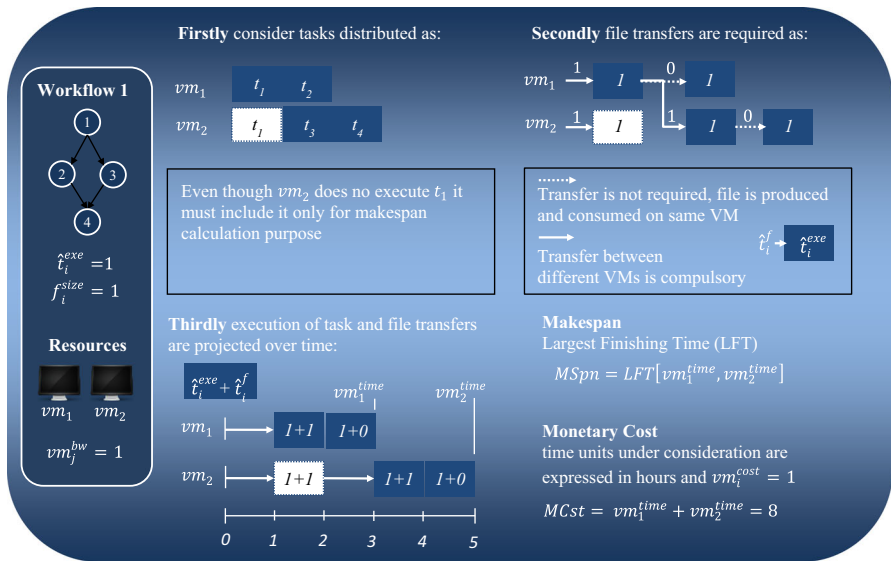


Fig. 6 Example to calculate makespan and monetary cost with Eqs. 1–6

example to calculate $MSpn$ and $MCst$ for a four-task workflow and a set of two VMs with unitary values for \hat{t}_i^{exe} , f_i^{size} , vm_j^{bw} and vm_j^{cost} . Firstly, consider the four tasks are equally distributed to the set of VMs as indicated. Secondly, files are transferred only between tasks residing on different VMs. Thirdly, tasks are presented on the timeline exhibiting its corresponding execution time \hat{t}_i^{exe} and its transfer time \hat{t}_i^f . Task t_1 must transfer its corresponding f_1^{size} , while t_2 does not require data transmission since it is allocated to same VM. In contrast, t_3 must transfer f_3^{size} from vm_1 to vm_2 , while t_4 does not require any data transmission. It is clearly seen that vm_1^{time} executes its set of tasks within three units of time, while vm_2^{time} executes its tasks in five units of time. Finally, $MSpn$ selects the largest value from $[vm_1^{time}, vm_2^{time}]$ obtaining the value of five. As for $MCst$, it obtains a value of eight units of time.

$$MSpn = LFT_{j=1}^{|VM|} [vm_j^{time}] \tag{1}$$

$$MCst = \sum_{j=1}^{|VM|} [vm_j^{time} vm_j^{cost}] \tag{2}$$

$$vm_j^{time} = LFT_{i=1}^{|vm_j^{queue}|} [\hat{t}_i^{total}] \tag{3}$$

$$\hat{t}_i^{total} = \hat{t}_i^f + \hat{t}_i^{exe} + \hat{t}_i^{parent} \tag{4}$$

$$\hat{t}_i^{parent} = \sum_{p=1}^{|parents|} \hat{t}_p^f + \hat{t}_p^{exe} \tag{5}$$

$$\hat{t}_i^f = \frac{f_i^{size}}{\min (vm_p^{bw}, vm_i^{bw})} \tag{6}$$

This model assumes: (1) every VM has a fixed bandwidth (vm_j^{bw}), number of cores (vm_j^{cores}), memory size (vm_j^{mem}) and disk size (vm_j^{disk}) and (2) the CUPA system negotiates with a cloud provider/broker in obtaining the number of VMs indicated by scheduling analysis. Since this study uses Pegasus as part of experimentation, workflow description follows its format including executable files, input data and DAX file which is an abstract description of the workflow and its internal dependencies. Additionally, workflow description must include the time to execute each of its tasks. This study prevents cases where this information is not available. For such cases, it employs a profiler to retrieve the missing data. Once information is complete, the analyzer produces scheduling plans with different finishing time, monetary cost and number of VMs. For this task, CUPA employs a scheduler based on the PSO optimization technique as described in Sect. 4.

4 Particle Swarm Optimization in solving the scheduling problem

For this study, the authors developed a scheduling approach based on the particle swarm optimization (PSO) mechanism to solve the aforementioned problem. PSO is a process to find a solution for nonlinear problems in terms of their optimization functions. PSO is based on particles continuously moving while aiming to obtain the coordinates that optimize the evaluation value as illustrated in Fig. 7. PSO is strongly related to swarming theory and has similarities with genetic algorithms (GAs) [30]. As compared with GA, PSO has lower demands in terms of computational power, memory capacity and computer coding with exceptional capabilities to solve different kinds of optimization problems [31].

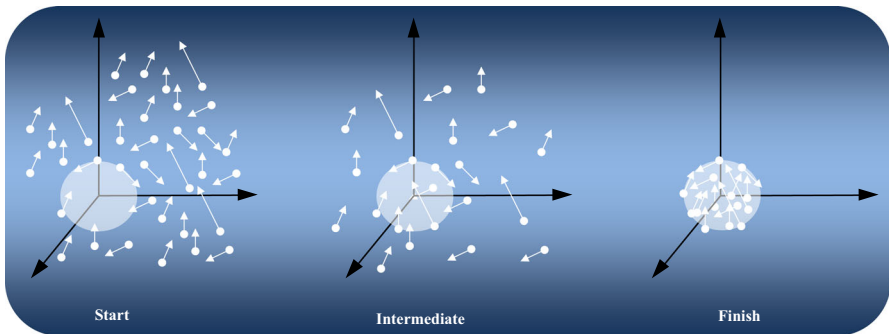


Fig. 7 The PSO process. Each particle (solution) moves through the solution space. Particles change velocity and direction on each iteration targeting a global best highlighted by a *white shadow*

4.1 Particle Swarm Optimization (PSO): the original model

In the original form of PSO [30], GBEST model is a searching technique in the solution space for an optimal answer. It is orientated for problems expressed with real numbers. In order to extend PSO's scope, a discrete version of the swarm algorithm is developed in [32]. The core of the original version was kept intact, while differing only on the discrete mode to manipulate the problem. Each particle in the PSO represents a solution and has a position and a velocity in the search space. Through a series of iterations, particles *swarm* through the solution space to find the maximum (or minimum) value for a given evaluation function. The following is the notation to introduce the discrete PSO (Algorithm 1).

On a population with size P , consider the position of the i th particle as $X_i^t = (x_{i1}^t, x_{i2}^t, \dots, x_{iD}^t)$ with D bits where $x_{id}^t \in \{0, 1\}$. A particle's velocity is then defined as $V_i^t = (v_{i1}^t, v_{i2}^t, \dots, v_{iD}^t)$ where $v_{id}^t \in R$. The PSO keeps a record of the particle's best position on $PBEST_i^t = (pbest_{i1}^t, pbest_{i2}^t, \dots, pbest_{iD}^t)$ as well as a global best solution ever found in $GBEST^t = (gbest_{i1}^t, gbest_{i2}^t, \dots, gbest_{iD}^t)$. Equation 7 presents the function to calculate the velocity v_{id}^t for the d th dimension of the i th particle on the t iteration. The ω term introduced in [33] is a particle's inertia to continue moving toward its original destination. Acceleration coefficients of c_1, c_2 act as a particles' memory, inclining it to move toward $PBEST_i^t$ and $GBEST^t$, respectively. The objective of v_{id}^t is to drive particles in the direction of a "superior" position in terms of its evaluation function value. Position X_i^t is updated on every iteration of the PSO.

Sigmoid function (Eq. 8) is employed to operate velocities, as probabilities values, in the interval of $[0, 1]$. Additionally, v_{id}^t is limited to a fixed range of values $[-V_{max}, +V_{max}]$ to prevent $s(v_{id}^t)$ from falling on the upper or lower bound of $[0, 1]$. In our experiments, also advised by [34], $V_{max} = 4$. Algorithm 1 presents the generic discrete version of the PSO for a maximization optimization. In step 1, it initializes an array of particles with random positions X_i^0 and velocities V_i^0 . In steps 2–20, it executes its main cycle. It employs function F to evaluate each particle's value in step 4; if a particle's value is greater than its previous best position, then $PBEST_i^t$ is updated by the particle's value. Similarly, the global best $GBEST^t$ value is compared, and updated if required, with P_i^t (steps 7–9). In the sub-cycles in steps 10–18, Algorithm 1 updates the velocities and positions for all D dimensions of a particle. In steps 11–12, each particle updates its velocity v_{id}^t and caps its values. Finally, based on the result of the sigmoid function, each particle sets the value for each dimension x_{id}^{t+1} for iteration $t + 1$. The main cycle continues until a termination criterion is met.

$$v_{id}^t = \omega v_{id}^{t-1} + c_1 r_1 (pbest_{id}^t - x_{id}^t) + c_2 r_2 (gbest_{id}^t - x_{id}^t) \quad (7)$$

$$s(v_{id}^t) = \frac{1}{1 + \exp(-v_{id}^t)} \quad (8)$$

Algorithm 1 Discrete PSO – The Global model

F: Evaluation Function

```

1  Initialize an arrangement of  $P$  particles with random positions
2  While a termination criterion is not met
3      For each  $X_i^t$  particle in  $N_p$ 
4          if  $F(X_i^t) > F(PBEST_i^t)$ 
5               $PBEST_i^t = X_i^t$ 
6          end
7          if  $F(P_i^t) > F(GBEST)$ 
8               $GBEST = P_i^t$ 
9          end
10         For each dimension  $d$  in  $D$ 
11             Update  $v_{id}^t$  (Eq. 7)
12             Limit  $v_{id}^t \in [-V_{max}, +V_{max}]$ 
13             if  $s(v_{id}^t) > \text{random}[0,1]$ 
14                  $x_{id}^{t+1} = 1$ 
15             else
16                  $x_{id}^{t+1} = 0$ 
17             end
18         end
19     end
20 end

```

4.2 PSO-DS as the scheduling engine for CUPA

In this section, we present our modified particle swarm optimization with discrete adaptation and a featured *SuperBEST* (PSO-DS)—an extension to the generic PSO—to solve the scheduling problem in this article. Following is the description of the PSO-DS particles, velocity and the introduction of a featured *SuperBEST* particle.

4.2.1 Adaptation of particle format

As in to [35], PSO-DS needs to unfold the original discrete PSO particles to interpret integer numbers to solve the scheduling problem. In PSO-DS, particles have an augmented format $X_i^t = (x_{i11}^t, x_{i21}^t, \dots, x_{inv}^t)$, $x_{ijk}^t \in \{0, 1\}$, where $x_{ijk}^t = 1$ if the j th task of the i th particle is executed in vm_k , and $x_{ijk}^t = 0$ otherwise. For ease of explanation, this study introduces a short format to represent particles, i.e., $X_i^t = (x_{i1}^t, x_{i2}^t, \dots, x_{in}^t)$, $x_{ij}^t \in \{vm_1, \dots, vm_v\}$, is the abstract representation of particle X_i^t where x_{ij}^t is the vm_k executing task j of particle i at time t . An example particle is presented in Fig. 8 for a workflow with four tasks $\mathbf{W} = \{t_1, t_2, t_3, t_4\}$, and a set of two resources $\mathbf{VM} = \{vm_1, vm_2\}$. Here, particle i is expressed in its long and abstract format X_i^t and X_i^t , respectively.

For a particle expressed in its abstract format X_i^t , the number of different values that each of its dimensions can have is v , given $\mathbf{VM} = \{vm_1, \dots, vm_v\}$. At the same time,

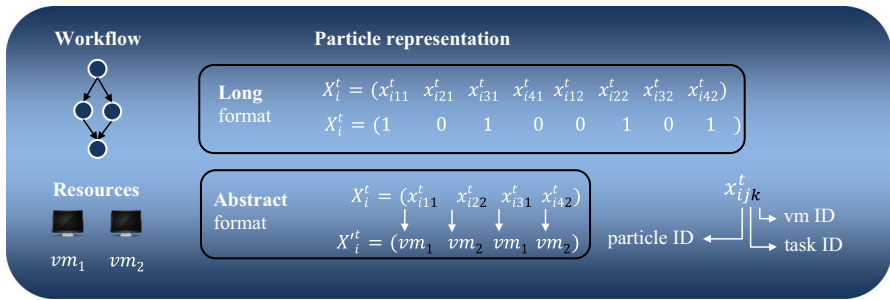


Fig. 8 Each particle represents a workflow scheduling configuration. The particle’s long format is used for velocity calculation, while its abstract representation expresses the scheduling distribution

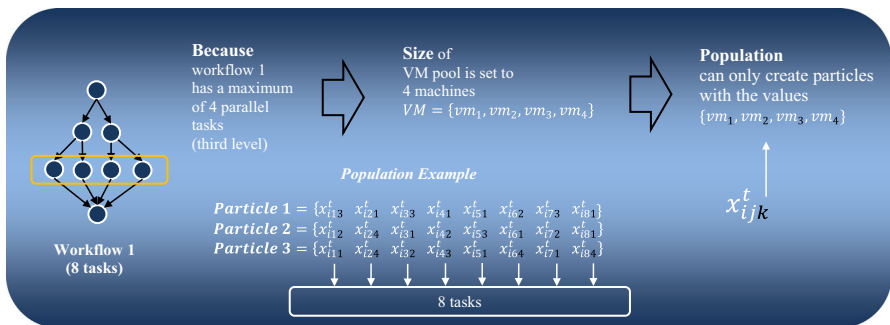


Fig. 9 Setting up the maximum number of values particle’s dimensions can have

v is driven by the number of parallel tasks in a given workflow. Figure 9 illustrates this concept; for Workflow 1, the maximum number of tasks that can be executed in parallel is four (in the third level of Workflow 1). As a consequence, VM is set to the values of $\{vm_1, vm_2, vm_3, vm_4\}$ because any additional VM (more than four) in the pool will remain idle during execution of this workflow.

4.2.2 Particle’s velocity adaptation

The velocity from Eq. 7 is transformed into Eq. 9 in order to follow each particle’s adaptation. Firstly, for X_i^t (expressed in its long format), the best position is defined as $PBEST_i^t = (pbest_{i11}^t, pbest_{i21}^t, \dots, pbest_{inv}^t)$, while the global best particle in the population is defined as $GBEST^t = (gbest_{11}^t, gbest_{21}^t, \dots, gbest_{nv}^t)$. Parameters ω , c_1 , r_1 , c_1 , r_1 and V_{max} have the same functions as in the original discrete PSO described in the previous section. Additionally, Eq. 8 is slightly modified to produce Eq. 10 for managing dimension velocities as a set of probabilities, i.e., representing velocities in the range of [0,1]. Here, each dimension’s velocity v_{ijk}^t is the probability of vm_k to execute t_j .

$$v_{ijk}^t = \omega v_{ijk}^{t-1} + c_1 r_1 (pbest_{ijk}^t - x_{ijk}^t) + c_2 r_2 (gbest_{jk}^t - x_{ijk}^t) \tag{9}$$

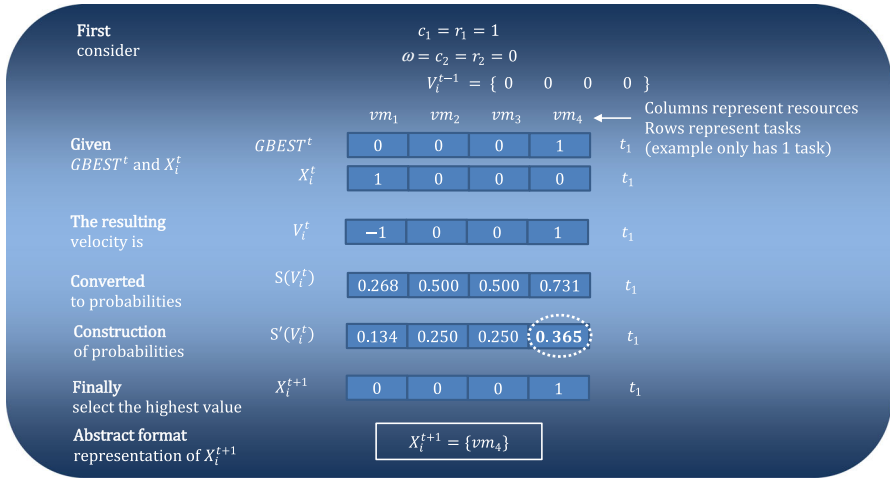


Fig. 10 Particle updates its velocity and position on each iteration. The process to calculate the velocity is done by expressing the particle in its long format

$$s(v_{ijk}^t) = \frac{1}{1 + \exp(-v_{ijk}^t)} \tag{10}$$

4.2.3 Scheduling reconstruction from particle velocity

PSO-DS has a population of solutions expressed as velocity probabilities requiring an interpretation to construct scheduling configurations. In this context, VMs compete to execute tasks, while each task can only be assigned to a single resource. Unlike [35], where construction of particles forces consecutive tasks to be assigned to different VMs, our approach allows to assign sets of successive tasks to the same VM to avoid unnecessary data transfers. Consider $s'(v_{ijk}^t)$ (Eq. 11) as the probability of assigning the j th task to the k th resource from the pool of v machines where $\sum_{k=1}^v s'(v_{ijk}^t) = 1$. During a process to be repeated for every task j in every particle i , only one dimension, namely k th, $x_{ijk}^t = 1$, while $\{x_{ijk'}^t = 0 | k' \neq k\}$; k is the index of the dimension with the maximum value. Figure 10 presents a velocity and position update example for a single-task workflow and a set of four VMs. Here, the $GBEST^t$ indicates that the task should be allocated to vm_4 , while the particle X_i^t has assigned the task to vm_1 . In the resulting $S'(V_i^t)$, the last dimension exhibits the highest probability to obtain 1, and thus in the updated particle X_i^{t+1} , vm_4 executes t_1 .

$$s'(v_{ijk}^t) = \frac{s(v_{ijk}^t)}{\sum_{k=1}^v s(v_{ijk}^t)} \tag{11}$$

4.2.4 SuperBEST particle and the GBEST

We defined a new particle, namely *SuperBEST^t*, in PSO-DS; *SuperBEST^t* is built using the most popular particles' elements in the population. Consider the set $X_j^{''t} = \{x_{j1}^{''t}, x_{j2}^{''t}, \dots, x_{jP}^{''t}\}$, $x_{ji}^{''t} \in \{vm_1, \dots, vm_v\}$, where $x_{ji}^{''t}$ is the value assigned to dimension j in the i th particle (expressed in its abstract format) for a population with P particles. In this case, $SuperBEST^t = (sb_1^t, sb_2^t, \dots, sb_n^t)$, where sb_j^t holds the value with highest frequency from X'' . Figure 11 illustrates the formation of the *SuperBEST^t*. Firstly, from a population of three particles X_1^t, X_2^t and X_3^t , a set of $X_j^{''t}$ vectors express the population; then the frequency of occurrence for each vm_k is counted; and finally, the *SuperBEST^t* is composed. The PSO-DS uses the *SuperBEST^t* to build the *GBEST^t*. For $GBEST_{Pbased}^t = \max_{i=1}^P [PBEST_i^t]$ and $GBEST_{Xbased}^t = \max_{i=1}^P [XBEST_i^t]$, the $GBEST^t = \max (SuperBEST^t, GBEST_{Pbased}^t, GBEST_{Xbased}^t)$.

4.2.5 Evaluation function and termination criterion

This study adopts a maximization optimization for the scheduling process in the PSO-DS. It employs an evaluation function E_{value} that integrates the makespan $MSPn$ and monetary cost $MCst$ with weight values w_1 and w_2 to control objective optimization. The variables max^{time} , min^{time} , max^{cost} and min^{cost} retain maximum and minimum values of makespan and economical cost through continuous update during PSO-DS processes. PSO-DS continues until E_{value} shows no improvement; PSO-DS outputs *GBEST* as the final solution.

$$E_{value} = w_1 \frac{(max^{MSPn} - MSPn)}{(max^{MSPn} - min^{MSPn})} + w_2 \frac{(max^{Cst} - MCst)}{(max^{Cst} - min^{Cst})} \tag{12}$$

Population with P size of 3	X_1^t (vm_2 vm_2 vm_1)		
	X_2^t (vm_1 vm_3 vm_1)		
	X_3^t (vm_2 vm_3 vm_1)		
Population Express as set of $X_j^{''t}$	X''_1 (vm_2 vm_1 vm_2)	X''_2 (vm_2 vm_3 vm_3)	X''_3 (vm_1 vm_1 vm_1)
Incidence of every dimension value	$vm_1 = 1$ $vm_2 = 2$ $vm_3 = 0$	$vm_1 = 0$ $vm_2 = 1$ $vm_3 = 2$	$vm_1 = 3$ $vm_2 = 0$ $vm_3 = 0$
Particle selects the dimension with the highest incidence	$SuperBEST^t$ (vm_2 vm_3 vm_1)		

Fig. 11 The *SuperBEST* particle formation. From every dimension, *SuperBEST* selects the value with the higher number of occurrence. *SuperBEST* is updated and included in PSO-DS

Algorithm 2 PSO-DS

E_{value} Evaluation Function, Population size P , V_{max} , Workflow W

```

1  Initialize an arrangement of  $P$  particles with random positions and velocities
2  While a termination criterion is not met
3    For all particles ( $i = 1$  to  $P$ )
4      if  $E_{value}(X_i^t) > E_{value}(PBEST_i^t)$ 
5         $PBEST_i^t = X_i^t$ 
6      end
7      if  $E_{value}(X_i^t) > E_{value}(GBEST^t)$ 
8         $GBEST_{Xbased}^t = XBEST_i^t$ 
9      end
10     if  $E_{value}(PBEST_i^t) > E_{value}(GBEST^t)$ 
11        $GBEST_{Pbased}^t = PBEST_i^t$ 
12     end
13   end
14   For all tasks ( $j = 1$  to  $n$ )
15      $X''_j = \{x''_{j1}, x''_{j2}, \dots, x''_{jp}\}$ 
16      $sb_j^t = higher\_incidence(X''_j)$ 
17      $SuperBEST^t \leftarrow sb_j^t$ 
18   end
19    $GBEST^t = \max(SuperBEST^t, GBEST_{Pbased}^t, GBEST_{Xbased}^t)$ 
20   For all particles ( $i = 1$  to  $P$ )
21     For all tasks ( $j = 1$  to  $n$ )
22       For all VMs ( $k = 1$  to  $v$ )
23          $v_{ijk}^t = \omega v_{ijk}^{t-1} + c_1 r_1 (pbest_{ijk}^t - x_{ijk}^t) + c_2 r_2 (gbest_{jk}^t - x_{ijk}^t)$ 
24          $v_{ijk}^t \in [-V_{max}, +V_{max}]$ 
25          $s(v_{ijk}^t) = 1 / (1 + \exp(-v_{ijk}^t))$ 
26          $x_{ijk}^t = 0$ 
27       end
28        $s'(v_{ij(0)}^t) = 0$ 
29       For all VMs ( $k = 1$  to  $v$ )
30          $s'(v_{ijk}^t) = s(v_{ijk}^t) / \sum_{k=1}^v s(v_{ijk}^t)$ 
31         if  $s'(v_{ijk}^t) > s'(v_{ij(k-1)}^t)$ 
32           Save  $k$ 
33         end
34       end
35        $x_{ijk}^t = 1$ 
36     end
37   end
38 end

```

4.2.6 The proposed PSO-DS algorithm

The resulting PSO-DS algorithm is presented in Algorithm 2. In step 1, it creates a population with size P where random positions X_i^t and velocities V_i^t are assigned to the population. Then steps 2–38 present the main loop. Steps 4–6 update $PBEST_i^t$ for every particle; $GBEST_{Xbased}^t$ and $GBEST_{Pbased}^t$ update their values if the current particle has a higher evaluation value. In steps 14–18, the algorithm builds the $SuperBEST^t$ particle. Following in step 19, PSO-DS selects $GBEST^t$ from the pool $\{SuperBEST^t, GBEST_{Pbased}^t, GBEST_{Xbased}^t\}$. In steps 23–26, velocity v_{ijk}^t is updated

and capped to the range $[-V_{max}, +V_{max}]$. Next, $s(v_{ijk}^t)$ expresses v_{ijk}^t as a probability in the interval $[0,1]$; its respective position x_{ijk}^t is set to 0 as a preliminary step to update each particle's position. Steps 28–35 present procedures to update each particle's position; it is repeated n times for a given workflow W with size n . Steps 29–34 form a loop to calculate $s'(v_{ijk}^t)$ for all k resources that are competing to execute the j th task in the i th particle. Finally, only the resulting k resource (Step 32) in x_{ijk}^t obtains the value of 1 for the j th task in the i th particle.

5 Experiments and analysis

This section evaluates the performance of the PSO-DS in the CUPA using three main tests. In experiment 1, we compare the execution of workflows using the Pegasus-WMS without any external scheduling policy and with the complete CUPA framework including the PSO-DS scheduler. Then in experiment 2, we analyze the need to guide the user in selecting a limited budget by comparing monetary costs when executing workflows with/without an unlimited budget. Finally in experiment 3, we compare the performance of the PSO-DS against Provenance [16], HEFT [15] and Flexible [17] and GA-ETI [20].

The experiment test bed consists of a private cloud with three Krypton Quattro R6010 with 4-way AMD OpteronTM 6300 series (64-Cores each). We selected the Pegasus-WMS (4.2) as the WMS; it is installed on Ubuntu 14.04 operating system. The VMware vSphere (5.5) manages computer resources and provides virtual machines with the aforementioned platform. As described in the CUPA, the PSO-DS performs as the scheduling engine with parameters set as shown in Table 1. In order to define the PSO-DS parameters, we produced preliminary experiments to tested ω in the range of $[0.9, 1.2]$ and $c_1 = c_2 = 2$ as advised in [34,36]. From those trials, a ω value of 1.2 produced the best results. Additionally [36], recommended a relatively small population of 20 particles but the nature of the scheduling problem demanded a larger population; preliminary tests exhibited that 100 particles give the best results. V_{max} is set to 4 to prevent $s(v_{id}^t)$ in Eq. 8 to continuously approaching the upper and lower bound of $[0,1]$. Finally, w_1 and w_2 have a value of 0.5 to give the same priority to monetary cost and execution time. We selected five scientific workflows from [1] to produce the experiments. Workflows represent applications from different scientific areas including astronomy, geology, biology, cosmic analysis and biotechnology. Their details are presented in Table 2.

5.1 Analysis 1: the need for a specialized scheduler on top of WMS

This first experiment stage has the objective of highlighting the need to add a specialized scheduling analysis on top of WMS. Table 3 provides makespan and monetary cost results for the PSO-DS and Pegasus-WMS. The epigenomics workflow presents the biggest difference in terms of time and cost due to its great parallelism level. In this application, PSO-DS converges to solutions where dependent tasks are executed on the same VM; as a consequence PSO-DS successfully decreases to its minimum

Table 1 PSO-DS setup

Parameter	Symbol	Value
Population	P	100
Makespan optimization	w_1	0.5
Monetary cost optimization	w_2	0.5
Social acceleration coefficient	c_2	2
Personal acceleration coefficient	c_1	2
Velocity limit	V_{max}	4
Inertia coefficient	ω	1.2

Table 2 Characteristics of the scientific workflows employed in experiments to test GA-ETI

	Nodes	w -Levels	Parallel tasks	Average file size (MB)	Average task execution time (s)	Dependencies patterns
Epigenomics	100	8	24	749	2346	(2)(3)(4)
Montage	100	9	62	20.6	11.34	(2)(3)(4)
Cybershake	100	5	48	1156.1	51.70	(1)(3)(4)
Ligo	100	8	24	55.6	222.0	(1)(4)(5)
Sipht	100	7	51	22.02	210.27	(4)(5)

Dependencies patterns: (1) process; (2) pipeline; (3) data distribution; (4) data aggregation; (5) data redistribution

Table 3 Results for makespan and monetary cost for PSO-DS and PEGASUS-WMS

	Makespan		Monetary cost	
	PSO-DS	Pegasus-WMS	PSO-DS	Pegasus-WMS
Epigenomics	20,099	89912	4.71	19.625
Cybershake	4377	6049	1.256	1.256
Sipht	3319	6532	0.628	1.57
Montage	160	435	0.785	0.785
Ligo	3560	6944	0.471	2.826

the number of data transfers. Similar scenarios are presented on the rest of the applications. For example, the Ligo workflow has three main groups of task running in parallel; PSO-DS converges to solutions where not all of the tasks are executed at the same time; since monetary cost is also considered as an optimization objective, it balances the optimization of makespan and cost. As for Pegasus, it presents higher makespan and cost values due to that fact that its only objective is to execute the application. Pegasus uses HTCondor as its internal DAG (direct acyclic graph) executor. HTCondor receives workflow and sends its tasks for execution with almost no control on which tasks to execute on a given VM or which data files would be replicated. In contrast, PSO-DS is able to evaluate a different number of scheduling configurations and chooses the one that contributes to the highest optimization value .

5.2 Analysis 2: the need to guide users in selecting a limited budget

In this experiment, we allow the PSO-DS to produce scheduling configurations relaxing the monetary cost optimization. Results are presented in Fig. 12; it first schedules for two VMs, then three, four and so on. For each workflow, three graphs are presented; the first graph presents evaluation function values; the second and third columns of graphs present their corresponding makespan and monetary cost. For each case, a red shadow highlights the values with function value above 80%. As readers will notice, makespan drops dramatically as the number of VMs increases, but once a low makespan is achieved, it does not decrease notably. In contrast, by incrementing the number of resources the makespan decreases by small amounts of timeframes but monetary cost increases linearly to the number of VMs.

With the exception of epigenomics, evaluation function values for every case present similar behavior; they rise as the number of VMs increases, then it reaches a peak and finally it drops. Peak time is presented when number of VMs is optimal for a particular case. For instance, when PSO-DS distributes Cybershake's tasks into four VMs, it obtains an optimal case with a makespan of 4377s at the cost of 1.256 dlls, even though the minimal achievable makespan is 3891s at the cost of 3.768 dlls. As for the epigenomics, its evaluation function starts rising as the number of VMs increases, it reaches its peak with five VMs, then it starts dropping, but it suddenly has a rise with 12 VMs. The reason for this behavior is that maximum number of parallel tasks is 24; for this reason, PSO-DS converges with a uniform distribution of these parallel tasks among 12 VMs. The reason PSO-DS does not produce an improved scenario for 24 VMs is that our scheduling model contemplates an hourly changing model of every VM, so although the configuration with 24 VMs presents the minimal makespan, its cost rises tremendously.

5.3 Analysis 3: Performance of PSO-DS in front up-to-date schedulers

In order to provide arguments for the competence of the PSO-DS, in this experiment, we compare it against the four schedulers previously presented. With the exception of Cybershake, results show the PSO-DS is able to provide better results especially for the cases with a low number of VMs and function values above 80% as highlighted in the previous section. An important reason for these positive results is that PSO-DS is designed to consider the most predominant factors affecting makespan such as task grouping that is based on their dependencies, file sizes and available number of VMs. It is important to highlight that whether HEFT and GA-ETI converge with similar makespan values they do it when the number of VMs increases. An important factor to emphasize is that optimal function values (above 80%) are presented as soon as makespan does the biggest drops. For example, in the Sipht result graph, in Fig. 13c, the makespan drops from 7188 to 2987s with two and five VMs, respectively, beyond that number of resources makespan does not provide a substantial improvement for any of the algorithms. This behavior is presented for the rest of workflow execution.

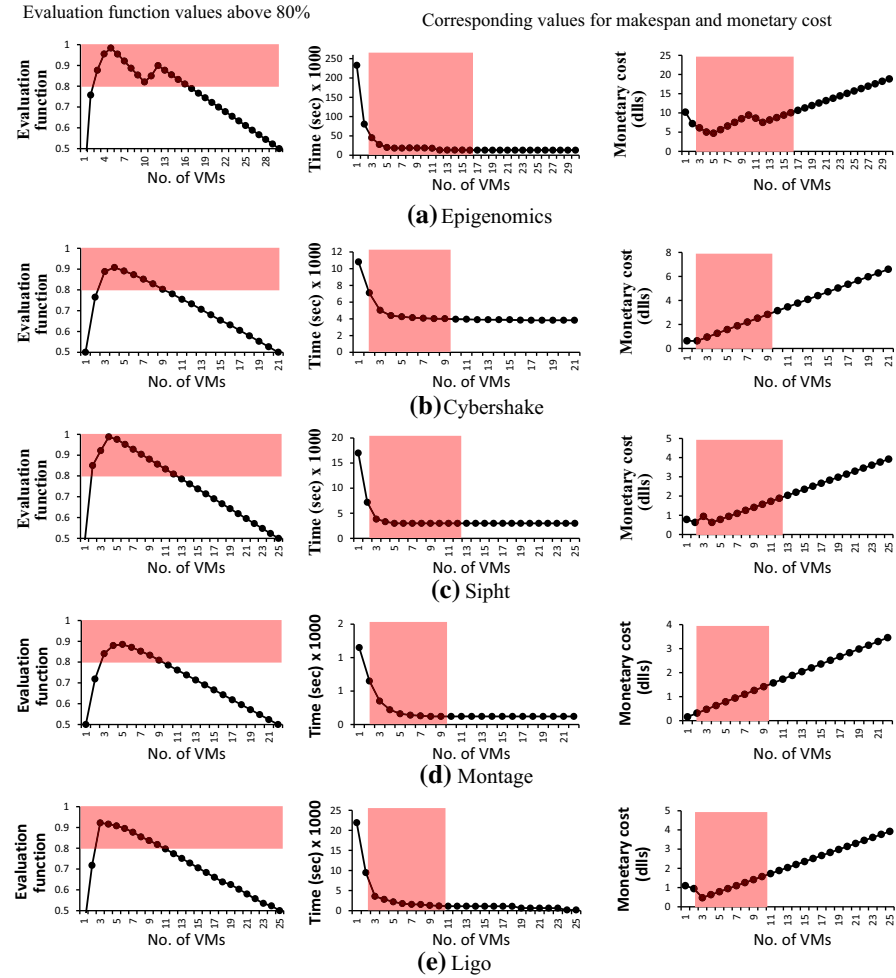


Fig. 12 Results for evaluation function values, makespan and monetary cost for the five scientific workflows highlighting the area where evaluation function values are above 80%

6 Discussion

This section provides a deeper analysis of the PSO-DS performance compared with other algorithms for solid reference.

6.1 PSO-DS performance on a Pareto front fashion

Figure 14 presents results of the five scheduling algorithms on a Pareto front fashion, i.e., on the *M_{Spn}* vs *M_{Cst}* graph. Pareto front is a concept in economics with application in engineering [37]. It is defined as positioning individuals where no one of them can improve its position without deteriorating another's. The Pareto front is built from

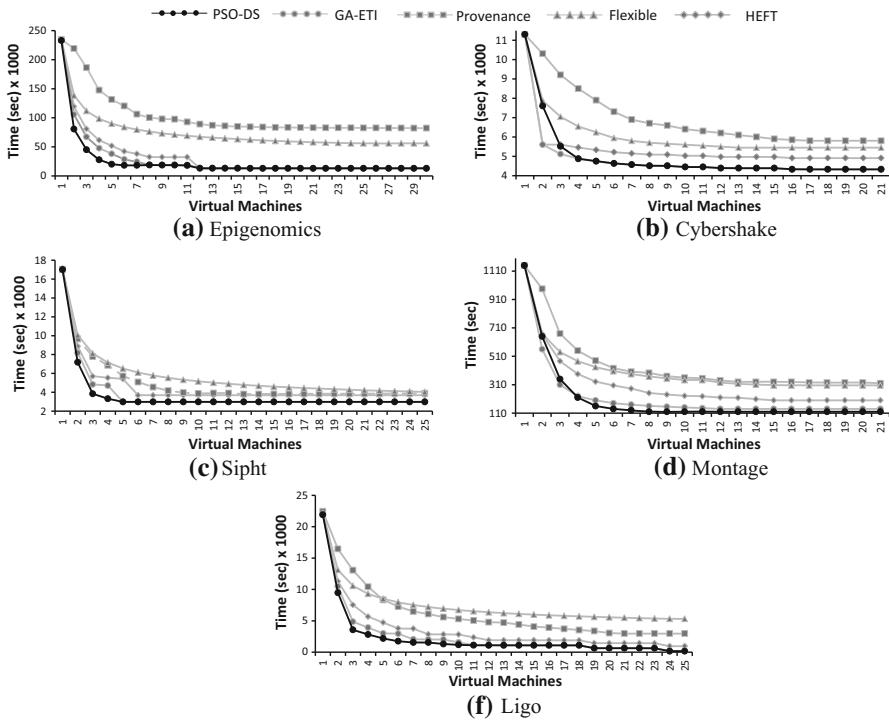


Fig. 13 Execution time results with a different number of VMs

connecting such individuals forming a curved graph which in engineering is used as a trade-off of values. In this case, we analyze only the epigenomics workflow since the remaining workflows present similar behavior. Figure 14 exhibits the results for makespan with its corresponding monetary cost. As readers will notice, the PSO-DS is able to converge to the values spread along the complete *MSPn* vs *MCst* curve. This analysis provides proof that PSO-DS does not get trapped in particular sections of the solution space and corroborates that the local best solution does not overdominate the search for a solution. In a similar manner, GA-ETI and HEFT distribute their values with great proximity to the PSO-DS due to their analysis and filtering of solutions. In contrast, flexible and provenance missed the opportunity to provide superior results due to their minimal workflow analysis.

6.2 Measure scheduling time

An important factor in scheduling algorithms is the time to run the scheduling process itself. Table 4 presents timeframes to produce a scheduling configuration for a workflow with 100 nodes. The PSO-DS and GA-ETI present the highest processing time with 1300 and 1450 ms, respectively, while the flexible executes the algorithm in 3.1 ms. The reason for this large difference is that flexible does not base its scheduling approach on an evolution of solutions; it rather chooses a final solution from a limited

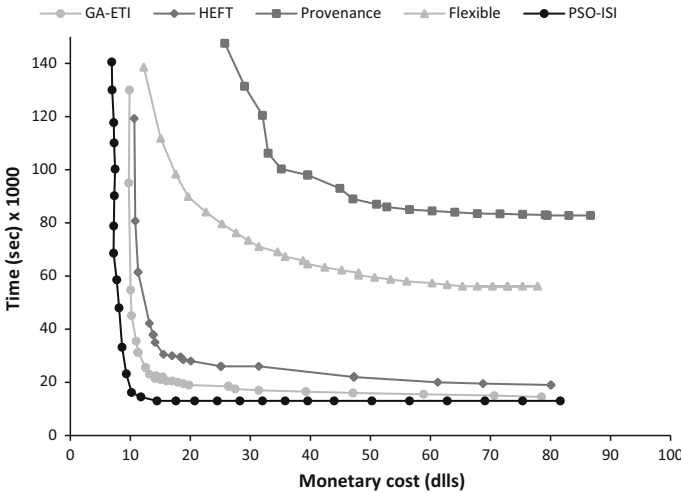


Fig. 14 Epigenomics’ *MSPn-MCst* graph to analyze where in the solution space schedulers search for a solution

Table 4 Scheduling time and its relation with final makespan

Scheduler	Scheduling time (ms)	Percentage of scheduling time compare with final makespan
PSO-DS	1300	1300 ms/13,190 s = ~0.00%
PSO (original)	1250	1250 ms/31,859 s = ~0.00%
GA-ETI	1450	1450 ms/16,190 s = ~0.00%
HEFT	16.1	16.1 ms/42,162 s = ~0.00%
Flexible	3.1	3.1 ms/89,912 s = ~0.00%
Provenance	68	68 ms/100,291 s = ~0.00%

number of configurations. Similarly, HEFT considers a single solution executing the algorithm on only 16.1 ms. On the contrary, PSO-DS and GA-ETI evaluate a group of solutions on a series of iterations unfolding solutions and executing their algorithms on a larger timeframe. However, none of the approaches has an excessive execution time compared with the final makespan as shown in column three of Table 4. Moreover, the high scheduling time as in PSO-DS yields a reduced final makespan.

7 Conclusion

In this article, we proposed CUPA, an architecture to execute scientific workflows in cloud computing systems. CUPA provides guidance to users in selecting the tools he/she needs in order to efficiently run his/her experiments. CUPA features the PSO-DS, a specialized workflow scheduler. PSO-DS is based on the PSO with a special adaptation to the scheduling problem including a discrete formatting of particles and an

enhanced super element. Using five workflows representing current scientific problems, PSO-DS was tested and proved its dominance against four cloud schedulers (HEFT, provenance, flexible and GA-ETI). Through experimentation, PSO-DS highlighted the need for a specialized scheduler on top of WMS. PSO-DS is able to provide superior results in terms of makespan and monetary cost compared against other schedulers, in particular in the cases with a small number of resources providing function values above 80%. Additionally, PSO-DS provides scheduling configuration with values spread along the complete $MSPn$ vs $MCst$ curve. PSO-DS's positive results exhibited the main factors to consider during the scheduling process in order to optimize time and cost; such characteristics include task grouping, job dependencies, file sizes and available number of VMs. Additionally, PSO-DS demonstrated that superior solutions execute parallel tasks sequentially on the same VM in order to lower file transferring. PSO-DS experiments underline the importance of not relaxing the monetary budget. Users may have an unlimited budget, or some schedulers may consider this assumption. By loosening the monetary budget, the user may obtain similar results at the expense of a pointless charge. To continue with this work, the authors aim to produce the CUPA system in order to make it available as a tool for other studies. As for future studies, the authors are focusing on the analysis of large sets of files (Big Data) for applications associated with biological viruses, terrorism and economic crisis behavior.

Acknowledgements The authors would like to thank the Commonwealth Scientific and Industrial Research Organisation (CSIRO) and Consejo Nacional de Ciencia Tecnología (Conacyt) for supporting this work.

References

1. Bharathi S, Chervenak A, Deelman E, Mehta G, Su M-H, Vahi K (2008) Characterization of scientific workflows. In: 3rd Workshop on Workflows in Support of Large-Scale Science, 2008. WORKS 2008, pp 1–10
2. Miao Y, Wang L, Liu D, Ma Y, Zhang W, Chen L (2015) A Web 2.0-based science gateway for massive remote sensing image processing. *Concurr Comput Pract Exp* 27:2489–2501
3. Liu P, Yuan T, Ma Y, Wang L, Liu D, Yue S et al (2014) Parallel processing of massive remote sensing images in a GPU architecture. *Comput Inform* 33:197–217
4. Deelman E, Blythe J, Gil Y, Kesselman C, Mehta G, Patil S et al (2004) Pegasus: Mapping scientific workflows onto the grid. In: undefined. Springer, Heidelberg, pp 11–20
5. HTCondor: High Throughput Computing. <http://research.cs.wisc.edu/htcondor/>
6. Gutierrez-Garcia JO, Sim KM (2012) Agent-based cloud workflow execution. *Integr Comput Aided Eng* 19:39–56
7. Jrad F, Tao J, Streit A (2013) A broker-based framework for multi-cloud workflows. In: Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds, pp 61–68
8. De Oliveira D, Ogasawara E, Baião F, Mattoso M (2010) Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In: 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), pp 378–385
9. Pandey S, Karunamoorthy D, Buyya R (2011) Workflow engine for clouds. *Cloud computing: principles and paradigms*, pp 321–344. doi:10.1002/9780470940105.ch12
10. Wang L, Chen D, Hu Y, Ma Y, Wang J (2013) Towards enabling cyberinfrastructure as a service in clouds. *Comput Electr Eng* 39:3–14
11. Chen D, Wang L, Wu X, Chen J, Khan SU, Kołodziej J et al (2013) Hybrid modelling and simulation of huge crowd over a hierarchical grid architecture. *Future Gener Comput Syst* 29:1309–1317
12. The Kepler Project. <https://kepler-project.org/>
13. Taverna Workflow Management System. <http://www.taverna.org.uk/>

14. Yang Y, Liu K, Chen J, Lignier J, Jin H (2007) Peer-to-peer based grid workflow runtime environment of SwinDeW-G. In: IEEE International Conference on e-Science and Grid Computing, pp 51–58
15. Topcuoglu H, Hariri S, M-y Wu (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans Parallel Distrib Syst* 13:260–274
16. de Oliveira D, Ocaña KA, Baião F, Mattoso M (2012) A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *J Grid Comput* 10:521–552
17. Tsakalozos K, Kilapi H, Sitaridi E, Roussopoulos M, Pappas D, Delis A (2011) Flexible use of cloud resources through profit maximization and price discrimination. In: IEEE 27th International Conference on Data Engineering (ICDE), 2011 pp 75–86
18. Ros S, Caminero AC, Hernández R, Robles-Gómez A, Tobarra L (2014) Cloud-based architecture for web applications with load forecasting mechanism: a use case on the e-learning services of a distant university. *J Supercomput* 68:1556–1578
19. Casas I, Taheri J, Ranjan R, Wang L, Zomaya AY (2016) A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems. *Future Gener Comput Sys*. doi:10.1016/j.future.2015.12.005
20. Casas I, Taheri J, Ranjan R, Wang L, Zomaya A (2016) GA-ETI: An enhanced genetic algorithm for the scheduling of scientific workflows in cloud environments. *J Comput Sci*
21. Burger D, Austin TM (1997) The SimpleScalar tool set, version 2.0. *ACM SIGARCH Comput Archit News* 25:13–25
22. Ekman M, Stenstrom P (2003) Performance and power impact of issue-width in chip-multiprocessor cores. In: Proceedings 2003 International Conference on Parallel Processing, pp 359–368
23. Gordon-Ross A, Vahid F (2005) Frequent loop detection using efficient nonintrusive on-chip hardware. *IEEE Trans Comput* 54:1203–1215
24. Krishna R, Mahlke S, Austin T (2003) Architectural optimizations for low-power, real-time speech recognition. In: Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, pp 220–231
25. Lau J, Schoenmackers S, Sherwood T, Calder B (2003) Reducing code size with echo instructions. In: Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, pp 84–94
26. Mathew B, Davis A, Fang Z (2003) A low-power accelerator for the SPHINX 3 speech recognition system. In: Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, pp 210–219
27. Suresh DC, Agrawal B, Yang J, Najjar W, Bhuyan L (2003) Power efficient encoding techniques for off-chip data buses. In: Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, pp 267–275
28. Zhang W, Kandemir M, Sivasubramanian A, Irwin MJ (2003) Performance, energy, and reliability tradeoffs in replicating hot cache lines. In: Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, pp 309–317
29. Zhang Y, Gupta R (2003) Enabling partial cache line prefetching through data compression. In: Proceedings 2003 International Conference on Parallel Processing, pp 277–285
30. Eberhart RC, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the 6th International Symposium on Micro Machine and Human Science, pp 39–43
31. Kennedy J (2011) Particle swarm optimization. In: Encyclopedia of machine learning. Springer, New York, pp 760–766
32. Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In: 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, pp 4104–4108
33. Shi Y, Eberhart R (1998) A modified particle swarm optimizer. In: 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence, pp 69–73
34. Kennedy J, Kennedy JF, Eberhart RC, Shi Y (2001) Swarm intelligence. Morgan Kaufmann, Burlington
35. Liao C-J, Tseng C-T, Luarn P (2007) A discrete version of particle swarm optimization for flowshop scheduling problems. *Comput Oper Res* 34:3099–3111
36. Shi Y, Eberhart RC (1998) Parameter selection in particle swarm optimization. In: International Conference on Evolutionary Programming, pp 591–600
37. Taheri J, Zomaya AY, Khan SU (2012) Genetic algorithm in finding Pareto frontier of optimizing data transfer versus job execution in grids. *Concurr Comput Pract Exp* 28(6):1715–1736