Soft Comput (2019) 23:1745–1765
https://doi.org/10.1007/s00500-017-2897-8

METHODOLOGIES AND APPLICATION

Smart PSO-based secured scheduling approaches for scientific workflows in cloud computing

J. Angela Jennifa Sujana[1] · T. Revathi[1] · T. S. Siva Priya[1] · K. Muneeswaran[1]

Published online: 8 November 2017

Abstract Owing to its manifold advantages in adapting cloud computing for real-world scientific workflow applications, we intend to use cloud computing for executing the scientific workflows. In the present work, we aim for scheduling the workflow in the scalable resources in the cloud. In general, security is a vital challenge in cloud and so we include security constraints into our optimization model. The main objective of our work is to find an optimized schedule having minimum makespan and cost and by satisfying security demand constraint. The users can submit their security demand to the cloud provider during negotiation. The workflow is initially scheduled with list-based heuristics, which is then optimized by Particle Swarm Optimization (PSO). Thus we device a Smart Particle Swarm Optimization (SPSO)-based secured scheduling to find the optimized schedule with minimum makespan and cost. The proposed method is capable of assigning the task in the scientific workflows to the best suitable virtual machine in the cloud. Hence, the resource allocation is addressed as well by our method. Besides, a variant of PSO algorithm called Variable Neighbourhood PSO is also experimented to overcome the local optima problem. Our experimental results show that the scheduled workflows with assured security are yielding better makespan than existing methods with minimum iterations, which is well suited for cloud environment.

Keywords Cloud computing · Secured scheduling · Scientific workflows · Particle Swarm Optimization

Communicated by V. Loia.

✉ J. Angela Jennifa Sujana
ang_jenefa@mepcoeng.ac.in

1 Mepco Schlenk Engineering College, Sivakasi, Tamilnadu 626 005, India

1 Introduction

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Mell and Grance 2011). This is the definition given by National Institute of Standards and Technology (NIST) for cloud computing. Cloud computing provides a variety of services; among these, Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) are the common one. All these services are accessed via Internet. The availability of Internet facilities everywhere makes cloud services reachable to everyone. Thus using cloud for scientific applications have also no exception. Automated service provisioning is the major research challenge in cloud computing as pointed out by Zhang et al. (2010). In the survey conducted by International Data Corporation (IDC), security is ranked as the vital challenge attributing to cloud computing. The security concern makes organizations to be hesitant in offloading their business workloads into cloud (Gens 2008). The NIST Cloud Computing Standards Roadmap Working Group has also indicated that security integration of a cloud system into existing enterprise security infrastructure is a must for majority of government systems with moderate and greater impact (Pritzker 2013).

Workflow application is the emerging paradigm for distributed computing infrastructures, which are now being used in areas like astronomy, bioinformatics, material sciences and physics. Workflow has a wide range of applications such as scientific, enterprise applications and real-world problems. Workflow concepts are well suited to fit for scenarios where several distributed entities work collaboratively together to achieve a common goal. Adopting the cloud computing for executing the workflow has many advantages. Firstly, it reduces the burden of the user from having a self-owned infrastructure. Secondly, it is cheap in terms of cost. Thirdly, it provides access anywhere. When adopting cloud computing for executing workflows, one has to focus on objectives such as cost to be paid for using the cloud services (Tan et al. 2014; Wu et al. 2010; Pandey et al. 2010; Xue and Wu 2012), time for completing the workflow (Wang et al. 2012; Bilogrevic et al. 2011; Tan et al. 2014; Zuo et al. 2014; Selvi and Govindarajan 2014; Rodriguez and Buyya 2014; Wu et al. 2010; Xue and Wu 2012), resources allocated for the tasks in the workflow (Selvi and Govindarajan 2014; Rodriguez and Buyya 2014; Pandey et al. 2010) and security demand of the user (Wang et al. 2012; Bilogrevic et al. 2011; Tan et al. 2014; Zeng et al. 2015). Earlier research works on other computing paradigm such as grid computing, high-performance computing, heterogeneous distributed systems and cluster computing also focus on these objectives. The literatures Topcuoglu et al. (2002); Tang et al. (2011); Li et al. (2015) focus on cost minimization, Topcuoglu et al. (2002); Xie and Qin (2006, 2008); Tang et al. (2010, 2011); Song et al. (2006); Liu et al. (2012); Abraham et al. (2006); Li et al. (2015); Sih and Lee (1993) focus on time minimization, Liu et al. (2012); Tang et al. (2010); Li et al. (2015) focus on resource allocation and Xie and Qin (2006, 2008); Tang et al. (2011); Song et al. (2006); Liu et al. (2012) focus on security constraints. The proposed work focuses on all these objectives in order to provide an efficient and unique solution for automated service provisioning in cloud for scientific workflows.

Workflow scheduling (Juve et al. 2013) is of course has to deal with some scheduling heuristics to generate a best schedule for all the tasks in the workflow. The correct execution sequence of workflow activities should be found for scheduling the workflow. A correct execution sequence should follow the constraints of the workflow model that may be of temporal constraints or causality constraints.

The scheduling strategies for workflow graphs are classified into two main categories, namely heuristic approach and metaheuristic approach (Topcuoglu et al. 2002). In the present work, we focus on using a metaheuristic search algorithm, Particle Swarm Optimization (PSO). Owing to the significant results of using PSO algorithm for workflow task scheduling in earlier works by Zuo et al. (2014), Rodriguez and Buyya (2014), Liu et al. (2012), Pandey et al. (2010), Xue and Wu (2012), Beegom and Rajasree (2014), we employ PSO in the present work for optimizing the schedule. We do modify the basic PSO algorithm by making a smart initialization of the population in the swarm. First we generate a schedule for the workflow, which has been submitted by the cloud user. We do consider the security requirement of the cloud user. This will help the cloud user to trust the cloud provider and confidently execute his/her workflow tasks in the cloud. The security risk has been mainly caused by the multitenancy nature of the cloud. There is a possibility of vulnerable neighbours or some disturbing (noisy) neighbours occupying the virtual machines in the same zone. They may try to perform some unethical hacking mechanisms or some type of attacks such as snooping, modification and spoofing. Snooping, modification and spoofing can be overcome by adopting security mechanisms such as confidentiality, integrity and authentication. Thus in this paper, we have integrated the security measures for satisfying the user QoS with the scheduling of the workflow applications. When we consider security services, it involves more overhead. This may be a bottleneck for the performance of the application. Normally before getting the IaaS the user will be signing the service-level agreement (SLA) with the cloud provider. Our model gives the user the right to specify their security demand as part of their SLA.

Similarly, the resources at the cloud provider side will be maintained with a security guaranteed level (SGL). Both the security demand as well as the security rank is configured based on the three security factors integrity, confidentiality and authentication. We design our SPSO optimization model by integrating the scheduling constrains and the security constraints. The scheduling constraints include the makespan of the schedule and the cost the user has to pay to the cloud provider. Security constraints consider the security demand of the cloud user. The SPSO model makes the optimistic decision in selecting the best schedule as well as the best virtual machine (VM) for executing the task.
The main contributions of our proposed work are as follows.

- We design a unique solution for automated IaaS provisioning with minimal decision time in cloud for scientific workflows.
- We propose Smart Particle Swarm Optimization (SPSO) which finds an optimized schedule that trade off between security and minimum makespan and cost for workflow execution in cloud.
- We consider the dynamic provisioning and heterogeneity of the virtual machine (VM) resource pool at the cloud provider side. The SPSO algorithm is used to solve not only the problem of producing a schedule defining the task to virtual machine mapping but also the number and type of VMs that are to be leased from the cloud provider.
- Security demand of the user is also merged and modelled into the optimization problem.
- A coding strategy is designed for encoding the particle to solve the multiobjective problem.

The remainder of this paper is organized as follows. Section 2 presents the related work in a nutshell. Section 3 describes the secured schedule architecture. Then the problem outline is briefed in Sect. 4 with the multiobjective focused in this paper. In Sect. 5, we explore the schedule primitives. This is followed by the schedule generation method in Sect. 6. Section 7 deals with the Smart Particle Swarm Optimization (SPSO) algorithm and Smart Variable Neighbourhood Particle Swarm Optimization (SVNPSO) algorithm. The experiments and performance analysis are enumerated in Sect. 8. Finally, we conclude this paper in Sect. 9.

2 Related Work

Scheduling precedence-constrained task such as workflows is NP-hard problem and it is studied by many researchers. The taxonomy of the workflow scheduling algorithms is list scheduling heuristics (Topcuoglu et al. 2002; Xie and Qin 2006, 2008; Tang et al. 2011, 2010; Li et al. 2015; Sih and Lee 1993; Canon et al. 2008), clustering heuristics (Topcuoglu et al. 2002), task duplication heuristics (Sujana et al. 2015; Tang et al. 2010) and guided random search such as GA (Song et al. 2006) and PSO (Zuo et al. 2014; Selvi and Govindarajan 2014; Rodriguez and Buyya 2014; Liu et al. 2012; Liu and Abraham 2007; Kalra and Singh 2015).

Sih and Lee (1993) focused on developing dynamic level scheduling. The priorities are changed dynamically to find the task processor pair. It is the first algorithms that computed an estimate of the availability of each computing resource and allowed a task to be scheduled to a processor in use. Topcuoglu et al. (2002) have designed the upward rank and downward rank. This has constituted for the Heterogeneous Earliest Finish Time (HEFT) and the Critical Path on a Processor (CPOP) algorithm. The task rank order is calculated using the communication cost in the task graph, i.e. workflow, in the work done by Tang et al. (2010) and they were able to achieve better results. Li et al. (2015) have used stochastic model for finding the task stochastic bottom level, which refers to the priority rank order of the task in the workflow. Canon and his team has compared 20 scheduling heuristics that can be applied to workflows in the form of directed acyclic graph and concluded that on average, for random graphs, HEFT is the best one in terms of robustness and schedule length (Canon et al. 2008). Arabnejad and Barbosa (2014) have proposed Predict Earliest Finish Time (PEFT)-based scheduling for heterogeneous computing systems. In this, they are introducing a look-ahead feature to the earlier HEFT without increasing the time complexity.

On the other hand, metaheuristic algorithms can give better optimized schedules. Particle Swarm Optimization (PSO) is a bioinspired algorithm inherited from birds flocking and fish schooling. It is one of the latest optimization algorithms that outperform genetic algorithm with minimal iterations. Zuo et al. (2014) have used PSO for scheduling deadline-constrained tasks in hybrid clouds. They have proposed self-adaptive PSO for effective utilization of inter cloud resources. They have focused on deadline alone with hybrid clouds. Chakraborty and his team (2016) have used PSO along with fuzzy rule constraints to solve multiobjective problem. CLOUDRB proposed by Selvi et al. Selvi and Govindarajan (2014) used PSO for job scheduling and resource allocation for high-performance computing applications. They have focused on scheduling with better resource allocation objective, which will minimize the makespan. Rodriguez and Buyya (2014) have proposed resource provisioning and scheduling strategy for scientific workflows on Infrastructure as a Service (IaaS) clouds. They present an algorithm based on PSO, which aims to minimize the overall workflow execution cost while meeting deadline constraints. Wu et al. (2010) have devised Revised Discrete Particle Swarm Optimization (RDPSO) to schedule applications among cloud services, by considering data transmission cost and computation cost. The RDPSO algorithm gives better performance on makespan and cost optimization compared to the classic PSO algorithm. Pandey et al. (2010) proposed a PSO-based algorithm to minimize the execution cost of a single workflow with load balancing on the available resources. They did not consider security constraints, but their work gave good results in terms of cost saving alone. Xue and Wu (2012) have devised Hybrid Particle Swarm Optimization (GHPSO) algorithm for scheduling workflows in cloud computing with cost minimization objective. They have embedded the crossover and mutation of genetic algorithm into the PSO, which has increased the time complexity of the algorithm. But we like to focus on time minimization also for quick decision-making. Beegom et al. Beegom and Rajasree (2014) have designed integer PSO, to solve the bi-objective task scheduling problem in cloud. They have considered the Pareto optimization with the user satisfaction in terms of QoS and cloud providers profit. Their work focuses only on independent tasks; hence, it cannot be applied for workflow scheduling. Liu et al. (2015) have also considered Pareto optimization-based multiobjectives, maximizing system reliability and minimizing schedule length in scheduling a distributed computing system. The schedule optimization is based on the Tabu search algorithm, which lacks in global optimum search compared to PSO.

However, many scheduling algorithm ignores security constraints, which is the most vital one in cloud environment. Considering the security-driven scheduling research works, Xie and Qin (2006; 2008) have focused on integrating the security module and the scheduling module. They have proposed security-aware real-time heuristic strategy for clusters, which integrates security requirements into the scheduling of real-time applications in clusters. They tried to prove that the overhead incurred in adding the security model does not result in performance bottleneck. They have also tested the security level provided by various algorithms. They have focused on the security parameters integrity, confidentiality and authentication.

Tang et al. (2011) have designed security-driven scheduling model for heterogeneous distributed systems. Their scheduling algorithm makes use of the Heterogeneous Earliest Finish Time (HEFT)-based approach for precedence-constrained tasks. The trust manager computes the security rank and it is used for finding the priority of the tasks in the heuristics. Wang and his team have devised a trust model based on Bayesian method. Then the trust value is integrated with the dynamic level scheduling (Wang et al. 2012). Their algorithm cloud DLS is applied to cloud environment and they were the first to consider security constraints for task scheduling in cloud. But they used only a soft security mechanism, i.e. the trust model. In the work done by Tan et al. (2014), a trust-based workflow scheduling algorithm was proposed. Their trust model mainly uses the fuzzy membership function approach for finding the trusted service. The best trustworthy service will be selected for each task in the workflow. This is applicable only for SaaS or PaaS type of services in cloud.

Bilogrevic et al. have introduced Privacy preserving scheduling for mobile devices using cloud computing (Bilogrevic et al. 2011). Song and his associates (2006) have developed Security-Assured Grid Job Scheduling. Their job scheduling method uses genetic algorithm, which is a metaheuristic search algorithm. The risk factor in terms of security for the jobs is measured. Their work focused on batch scheduling, which deals with independent tasks. Hence, it cannot be applied to workflow scheduling, where precedence constraint tasks exist. Zeng et al. (2015) have devised security-aware and budget-aware (SABA) workflow scheduling strategy in clouds. Their work uses security overhead and monetary cost constraints while minimizing time. The scheduling heuristic is towards the concept of immovable dataset. Our paper is towards optimized scheduling using metaheuristic approach and in workflows data gets transferred from parent task to child task. Liu and his team (2012) have applied the PSO for optimizing the schedule for workflow applications in distributed environment. They have considered minimizing the makespan with security constraints in their scheduling approach. However, the security constrain is considered as a single parameter, which does not take any security mechanism into account.

Pondering on the security objective, security has been incorporated either as trust model (Tang et al. 2011; Wang et al. 2012; Tan et al. 2014) or with security mechanism (Xie and Qin 2006, 2008; Song et al. 2006; Liu et al. 2012; Zeng et al. 2015). Among these, trust-based security mechanism is like soft security model. It cannot provide a more realistic secured environment for cloud. Hence, we choose to use security mechanisms such as integrity, confidentiality and authentication for the virtual machines. Since we propose to develop automated IaaS provisioning with minimal decision time for scientific workflows, we go for merging list scheduling heuristics into PSO algorithm. This has resulted in the Smart Particle Swarm Optimization (SPSO) algorithm and Smart Variable Neighbourhood Particle Swarm Optimization (SVNPSO) algorithm. These algorithms aim to reduce the number of iterations of the PSO algorithm considerably, to support quick decision time for automated IaaS for scientific workflows. The proposed SPSO algorithm is smart since it not only reduces the number of iterations, but also considers the dynamic provisioning and heterogeneity of the VM resource pool at the cloud provider side. In addition to that, a particle coding strategy is designed to solve the multiobjective problem.

3 Secured scheduling architecture

Any complex applications with precedence constraint can be modelled as a workflow. Scientific workflows are logically connected dependent tasks. A set of rules defines the linkage between dependent tasks. In such a relationship, dependent (child) task gets executed only after its parent task gets executed. Tasks without a parent are termed as entry tasks and those do not have a child are called as exit tasks. Figure 1 shows a sample workflow with ten different tasks. The entry task is T_1 and the exit task is T_{10}. The task T_2 and T_3 are the child tasks of the parent T_1. The child tasks can be executed only after the execution of their parent task. The best optimized task–VM pair (task, VM) is to be found.

The system architecture for secured scheduling is shown in Fig. 2. The cloud consumers who want to use the cloud

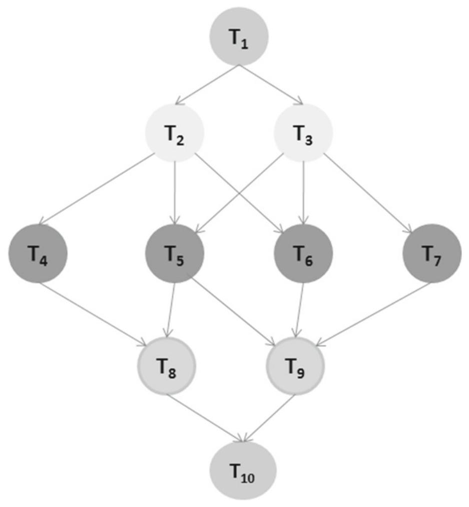

Fig. 1 Sample workflow graph

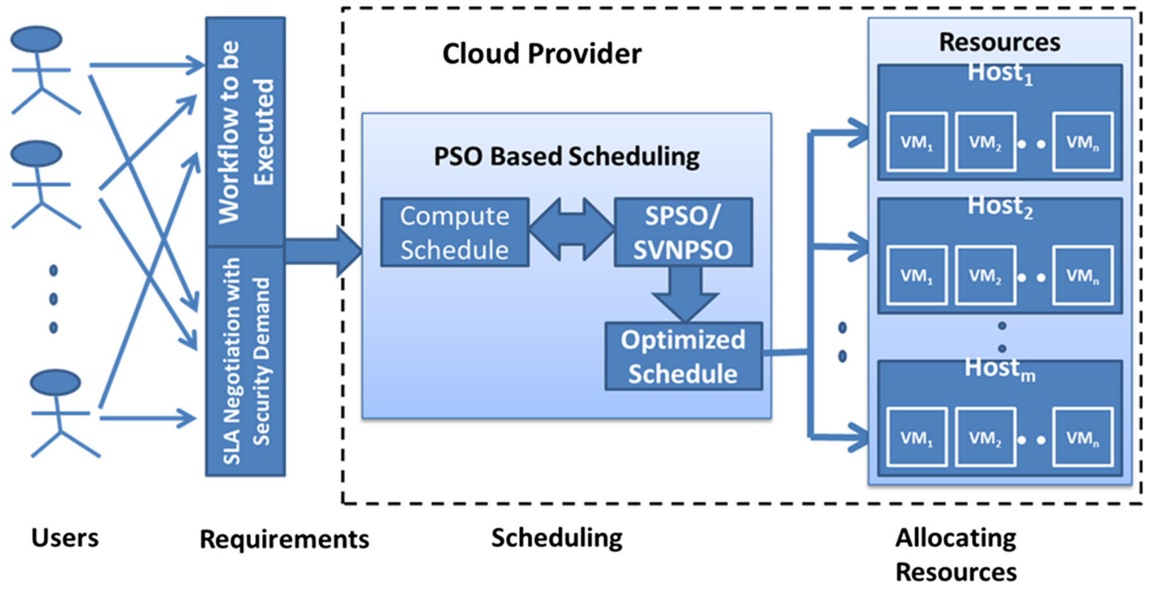

Fig. 2 System architecture for secured scheduling

for executing their workflow applications will submit their request to the cloud provider. Initially before performing the scheduling and resource allocation process, the cloud consumer will negotiate with the provider for the user's quality of service (QoS) in terms of service-level agreement (SLA). We consider the following parameters in the SLA, namely SGL, performance, availability, robustness, backup and service initialization time. Among them, SGL is the main parameter, which we focus in our work. As a part of the SLA negotiation, the user will provide their Security Demand (SD) to the provider. Similarly, the provider will provide the different SGLs available for different services along with their rates, i.e. the amount the user has to pay for their usage. So in this negotiation the security demand by the customer will be finalized. Now the user will submit their workflow along with the SD signed in the SLA. Now the cloud provider has to find an optimized schedule for the submitted workflow and also the best virtual machine (VM) that will optimize the makespan of the algorithm.

The tasks in the workflow are precedence constrained. Thus the child task in the workflow can be executed only after the execution of its parent. To make all the tasks in the workflow to execute in minimal makespan, we must generate a good schedule. Now, this schedule must incorporate the precedence constraint of the tasks. We consider the execution time and earliest start time to find the schedule order. Now we incorporate the swarm intelligence to find the optimized schedule. The objective function of the swarm optimization is to minimize the makespan and the cost subjected to security demand being satisfied. The overall execution time of the generated schedule gives the makespan. The total execution cost is decided by the required makespan of the workflow. If the makespan is to be reduced, the cost is increased due to the increased high end VM requirements. We consider the solution only when the security demand of the workflow is satisfied. Thus after some minimal number of iterations, the algorithm will find the best optimized schedule. We summarize the various notations used in this paper in Table 1.

4 Problem formation and outline

The cloud providers offer the IaaS with different types of VMs. Each VM type has the specific VM configuration as given in Table 3. The VMs are grouped into three categories, namely *green, yellow and red zones*. Each zone is supposed to have their own security levels. The SGL is used to represent the percentage of security that can be provided for the VMs at the cloud provider side. The three types of zones support different range of SGL. We fix the SGL range for the green zone as [0.9,1]. Similarly, the SGL range for the yellow zone is [0.70,0.89] and for the red zone is [0.4,0.69]. The SGL will address the three attributes of security, namely authentication, confidentiality, integrity. Hence, apart from other specification parameters the SGL of a vm_j is described as given in Eq. (1).

$$\mathrm{SGL}_{\mathrm{VM}_j} = \mathrm{Avg}\,\{\mathrm{SGL}_\mathrm{a}, \mathrm{SGL}_\mathrm{c}, \mathrm{SGL}_\mathrm{i}\} \tag{1}$$

The values SGL_a, SGL_c, SGL_i denote the Security Guaranteed Level values that a particular VM will assure in terms of authentication, confidentiality and integrity. Hence, the SGL can be expressed as a percentage value, which lies in the interval [0–1]. The different security guaranteed level

Table 1 Definitions of notations

Notation	Definition
t_i	The ith task t_i in the workflow W =(T,E)
$\mathrm{vm_j}$	The *jth virtual machine* available for allocation from the pool of resources
$\mathrm{SGL_a}$	Security Guaranteed Level for authentication
$\mathrm{SGL_c}$	Security Guaranteed Level for confidentiality
$\mathrm{SGL_i}$	Security Guaranteed Level for integrity
$\mathrm{SGL_{VM_j}}$	Security Guaranteed Level of $\mathrm{jthvm_j}$
wt_a	Weight or percentage of the security demand for authentication factor
wt_c	Weight or percentage of the security demand for confidentiality factor
wt_i	Weight or percentage of the security demand for integrity factor
SD_{t_i}	Security demand made by the cloud user for the ith task t_i in the workflow W.
OET	Overall execution time (OET)
OEC	Overall execution cost (OEC)
ω	Weight factor
$\mathrm{t_{entry}}$	Entry task
$\mathrm{t_{exit}}$	Exit task
$\mathrm{child(t_j)}$	The set of child tasks of a task $\mathrm{t_i}$
$\mathrm{parent}(t_i)$	The set of parent tasks of the task $\mathrm{t_i}$
W	Workflow application
$w(\mathrm{vm_j})$	Computation capacity of a VM
$w(t_i)$	Computation cost of a task
$\mathrm{ET(t_i}, vm_\mathrm{j})$	Execution time of the task $\mathrm{t_i}$ on the jth VM
$\overline{\mathrm{ET}(\mathrm{t}_i)}$	Average execution time of the task $\mathrm{t_i}$
$\mathrm{CC(t_i, t_j)}$	Communication cost between a parent and child task along an edge $e(i, j)$
$\overline{\mathrm{CC}}\mathrm{(t_i, t_j)}$	Average communication cost of an edge $e(i, j)$
$\mathrm{late(vm_j)}$	Latency of a VM
bw_k	The bandwidth of the link between the VMs
$\mathrm{EST(t_i, vm_j)}$	Earliest start time (EST) is the earliest possible start time of a task on a VM.
$\mathrm{ReadyTime(t_i, vm_j)}$	Time in which the jth VM is ready for executing the task $\mathrm{t_i}$.
$\mathrm{EET}(t_i, \mathrm{vm}_j)$	Earliest end time (EET) of a task is the earliest possible completion time of a task on a VM.
OET	Overall execution time (OET) is the total schedule length
OEC	Overall execution cost (OEC) is the total cost incurred in utilizing the cloud resource and the user has to pay to the provider
$\mathrm{ReleaseTime}(\mathrm{vm}_j)$	Time at which the VM has finished the execution of the task in the workflow and is released to the provider
$\mathrm{PRank}(t_i)$	Priority rank of the task
$\lvert\mathrm{VM_{type}}\rvert$	Number of instance type of virtual machines
$\lvert\mathrm{W_{Level}}\rvert$	Number of levels in the DAG representation of the workflow
$\vec{p_i}(t)$	Particles' position
$\vec{v_i}(t)$	Velocity of the particle
$\propto$	Inertia weight
r_1, r_2	Random numbers
$c_1,\ c_2$	Acceleration constants
$\overrightarrow{p_{\mathrm{pbest}}}(t)$	Particle's best position during the swarm
$\overrightarrow{p_{\mathrm{Gbest}}}(t)$	Global best position of all the particles in the swarm, i.e. population

Table 1 continued

Notation	Definition
v_{update}	Velocity update used in SVNPSO, when the swarm movement is not giving better results
v_{th}	Threshold velocity
$P_{\text{Security}}\left(t_i,\ \text{vm}_j\right)$	Probability of security in scheduling a task t_i to a VM vm_j
$P_{\text{security}}(T)$	Probability of security for the entire task set T

Table 2 Cryptographic algorithms for confidentiality and its Security Guaranteed Level (SGL) (Xie and Qin 2006)

Cryptographic algorithms	Security Guaranteed Level	Computation time (ms)
SEAL	0.08	168.75
RC4	0.14	96.43
Blowfish	0.36	37.5
Knufu/Khafre	0.40	33.75
RC5	0.46	29.35
Rijndael	0.64	21.09
DES	0.90	15
IDEA	1.00	13.5

values will be attained by implementing the corresponding algorithms given in Table 2 for confidentiality. Similarly, the type of security mechanism that was provided for each type of security characteristics is evaluated in the work by Xie and Qin (2006; 2008). Table 2 lists the methods that can be adopted to secure the VMs in terms of confidentiality.

Figure 3 represents the flow of the proposed work for processing the given scientific workflow. The workflow parser will parse through the different tasks in the given workflow and initiate the workflow engine. Now an optimized scheduling with suitable virtual machine selection has to be done along with security constrain checking. Thus the heuristic approach first selects the security zone followed by the schedule generation and resource mapping. These activities are done by the first algorithm genSchedule. This generated schedule is then optimized by the Particle Swarm Optimization algorithms, namely Smart Particle Swarm Optimization (SPSO) and Smart Variable Neighbourhood Particle Swarm Optimization

On the other hand, we use another parameter used from the user perspective, i.e. Security Demand (SD). Similar to the SGL the Security Demand can also be represented as a percentage value normally in the interval [0,1]. The SD is obtained by the provider as part of the SLA that is negotiated between the provider and the user. For a high value of SD such as 95%, requested by the user the provider has to assign a most secured VM and it is to

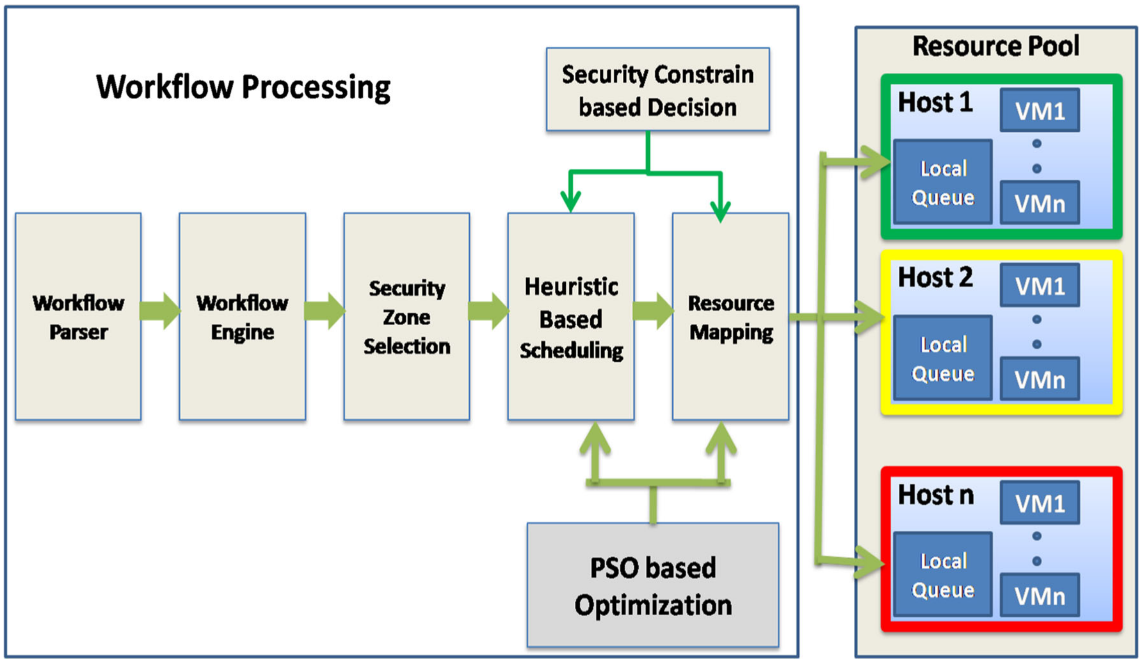

Fig. 3 Schematic representation of workflow processing with security constraints and resource allocation (color figure online)

be from the green zone. Since more sophisticated hardware and algorithms has to be used for such request the cost charges for those VMs will be high compared to the VMs procurement from yellow and red zones. The SD is also obtained as a three-valued parameter each representing the weight values for authentication, confidentiality and integrity. The system provides flexibility in specifying the SD by the user. If the user is not bothered about the security with different parameters, then they can specify it as a single-valued one. In that case, the same weight value will be considered for all the three parameters.

$$\mathrm{SD}_{t_i} = \mathrm{Avg}\{\mathrm{wt_a}, \mathrm{wt_c}, \mathrm{wt_i}\} \tag{2}$$

Equation (2) represents the weight of the security demand for each security parameter authentication, confidentiality and integrity as $\mathrm{wt_a}$, $\mathrm{wt_c}$ and $\mathrm{wt_i}$, respectively. The security demand for the workflow will be applied to all the tasks in the workflow by default. If the consumer wants different SD for each task, then the consumer can specify as an array list. This choice is provided by means of two modes, namely default mode and selective mode. The consumer is free to choose either the default mode or the selective mode to specify the SD value specification in the SLA. In the default mode, the consumer specifies their security demand as a single value. This value is uniquely applied to all the tasks in the workflow. On the other hand, in the selective mode the consumer will demand for different proportion of security demand for each task in the workflow.

4.1 Problem objective

The proposed work aims at devising a metaheuristic-based scheduling for scientific workflow. A workflow application is modelled as a directed acyclic graph, $W = (\mathrm{T,E})$, as shown in Fig. 1, where T is the set of n tasks and each task $t_i \in T$ represents an workflow application task. E is the set of communication edges between tasks and each e (i, j) $\in$ E represents the task dependence constraint such that task T_i should finish its execution before task T_j can be started.

The $parent(t_i)$ and $child(t_i)$ denote the set of predecessors and successors of task t_i, respectively. There is an entry task and an exit task in a workflow W. The entry task T_{entry} is the starting task of the application without any predecessors, while the exit task T_{exit} is the final task with no successors. In case there is no common entry task, a dummy task will be appended to have a single-entry task. Similarly, if there is no common exit task is available then a dummy task will be appended to have a single-exit task.

The objective of our proposed work is to minimize the overall execution cost (OEC) and overall execution time (OET) such that the security demand of the user is satisfied. The proposed work aims at devising a balanced multiobjective schedule model. It provides a balance between the time and cost by introducing a weight factor ω. The weight factor would be used to choose the importance between the two factors. If the customer wants to minimize the cost than the time, then they can give more weightage to the weight factor ω. Similarly if the customer wants to minimize the time taken for execution more by compromising for more payment for the high end VMs, then they can assign lesser weightage value for the weight factor ω. The objective function is thus defined in Eq. (3). Hence, the fitness of our problem can be defined as

$$\begin{aligned} F &= \mathrm{Min}\{\omega * \mathrm{OEC} + (1-\omega) * \mathrm{OET}\} \\ &\quad Subjected\ to\ SD_{t_i} \le \mathrm{SGL}_{\mathrm{VM}_j} \end{aligned} \tag{3}$$

where OEC and OET is a set of values associated with the varied virtual machines deployment for the required SD of ith task deployed on the jth VM with the required SGL. While computing the objective function, the computational time associated with the appropriate SGL is considered.

4.2 Security risk analysis

Since the cloud environment is risk-prone, we do evaluate the security of our proposed system. We define probability of security in scheduling a task t_i to a VM vm_j as $P_{\mathrm{Security}}\left(t_i, \mathrm{vm}_j\right)$. The security of a task execution on a virtual machine increases with the increase in the difference of the security demand of the task and the Security level of the VM. This change follows an exponential distribution. The probability of security is defined by an exponential distribution modelled as a function of the difference between SD and SGL as defined in Eq. (4). The security risk coefficient λ is a fraction number. The negative exponent indicates that the failure rate grows with the difference between SD_{t_i} and $\mathrm{SGL}_{\mathrm{vm_j}}$. The task failure at a site could result from VM hijacking, VM theft, Hyper Jacking, severe network attack or in accessibility from a security imposed barricade.

$$P_{\mathrm{Security}}\left(t_i, \mathrm{vm}_j\right) = \begin{cases} 0, & \mathrm{SD}_{t_i} \ge \mathrm{SGL}_{\mathrm{vm}_j} \\ 1 - e^{\lambda\left(\mathrm{SD}_{t_i} - \mathrm{SL}_{\mathrm{vm}_j}\right)}, & \mathrm{SD}_{t_i} < \mathrm{SGL}_{\mathrm{vm}_j} \end{cases} \tag{4}$$

The probability of security of a task is thus computed using Eq. (5) based on the assignment of the task t_i to virtual machine vm_j. The constant κ_{ij} is assigned 1 if the virtual machine vm_j is assigned to task t_i else it is assigned 0.

$$P_{\text{security}}(t_i) = \sum_{j=1}^{m} \kappa_{ij} P_{\text{security}}\left(t_i, \text{vm}_j\right)/m \tag{5}$$

When considering the entire workflow task set T, the proposed methods' probability of security is given by $P_{\text{security}}(T)$ as given in Eq. (6). $P_{\text{security}}(T)$ specifies the probability of security so that all tasks are free from being attacked during their executions.

$$P_{\text{security}}(T) = \text{Average}\left(P_{\text{security}}(t_i)\right) \tag{6}$$

5 Schedule primitives:

A common representation of a workflow application is the directed acyclic graph (DAG), which includes the characteristics of an application programme such as the execution time of tasks, the data size to communicate between tasks and task dependencies. We model the workflow as a directed acyclic graph, $W = (T,E)$, where T represents the set of tasks $T = \{t_1, t_2, ..., t_n\}$ and each task $t_i \in T$, $1 \le i \le n$ represents an workflow application task. Similarly E represents the set of communication edges between tasks and each edge e(i,j) ∈ E represents that the task t_j depends on the task t_i, i.e. task t_i should be executed prior to the execution of the task t_j.

The task which has in degree as 0 is designated as t_{entry} entry task and similarly the task without degree as 0 is designated as t_{exit} exit task. The parent tasks of the task t_i are denoted parent(t_i) and the child tasks of a task t_i are represented by child(t_j). In cloud environment, the provider has enormous amount of resources. The user has to pay for the resources. Let us consider the consumer has to purchase m number of VMs from the provider. These VMs will be of different instance types. The VMs computation capacity depends on the processor cores and the MIPS of the VM. Thus executing each task on different VMs will depend on the MIPS and the number of cores. We denote the computation capacity of each VM as $w\left(\text{vm}_j\right)$. The set of VMs is represented by $\text{VM} = \{\text{vm}_1, \text{vm}_2, ..., \text{vm}_m\}$. Similarly each task has its own computation cost and it is designated as $w(t_i)$ for the task t_i. The execution time of the task t_i on the jth VM of kth type is termed as $\text{ET}\left(t_i, \text{vm}_j\right)$ and it is defined in equation (7).

$$\text{ET}\left(t_i, \text{vm}_j\right) = \frac{w(t_i)}{w\left(\text{vm}_j\right)} \tag{7}$$

The average execution time of the task t_i is given in Eq. (8).

$$\overline{\text{ET}(t_\iota)} = \frac{\sum_{j=1}^{m} \text{ET}\left(t_i, \text{vm}_j\right)}{m} \tag{8}$$

The communication cost of an edge $e(i, j)$ can be calculated as given in Eq. (9).

$$\text{CC}\left(t_i, t_j\right) = \text{late}\left(\text{vm}_j\right) + \frac{w\left(e\left(i, j\right)\right)}{\text{bw}_k} \forall 1 \le j \le m,\ 1 \le i \le n \tag{9}$$

The average communication cost of an edge $e(i, j)$ can be calculated as given in Eq. (10).

$$\overline{\text{CC}}\left(t_i, t_j\right) = \frac{\sum_{j=1}^{m} \text{late}\left(\text{vm}_j\right)}{m} + \frac{w\left(e\left(i, j\right)\right)}{\sum_{k=1}^{o} \text{bw}_{k/k}} \tag{10}$$

where $\text{late}\left(\text{vm}_j\right)$ denotes the latency that a VM will have for executing any job. This will include the boot-up time for any VM to start up and execute the task. If two tasks are assigned within the same VM, then the latency $late(\text{vm}_j)$ for those two tasks is assumed to be 0. $w(e(i, j))$ is the amount of data transferred between the two tasks i and j. The bandwidth of the link between the VMs is denoted by bw_k, where k represents the number of links that exists between the VMs used. It is to be noted that the bandwidth within a data centre is the maximum that can be provided. Hence, it is assumed to be uniform with in a data centre.

The earliest start time (EST) is the earliest possible start time of a particular task on a VM. This can be computed by Eq. (11)

$$\begin{aligned}\text{EST}\left(t_i, \text{vm}_j\right) = \max\{&\text{ReadyTime}(t_i, \text{vm}_j),\\ &\max_{t_m \in \text{parent}(t_i)} \left\{\text{EET}(t_m, \text{vm}_j) + \text{CC}(t_m, t_i)\right\}\\ &\text{ReadyTime}\left(t_{\text{entry}}, vm_j\right) = 0\end{aligned} \tag{11}$$

The EST is defined by the considering the maximum of the ready time of vm_j and the earliest end time of its parent task on that VM. The $\text{ReadyTime}\left(t_i, \text{vm}_j\right)$ denotes the time in which the jth VM is ready for executing the task t_i. The same VM can be used to execute a task, provided it has completed its earlier task and ready for executing the next task. The ready time of the entry task is considered as 0. The earliest end time of a task is evaluated by considering the earliest start time of a task and the execution of the task on a VM. The EST of a task is evaluated by finding the maximum of the ready time of a VM for the task and the earliest end time of the entire ancestral task along with the data transfer time (Communication Cost). We consider initially all the VMs are free and are ready to execute any task for simplicity. If any two tasks are allotted with the same VM, then the communication cost of those two tasks $\text{CC}(t_m, t_i) = 0$.

The earliest end time (EET) of a task is the earliest possible completion time of a particular task on a VM. This can be computed by Eq. (12)

$$\text{EET}\left(t_i, \text{vm}_j\right) = \text{EST}\left(t_i, \text{vm}_j\right) + \frac{w(t_i)}{w\left(\text{vm}_j\right)} \forall 1 \le i \le n, 1 \le j \le m \tag{12}$$

Overall execution time (OET) is the total schedule length can be obtained by considering the EET of exit task and it is defined in Eq. (13). If there is no common exit task, then a dummy exit task will be appended to make a common end point in a workflow.

$$\mathrm{OET} = \mathrm{makespan} = \mathrm{EET}\left(t_{\mathrm{exit}}, \mathrm{vm}_j\right) \tag{13}$$

Overall execution cost (OEC) is the total cost incurred in utilizing the cloud resource for executing the workflow. OEC is the amount the user has to pay to the cloud provider. OEC of the workflow is given in Eq. (14).

$$\mathrm{OEC} = \sum_{j=1}^{m} \mathrm{Cost}\left(\mathrm{vm}_j\right) * \frac{\left[\mathrm{ReleaseTime}\left(\mathrm{vm}_j\right) - \mathrm{ReadyTime}\left(\mathrm{vm}_j\right)\right]}{\tau} \tag{14}$$

where $\mathrm{Cost}\left(\mathrm{vm}_j\right)$ is the cost fixed by the cloud provider for that VM type for one unit of time. We have used the cost values as per the Amazon AWS EC2 services. The $\mathrm{ReadyTime}\left(\mathrm{vm}_j\right)$ is the time at which the chosen VM is ready for executing the task in the workflow and the $\mathrm{ReleaseTime}\left(\mathrm{vm}_j\right)$ is the time at which the VM has finished the execution of the task in the workflow and is ready to shut down. The time unit by which the metering is done is denoted by τ.

5.1 Priority rank

The tasks in the graph G are first ordered with their priority. In the list scheduling heuristics, the tasks are first prioritized based on the rank of the tasks. Similarly, in our schedule generation we adopt the prioritization phase for ordering the tasks. The priority rank of the tasks is evaluated by Eq. (15). The prioritization is done from the exit task to the parent. The exit task's priority rank is derived from the average execution time of the exit task in the workflow. Similarly, the priority rank of other parental tasks is calculated by the recursive calculation of priority rank of parent task and communication cost added with the average execution time of the task.

$$\mathrm{PRank}\left(t_i\right) = \mathrm{ET}\left(t_i\right) + \max_{t_j \in \mathrm{child}(t_i)} \left\{\mathrm{PRank}\left(t_j\right) + \overline{\mathrm{CC}}\left(t_i, t_j\right)\right\}$$
$$\mathrm{PRank}\left(t_{\mathrm{exit}}\right) = ET\left(t_{\mathrm{exit}}\right) \tag{15}$$

6 Schedule generation

The VM types that we used for our experiments are listed in Table 3. Besides different types of VMs, the provider maintains the VMs in three different security zones. To select the initial set of resource pool, we need to provide all the heterogeneous types of VM. On the other hand, we have to be more concerned about the initial pool of VMs which is available for the PSO algorithm to search with. In order to make the initial pool of resources optimal, we devise a strategy, such that the search space is not exhaustive as well as it is not minimal. Selection of optimal initial resource pool paves the way for earlier convergence.

Our strategy for devising the initial resource pool is based on the security demand submitted by the customer. Instead of choosing the VMs from all the zones, we incorporate the strategy of selecting the VMs only from the zone in which the security demand value matches with the SGL interval of that zone. Also for allowing different tasks to choose different VMs we consider the tasks in parallel in the workflow. The initial resource pool is selected in such a manner that the total number of available VMs for consideration is equal to the $|\mathrm{VM}_{\mathrm{type}}| \times |\mathrm{W}_{\mathrm{Level}}|$. Thus the number of VMs selected in the initial resource pool is product of the total number of VM types $|\mathrm{VM}_{\mathrm{type}}|$ and the total number of levels $|\mathrm{W}_{\mathrm{Level}}|$ in the directed acyclic graph G representing the tasks.

Table 3 List of virtual machine types used in experiments

Type	vCPU	Memory (GB)	Storage (GB)	MIPS	Cost ($)
t2.small	1	2	1 × 160	1000	0.06
t2.medium	2	4	2 × 240	2000	0.12
m3.medium	1	3.75	1 × 400	2000	0.10
m3.large	2	7.5	2 × 320	4000	0.18
m3.xlarge	4	15	4 × 320	8000	0.24

Algorithm 1 genSchedule(W, VMs)

```
//Input: W = (T,E) //W - Workflow DAG with W_Level levels, T- Set task nodes, E-set of edges
//$VM_j^k$- a set of Virtual machines of type k, $1 \le j \le M$ , M is the max virtual machine in type k //and
VM represents the total number of initialized VMs available in the initial resource pool.
//Output: Schedule S
1.  Initialize OEC and OET to 0
2.  $S = \phi$ //S is the schedule
3.  Compute $ET(t_i, vm_j)\ \forall\, t_i \in T, vm_j \in VM$ using equation (7)
4.  Compute $\overline{CC}(t_i, t_j)\ \forall\, e(i,j) \in E$ using Equation (10)
5.  Compute the priority rank of tasks $PRank(T)$ using Equation (15)
6.  $T_O$ = RevTopSort( T, $PRank(T)$ ); //Sort the task list using Reverse Topological Sort
    considering the priority rank
7.  While($T_0 != \phi$)
8.  {
9.          $t_i = T_o.removeFront()$
10.         For each k in $|VM_{type}|$
11.         {
12.                 $vm_j^k$ = Randomly select one VM from the VM pool.
13.                 If($SD_{t_i} \le SGL_{VM_j}$)
14.                 {
15.                 Compute the EET$(t_i, vm_j^k)$, $1 \le i \le n, 1 \le j \le |VM^k|$ using equation (12)
16.                 Compute the OET$(t_i, vm_j^k)$ and OEC$(t_i, vm_j^k)$, $1 \le i \le n$, $1 \le j \le |VM^k|$ using
                    equation (13) and (14)
17.                 }
18.         }
19.         For each $t_i$ in $T$
20.         {
21.                 $S_i$ =Select the Task-VM pair $(t_i, vm_j)$which has the minimum OET$(t_i, vm_j^k)$
                    and OEC$(t_i, vm_j^k)$ values
22.                 $S = S \cup S_i$ //Append the Task-VM pair to the schedule
23.         }
24. } //end of while
25. return S
```

The above schedule generation algorithm works on the basis of list scheduling heuristics. The input VM_j^k is initialized from the available list of virtual machines at the data centre based on the total number of VM types and the total number of levels in the directed acyclic graph W representing the workflow tasks. This same VM pool will be used in the subsequent Smart Particle Swarm Optimization (SPSO) algorithm. Overall execution time (OET) and overall execution cost (OEC) are initialized to zero. Then the computation of the execution time, communication cost and priority rank is done. The priority rank is calculated from exit task. Thus it represents the bottom level of the task in the workflow DAG. Hence, we sort it by the reverse topological sorting method. The while loop comprising the lines from 7 to 22 creates the schedule and identifies the best VM for the task. The first for loop from line 10 to 18 focuses on the computation of the earliest end time of the tasks on different types of VM instances provided the Security demand of the user is satisfied by the VM. The OET and OEC values are also computed. Minimizing these time and cost values is our main objective. Once the values of OET and OEC are computed, the algorithm checks for the best feasible task–VM pair for a particular task. This is done by the for loop at line 19. Line 20 assigns the best feasible task–VM pair to the schedule for the ith task. Then it is appended to the final schedule S, at the end of the while loop.

The output generated by this algorithm is converted to the particle's position in the SPSO algorithm. Now, we have all the information needed to begin decoding the particle's position and constructing the schedule. The search space available for the particle is the VM set which has the initialized VM pool. The virtual machine assignment made at line 20 gives the resource mapping done for the task t_i. Now, this is to be optimized with the SPSO algorithm for getting the optimized schedule for the given workflow application. The fitness function will use the OEC and OET calculated by this algorithm.

The time complexity of this genSchedule(W, VMs) is $O(n^2)$. Here n is the number of tasks in the workflow W. The while loop at line 7 iterates for 'n' times considering one task at a time. The first inner for loop at line 10 iterates only for the number of VM instance types, which is always less than the number of tasks. On the other hand, the for loop at line 19 will be executed for another 'n' number of times. Thus the overall time complexity of this algorithm is $O(n^2)$.

7 PSO-based Optimization

7.1 Smart Particle Swarm Optimization—SPSO algorithm

Particle Swarm Optimization (PSO) is a heuristic search method which is inspired by the swarming or collaborative behaviour of biological populations such as flocks of birds, schools of fish and herds of animals. PSO is a population-based search method. The population can be assumed like the set of points, particle or potential solution in the problem of study. In PSO, the population move from a set of points to another set of points, in each iteration with a velocity. The movement of the particle contributes to probable improvement. The velocity function is devised using a combination of deterministic and probabilistic rules.

7.1.1 Particle encoding strategy

In our work, we use the Particle Swarm Optimization algorithm by representing the tasks in the workflow as particle. Each particle maintains a position, composed of the candidate solution and its evaluated fitness, and its velocity.

To address the multiobjective problem, we define the encoding strategy for the particle with its dimension. We design our particle to be of four dimension. A particle $p_i = \left(t_i, \mathrm{vm}_j, \text{ vm Type}, \text{ SGL}_{\mathrm{vm}_j}\right)$. The first dimension represents the ith task in the workflow, i.e. the task for which the particle is designated. The second dimension vm_j represents the VM id that was selected in the current iteration for evaluating the fitness based on the previous position and the velocity. The third dimension represents the vm Type as specified in Table 3. Finally, the fourth dimension represents the SGL of the VM according to Eq. (1). Thus each particle can be viewed as a structure represented in Table 4. The figure shows the particle's dimension for the sample workflow shown in Fig. 1 as a population. Each task in the workflow have four dimension or can be said as a coordinate associated with it. The dimension represents the position and the velocity at which the task can move in the solution space. Hence, the position of the particle is consolidated by the dimension of the particle.

Table 4 Particle structure representation for the sample workflow shown in Fig. 1

Tasks in the workflow	Dimension of the particle corresponding to the task
T_1	(1,1, t2.medium, 0.9)
T_2	(2,3, m3.medium, 0.95)
T_3	(3,2, m3.large, 0.9)
T_4	(4,3, m3.medium, 0.95)
T_5	(5,1, t2..medium, 0.9)
T_6	(6,3, m3.mediium, 0.95)
T_7	(7,4, m3.medium,0.95)
T_8	(8,2, m3.large, 0.9)
T_9	(9,1,t2.medium, 0.9)
T_{10}	(10,4,m3.medium,0.95)

7.1.2 Basic terms

The PSO algorithm works in such a way that these generated particles swarm as a population such as the swarm of birds searching for the optimal solution. In our proposed work, the particle moves with a velocity searching for the best schedule and the best VM that can be allocated for it, so that it can give best optimized results. PSO algorithm does have two main parameters generally called as Pbest and Gbest. Pbest is the particle's best position that it maintained during its swarm. Similarly, Gbest is the global best that the population experienced during the swarm. During the execution of the PSO algorithm, the particle's velocity is stochastically accelerated towards its best position and towards the global best. The particle's position is represented as $\vec{p}_\imath(t) = (p_{i1}(t), p_{i2}(t), \ldots, p_{in}(t))$ at time t and the velocity of the particle is represented as $\vec{v}_\imath(t) = (v_{i1}(t), v_{i2}(t), \ldots, v_{in}(t))$. At the next time step $(t+1)$, the particle moves to a new position and the updated position can be computed using Eq. (17). In the proposed work the task t_i moves to another virtual machine vm_j in search for the best optimal virtual machine. The search for the next virtual machine depends on the velocity of the particle. The speed in which the particle moves is determined by the velocity. The velocity vector is updated using Eq. (16). The particles move with this velocity in search of a new solution with better fitness. The fitness of the particle is evaluated using Eq. (3).

$$\vec{v}_\imath(t+1) = \alpha\,\vec{v}_\imath(t) + c_1 r_1\left(\overrightarrow{p_{\mathrm{pbest}}}(t) - \vec{p}_\imath(t)\right) + c_2 r_2\left(\overrightarrow{p_{\mathrm{Gbest}}}(t) - \vec{p}_\imath(t)\right) \tag{16}$$

$$\vec{p}_\imath(t+1) = \vec{p}_\imath(t) + \vec{v}_\imath(t+1) \tag{17}$$

where $\propto$ is the inertia weight. It is used to control the influence of earlier velocity on the new velocity. Larger inertia weight makes an extensive search, whereas smaller inertia weights focus on smaller region. In our work, we have to make the task select a best VM for it so that its makespan and money to be paid to the cloud provider have to be minimized. Hence, we choose smaller value for inertia weight. This helps in fast convergence too. The symbols $r_1 r_2$ are random numbers which are randomly selected in the interval [0,1]. These random values make the candidate solution distinct. The symbols c_1 and c_2 are called as acceleration constants. Kennedy has studied (Kennedy 1998; Kennedy et al. 2001) the effect of these variables on the particle lights and asserted the favourable condition for the swarm is $c_1+c_2 \leq 4$. Otherwise it may explode towards infinity. The second term in Eq. (17) is the cognitive component and the third term is the social component influencing the swarm movement.

A sample value of the particle's position at an iteration is shown in Table 5. For simplicity, we have shown this with four VMs and ten tasks. The position value computed for each task is given in Table 5. The coordinate values will be rounded off and the resultant value represents the selected VM for each task. Based on this allocation the overall execution ost (OEC) and overall execution time (OET) will be computed. After updating the particle's position and velocity in the next iteration the fitness of the new solution will be checked. This process will be repeated till convergence in the fitness function.

7.1.3 Smart Particle Swarm Optimization (SPSO) algorithm

The proposed Smart Particle Swarm Optimization (SPSO) algorithm has a smart initialization procedure. Traditional PSO algorithms used to initialize the particle's position in a random way. But here we use a schedule generation algorithm 'genSchedule', which works out a list-based heuristics for the given workflow. The output of this 'genSchedule' algorithm is itself an optimal schedule. But here we want to optimized the generated schedule further by using the metaheuristic approach. The idea of integrating the list-based heuristic with the metaheuristic approach is twofold. First, we will get the best optimized schedule with resource allocation for the given workflow. Second, the number of iterations is reduced to fewer numbers, thus reducing the execution time of the algorithm, which is very critical in cloud environments.

At each iteration, the fitness value will be compared with the Pbest and then with Gbest. If the new values are giving better results, then the new values will be updated in Pbest and Gbest accordingly. The detailed methodology is given in SPSO algorithm given below.

Algorithm 2 : SPSO(S, |T|)

Input: Number of tasks in workflow |T|, Generated schedule S from algorithm 1.

Output: Optimized schedule as particle's Position.

```
1.  $P_{Dim} = |T| * 4$; //Initialize the Particle's population dimension with the No. of tasks in
    Workflow*4
2.  For (i=1;i<$P_{Dim}$;i++)
3.  {
4.  $\overrightarrow{p_i}(t) = S$; //Initialize Particle's position $\overrightarrow{p_i}(t)$from the generated schedule 'S' from algorithm 1
5.  $\overrightarrow{v_i}(t) = random(\,)$; //Initialize Particle's velocity randomly
6.  Pbest=$\overrightarrow{p_i}(t)$; //Initialize Pbest with the initial position of the particle.
7.  Gbest = $argmin\left(\overrightarrow{p_i}(t)|\overrightarrow{p_i}(t) \in p_{i1}(t), p_{i2}(t), \ldots, p_{iP_{Dim}}(t)\right)$; //Initialize Gbest with the best
    position among all the particles.
8.  }
9.  Do // Start of Until loop
10. {
11. FOR each $\overrightarrow{p_{ij}}(t)$ in $\overrightarrow{p_i}(t)$
12. {
13.         Calculate its fitness value using equation (3); // Fitness value will be calculated with
    OET and OEC subjected to satisfying the security constraints.
14.         IF fitness of current particle$\overrightarrow{p_{ij}}(t)$is better than Pbest then
15.                 Pbest=$\overrightarrow{p_{ij}}(t)$; //Update Pbest with current particle's value
16.         End IF
17.         IF fitness of current particle$\overrightarrow{p_{ij}}(t)$is better than Gbest then
18.                 Gbest=$\overrightarrow{p_{ij}}(t)$; // Update Gbest with current particle's value
19.         End IF
20.         $\overrightarrow{v_i}(t+1) = \overrightarrow{v_i}(t)$;  // Update the particle's velocity using equation (16)
21.         $\overrightarrow{p_i}(t+1) = \overrightarrow{p_i}(t)$; //Update the particle's position using equation (17)
22. } // End FOR
23. } UNTIL (Results converge) //End UNTIL
```

Table 5 Allocation of VM for different task's based on its position

Tasks in the workflow acting as particle	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
Position	1.3	2.2	3.6	1.4	3.2	2.4	4.3	1.2	2.1	4.2
VM assigned	1	2	3	1	3	2	4	1	2	4

The algorithm SPSO(S, |T|) takes the schedule generated by algorithm 1 and the number of tasks as input. The dimension of the population is set to the number of tasks in the workflow *4. The particles are going to represent the tasks in the scientific workflow and the position will represent the virtual machine selected. This is described in Table 3. The for loop from lines 2 to 8 will do the needed initialization. At line 4, the particles position is initialized with the position from the schedule. That is, the particles are initialized with the virtual machine assignment decided from the schedule S. This initialization makes the SPSO algorithm smarter by initializing the particles in the near optimal locations. This will aid in faster convergence of the schedule. The velocity is initialized randomly at line 5. The particle's Pbest is initialized with the initial particle's position at line 6. The global best of the population, Gbest is initialized at line 7. The Gbest is initialized in such a way that the best of the entire particle's position is selected. Lines 9–23 will be executed for each iterations of the swarm search. Line 13 calculates the fitness value for each particle. Then it is compared with the Pbest. If a better result is obtained in the search, then the Pbest is updated. Similarly, lines 17–19 check for the improvement in the Gbest. If any improvement is identified, then the Gbest will be updated with that improvement. In lines 20 and 21, the velocity and the positions are updated, so that the particles move in the right direction by using the knowledge gained in the current iteration. The until loop will be iterated till the objective values of OET and OEC converge. Our SPSO algorithm does converge within 20 iterations on an average.

The time complexity of this SPSO(S, |T|) is $O(qn)$. Here n is the number of tasks in the workflow W. The for loop from line 2 to 8 will be executed for 'n' times. Because the particles population dimension is n. The do–until loop starting at line 9 will be executed for 'q' times. The q value set in the experiment is 25. On an average case, the results converged within 17 iterations for smaller workflows and around 20 for larger workflows. The for loop starting at line 11 will be iterated for 'n' times accounting for the 'n' times in the time complexity. Hence, the time complexity of this algorithm is $O(qn)$.

7.2 Smart Variable Neighbourhood Particle Swarm Optimization (SVNPSO) algorithm

The major drawback of PSO algorithm is that at some worst cases it will suffer from local minima problem. In such cases, the results will not converge even with more number of iterations. In order to escape from local minima problem, we have also tried another variant of PSO algorithm called Variable Neighbourhood Particle Swarm Optimization (VNPSO). It is a combination of the Variable Neighbourhood Search (VNS) and Particle Swarm Optimization (PSO) (Liu et al. 2012; Abraham et al. 2006). The VNS approach is used for various applications. One among is the usage for scheduling. Marinakis and Marinaki (2013) has used the variable neighbourhood search strategy and a path re-linking strategy for permutation flow shop scheduling. VNPSO algorithm is based on repeated exploration of neighbourhoods in progressively increasing size to identify better local optima using so called shaking strategies. Shaking phase normally generates a new point at random from the kth neighbourhood of the particle 'p'. The Variable Neighbourhood Search framework exploits systematically the idea of neighbourhood change, both in the descent to local minima and in the escape from the valleys which contain them (Liu et al. 2012; Hansen et al. 2010; Liu and Abraham 2007). These searches are done in steps and are repeated until a given termination condition is met. Since our algorithm's fitness function checks for the security demand being satisfied in its constraints, we name our algorithm as Smart Variable Neighbourhood Particle Swarm Optimization (SVNPSO) algorithm.

Like the previous algorithm, here also we have used the smart initialization of the particles in the population. The population size is the number of tasks in the workflow.

7.2.1 Basic terms

In SVNPSO, we use a threshold velocity to identify when to apply the shaking strategy. If the velocity of the particle decreases below the threshold velocity v_{th}, the new updated velocity is computed using Eqs. (18) and (19).

$$\vec{v}_\iota\,(t+1) = \alpha v_{\text{update}} + c_1 r_1 \left(\overrightarrow{p_{\text{pbest}}}\,(t) - \vec{p}_\iota\,(t)\right) + c_2 r_2 \left(\overrightarrow{p_{\text{Gbest}}}\,(t) - \vec{p}_\iota\,(t)\right) \tag{18}$$

$$v_{\text{update}} = \begin{cases} \vec{v}_\iota & \text{if } |\vec{v}_\iota| \geq v_{\text{th}} \\ x v_{\max/\mu} & \text{if } |\vec{v}_i| < v_{\text{th}} \end{cases} \tag{19}$$

The updated velocity is represented by v_{update} and it is computed using Eq. (19), where x represents a random numbers generated by uniform distribution in the interval $[-1,1]$. The efficiency of the algorithm is characterized by two parameters v_{th} and x. The value of x is responsible for changing the direction of the variable neighbourhood search.

The value of v_{th} is responsible for escaping from the local minima. A larger value for v_{th} makes the particle to be in its flying state, which prevents them from converging to a solution (Liu et al. 2012).

7.2.2 SVNPSO algorithm

the search by the swarm particles. When there is no improvement, it is incremented or else it is reinitialized to 0. This helps in making decision whether to apply the shaking strategy or not. We define a limit value for this as 7. That means for 7 consecutive swarm search iterations there is no possibility of improvement. This is checked in line 26. When it

Algorithm 3: SVNPSO (S, |T|)

Input: Number of tasks in workflow |T|, Generated schedule S from algorithm 1.

Output: Optimized schedule as particle's Position.

```
1.  $P_{Dim} = |T| * 4$; //Initialize the Particle's population dimension with the No. of tasks in Workflow*4
2.  Initialize $l = 0$ ; // Initialize Neighbourhood search limit
3.  FOR (i=1;i<$P_{Dim}$;i++)
4.  {
5.  $\overrightarrow{p_i}(t) = S$; //Initialize Particle's position $\overrightarrow{p_i}(t)$from the generated schedule 'S' from algorithm 1
6.  $\overrightarrow{v_i}(t) = random()$; //Initialize Particle's velocity randomly
7.  Pbest = $\overrightarrow{p_i}(t)$; //Initialize Pbest with the initial position of the particle.
8.  }
9.  DO // Start of Until loop
10. {
11. FOR each $\overrightarrow{p_{ij}}(t)$ in $\overrightarrow{p_i}(t)$
12. {
13.         Calculate its fitness value using equation (3); // Fitness value will be calculated with OET and
    OEC subjected to satisfying the security constraints.
14.         Gbest = $argmin\left(\overrightarrow{p_i}(t)|\overrightarrow{p_i}(t) \in p_{i1}(t), p_{i2}(t), \ldots, p_{iP_{Dim}}(t)\right)$ ; //Initialize Gbest with the best
    position among all the particles.
15.         IF fitness of current particle $\overrightarrow{p_{ij}}(t)$is better than Gbest then
16.                 $l = 0$
17.         ELSE
18.                 $l = l + 1$
19.         End IF
20.         IF fitness of current particle $\overrightarrow{p_{ij}}(t)$is better than Pbest then
21.                 Pbest = $\overrightarrow{p_{ij}}(t)$; //Update Pbest with current particle's value
22.         End IF
23.         IF fitness of current particle $\overrightarrow{p_{ij}}(t)$is better than Gbest then
24.                 Gbest = $\overrightarrow{p_{ij}}(t)$; // Update Gbest with current particle's value
25.         End IF
26.         IF($l < 7$) then
24.         $\overrightarrow{v_i}(t+1) = \overrightarrow{v_i}(t)$;  // Update the particle's velocity using equation (16)
25.         $\overrightarrow{p_i}(t+1) = \overrightarrow{p_i}(t)$; //Update the particle's position using equation (17)
27.         ELSE
28.             $\overrightarrow{v_i}(t+1) = \overrightarrow{v_i}(t)$; // Update theparticle's velocity using equation (18)&(19)
29.         $\overrightarrow{p_i}(t+1) = \overrightarrow{p_i}(t)$;  //Update the particle's position using equation (17)
30.         End IF
31. }/ End FOR
32. }UNTIL (Results converge) //End UNTIL
```

The algorithm SVNPSO(S, |T|) takes the schedule generated by algorithm 1 and the number of tasks as input. The particles usage is the same as in the previous algorithm. The variable l used in line 2 represents the neighbourhood search limit. This will check whether there is any improvement in is understood that for seven consecutive iterations there is no improvement; then, we update the velocity using v_{update} which implements the shaking strategy. The for loop from lines 3 to 8 will do the needed initialization. At line 5, the particles' position is initialized with the position from the schedule. That is, the particles are initialized with the virtual machine assignment decided from the schedule S. This initialization makes the SVNPSO algorithm smarter in such a way the particles are initialized with the near optimal locations. This will aid in faster convergence of the schedule. The velocity is initialized randomly at line 6. The particle's Pbest is initialized with the initial particle's position at line 7. The global best of the population, Gbest is initialized at line 14. The Gbest is initialized in such a way that the best of the entire particle's position is selected. Lines 9–32 will be executed for each iterations of the swarm search. Line 13 calculates the fitness value for each particle by considering the OET and OEC values subjected to satisfying the security constraints. The particles' fitness value is compared with the Pbest. If a better result is obtained in the search, then the Pbest is updated from lines 20–22. Similarly, lines 23–25 check for the improvement in the Gbest. If any improvement is identified, then the Gbest will be updated with that improvement. In lines 26 and 30 the velocity and the positions are updated based on the neighbourhood search limit, so that the particles move in the right direction by using the knowledge gained in the current iteration. The until loop will be iterated till the objective values of OET and OEC converge.

The time complexity of this SVNPSO(S, |T|) is also $O(qn)$. Here n is the number of tasks in the workflow W. The for loop from line 3 to 8 will be executed for 'n' times. The particles' population dimension is n. The do–until loop starting at line 9 will be executed for 'q' times. The q value set in the experiment is 30. On an average case, the results converged within 20 iterations for smaller workflows and around 22 for larger workflows. This SVNPSO algorithm will suit for overcoming local minimum. The for loop starting at line 11 will be iterated for 'n' times accounting for the 'n' times in the time complexity. Hence, the time complexity of this algorithm is $O(qn)$.

8 Experimentation and performance analysis

We implemented the proposed algorithms using WorkflowSim Chen and Deelman (2012) to evaluate their performance. WorkflowSim is the extension of CloudSim Simulator (Calheiros et al. 2011). It is an open-source simulator and it provides a complementary level of managing the workflows. The workflow applications considered to evaluate the proposed system are the real-world scientific workflow applications such as Montage (2015,Berriman et al. 2004), Epigenomics (USC Epigenome Center 2015) and CyberShake (Graves et al. 2010). All these workflows are delivered to the scientific community in a robust and scalable way by the Pegasus Workflow Management System tools (Pegasus Workflow Management System 2015). All the three workflows, namely Montage, Epigenomics and Cybershake, have

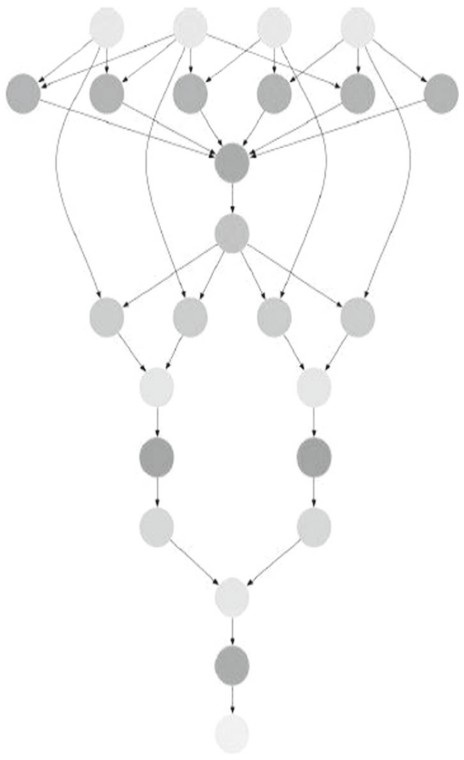
Fig. 4 Montage workflow structure

different structures as given in Figs. 4, 5 and 6 and have different computational and data characteristics. Workflow management system (WfMS) is used to manage scientific workflows. The scientific workflows can be characterized and used for testing the workflow scheduling (Workflow Generator 2015).

WorkflowSim consists of a Workflow Mapper to map abstract workflows to concrete workflows depending on their execution sites; a Workflow Engine to handle the data dependencies; and a Workflow Scheduler to match jobs to resources.

The DAX is a description of an abstract workflow in XML format that is used as the primary input in our model. The documentation of the schema and its elements can be found in (Schema of workflow in XML format 2015). The DAX file consists of the following three parts:

1. List of all referenced files (optional).
2. Definition of all jobs (1..*).
3. List of control flow dependencies (optional).

Let the unit of time be represented as τ. If there is any partial utilization of the leased VM, that can be also be considered as full time utilization. For example, consider that $\tau = 60$ min then, if a VM is used for 61 minutes, then the user will pay for 2 periods of 60 min, that is, 120 min. Also, there is no limitation for the number of VMs to be leased.

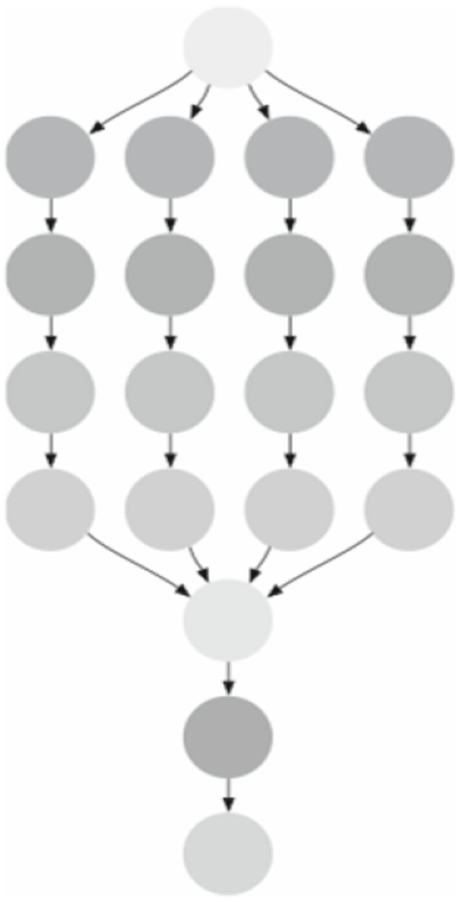

Fig. 5 Epigenomic workflow structure

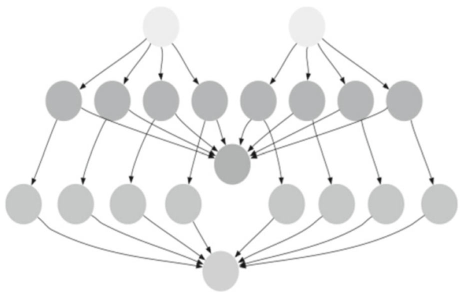

Fig. 6 Cybershake workflow structure

The different types of VMs and cost per unit time of the VMs used in our experiments are given in Table 3.

8.1 Performance analysis with real workflows based on makespan

Montage is an application for constructing custom astronomical image mosaics of the sky. We considered Montage graphs with 25, 50 and 100 tasks. The Epigenomics workflow is used to map the epigenetic state of human cells on a genome-wide scale. As was the case for the other real application graphs, the structure of this application is known. In our experiment, we used graphs with 24, 46 and 100 tasks. The CyberShake workflow is used to characterize earthquake hazards in a region using the Probabilistic Seismic Hazard Analysis technique. The CyberShake workflows with 30, 50 and 100 tasks are used for experimentation. The Montage 25 tasks, Epigenomics 24 tasks and Cybershake 30 tasks workflow are considered as the small group. Similarly Montage 50 tasks, Epigenomics 46 tasks and Cybershake 50 tasks workflow are considered as the medium-sized workflows. Finally Montage 100 tasks, Epigenomics 100 tasks and Cybershake 100 tasks workflow are considered as the large-sized workflows.

To show the efficiency of our proposed SPSO and SVNPSO algorithms, we have considered two algorithms with list heuristic approach, namely HEFT algorithm (Topcuoglu et al. 2002) and PEFT algorithm (Arabnejad and Barbosa 2014). At the same time, we consider two more PSO-based algorithms, namely PSO algorithm (Rodriguez and Buyya 2014), RDPSO algorithm (Wu et al. 2010). These four algorithms are used to compare with our proposed SPSO and SVNPSO algorithms.

The experiments were executed for 50 times and the results were plotted with the average value of those fifty experiments. Thus the makespan plotted in Table 6 is the average makespan of 50 different trials.

PSO Parameters: The population size is fixed as the number of tasks in the workflow *W*. The search space for the particles is the initial virtual machine pool initialized by $|VM_{type}| \times |W_{Level}|$. For instance, for the sample workflow given in figure the number of VMs in the initial pool is 25 VMs (5×5=25). The number of VM types considered is 5 as per Table 3 and the number of level is 5 as per Fig. 1. In Sect. 5, we have considered that the consumer has to purchase *m* number of VMs from the provider. This m will be decided by the SPSO algorithm depending on the security guaranteed level being satisfied. In our experiments, the minimum number of VM being allocated for the tasks in the workflow is 3 and the maximum number depends on the dynamic allocation of the algorithm depending on the availability of the VM and the SGL being satisfied.

We have varied the value of $\propto$, the inertia weight from 0.2 to 0.7. We observed that this inertia weight has a higher impact on the convergence of results. The best results can be had when the inertia is fixed as 0.4–0.6. On an average case, the results converged within 17 iterations for smaller workflows and around 20 for larger workflows for the SPSO algorithm. For the SVNPSO algorithm, the results converged within 20 iterations for smaller workflows and around 22 for larger workflows on an average. The extra iterations needed are due to the search for overcoming the local minima. The values of c_1 and c_2 are varied from 1.5 to 2.0 in our experiments. But change in these values does not affect the convergence rate much. On an average, we have preferred 1.9 for both c_1 and c_2, so that the sum of c_1 and c_2 is within 4.

Table 6 Tabulation of makespan for different sized scientific workflow applications

Size	Workflow used	HEFT	PEFT	RDPSO	PSO	SPSO	SVNPSO
Small	Montage	243	210	223	136	49.833	51.823
	CyberShake	356	324	566	213	181	186
	Epigenomics	292	245	278	253	195	212
Medium	Montage	162	142	156	112	68.25	77.33
	CyberShake	454	421	1242	402	312	310
	Epigenomics	440	425	435	398	356	365
Large	Montage	332	310	325	256	212	230
	CyberShake	654	612	1132	578	488	498
	Epigenomics	624	596	612	568	534	545

Table 7 Tabulation of speedup for different sized scientific workflow applications

Size	Workflow used	HEFT	PEFT	RDPSO	PSO	SPSO	SVNPSO
Small	Montage	16.2	15.4	14.911	12.673	9.937	13.108
	CyberShake	26	23	23.14	19.57	16.94	18.725
	Epigenomics	10.2	9	6.99	10.03	6.96	7.431
Medium	Montage	27	24	24.9	21.371	16.169	24.67
	CyberShake	36	33	32.66	28.32	25.14	32.13
	Epigenomics	14	12	11.613	18.44	10.472	11.731
Large	Montage	44	42	40.4	38	27.5	33
	CyberShake	56	53	49.5	47	41	46
	Epigenomics	17	16	12	36	14	13

The makespan of Montage, Cybershake and Epigenomics workflow applications for small, medium and large task groups is tabulated in Table 6. The results show that the proposed SPSO and SVNPSO algorithm gives a better makespan than the existing algorithms such as HEFT, PEFT, RDPSO and PSO algorithms. In this, the RDPSO algorithm doesn't work good for Cybershake, which is more breadthwise distributed. Also we can infer that SVNPSO algorithm achieves good makespan for random structures such as partial parallel and sequential. But for workflow such as Epigenomics, which has a good parallel structure, it lags a little bit in yielding a good makespan. The structure of Montage and Cybershake gives good random structure. Montage has both depth and breadth in its structure, whereas Cybershake has breadth structure. Epigenomics has good depth structure. Thus we have considered both depth and breadth structure in the workflow for our experiments and our SPSO and SVNPSO is able to achieve good makespan and also able to find the possibly secured VM for the tasks to be allocated.

8.2 Performance analysis of real workflows based on speedup

Speedup The speedup of our algorithm is calculated as the fraction of cumulative computation costs and fitness function computation of the tasks in the workflow with the makespan of the output schedule. It is defined in Eq. (20).

$$\text{speed up} = \frac{\sum_{t_i \in T} \overline{\text{ET}}\,(t_i) + \text{time taken for F}}{\text{makespan}} \tag{20}$$

The speedup of Montage, Cybershake and Epigenomics workflow applications for small, medium and large task groups is tabulated in Table 7. The results show that the proposed PSO algorithm gives the best speedup compared to other algorithms. For the Epigenomics workflow, which has a good depth structure the SVNPSO algorithm works better compared to other workflows.

The SPSO and SVNPSO algorithms have converged with minimum number of iterations. Our results for overall execution cost and overall execution time has converged within 20 iterations.

8.3 Performance analysis with synthetic workflows

To evaluate the performance of the proposed algorithm, with larger-sized workflows we have used workflow generator. The workflow generator is used to generate synthetic workflows with various structures. The workflow graph generator uses several parameters such as number of tasks as DAG size, number of levels, link density, maximum and minimum expected values of task processing times and inter-

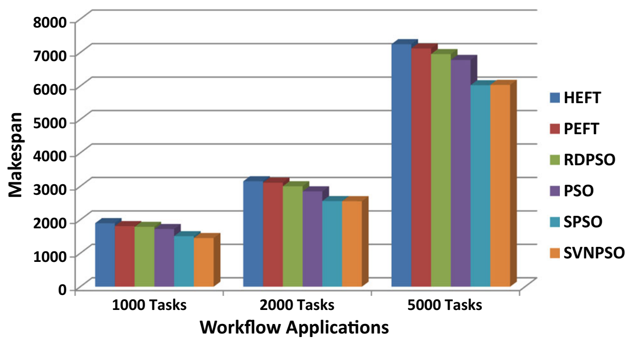

Fig. 7 Comparison of makespan with different sized workflows

task communication times. The task processing time on every VM follows uniform distribution. The communication time on each edge is based on uniform distribution. The communication-to-computation ratio (CCR) encodes the complexity of the computation of a task depending on the number of elements in the workflow application. We have generated workflows of sizes 1000, 2000 and 5000. We have also experimented with different CCR values. The range of values we used in our simulation is 0.1, 0.5, and 1 for CCR. In order to differentiate the virtual machines' computational capacity, each virtual machine has been created by varying MIPS rate and PesNumber according to Table 3.

We have generated workflows with 1000, 2000 and 5000 tasks and plotted the makespan in Fig. 7. It can be inferred from the diagram that the proposed SPSO an SVNPSO algorithm gives better results even for larger workflows. On an average, the SPSO algorithm gives 18.7 % improvement with respect to HEFT algorithm, 14.5% improvement with respect to RDPSO algorithm and 11.2% improvement with respect to PSO algorithm. Similarly, the SVNPSO algorithm gives 19.6 % improvement with respect to HEFT algorithm, 15.5% improvement with respect to RDPSO algorithm and 12.3% improvement with respect to PSO algorithm. This proves the efficiency of our proposed SPSO and SVNPSO algorithm.

We have tested the synthetic workflows with different CCR values for workflows. The communication-to-computation ratio (CCR) of a workflow is used to represent graph characteristics, which is either communication-intensive or computation-intensive. For higher value of CCR, the graph is more communication-intensive and if the CCR value is low then graph is computation-intensive. It is calculated by the average communication cost divided by the average computation cost on a target computing system. The range of CCR values that we used in our simulation is 0.1, 0.5 and 1. The comparison of makespan with different CCR values for workflows with 1000 tasks, 2000 tasks and 5000 tasks are shown in Table 8. It can be noted that the proposed SPSO and SVNPSO algorithms give better makespan compared to other algorithms. The RDPSO algorithm which didn't work well for Cybershake workflow shows the same effect when the CCR values are increased. The SPSO and SVNPSO algorithm is able to give improvements even with higher CCR values, which proves the consistency of our approach.

8.4 Performance analysis on security

Probability of security with different risk coefficient λ for the SPSO algorithm is plotted in Figs. 8 and 9. Similarly, the probability of security with different risk coefficient λ for the

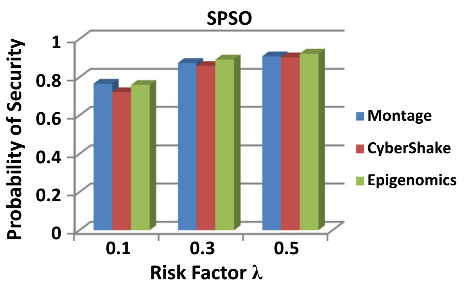

Fig. 8 Probability of security with different risk coefficient λ for small workflow applications using SPSO algorithm

Table 8 Tabulation of makespan for different sized synthetic workflows

Size	CCR value	HEFT	PEFT	RDPSO	PSO	SPSO	SVNPSO
1000	0.1	1897	1812	1789	1723	1510	1456
	0.5	2276	2210	3021	2100	1895	1890
	1	4100	3950	4966	3789	3554	3521
2000	0.1	3152	3112	3002	2850	2560	2555
	0.5	4123	4012	4556	3750	3550	3560
	1	5988	5850	6952	5600	5200	5190
5000	0.1	7250	7122	6952	6778	6023	6033
	0.5	8120	8025	8788	7800	7500	7560
	1	9452	9396	11242	9152	8950	8965

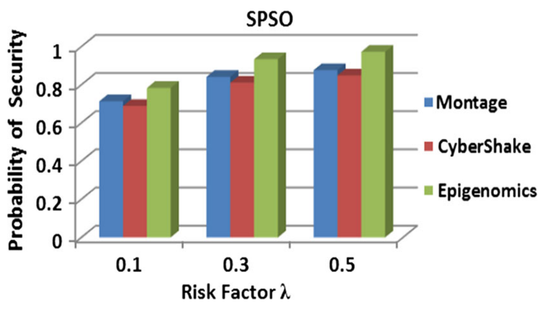

Fig. 9 Probability of security risk with different risk coefficient λ for medium workflow applications using SPSO algorithm

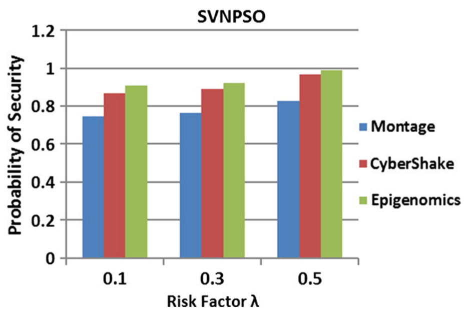

Fig. 10 Probability of security with different risk coefficient λ for small workflow applications using SVNPSO algorithm

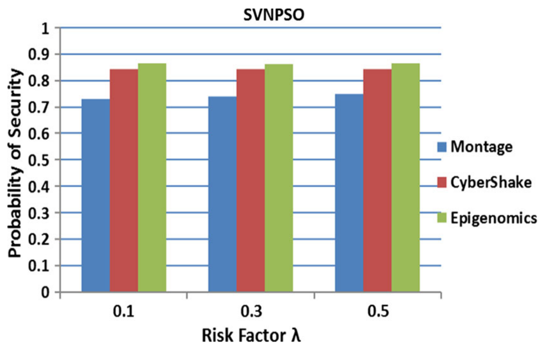

Fig. 11 Probability of security risk with different risk coefficient λ for medium workflow applications using SVNPSO algorithm

SVNPSO algorithm is plotted in Figs. 10 and 11. From the bar chart, we can infer that the probability of security is in the interval [0.1]. The proposed model ensures to increase the security as the risk factor increases. Our method will allocate the VMs, which are having better security Level. This has been proved with the probability of security plotted in Figs. 8, 9, 10 and 11. Thus the proposed model generates a good secured schedule for the workflow applications in cloud computing.

9 Conclusion

In the present work, we have proposed SPSO and SVNPSO algorithm with secured scheduling approach for scientific workflows in cloud computing. It gives an effective method for the utilization of the resources by proposing to use the metaheuristic optimization for resource allocation and scheduling. The proposed method for scheduling the workflow in the IaaS of cloud uses the naturally driven optimizing algorithm, and PSO provides an automated IaaS provisioning with minimal decision time in cloud for scientific workflows. The proposed algorithm provides a trade-off between security and minimum makespan and cost for workflow execution in cloud. Security constraints are introduced into the optimization model. The proposed algorithms SPSO and SVNPSO uses the coding strategy designed for encoding the particle to solve the multiobjective problem. Experimental results show that the proposed algorithm gives comparatively good results than the previous methods with minimum iterations which is well suited for cloud environment.

Compliance with ethical standards

Conflict of interest Our research work doesn't have any conflict of interest. The authors declare that they have no conflict of interest.

Human and animal rights This research work doesn't involve human participants and/or animals.

References

Abraham A, Liu H, Chang TG (2006) Variable neighborhood Particle Swarm Optimization Algorithm. In: GECCO '06 Seattle, WA, USA Copyright ACM

Arabnejad H, Barbosa JG (2014) List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans Parallel Distrib Syst 25:682–694

Beegom ASA, Rajasree MS (2014) A Particle Swarm Optimization based pareto optimal task scheduling in cloud computing. In: Tan Y, Shi Y, Coello CAC (eds) Advances in Swarm Intelligence: 5th international conference, ICSI 2014, Hefei, China, October 17–20, 2014, Proceedings, Part II. Springer International Publishing, Cham, pp 79–86. https://doi.org/10.1007/978-3-319-11897-0_10

Berriman GB, Deelman E, et al. (2004) Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In: SPIE conference on astronomical telescopes and instrumentation

Bilogrevic I, Jadliwala M, Kumar P, Walia SS, Hubaux JP, Aad I, Niemi V (2011) Meetings through the cloud: privacy-preserving scheduling on mobile devices. J Syst Softw 84:1910–1927

Calheiros RN, Ranjan R, Beloglazov A, DeRose CA, Buyya R (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exp 41(1):23–50

Canon LC, Jeannot E, Sakellariou R, Zheng W (2008) Comparative evaluation of the robustness of dag scheduling heuristics. In: Gorlatch S, Fragopoulou P, Priol P (eds) Grid computing: achievements and prospects. Springer, New York, pp 73–84

Chakraborty D, Guha D, Dutta B (2016) Multi-objective optimization problem under fuzzy rule constraints using particle swarm optimization. Soft Comput 20(6):2245–2259

Chen W, Deelman E (2012) WorkflowSim: a toolkit for simulating scientific workflows in distributed environments. In: IEEE 8th International Conference on E-Science (e-Science), pp 1–8

Gens F (2008) IT cloud services user survey, pt.2: top benefits & challenges, October 2008. URL: http://blogs.idc.com/ie/?p=210

Graves R, Jordan T, Callaghan S, Deelman E, Field E et al (2010) Cybershake: a physics-based seismic hazard model for southern California. Pure Appl Geophys 168(3–4):367–381

Hansen P, Mladenovic N, Perez JM (2010) Variable neighbourhood search: methods and applications. Ann Oper Res 175(1):67–407

Juve G, Chervenak A, Deelman E, Bharathi S, Mehta G, Vahi K (2013) Characterizing and profiling scientific workflows. Future Gener Comput Syst 29(3):682–92

Kalra M, Singh S (2015) A review of metaheuristic scheduling techniques in cloud computing. Egypt Inf J 16:275–295

Kennedy J (1998) The behavior of particles. In: Porto VW, Saravana N, Waagen D, Eiben AE Proceedings of the 7th conference on evolutionary programming, pp 581–589

Kennedy J, Eberhart RC, Shi Y (2001) Swarm intelligence. Morgan Kaufmann Publishers, San Francisco

Li K, Tang X, Veeravalli B, Li K (2015) Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. IEEE Trans Comput 64(1):191–204

Liu H, Abraham A (2007) An hybrid fuzzy variable neighborhood particle swarm optimization algorithm for solving quadratic assignment problems. J Univers Comput Sci 13(9):1309–1331

Liu H, Abraham A, Snasel V, McLoone S (2012) Swarm scheduling approaches for work-flow applications with security constraints in distributed data-intensive computing environments. Inf Sci 192:228–243

Liu G, Zeng Y, Li D, Chen Y (2015) Schedule length and reliability-oriented multi-objective scheduling for distributed computing. Soft Comput 19(6):1727–1737

Marinakis Y, Marinaki M (2013) Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem. Soft Comput 17(7):1159–1173

Mell P, Grance T (2011) The NIST definition of cloud computing. Special Publication, pp 800–145

Montage: an astronomical image mosaic engine (2015) http://montage.ipac.caltech.edu/

Pandey S, Wu L, Guru SM, Buyya R (2010) A Particle Swarm Optimization-based Heuristic for scheduling workflow applications in cloud computing environments. In: 24th IEEE international conference on advanced information networking and applications, pp 400–407. https://doi.org/10.1109/AINA.2010.31

Pegasus Workflow Management System (2015) https://pegasus.isi.edu/projects/pegasus/

Pritzker P (2013) NIST cloud computing standards roadmap Working Group NIST Cloud Computing Standards Roadmap. National Institute of Standards and Technology Special Publication, pp 500–291, 108 pages

Rodriguez MA, Buyya R (2014) Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Trans Cloud Comput 2(2):222–235

Schema of workflow in XML format (2015) http://pegasus.isi.edu/wms/docs/schemas/dax-3.4/dax-3.4.html

Selvi T, Govindarajan K (2014) CLOUDRB: a framework for scheduling and managing high-performance computing (HPC) applications in science cloud. Future Gener Comput Syst 34:47–65

Sih GC, Lee EA (1993) A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architecture. IEEE Trans Parallel Distrib Syst 4(2):175–187

Song S, Hwang K, Kwok Y-K (2006) Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling. IEEE Trans Comput 55(6):703–719

Sujana JAJ, Revathi T, Malarvizhili M (2015) Scheduling of scientific workflows in cloud with replication. Appl Math Sci 9(46):2273–2280

Tan WA, Sun Y, Li LX, Lu GZ, Wang T (2014) A trust service-oriented scheduling model for workflow applications in cloud computing. IEEE Syst J 8(3):868–878

Tang X, Li K, Liao G, Li R (2010) List scheduling with duplication for heterogeneous computing systems. J Parallel Distrib Comput 70:323–329

Tang X, Li K, Zeng Z, Veeravalli B (2011) A novel security-driven scheduling algorithm for precedence-constrained tasks in heterogeneous distributed systems. IEEE Trans Comput 60(7):1017–1029

Topcuoglu H, Hariri S, Wu MY (2002) Performance-effective and low complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274

USC Epigenome Center (2015) http://epigenome.usc.edu

Wang W, Zeng G, Tang D, Yao J (2012) Cloud-DLS: dynamic trusted scheduling for cloud computing. Expert Syst Appl 39:2321–2329

Workflow Generator (2015) http://vtcpc.isi.edu/pegasus/index.php/WorkflowGenerator

Wu Z, Ni Z, Gu L, Liu X (2010) A revised discrete particle swarm optimization for cloud workflow scheduling. In: Proceedings of IEEE international conference on computational intelligence and security (CIS), pp 184–188

Xie T, Qin X (2006) Scheduling security-critical real-time applications on clusters. IEEE Trans Comput 55(7):864–879

Xie T, Qin X (2008) Security-aware resource allocation for real-time parallel jobs on homogeneous and heterogeneous clusters. IEEE Trans Parallel Distrib Syst 19(5):682–697

Xue S, Wu W (2012) Scheduling workflow in cloud computing based on hybrid Particle Swarm Algorithm. TELKOMNIKA 10(7):1560–1566

Zeng L, Veeravalli B, Li X (2015) SABA: a security-aware and budget-aware workflow scheduling strategy in clouds. J Parallel Distrib Comput 75:141–151

Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: state-of-the-art and research challenges. J Internet Serv Appl 1:7–18

Zuo X, Zhang G, Tan W (2014) Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud. IEEE Trans Autom Sci Eng 11(2):564–573