Journal of Computational Science 26 (2018) 318–331

Contents lists available at ScienceDirect

Journal of Computational Science

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

GA-ETI: An enhanced genetic algorithm for the scheduling of scientific workflows in cloud environments

Israel Casas[a,b,*], Javid Taheri[c], Rajiv Ranjan[b,e], Lizhe Wang[d], Albert Y. Zomaya[a]

[a] School of Information Technologies, The University of Sydney, Sydney, Australia
[b] Data61, CSIRO, Australia
[c] Department of Mathematics and Computer Science, Karlstad University, Karlstad, Sweden
[d] School of Computer Science, China University of Geosciences, China
[e] China University of Geosciences, Wuhan, China

ARTICLE INFO

Article history:
Received 17 May 2016
Received in revised form 8 August 2016
Accepted 31 August 2016
Available online 4 September 2016

Keywords:
Cloud computing
Scientific workflow
Scheduling algorithms
Genetic algorithm
Virtual machine

ABSTRACT

Over recent years, cloud computing has become one of the main sources of computer power to run scientific experiments. To cope with these demands, cloud providers need to efficiently match applications with computing resources to maintain an acceptable level of customer satisfaction. A correct match or scheduling of scientific workflows relies on the ability to fully analyze applications prior to execution, analyze characteristics of available computing resources, provide users with several scheduling configurations, and guide users to select the optimal configuration to execute workflows. To date, different schedulers have been proposed to execute complex applications on cloud environments; nevertheless, none exists, to the best of our knowledge, to provide all the aforementioned features. GA-ETI, the scheduler proposed in this work, is designed to address all aforementioned concerns by providing several efficient solutions (in a Pareto Front fashion) to run scientific workflows on cloud resources. Flexibility of optimization procedure of GA-ETI allows it to easily adapt to different types of scientific workflows and produce schedules that effectively exploit/consider the relationship between jobs and their required data. GA-ETI acts as an interface between cloud user and cloud provider in receiving an application, analyzing it, and distributing its tasks among selected resources. GA-ETI differs from the majority of proposed schedulers because it can adapt to the size of both jobs and virtual machines, it includes a monetary cost model (from a public cloud), and it considers complex interdependencies among tasks. We test GA-ETI with five well-known benchmarks with different computing and data transfer demands in our VMware-vSphere private cloud. Through experimentation, GA-ETI has been proved to reduce makespan of executing workflows between 11% and 85% when compared to three up-do-date scheduling algorithms without increasing the monetary cost. GA-ETI opens the way to develop a top-layer-scheduler for a workflow manager system to provide a complex analysis and include different optimizing objectives.

1. Introduction

Cloud computing is facilitating an impressive shift in how organizations meet their computing needs. Through massive integration of powerful computing servers and enormous data storage units, cloud systems deliver three deployment models: IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service). Cloud-powered applications (i.e. applications using cloud deployment models as a platform) are globally used, causing a dramatic growth of cloud resource usage. Scheduling this extraordinary number of computing tasks and distributing the data files they require are critical concerns for cloud providers.

The scientific community is providing ever-growing problems to the information technology community, demanding overwhelming computing power to fulfill their needs [1]. As an example, nucleotide sequencing machines in genomics are producing more data each year than non-dedicated computing machines are able to process; as a reference, nucleotide sequencer capacity reported a growth of three to five times per year while computer processor speed only doubles every two years (following Moore's Law benchmark) [2,3]. Big-data analytics to process

* Corresponding author at: School of Information Technologies, The University of Sydney, Sydney, Australia.
E-mail addresses: israel_fime@hotmail.com, icas8033@uni.sydney.edu.au (I. Casas), javid.taheri@kau.se (J. Taheri), rajiv.ranjan@csiro.au (R. Ranjan), lizhe.wang@gmail.com (L. Wang), albert.zomaya@sydney.edu.au (A.Y. Zomaya).

https://doi.org/10.1016/j.jocs.2016.08.007
1877-7503/

extraordinarily large amounts of data, in the order of terabytes and beyond, are another type of scientific application that requires and demands efficient storage systems [4,5]. For instance, the Large Hadron Collider [6] produced ~13 petabytes of data in 2010 and the Large Synoptic Survey Telescope [7] coming online in 2016 is projected to produce 10 petabytes of data a year [3]. Similarly, it is calculated that labs and hospitals around the globe are able to provide around 15 quadrillion nucleotides per year, i.e. 15 petabytes of compressed genetic data.

As well as their storage requirements, scientists aim to obtain analytical results from collected data. To accomplish this task, researchers automate their experiments as scientific workflows, i.e. scripts to call in data and computer programs to analyze and obtain insights from retrieved data. Cloud computing presents the best environment to execute workflows due to 1) large computing power and extraordinary volume data storage, 2) unlikely grids, any public user can access resources at a cost established by a cloud provider, 3) it doesn't require an initial investment in supercomputers or specialized clusters, 4) in contrast to clusters, resources can scale up and down adjusting to workflow demands and 5) users can access necessary resources immediately in contrast to supercomputing where a waiting period of weeks may be common.

Cloud systems are linked to scientific workflows through a scheduling platform. This platform, commonly referred to as the scheduler, receives an application from the user, analyzes it and assigns it to a computing resource. It is expected that scheduling analysis has a relatively small duration in comparison with the total workflow execution time. However, cloud schedulers often require complex analysis since they manage a large number of variables such as network bandwidth, instance types, tasks' computing demands, data file sizes, and dependencies among others. Given the extraordinary number of possible solutions this scenario provokes, the scheduling of workflows falls into the type of an NP-complete problem [8], i.e. a problem that cannot be solved within polynomial time using current computing systems.

After deep analysis of a large number of scheduler proposals, we observe that there is not an accurate investigation of cloud schedulers that manages both computing and data intensive applications with task interdependencies considering a number of resources as a variable within its process [9–16]. The investigation also discovered that a lack of a proper cost model [9,12,15] prevents current algorithms from analyzing a realistic scenario. Every public cloud provider offers its resources at a price per quantum of time, usually hours. Whether the user has an unlimited budget or not, idle resources stop cloud providers from assigning those resources to a different user, affecting the complete system efficiency.

Additionally, we noticed schedulers do not create realistic scenarios, most of them realize experimentation over a fixed pool of resources; few of them dually optimize execution time and monetary cost using a public cloud pricing model. Runtime and monetary cost objectives have great importance to the execution of workflows: on one hand, execution time has a direct impact on variables such as reliability, security and energy consumption; on the other hand, monetary cost represents the pay-as-you-go model of public cloud providers. GA-ETI is an approach we developed to address this problem, it is able to (1) evaluate configurations with a different number of resources, (2) employ the Amazon EC2 cloud pricing model [17] and (3) converge to an optimal, minimizing makespan and monetary cost.

For the reasons outlined above, this paper addresses the problem of scheduling scientific workflows in cloud environments employing a genetic algorithm. The contributions are: (1) a cloud scheduler for the optimization of execution time and monetary cost; (2) modification of mutation operator to scale up/down the number of resources to provide the exact number of required VMs to a user; additionally provide estimation of (i) makespan and (ii) monetary cost; and (3) a scheduler orientated for computational and data-intensive scientific workflows contemplating (i) time to transfer data files, (ii) time to execute each workflow task and (iii) dependencies among tasks. Results demonstrate that GA-ETI successfully balances the conflicting objectives and outperforms up-to-date cloud schedulers in scheduling current scientific applications.

This work is divided into eight sections organized as follows: Section 2 presents related work; Section 3 outlines the system model; Section 4 discusses the problem statement; Section 5 provides a detailed analysis of the GA-ETI; Section 6 analyzes experiments; Section 7 provides a discussion of results; finishing with Section 8, the conclusion.

2. Related work

Numerous algorithms have already been proposed targeting the scheduling problem in cloud environments. From an extensive analysis, we identified two distinctive characteristics. Firstly, applications are orientated for BoT (Bag of Tasks) or DAGs (Direct Acyclic Graph). BoT applications have parallel tasks independent from each other without contemplating task or data dependencies [18,19] while DAGs are an organization of nodes (tasks) connected by edges (data files) where node weight denotes computing demands and edge weight denotes file size. Secondly, the selection of VM pool size is (i) driven by a monetary cost constraint, (ii) selected by the user or (iii) computed by the scheduler. To date, most DAG schedulers still miss important opportunities to manage specific task dependency patterns as in scientific workflows and only a few of them offer a complete framework computing the correct number of resources, leaving this decision to the user without any guidelines or in the best case it is indirectly driven by a monetary constraint.

Kloh et al. [10] developed a bi-criteria approach to schedule DAGs in cloud environments with monetary cost constraint guidance to select a number of resources. This approach optimizes two variables out of runtime, cost and/or reliability. The process starts with the application owner selecting two objective variables, then incorporating them into well-known scheduling mechanisms including bi-criteria scheduling [20], dynamic scheduling [21], fault tolerance policies [22], cost-based scheduling [23], multiple QoS constrained schedule strategy [24], and scheduling decisions based on service level agreements [25]. Through experimentation, the authors prove this bi-criteria scheduler is superior to the Join the Shortest Queue (JSQ) [26]. Although this work employs a number of resources selected at a high level of abstraction, it does not provide a guarantee of optimum system utilization. Additionally, it does not provide information to application owners to decide which service class better suits execution of their application.

Achar et al. [14] created a scheduling algorithm orientated for BoT applications with VM pool size selection dictated by the user. This approach first prioritizes tasks and VMs based on MIPS (Million Instructions per Second). Then it groups tasks and selects the best group of VMs to execute them. This algorithm obtains high resource utilization with low execution times compared to FCFS (First Come First Serve). Nonetheless, they employed a different number of VMs in their experiments without any methodology to select them, leaving this selection to the user with no guidance provided. Additionally, this algorithm does not consider network behavior as a key factor in scientific applications with large data file sizes.

A scheduler based on the Ordinal Optimization (OO) method was developed in [13]. It executes scientific workflows on a fixed number of resources. The authors' objective was to decrease the scheduling time overhead by reducing the solution space. For this reason, they modified the OO method originally developed for

automated systems. Experiments prove this modified OO presented a reduced scheduling overhead compared with the Monte Carlo method, an algorithm based on repetitive random sampling. However, the modified OO neither scales up nor down the number of resources nor provides user guidance for this selection.

Deng et al. [9] produced a linear programming algorithm to map applications to a static pool of resources/VMs with the objective of reducing monetary cost and response time. The nature of the scheduling decisions, based on arrival time, identifies this approach as a BoT scheduler. Although the scheduler can handle a different number of resources, this solution lacks a precise policy to select the number of VMs and/or to limit the number of resources by a monetary constraint. Furthermore, it does not consider applications with interdependent tasks.

Moschakis et al. [11] delivered a study for the execution of tasks on a different number of resources/VMs arranged on the Amazon Elastic Compute Cloud (EC2). This dynamic scheduling approach considers BoT applications arriving with exponential distribution time. Their experiments evaluated AFCFS (Adaptive First Come First Serve) and LJFS (Largest Job First Serve). AFCFS executes tasks as soon as they arrive while the LJFS executes tasks previously prioritized according to their computing requirement demands. Nevertheless, this approach lacks an analysis to handle applications with task dependencies; for this reason, this scheduler is inappropriate for scientific workflows.

Tsakalozos et al. [12] proposed a scheduler with a flexible selection of VMs that balances cloud revenue and user budget. FSV (Flexible Selection of VMs), as we will refer to this algorithm for the rest of this article, calculates the number of VMs required to execute an application based on its computing demands. This scheduler has its base in microeconomics. On one side, the users' objective is to execute their application spending their budget. On the other side, cloud providers aim to maximize their revenue. In a microeconomic context, both parties must reach an equilibrium satisfying their requirements. This scheme provides a fair position for application owner and cloud provider. However, it only balances the number of VMs based on computing demands, i.e. it uses the maximum number of VMs without considering monetary cost optimization, as a consequence users spend their entire budget without contemplating options with a different number of VMs.

Oliveira et al. [16] propose a scheduler where the number of VMs scales up and down based on provenance data captured during execution. Provenance, as we will refer to this scheduler for the rest of this article, bases its analysis on a realistic scenario where application owners need to execute workflows considering dependencies among files and tasks. This approach optimizes execution time, monetary cost, and reliability. It first groups tasks requiring similar input files and then selects a set of VMs to execute them based on the defined cost function. Experiments proved Provenance superior to the MapReduce [27]. However, Provenance does not produce a full scheduling plan before execution: i.e. it first schedules and executes groups of tasks as they become ready for execution. This prevents it from analyzing the full workflow in a single analysis. Finally, Provenance does not help users to select an optimal configuration based on their needs.

HEFT (Heterogeneous Earliest-Finish-Time) [15] is a well-known scheduler for achieving high performance in computing environments. HEFT addresses the scheduling problem for heterogeneous systems with a low scheduling overhead time. HEFT first organizes tasks based on their rank-values. HEFT assigns a rank to each task based on (1) their computation demands and (2) computation demands from their descended tasks. The second parameter is calculated recursively upwards from the last workflow node toward the first one. Tasks are then organized into a single list in decreasing order. After that, HEFT chooses the task on top and assigns it to the VM that will execute the task in the earliest finishing time. This process is repeated until all tasks are scheduled. Though HEFT provides reasonable scheduling plans in a relatively short period of time, its scheduling decisions only analyze a single task without considering the impact of this decision on descendent nodes.

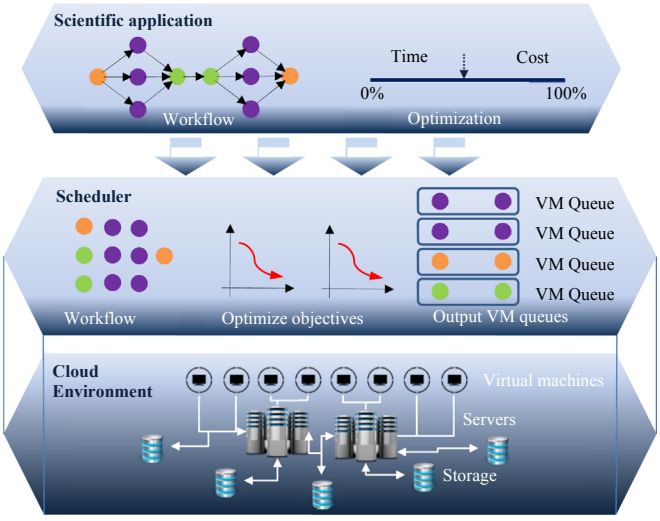

Fig. 1. The scientific workflow execution in a cloud environment is described as a layered architecture. The top layer comprises the information a user provides; the Scheduling layer links the scientific application to cloud environments, and the bottom layer presents a cloud environment.

From this related work we observe that most schedulers generate scheduling plans based on a fixed number of resources (e.g. VMs). Some exceptions permit users to use different budgets, directly affecting the number of VMs and execution time. In addition, some schedulers are designed for a BoT type application with no dependencies between tasks. This factor prevents them from being used in scheduling of scientific workflows on clouds.

In order to address the limitations of existing scheduling solutions, we developed GA-ETI, a scheduler based on a Genetic Algorithm. In summary, GA-ETI (1) has a well-organized structure to efficiently map tasks with file dependencies for scientific workflows, (2) enables cloud users to optimize execution time and monetary cost, and (3) scales up/down the number of resources during the scheduling process and provides users with the efficient number of VMs for their particular applications.

3. Architecture of GA-ETI

This section presents the study model and parameter definitions in order to define the scheduling problem in the next section. In this study, the architecture for scientific workflow execution in cloud environments is divided into three layers: (1) a **Scientific Application** aims to be executed on a cloud system, optimizing runtime and monetary cost; (2) a **Scheduler** stage acting as a bridge between the cloud environment and the scientific application with the goal to distribute a workflow's tasks; and (3) a **Cloud Environment** containing a group of servers offering VMs on a pay-as-you-go basis. An illustration of the aforementioned framework and parameter definition is presented in Fig. 1 and Table 1, respectively. The first layer, Scientific application, considers a workflow $\boldsymbol{W}$ aiming to be executed in a cloud system where a user is able to indicate optimization levels for execution time and monetary cost, w_1 and w_2 respectively. The second layer, Scheduling, receives workflow description, analyzes it and distributes tasks among the available resources assembling a queue for each VM. Finally, a Cloud Environment is a physical place hosting a group of servers and storage devices providing computing power to clients through

Table 1
Parameter definitions for the execution of workflows on cloud systems.

Parameter	Description
$\mathbf{W} = \{t_1, \ldots, t_n\}$	Set of tasks for the workflow W
$\hat{t}_f^i$,	Time to transfer file f required by task t_i
f_i^{size}	Size of the ith file
$\hat{t}_i^{exe}$	Task execution time
$t_i^{parents}$	Set of parents for t_i
$MSpn$	Makespan to execute workflow (see Eq. (1))
$MCst$	Monetary cost to execute workflow (see Eq. (2))
vm_j^{queue}	Scheduling queue assigns to vm_j
$\hat{vm}_j^{time}$	Estimated time to execute vm_j^{queue} on vm_j
$idle_{vm}^{task}$	Time vm remains idle waiting for the execution of $task$'s parents
$\boldsymbol{VM} = \{vm_1, \ldots, vm_{\|VM\|}\}$	Pool of |VM| machines
vm_j^{cores}	Total number of virtual cores in vm_j
vm_j^{cost}	Monetary cost for vm_j per quantum of time
vm_j^{mem}	VM's main memory size
vm_j^{disk}	VM's hard disk size
vm_j^{bw}	VM's network bandwidth

virtualization. Each client has the option to select a number of resources to build his/her particular group of resources ***VM***.

This model contemplates six assumptions: (1) scheduler analyzes and executes one workflow at a time, (2) scheduler accepts both computing and data-intensive workflows, (3) every VM has a fixed bandwidth (vm_j^{bw}), number of cores (vm_j^{cores}), cost per quantum of time (vm_j^{cost}), memory size (vm_j^{mem}) and disk size (vm_j^{disk}), (4) our proposed scheduler, GA-ETI, negotiates with the cloud provider to obtain the required VMs prior to execution of each workflow, (5) users must supply or estimate execution time $\hat{t}_i^{exe}$ for every task of W, as envisioned on Pegasus-WMS, and (6) resource deployment assumes each VM executes one task at a time.

In order to provide a realistic application to test the proposed solution, this study includes five scientific workflows representing different scientific applications extracted from [28] as presented in Fig. 2. On each workflow, nodes are represented by a circle containing a single task t_i with its input set of file(s) of size f_i^{size}. Depending on how nodes are related, five main workflow structures/distributions are highlighted: (1) *Pipeline* structure connects nodes serially, (2) *Data distribution* highlights a set of nodes requiring a single set of input files, (3) *Data aggregation* represent nodes requiring files from at least two other nodes, and (4) *Data redistribution* highlights nodes combining structures (2) and (3) requiring and producing files for multiple nodes. In order to organize workflow analysis we define *w-level* as the number of workflow levels and *parallel-tasks* as the maximum number of tasks a workflow can execute in parallel. For instance, the Montage workflow has nine *w-levels* and *parallel-tasks* has a value of six as exhibited in Fig. 2.a.

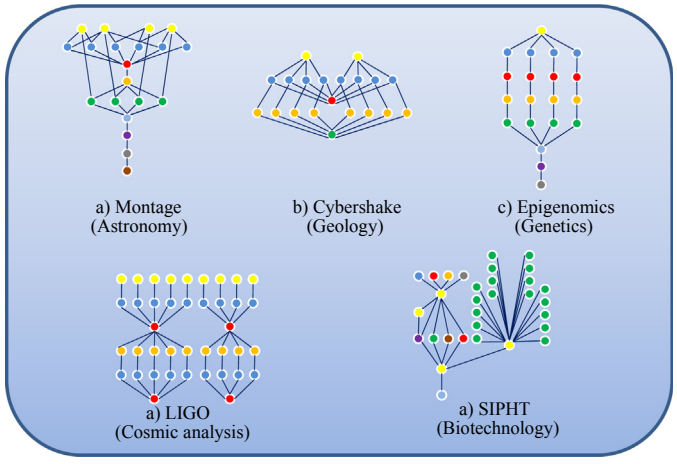

Fig. 2. Illustration of five well-known scientific workflows exhibiting different dependency patterns representing applications from different research areas. The proposed approach can be used to schedule each of these workflows.

This study based its model on our last work [29], research focusing on the balancing of task queues in the execution of scientific applications in cloud environments. Specific modifications to its model were made to enhance the output of this research: firstly, we introduce the term $idle_{vm}^{task}$ to decrease the overhead scheduling time generated when calculating the execution of tasks from a given VM. This term comprises the execution time from all parent tasks from a particular task. It is embedded in the vm_j^{queue} for time calculation purposes only. In this sense, the model avoids recurrent execution time calculation whenever it finds task interdependency with other VMs; as a result analysis of a complete scheduling configuration time obtained a reduction of 40% on our experimental practice. Secondly we include a resource utilization constraint in order to procure an efficient usage of resources. For our experiments, this study takes only the scheduling configurations with the highest utilization resource values. Fig. 3 presents an example for the calculation of utilization for a set of three VMs. Resource vm_3 remains busy for two hours while vm_1 and vm_2 execute their loads in 1.8 h. For this reason, the cloud provider charges the user for two hours for each machine causing a utilization resource of 0.933.

4. Problem statement: scientific workflow scheduling

Scientific Workflow Scheduling (SWS) is a problem defined as assigning tasks to virtual machines to minimize: (1) total makespan to execute all workflow tasks and (2) monetary cost a user pays to have his/her application completed. To formally express this problem, assume tasks are clustered and assigned to several VM queues $\{vm_1^{queue}, vm_2^{queue} \ldots vm_{|VM|}^{queue}\}$ to be executed by $\{vm_1, vm_2 \ldots vm_{|VM|},\}$ respectively. A cluster of tasks is defined as a decomposition of a workflow's tasks set into disjoint subsets of which the union is the original set. For instance, a pool of |VM| = 2 machines with $vm_1^{queue} = \{t_1, t_2\}$ and $vm_2^{queue} = \{t_3, t_4\}$ is executing a workflow of four tasks $W = \{t_1, t_2, t_3, t_4\}$.

Given the above description, an SWS problem is stated as defining a pool of resources ***VM*** and assigning workflow (***W***) tasks to each virtual machine to minimize their execution time:

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

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

Makespan (Eq. (1)), interchangeably referred to as runtime and execution time through the text, is the value of the *LFT* (Latest Finishing Time) from all the VMs executing workflow W, while monetary cost is the sum of all VMs' cost multiplied by their respective round up runtime to the closest integer as expressed in Eq. (2). In order to reduce monetary cost presented in [29], this problem statement allows machines to be launched at different times by the cloud provider. With this modification our proposed SWS solver has the freedom to employ a particular VM for a specific interval. This modification led to a resource utilization improvement of up to 30%. Similarly, *MCst* only considers the amount of time each machine is hired, yet cloud providers charge an hourly rate. Eq. (3) expresses the time each VM takes to execute its corresponding load, where vm_j^{queue} refers to the list of tasks assigned to vm_j. In order to reduce the scheduling overhead time presented in [29], this study

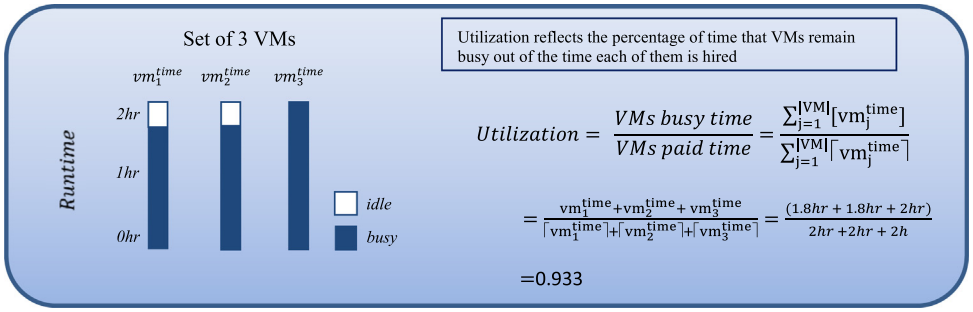

Fig. 3. Resource utilization example. This parameter measures the time VMs remain active during workflow execution.

introduces the term $idle_{vm}^{task}$ (Eq. (4), exectuion time of parent task) in the calculation of vm_j^{time}. In this sense, every vm_j^{queue} would contain the complete information from all tasks to calculate its total runtime and monetary cost without waiting for its calculation on a different vm_j^{queue}. This will enable the SWS solver to reduce the overhead time at the scheduling stage. An illustration of vm_j^{time} calculation is presented in Fig. 4.

$$vm_j^{time} = \sum_{i=1}^{|vm_j^{queue}|} \left[\hat{t}_i^{total} + idle_{vm}^{task}\right] \tag{3}$$

$$idle_{vm}^{task} = \sum_{i=1}^{|parents|} \hat{t}_i^{total} \tag{4}$$

The value of $\hat{t}_i^{total}$ expressed in Eq. (5), contemplates the time to transfer the required n number of files and the time to run a task's executable program, $\hat{t}_i^{exe}$, which is a value provided by the user

$$\hat{t}_i^{total} = \sum_{f=1}^{n} \hat{t}_i^{f} + \hat{t}_i^{exe} \tag{5}$$

Finally, the time to transfer each file, $\hat{t}_i^{f}$, depends on the bandwidth of the VM's parents vm_p^{bw} andvm_i^{bw}, expressed as:

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

5. Genetic algorithm with efficient tune-In of resources

The Genetic Algorithm (GA) is a metaheuristic motivated by genetic evolution with important features for combinational optimization. It is a robust technique to solve complex problems in engineering and science due to its ability to detect a global optimum in the complete search space [30]. Contrary to current heuristics solutions [11,13,14] GA does not build a single solution. Instead, it

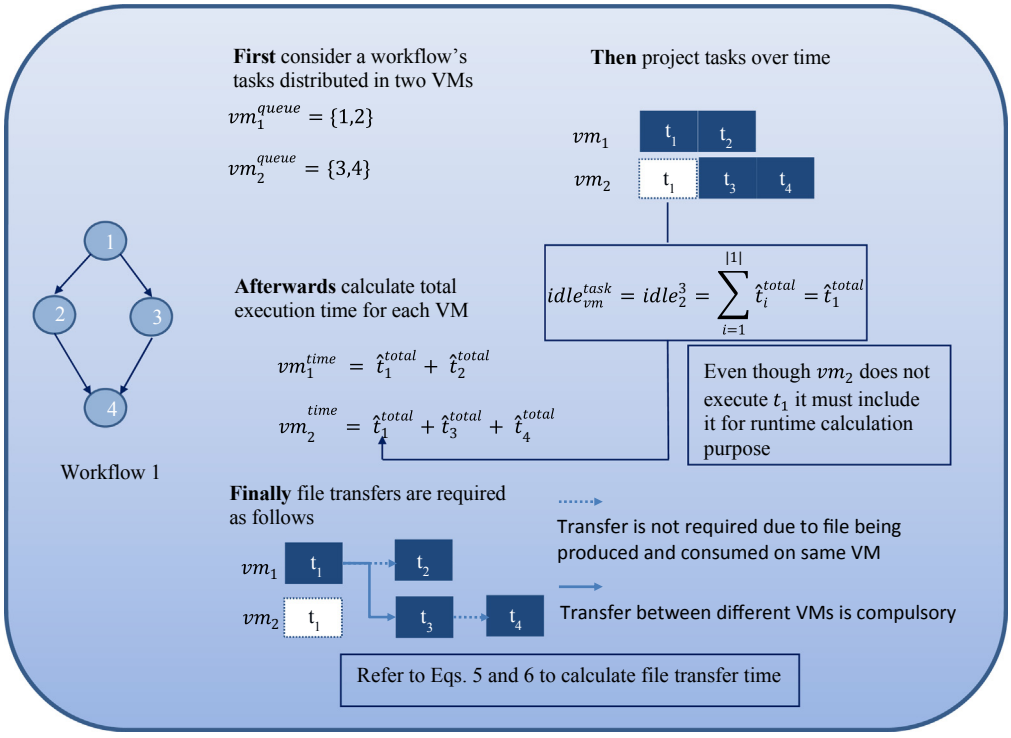

Fig. 4. Example of runtime calculation for a 4-task workflow. Runtime calculation includes the waiting time for each parent task whether or not they are assigned to the same VM.

applies genetic operators to current configurations (parents) with the objective of generating stronger solutions (offspring) from evolution [31]. For these reasons, this study modified the GA into the GA-ETI to solve the SWS problem. This section presents the fundamentals of the GA and its adaptation for the cloud scheduling problem.

5.1. The genetic algorithm (GA)

The original GA has an initial population that starts the GA with a group of possible solutions [32]. Each chromosome is a string of genes encoding a specific solution. The particular nature of an optimization problem defines chromosome and gene characteristics. Through the genetic process, GA selects fittest chromosomes, combining them to produce a final strong solution. The first phase to produce a new population is the selection operator. Its objective is to select chromosomes to produce the next population [31]. A frequently used selection technique is the roulette wheel where each chromosome is allocated a portion of the wheel according to its fitness value, hence chromosomes with greater values are allocated more slots with more chances to be selected for the next population. Then, the genetic operators combine chromosomes to hopefully produce chromosomes with higher fitness values: (1) Crossover splits and combines genes between two selected chromosomes according to a predefined probability; (2) Mutation randomly selects genes from a chromosome and changes their values according to another predefined probability. Additionally, the fittest chromosomes are directly copied to the next population. Finally, GA terminates when it meets selected criteria. The most used criteria are total execution time, the number of iterations, fitness value, and conditional minimum improvement [30–33]. The fitness function in GA evaluates the quality of each chromosome. For maximization problems, the fitness function is proportional to the problem cost function while minimization problems use the inverse value of this equation.

5.2. The GA-ETI in solving the SWS problem

This section presents the enhanced Genetic Algorithm with Efficient Tune-In of resources inspired by the fundamentals of the genetic process.

$$Fittest\ Solution = Max_{i=1}^{total} F_{fitness}$$

Algorithm 1: GA-ETI

Input: Workflow *W*, *VM* set
Output: Scheduling plan
1: **Generate** preliminary population (Algorithm 2)
2: Initial population ⟵ Select fittest chromosomes
4: **While** time **or** cost **still improve**
5: Evaluation
6: Selection
7: Crossover
8: - Conventional crossover
9: - Clustered crossover
10: Mutation
11: - Swap
12: - Increment VMs
13: - Decrement VMs
14: **End**
15: $Fittest\ Solution = Max_{i=1}^{total} F_{fitness}$

GA-ETI's objective is to schedule workflows to cloud environments to optimize monetary cost and execution time. We carefully adjust genetic operators to distribute a workflow's tasks on VM queues. Algorithm 1 presents the GA-ETI with its featured components. Firstly in step 1, GA-ETI uses Algorithm 2 to generate a preliminary population with a size *IP* greater than a regular population. Then, step 2 reduces this preliminary population to a regular size selecting the fittest chromosomes. Steps 4–14 develop the main loop. Step 5 evaluates every chromosome using Eq. (7). Then in step 6 a quarter of the fittest chromosomes are directly copied to the next population for elitism and the rest of the chromosomes are selected using the roulette wheel. Afterwards, steps 7–9 apply one-point and multiple-point crossover to a selected grouped of chromosomes to produce offspring followed by a mutation operator in steps 10–13. The algorithm stops when neither *MSpn*n or *MCst* present an improvement by returning chromosomes with the highest fitness value (step 15).

$$F_{fitness} = w_1 \frac{(max^{MSpn} - MSpn)}{(max^{MSpn} - min^{MSpn})} + w_2 \frac{(max^{Cst} - MCst)}{(max^{Cst} - min^{Cst})} \quad (7)$$

5.2.1. Chromosome and fitness function description

Chromosomes represent a complete workflow scheduling where each gene represents a task and the required VM to execute it; hence, chromosome length equals the size of the given workflow $|W|$. Fig. 5 describes a chromosome with an example. On it, the position of $gene_1$ represents $task_1$, $gene_2$ represents $task_2$ and so on. Similarly, the value of $gene_1$, **1**, expresses that vm_1 executes the represented task, in this case $task_1$; value of $gene_2$, **1**, assigns $task_2$ to vm_1 and so on.

Eq. (7) assigns a fitness value to each chromosome based on its makespan and monetary cost on every iteration of Algorithm 1 (steps 4 − 14). $F_{fitness}$ keeps a record of maximum (max^{MSpn} and max^{Cst}) and minimum (min^{MSpn} and min^{Cst}) values of *MSpn* and *MCst* in order to provide a global evaluation to each solution; these values update on each iteration on the main loop of GA-ETI. Additionally, the fitness equation enables the user to assign priority to a given optimization objective employing w_1 and w_2 as time and cost optimization weights respectively where $w_1 + w_2 = 1$.

Algorithm 2: Creating the Initial Population

Input: Workflow *W*
Output: Initial population
1: *genes* = number of tasks on workflow
2: *largest-level* = largest workflow level
3: *parallel-tasks* = number of the tasks in *largest-level*
4: **Set** *IP*
5: **For** j = 1:IP
6: **For** k = 1: size of population
7: **For** i = 1:genes
8: $gene_i$ = random value from [1 to *parallel-tasks*]
9: $chromosome_k \longleftarrow gene_i$
10: **End**
11: **End**
12: pre_initial_population ⟵ $chromosome_k$
13: **End**
14: **Return** pre_initial_population

5.2.2. Pre-initial population

Algorithm 2 leads to an initial population with fittest chromosomes for the GA-ETI. This algorithm first produces a larger initial pre-population, then it reduces the population selecting the best chromosomes to build the first generation. Algorithm 2 firstly identifies the number of *genes*, *largest-level* and *parallel-tasks* and then assigns a random VM to each task in steps 1–3. Since workflows do not require an unlimited number of resources, the algorithm limits the size of ***VM*** to *parallel-tasks* which are the maximum number of tasks that can run in parallel. The main loop in steps 4–13 executes *IP* times to build the pre-initial population. The loop in steps 7–10 assigns a random value to each gene on every chromosome with values from 1 to *parallel-tasks*. Finally, step 12 returns the initial population of a regular size. Founded on our practical tests, the best results were obtained with an *IP* value of 10.

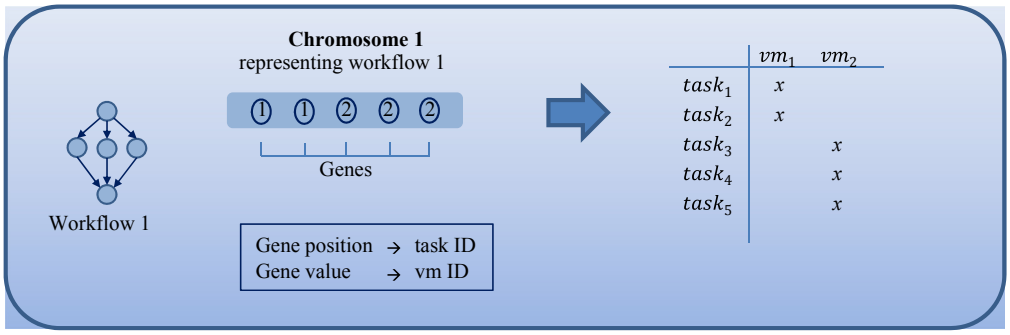

Fig. 5. Chromosome representation. Each chromosome represents a complete workflow scheduling, genes comprising task and VM identification numbers. Gene position represents a task while gene value corresponds to the VM executing that task.

5.2.3. Genetic operators: selection, crossover, mutation

GA-ETI makes use of the roulette wheel for selection of chromosomes for genetic operators. The roulette wheel is a selection process simulating a partitioned spinning wheel. Partition size depends on fitness of its elements. In this study, each element is a chromosome and its partition size depends on its fitness value. Fig. 6 presents a graphical description of the roulette wheel. First it assigns size (p_i) of each partition to every element (chromosome). Then, all partition sizes are assigned to the wheel. Finally, the roulette wheel spins and selects a winner.

GA-ETI adapts the conventional crossover and swap mutation from the original GA to be used in our model. Additionally, a modified crossover and new increment and decrement mutation operators were designed and added to the GA-ETI to produce a powerful tool. Description of these mechanisms is as follows.

5.2.3.1. Conventional crossover. This operator is the accurate adaptation of the original GA crossover into the scheduling problem. It allows the breaking of a pair of chromosomes into a limited number of pieces and then combining their parts in order to produce offspring. Number and location of breaking points are chosen arbitrarily. Fig. 7a presents an instance of this process. Firstly, chromosomes 1 and 2 are selected using the roulette wheel from the population. Then, step 2 highlights that chromosomes can break on any number and location; in this case, only one crossover point is used, dividing chromosomes into two parts each. Finally, chromosomes are combined building offspring 1 and 2.

5.2.3.2. Clustered crossover. This enhanced crossover operator is specially adapted to the scheduling problem. While conventional crossover breaks chromosomes at any location, clustered crossover does not separate genes from the same workflow. This procedure allows GA-ETI to produce newborns combining clusters of genes representing workflow levels. Fig. 7b presents an example of this procedure. It first selects a pair of chromosomes in step 1, then step 2 presents clusters of genes for each chromosome. Workflow 1 is split into four-cluster chromosomes, each one representing a level from the given workflow. From this chromosome division a crossover breaking point(s) is then selected. Finally, chromosomes mix with each other, producing offspring.

5.2.3.3. Swap mutation operator. This study adapts the original GA swap operator to be applied with the GA-ETI. The swap operator produces an offspring from a single chromosome, it first selects a pair of genes and then it swaps their values. A pair of gene values is interchanged in each swap operation. Fig. 8a presents an example of this operation. In step 1, a random pair of genes is selected from the parent chromosome. Then in step 2, selected genes swap their values, producing offspring 1.

5.2.3.4. Increment and decrement mutation operators. Increment and decrement instruments are a modification of the mutation operator to change the number of VMs that a given chromosome uses. Fig. 8b–c explain these operators with an example. The decrement process in Fig. 8b reduces the number of VMs, i.e. gene values. In this example, chromosome 1 has three different gene values (1, 2 and 3) while offspring 1 ends up with only two different values of genes (1 and 3). The procedure starts with step 1, it first selects a random gene, and then it selects every gene with a similar value. In step 3, it lists the different gene values presented on chromosome 1. Finally, in step 4, it selects a random value from the list in step 3 and replaces selected gene(s) from step 2. As for the increment operator in Fig. 8c, it adds a new gene value, i.e. a new VM to the chromosome. As this example shows, offspring 1 ends up with an additional gene value. This operator first selects a random gene value in step 1. Then in step 2, it lists the available VMs that GA-ETI can use, but are not part of chromosome 1, in this particular

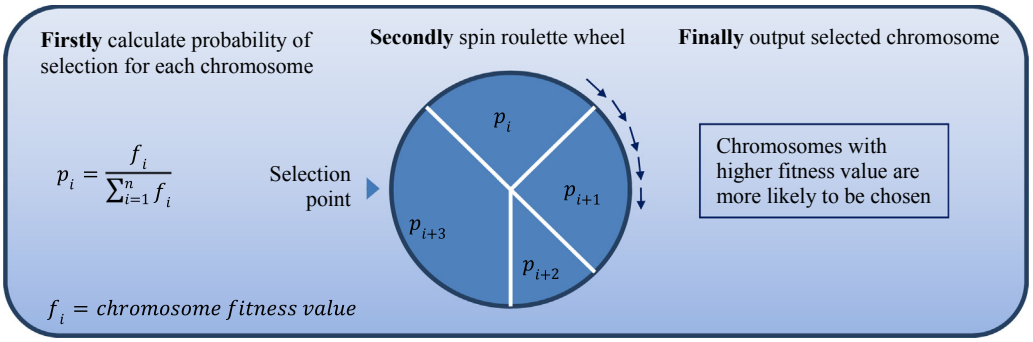

Fig. 6. Roulette wheel. The goal of the roulette wheel in GA is to choose a random chromosome from the population where each chromosome has a probability of being chosen proportional to its fitness value.

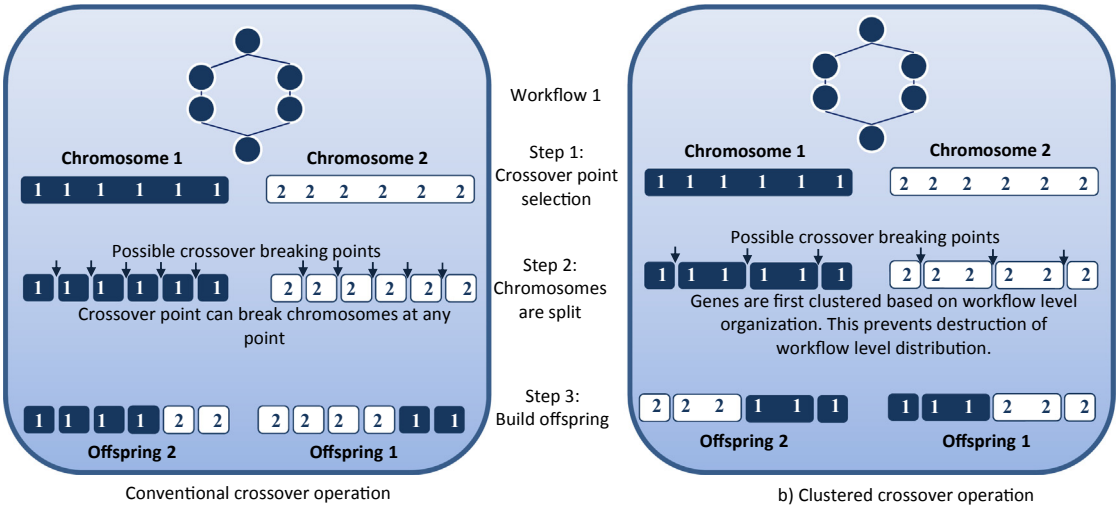

Fig. 7. Crossover operation example. Chromosomes 1 and 2 provide solutions for Workflow 1. On one side, conventional crossover selects any breaking point on chromosomes. On the other side, clustered crossover groups genes in clusters, preventing algorithms from destroying workflow level configurations.

case 4–9. Finally, in step 3, a random value from the mentioned list replaces the selected gene from step 1.

5.2.4. GA-ETI algorithm complexity

In order to measure GA-ETI's complexity, this study defines the growing order of the algorithm. For this purpose, let's first define $T(t, s, l)$ as the number of time units GA-ETI needs in order to produce a scheduling configuration of a given workflow. GA-ETI is divided into five different stages: initial population generation, evaluation, selection, crossover and mutation. The complexity of each stage is extracted from Table 2, hence $T(t, s, l)$ obtains the value of $(t)(s)(IP) + (s) + (s - l) + (0.6)(s)c + (0.3)(s)$. Since values of IP, l and c are constants and s does not depend on the number of tasks then $T(t, s, l) = O(t)$ where $O(t)$ expresses the growing order of GA-ETI as a linear function of the number of tasks in the workflow.

As readers will notice, the number of VMs does not affect complexity since it is only considered as a pool of values where genes initially obtain their identification number (see stage 1 in Table 2). Furthermore, the size of population S appears at every stage but

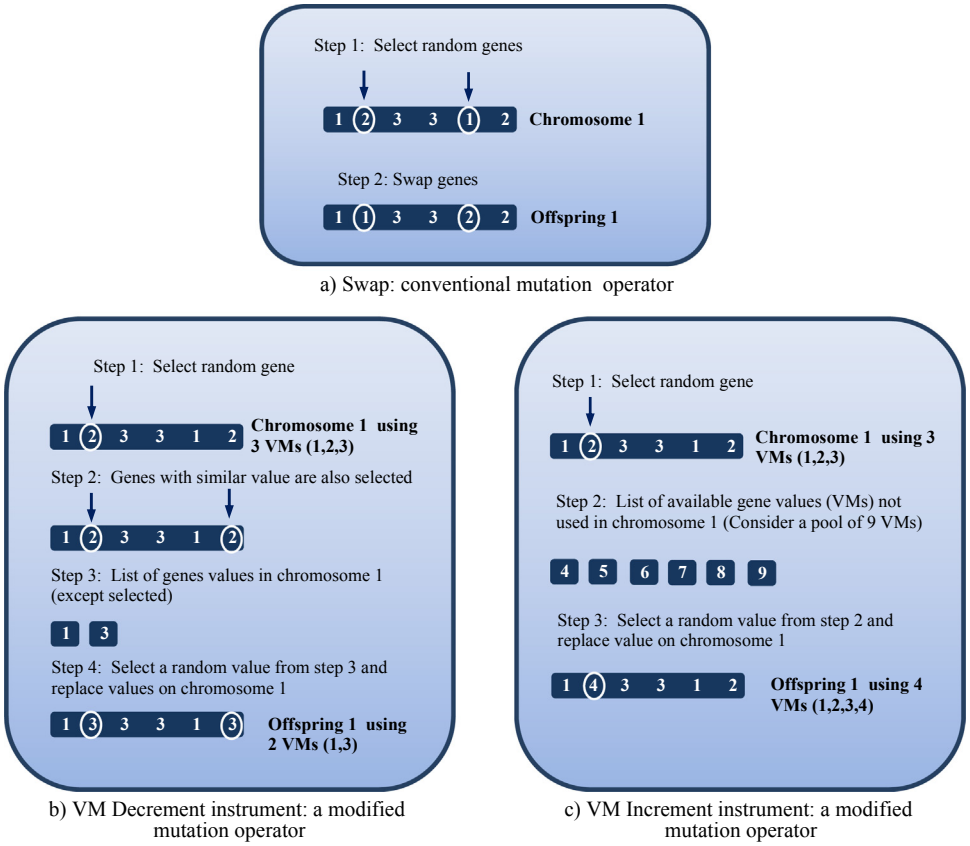

Fig. 8. Example of mutation operators: a) Presents swap conventional mutation process; b) and c) present the modified mutation process to decrement/increment number of VMs on chromosomes.

Table 2
Parameter description to determine GA-ETI algorithm complexity.

Stage	Description	Complexity
Generate initial population	Assign a random number [1:*VM*] to each gene (t) from the complete population (s) augmented	(t)(s)(IP)
Evaluation	Calculates $F_{fitness}$ for each chromosome	(s)
Selection	Run roulette wheel (p) times to build a new population taking off number of chromosomes from elite operator	(s–l)
Crossover	The complete population has a maximum probability of 0.6 to go through crossover operator where each operation depends on a constant number of crossing points	(0.6)(s)(c)
Mutation	The complete population has a maximum probability of 0.3 to go through mutation operator where the number of interchangeable genes remains constant	(0.3)(s)(c)

(t) Number of tasks (genes); (s) Size of population (number of chromosomes); (IP) Pre-initial population factor; (l) Elite size group; *VM* Maximum number of virtual machines; (c) Constant.

Table 3
Characteristics of the scientific workflows employed on 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)

(1) Process; (2) Pipeline; (3) Data distribution; (4) Data aggregation; (5) Data Redistribution.

is not affected by type and size of workflow. Experimentation also reveals that the number of iterations is not affected by either workflow type or size even for the unmodified GA. Growing order of GA-ETI depends only on the number of tasks.

6. Results

The five scientific workflows presented in Section 3 are used to gauge the efficiency of our specific scheduling approach in this work. Table 3 presents details of these workflows. To evaluate the performance of GA-ETI we employed our private VMware-vSphere (version 5.5) private cloud to validate our solutions. Our cloud consists of three Krypton Quattro R6010 s with 4-way AMD OpteronTM 6300 series (64-Cores each). For system management, we employed Pegasus-WMS (4.2) on Ubuntu 14.04 where GA-ETI was implemented with the parameters shown in Table 4. The inputs for our experiments are workflow files including (i) executable files, (ii) data files and (iii) workflow dependency description. The goal of experiments is to test our proposed scheduler and analyze its behavior against up-to-date scheduling algorithms.

This study compares GA-ETI against three up-to-date schedulers in the same field. This algorithm selection includes: Provenance [16] , HEFT [15] and FSV[12] . In summary, Provenance groups tasks on queues depending on their historical execution time and file sizes; HEFT creates a pre-schedule queue based on a critical path and then distributes tasks following an earliest finishing time; and FSV emulates HTCondor's behavior [34] to execute tasks on available VMs. To manage the number of VMs, Provenance increments the number of VMs as long as the monetary cost does not exceed a user's budget; HEFT and FSV use as many VMs as are available.

Table 4
GA-ETI setup for population and genetic operators.

Parameter	Symbol	Value
Crossover probability	p_c	0.60
Swap mutation	p_s	0.25
Increment mutation	p_{inc}	0.20
Decrement mutation	p_{dec}	0.20
Pre-initial population factor	IP	10
Initial population	P	500
Time weight constraint	w_1	0.5
Cost weight constraint	w_2	0.5

6.1. GA-ETI results

For this first experimental stage, the scheduling algorithms have access to as many VMs as parallel tasks in the workflow. Table 5 presents the results. For instance, LIGO is able to use a pool of 24 VMs (see *parallel-tasks* in Table 3). For the Epigenomics workflow, GA-ETI and HEFT produce similar runtime results (21190 and 22890seconds, respectively) as its nodes have a very uniform distribution allowing schedulers to allocate tasks evenly among VMs. In contrast, FSV and Provenance presented higher time values (67325 and 89011seconds, respectively). On one hand FSV allocates tasks to any available VM without considering dependencies causing duplication of data files, on the other hand, Provenance has an internal grouping offset value that groups tasks based on previous executions and not on current tasks. As for the Cybershake workflow, it presents a simple dependency pattern among tasks allowing FSV to obtain similar results to GA-ETI and HEFT; in contrast, Provenance is prevented from delivering better results due to its task grouping policy. Montage workflow highlights the need to analyze dependencies between tasks; for this workflow, GA-ETI's scheduling policy allows groups of tasks sharing a common parent task to be allocated on the same VM in order to lower file transfer time. In summary, HEFT outperformed Provenance and FSV due to its simplistic nature to allocate tasks to VMs, even though HEFT does not analyze job dependencies which prevents it from delivering lower values for time and monetary cost as exhibited by GA-ETI.

7. Analysis and discussion

To show the efficiency of our approach, we also analyzed it from the following perspectives and how each scheduler performs.

Table 5
Execution time and monetary cost results for FSV, GA-ETI, HEFT and Provenance.

	Epigenomics	Cybershake	Sipht	Montage	Ligo
	Time (s)	Time (s)	Time (s)	Time (s)	Time (s)
GA-ETI	**21190**	**4619**	**3587**	**270**	**3486**
HEFT	22890	5199	3687	385	4717
FSV	67325	6549	6106	475	8508
Provenance	89011	8711	5090	550	8340

Bold values highlight the lowest execution time for each workflow.

7.1. Scheduling strategy differences between FSV, GA-ETI, HEFT and provenance

In this section, GA-ETI's behavior is analyzed and compared with the other algorithms. All approaches are forced to produce scheduling plans for all possible VMs. Schedulers start mapping for a single machine incrementing the number of VMs until the execution time does not improve. We have chosen this criterion since increasing the number of VMs beyond such a point only increments monetary cost. Fig. 9 presents these results.

The previous section exhibited HEFT and GA-ETI presenting the lowest runtime for the five workflows. Still, GA-ETI contributed lower values caused by its scheduling policies. For instance, the Cybershake workflow presents particular dependencies where parallel nodes on the second level execute on 127.55 s (63.35 s for task execution plus ∼64 s for 791MB input file transfer on a 100Kbps network); internally, GA-ETI converged to solutions where groups of three of these parallel tasks are assigned to a single VM executing them serially in 254.05 s transferring input files only once to the same VM. In contrast, HEFT executes them in parallel in 127.55 s, transferring input set files to the different machines' VM. Overall, these decisions mean that HEFT requires redundant file transfers and to execute the application in 5199 while GA-ETI only required 4619seconds.

GA-ETI outperforms Provenance because the latter makes groups of tasks based on historical data. For example, the LIGO workflow on its second level has tasks that execute in ∼400 s setting grouping factor to be ∼400. As a consequence on the following level, it tries to group as many tasks as possible to fulfill a total of ∼400 s, even though the next tasks execute in only ∼5 s causing the algorithm to group all tasks on the same VM. In contrast, GA-ETI provides flexibility to allocate tasks according to actual execution times. Finally, GA-ETI overcame FSV because this latter executes workflows using Pegasus and HTCondor's default scheduling policies that are based on VM availability.

7.2. Workflows' particular challenges

Workflows have inherent characteristics such as dependency patterns, execution times and file sizes. These features are targeted in different ways by each scheduling algorithm. This section offers a deep analysis of these characteristics and how algorithms handle them.

For instance, 96% of the nodes from Epigenomics are grouped into 24 pipelines with four nodes in each group. Thus, when HEFT and GA-ETI equally distribute parallel tasks among different resources, execution time drops significantly to 13190sec as shown in Fig. 9a with the 12 VM configuration. As readers will note, a doubling of numbers of resources to 24 does not offer a proportional improvement since the execution time only drops to 13016sec. This is caused by the size of the input files; when parallel tasks are executed on different VMs extra replica files must be transferred as well, incrementing total execution time. For these reasons, algorithms don't employ more resources beyond a particular number of VMs. As for the Cybershake workflow, it exhibits data orientated nodes distributed mainly on two workflow levels. At the scheduling stage, HEFT and GA-ETI achieve similar values with 5105sec with two VMs as shown in Fig. 9b; this is caused by the high task parallelism and uniformity of task execution times. For the same application, Provenance presents a reasonable performance; it sets the grouping value to the highest value (3450sec) executing the workflow at a better time (5000sec.) although it uses 12 VMs causing a higher cost. As for the Sipht application, it has less uniformity in terms of task execution time, dependencies and parallelism when compared with the other applications. In general, all four algorithms do not need a high number of VMs (compared with the number of parallel tasks) to reach minimum execution time for their applications.

Table 6
Optimal number of VMs for HEFT, Provenance, FSV and GA-ETI.

	HEFT	Provenance	FSV	GA-ETI
Sipht	6	6	5	5
Cybershake	3	7	4	3
Epigenomics	6	8	5	24
Ligo	8	18	9	5
Montage	5	6	5	4

7.3. Empirical validation of results from provenance, FSV, HEFT and GA-ETI

In this section, we select the best scheduling configurations from the different algorithms and empirically validate them using our private cloud. In this context, we consider the best scheduling to be the configurations with the optimal outcome in terms of execution time and monetary cost. Results are presented in Table 6; Fig. 10 shows execution time and monetary cost.

Results show that the number of tasks in a workflow does not influence the final number of resources a given application needs. The number of VMs is related to (1) workflow computational requirements, (2) file transfer demands, and (3) task dependency constraints. For each workflow, all approaches select a similar number of resources with only two specific exceptions: GA-ETI on Epigenomics, and Provenance for Ligo. For the first case, GA-ETI converges to solutions employing as many VMs as the number of tasks on the largest workflow level (24). For the second case, Provenance selects a high number of VMs due to the high number of tasks it groups on its first level.

To empirically validate our experiments, we run the best scheduling configurations from each algorithm to expose the difference between the "calculated" and the "actual" execution time when scheduling decisions are implemented/enforced by running applications on top of the Pegasus-WMS (4.2). Results are presented in Table 7. Discrepancies between calculation and Pegasus experiments exhibit the difference between the calculated time (from our formulas and models) and the actual run times; the discrepancies, in percentage, were ∼2.24%. For the Montage application, discrepancies reached a maximum of 3.45%. The reason for this difference is that the application executes in a relatively short time (∼369 s) where difference represents a higher percentage value.

Table 7
Discrepancies between calculation and Pegasus experiments; run time values are expressed in seconds.

		HEFT	Provenance	FSV	GA-ETI
Epigenomics	Calculation	42163	100291	89912	6316
	Experimental	43087	101605	88009	6478
	Discrepancy	2.19%	1.31%	2.12%	2.56%
Cybershake	Calculation	5199	6400	6049	4619
	Experimental	5286	6491	6157	4521
	Discrepancy	1.67%	1.42%	1.79%	2.12%
Sipht	Calculation	3687	5090	6532	2987
	Experimental	3759	5205	6705	2942
	Discrepancy	1.95%	2.26%	2.65%	1.51%
Montage	Calculation	332	426	435	270
	Experimental	321	440	450	279
	Discrepancy	3.31%	3.29%	3.45%	3.33%
Ligo	Calculation	2846	3531	6944	3486
	Experimental	2909	3597	7039	3400
	Discrepancy	2.21%	1.87%	1.37%	2.47%

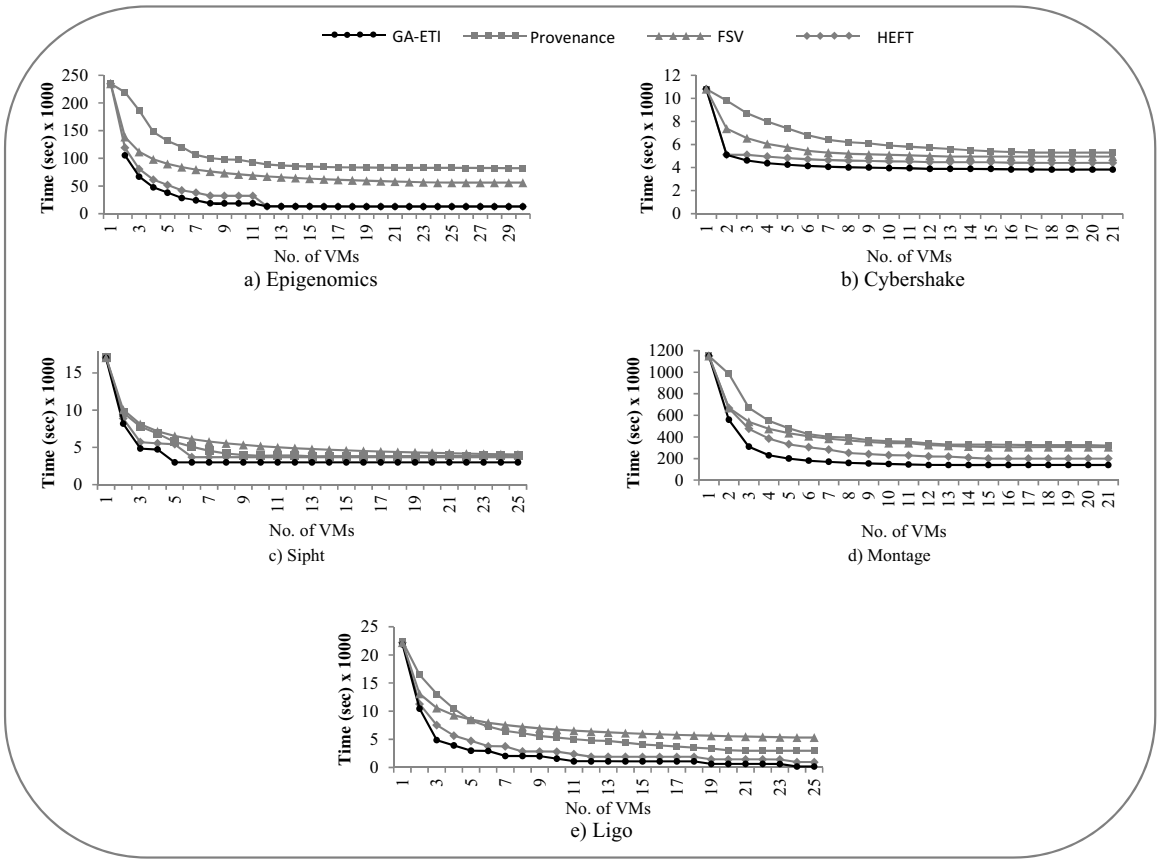

Fig. 9. Execution time results with a different number of VMs.

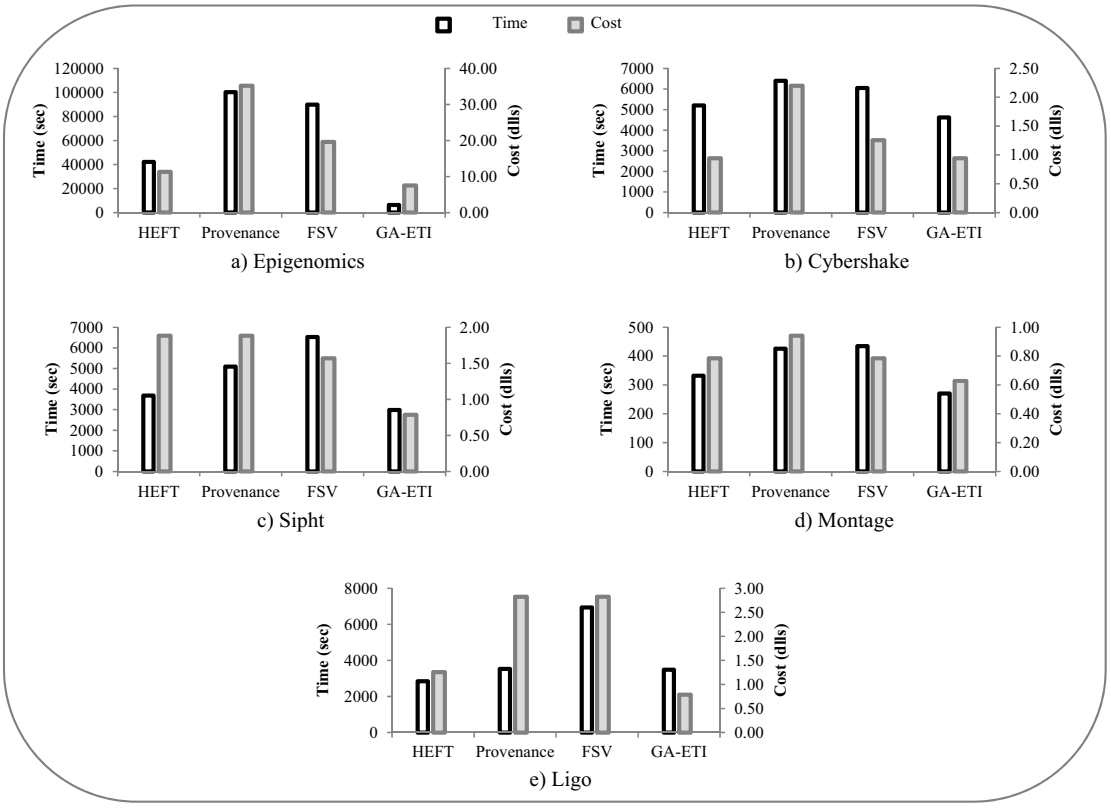

Fig. 10. Execution time and monetary cost selecting for best configuration for each scheduling algorithm.

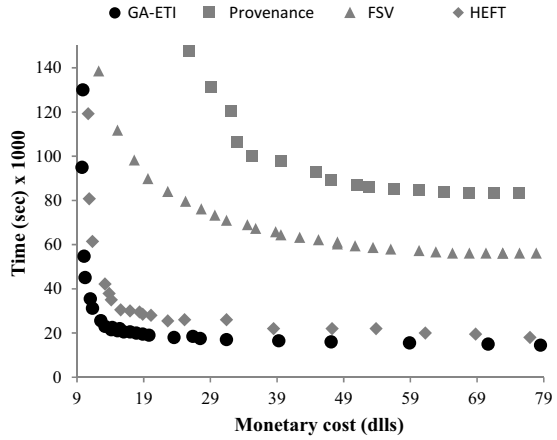

Fig. 11. Epigenomics' MSpn – MCst graph to analyze where in the solution space schedulers search for a solution.

7.4. Global search space

This section presents results for makespan and monetary cost from the four schedulers in order to examine results distributed in the *MSpnvsMCst* space. Fig. 11 presents the *MSpnvsMCst* graph for the Epigenomics workflow since the rest of the applications present similar behavior. On one hand, it is shown that Provenance and FSV present a semi-distribution of their results in the makespan-cost space. On the other hand, HEFT and GA-ETI present a stronger distribution of solutions along the space. This exercise proves GA-ETI considers chromosomes distributed over the complete search space without being trapped at isolated locations. Additionally, GA-ETI's algorithm configuration allows it to consider solutions from the complete solution space without elite chromosomes driving it to specific regions.

7.5. GA-ETI parameters

On this last experiment section, we allowed GA-ETI to produce generations until no benefit is observed. Since workflows present similar behavior in terms of population evolution, this section only presents the results from a single workflow type. Additionally, we included three different workflow sizes for a deep analysis. For evaluation purposes, GA-ETI is compared against the general GA. Original GA uses conventional crossover and a swap mutation while GA-ETI additionally employs clustered crossover, increment and decrement mutation mechanisms. Fig. 12 presents results for population evolution on the Epigenomics workflow.

GA-ETI is able to converge to a satisfactory solution with fewer generations due to its enhanced crossover and mutation operators. These mechanisms complement each other, transforming the original GA into a potent tool to resolve the programming problem. On one side, clustered crossover avoids random selection of crossover points, instead, it first identifies workflow levels then it breaks chromosomes into clusters that later combine to produce offspring. This procedure allows the algorithm to combine the clusters of genes instead of chromosomes being randomly divided. On the other side, increment/decrement mutation provides an instrument to add/remove a particular VM from a chromosome allowing the algorithm to restructure that particular chromosome. The application of the mentioned operators allows GA-ETI to reduce randomness, an inherent characteristic from the original GA in converging to a final result.

A closer look at these graphs also reveals thought-provoking facts on execution time graphs. The difference between execution time obtained at the beginning and end of algorithms is minimal for both attacks. This is caused due to algorithms having access to an unlimited number of VMs allowing algorithms to take advantage of parallelism. This usually results in high monetary costs, for this reason the main challenge of algorithms is to allocate tasks to a reduced number of VMs while maintaining a low execution time.

8. Conclusions and future work

In this paper, we presented the GA-ETI, a scheduler for scientific applications for cloud systems to concurrently optimize their execution makespan and monetary cost. GA-ETI enhanced the original GA through purposeful/tailor-made modification to its crossover and mutation operators. GA-ETI uses enhanced crossover to combine clusters of genes rather than randomly divided chromosomes; it also employs increment/decrement mutations to add/remove

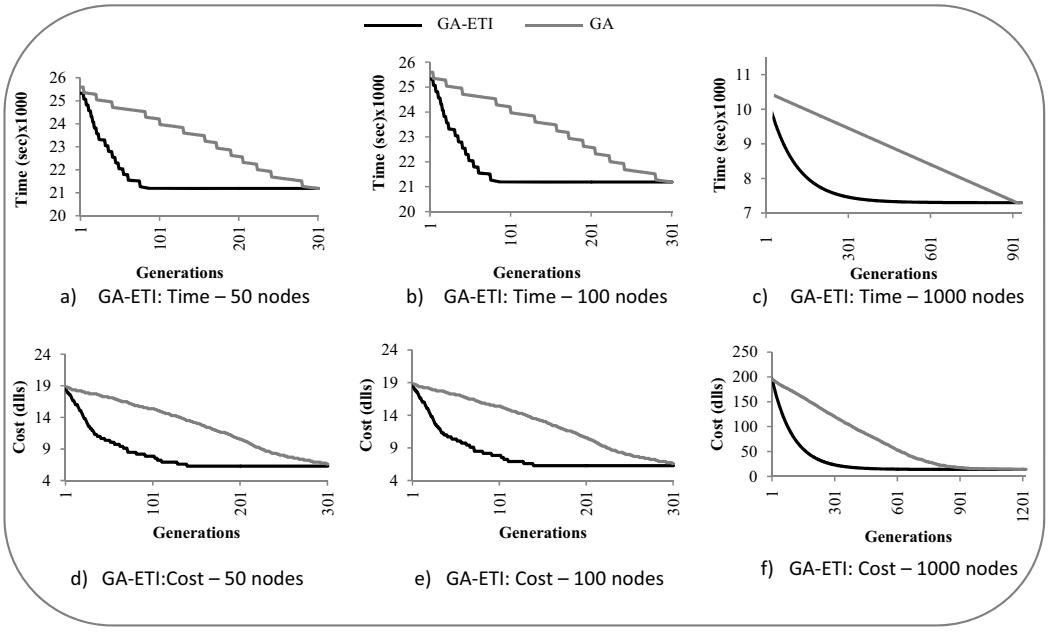

Fig. 12. Analysis of the number of generations for the GA-ETI and the general GA applied to the Epigenomics workflow with 50, 100 and 1000 nodes.

virtual machines from a given chromosome. Both modifications yield reduced inherent randomness compared to the original GA. Using five workflows to represent a variety of current scientific problems, GA-ETI was tested and proved its superiority against three (HEFT, Provenance and FSV) well-known/up-to-date schedulers in this field. GA-ETI solutions had lower makespan and monetary cost when compared with solutions provided by HEFT. Unlike FSV, GA-ETI produces a complete scheduling configuration prior to execution with better qualities. In contrast to Provenance, GA-ETI produces its own scheduling configuration and uses a workflow manager system only as a middleware to execute scheduling decisions. GA-ETI also revealed that, despite the general impression, optimal execution of workflows does not require a high number of resources (compared to the number of parallel nodes) in most cases. To continue this work, we aim to develop/incorporate cloud pricing models to consider fluctuation of the hiring cost of VMs during scheduling. We also aim to focus on performance oscillation in cloud environments and its impact on execution of applications.

Acknowledgments

The authors would like to thank the Commonwealth Scientific and Industrial Research Organization (CSIRO) and Consejo Nacional de Ciencia Tecnología (CONACYT) for supporting this work. Professor Zomaya's work is supported by the Australian Research Council Discovery Grant number DP130104591.

References

[1] A. Iosup, S. Ostermann, M.N. Yigitbasi, R. Prodan, T. Fahringer, D.H. Epema, Performance analysis of cloud computing services for many-tasks scientific computing, IEEE Trans. 22 (2011) 931–945.
[2] Illumina. Available: https://www.illumina.com/.
[3] (2013, July) IEEE SPECTRUM.
[4] B. Martini, K.-K.R. Choo, Cloud storage forensics: ownCloud as a case study, Digital Invest. 10 (2013) 287–299.
[5] D. Quick, K.-K.R. Choo, Big forensic data reduction: digital forensic images and electronic evidence, Cluster Comput. (2016) 1–18.
[6] The Large Hadron Collider. Available: http://home.web.cern.ch/topics/large-hadron-collider.
[7] The Large Synoptic Survey Telescope. Available: http://www.lsst.org/.
[8] J. Zhang, K. Hwang, C. Wu, Ordinal optimized scheduling of scientific workflows in elastic compute clouds, Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on (2011) 9–17.
[9] L. Deng, Q. Yu, J. Peng, Adaptive scheduling strategies for cloud-based resource infrastructures, Secur. Commun. Networks vol. 5 (2012) 1102–1111.
[10] H. Kloh, B. Schulze, R. Pinto, A. Mury, A bi-criteria scheduling process with CoS support on grids and clouds, Concurrency Comput. 24 (2012) 1443–1460.
[11] I.A. Moschakis, H.D. Karatza, Evaluation of gang scheduling performance and cost in a cloud computing system, J. Supercomput. 59 (2012) 975–992.
[12] H. Tsakalozos, E. Kllapi, M. Sitaridi, D. Paparas, A. Delis, Flexible use of cloud resources through profit maximization and price discrimination, Data Engineering (ICDE), 2011 IEEE 27th International Conference on (2011) 75–86.
[13] F. Zhang, J. Cao, K. Li, S.U. Khan, K. Hwang, Multi-objective scheduling of many tasks in cloud platforms, Future Gener. Comput. Syst. 37 (2014) 309–320.
[14] P. Achar, D. Shwetha, H. Pooja, Optimal scheduling of computational task in cloud using Virtual Machine Tree, Emerging Applications of Information Technology (EAIT), 2012 Third International Conference on (2012) 143–146.
[15] H. Topcuoglu, S. Hariri, M.-y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. 13 (2002) 260–274.
[16] D. de Oliveira, K.A. Ocaña, F. Baião, M. Mattoso, A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds, J. Grid Comput. 10 (2012) 521–552.
[17] A. EC2. Amazon EC2. Available: aws.amazon.com/ec2.
[18] C. Anglano, M. Canonico, Scheduling algorithms for multiple bag-of-task applications on desktop grids: a knowledge-free approach, Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on (2008) 1–8.
[19] A. Sulistio, R. Buyya, A time optimization algorithm for scheduling bag-of-task applications in auction-based proportional share systems, SBAC-P2005 CE, 17th International Symposium on Computer Architecture and High Performance Computing (2005) 235–242.
[20] S. Wieczorek, R. Prodan, T. Fahringer, Bi-criteria scheduling of scientific workflows for the grid, Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on (2008) 9–16.
[21] M. LAVC, Strategies for dynamic workflow scheduling on grids PhD. PhD, COPPE/UFRJ, COPPE/UFRJ, 2007.
[22] L. Ramakrishnan, D.A. Reed, Performability modeling for scheduling and fault tolerance strategies for scientific workflows, Proceedings of the 17th International Symposium on High Performance Distributed Computing (2008) 23–34.
[23] J. Yu, R. Buyya, C.K. Tham, Cost-based scheduling of scientific workflow applications on utility grids, e-Science and Grid Computing, 2005. First International Conference on (2005) (8 pp.-147).
[24] L. Xu, H. Wang, Y. Bi, A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing, Parallel and Distributed Processing with Applications, 2009 IEEE International Symposium on (2009) 629–634.
[25] M. Bandini, A.R. Mury, B. Schulze, R. Salles, A grid QoS decision support system using service level agreements, in: Congresso Da Sociedade Brasileira De Computacao De 2009, CSBC 09. Sociedade Brasileira De Computacao, Porto Alegre, RS, Brazil, 2009.
[26] H.-C. Lin, C. Raghavendra, An approximate analysis of the join the shortest queue (JSQ) policy, IEEE Trans. 7 (1996) 301–307.
[27] J. Dean, S. Ghemawat, MapReduce: a flexible data processing tool, Commun. ACM 53 (2010) 72–77.
[28] A. Bharathi, E. Chervenak, G. Deelman, M.-H. Su, K. Vahi, Characterization of scientific workflows, Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS 2008) (2008) 1–10.
[29] I. Casas, J. Taheri, R. Ranjan, L. Wang, A.Y. Zomaya, A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems, Future Generation Computer Systems (2016).
[30] Z. T. Wang, Y. Liu, Y. Xu Chen, Load balancing task scheduling based on genetic algorithm in cloud computing, Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on (2014) 146–152.
[31] C.-L. Chen, V.S. Vempati, N. Aljaber, An application of genetic algorithms for flow shop problems, Eur. J. Oper. Res. 80 (1995) 389–396.
[32] K.-S. Tang, K.-F. Man, S. Kwong, Q. He, Genetic algorithms and their applications, Signal Process. Mag. IEEE 13 (1996) 22–37.
[33] H. K. Zhu, L. Song, J. Gao Liu, Hybrid genetic algorithm for cloud computing applications, Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific (2011) 182–187.
[34] HTCondor, High Throughput Computing, 2016 (Available: http://research.cs.wisc.edu/htcondor/.).

Israel Casas is a researcher on The Information Technology School at The University of Sydney, Australia. He is a member of the Centre for Distributed and High Performance Computing at mentioned School. His main research interests include Cloud Computing, Parallel Computing, Optimization Techniques, Embedded Systems and Microcontrollers. Casas started his research experience at the Electronic and Computer Department at Monterrey Institute of Technology and Higher Education. Mexico. He has also contributed with the University of California (Irvine) for the exploration and evaluation of embedded systems with software focus.

Javid Taheri received his Bachelor and Masters of Electrical Engineering from Sharif University of Technology, Tehran, Iran in 1998 and 2000, respectively. His Master was in the field of Intelligent Control and Robotics. His Ph.D. is in the field of Mobile Computing from the School of Information Technologies in the University of Sydney, Sydney, Australia. He is currently working as a Postdoctoral research fellow at same school. His main areas of research are Optimization Techniques, Artificial Intelligence, Vehicular Ad-hoc Networks, Scheduling, and Parallel Computing.

Rajiv Ranjan is a Scientist in the CSIRO ICT Center, Information Engineering Laboratory, Australian National University, Canberra, where he is working on projects related to cloud and service computing. Previously, he was a Senior Research Associate (Lecturer level B) in the School of Computer Science and Engineering, University of New South Wales (UNSW). Dr. Ranjan has a Ph.D. (2009) in Computer Science and Software Engineering from the University of Melbourne. He completed Bachelor of Computer Engineering from North Gujarat University, India, in 2002. Dr. Ranjan is broadly interested in the emerging areas of cloud, grid, and service computing. The main goal of his current research is to advance the fundamental understanding and state of the art of provisioning and delivery of application services in large, heterogeneous, uncertain, and evolving distributed systems. Dr. Ranjan has more than 50 research publications in journals with high impact factor (according to JCR published by ISI), in proceedings of IEEE's/ACM's premier conferences and in books published by leading. Dr. Ranjan has often been invited to served as Guest Editor for leading distributed systems and software engineering journals including Future Generation Computer Systems (Elsevier Press), Concurrency and

Computation: Practice and Experience (John Wiley & Sons), and Software: Practice and Experience (Wiley InterScience). He was the Program Chair for 2010 and 2011 Australasian Symposium on Parallel and Distributed Computing and 2010 IEEE TCSC Doctoral Symposium. He serves as the editor of IEEE TCSC Newsletter. He has also recently initiated (as chair) IEEE TCSC Technical area on Cloud Computing

Lizhe Wang is a Professor at Institute of Remote Sensing & Digital Earth, Chinese Academy of Sciences (CAS) and a ChuTian Chair Professor at School of Computer Science, China University of Geosciences (CUG). Prof. Wang received his B.E. & M.E from Tsinghua University and Doctor of Engineering from University Karlsruhe (Magna Cum Laude), Germany. Prof. Wang is a Fellow of IET, Fellow of British Computer Society. Dr. Wang serves as Associate Editor of IEEE Transaction on Computers and IEEE Transaction on Cloud Computing. His main research interests include high performance computing, e-Science, and spatial data processing.

Albert Y. Zomaya is the Chair Professor of High Performance Computing & Networking and Australian Research Council Professorial Fellow in the School of Information Technologies, Sydney University. He is also the Director of the Centre for Distributed and High Performance Computing which was established in late 2009. Dr. Zomaya published more than 500 scientific papers and articles and is author, co-author or editor of more than 20 books. He is currently the Editor in Chief of the IEEE Transactions on Computers and Springer's Scalable Computing and serves as an associate editor for 22 leading journals. Dr. Zomaya is the Founding Editor of the Wiley Book Series on Parallel and Distributed Computing. Dr. Zomaya was the Chair the IEEE Technical Committee on Parallel Processing (1999–2003) and currently serves on its executive committee. He is the Vice–Chair, IEEE Task Force on Computational Intelligence for Cloud Computing and serves on the advisory board of the IEEE Technical Committee on Scalable Computing and the steering committee of the IEEE Technical Area in Green Computing. Dr. Zomaya has delivered more than 130 keynote addresses, invited seminars, and media briefings and has been actively involved, in a variety of capacities, in the organization of more than 600 conferences.