Reliability Aware Cost Optimization for Memory Constrained Cloud Workflows

E Cao[1], Saira Musa[1], Jianning Zhang[1], Mingsong Chen[1](✉), Tongquan Wei[1], Xin Fu[2], and Meikang Qiu[3]

[1] Shanghai Key Lab of Trustworthy Computing, East China Normal University, Shanghai, China
mschen@sei.ecnu.edu.cn

[2] Department of Electrical and Computer Engineering, University of Houston, Houston, USA

[3] Department of Computer Science, Texas A&M University, Commerce, TX, USA

Abstract. Due to the increasing number of constituting jobs and input data size, the execution of modern complex workflow-based applications on cloud requires a large number of virtual machines (VMs), which makes the cost a great concern. Under the constraints of VM processing and storage capabilities and communication bandwidths between VMs, how to quickly figure out a cost-optimal resource provisioning and scheduling solution for a given cloud workflow is becoming a challenge. The things become even worse when taking the infrastructure-related failures with transient characteristics into account. To address this problem, this paper proposes a soft error aware VM selection and task scheduling approach that can achieve near-optimal the lowest possible cost. Under the reliability and completion time constraints by tenants, our approach can figure out a set of VMs with specific CPU and memory configurations and generate a cost-optimal schedule by allocating tasks to appropriate VMs. Comprehensive experimental results on well-known scientific workflow benchmarks show that compared with state-of-the-art methods, our approach can achieve up to 66% cost reduction while satisfying both reliability and completion time constraints.

Keywords: Workflow scheduling · Cost optimization · Reliability constraint · Soft error · Evolutionary algorithm

1 Introduction

Along with the increasing popularity of cloud services in a pay-as-you-go manner, more and more enterprises and communities adopt cloud platforms to deploy

Supported by the grants from National Key Research and Development Program of China (No. 2018YFB2101300), Natural Science Foundation of China (No. 61872147) and National Science Foundation (No. CCF-1900904, No. CCF-1619243, No. CCF-1537085 (CAREER)).

their commercial or scientific workflows to facilitate the distribution of data and computation-intensive applications [1]. However, as modern workflows grow rapidly in terms of the number of constituting jobs and input data size, their task allocation and scheduling complexity is skyrocketing. The scheduling of workflows requires large number of virtual machines (VMs), which makes the workflow execution cost a great concern to cloud service providers.

Since the resource allocation problem for cloud workflows is NP-complete [1], various heuristics have been proposed to find near-optimal schedules quickly. However, as more and more data center servers adopt CMOS-based processors, few of existing methods take transient faults (i.e., soft errors) [2,3] into account. Typically, a modern CMOS processor consists of billions of transistors where one or more transistors form one logic bit to hold the logic value 0 or 1. Unfortunately, various phenomena (e.g., high energy cosmic particles, cosmic rays) can result into the notorious soft error where the binary values held by transistors are changed by mistake, and the probability of incorrect results or system crashes during cloud workflow execution becomes increasingly higher.

As a reliable fault-tolerance mechanism, the checkpointing with rollback-recovery [4] has been widely adopted to improve the reliability of cloud workflow execution. By periodically saving VM execution states in some stable storage at specified checkpoints, the rollback-recovery can restore the system with the latest correct state to enable re-execution when an execution error is detected. However, the unpredictable overhead of checkpointing with rollback-recovery operations prolonged the execution time of workflow jobs due to re-execution which not only cause severe temporal violations [1], but also increase the overall execution cost.

To achieve increasing profit in the fierce cloud computing market, cloud service providers need to explore efficient cloud workflow schedules involving both resource provisioning (i.e., a set of VMs with specific processing and storage configurations) and allocation (i.e., assignment of workflow tasks to the VMs without violating VM memory constraints) to minimize the execution cost. In this paper, we propose a novel approach that can generate cost-optimal and soft error resilient schedules for workflow applications considering the overhead of both checkpointing with rollback-recovery and inter-VM communications. This paper makes following three major contributions:

– Under the constraints of VM memory size and overall workflow makespan, we formalize the cost-optimization problem of task scheduling for cloud workflows considering the overhead of both checkpointing with rollback-recovery and inter-VM communication.
– Based on two-segment group genetic algorithm (TSG-GA), we propose a soft error aware cost-optimized workflow scheduling approach that can quickly figure out a schedule with cost-optimal resource provisioning and task-to-VM allocation for a given workflow application.
– We evaluate our approach on well-known complex scientific benchmarks and show the effectiveness of the proposed approach.

The rest of this paper is organized as follows. Section 2 presents Section the related work. Section 3 formalizes the cost optimization problem for cloud workflow scheduling considering both resource and reliability constraints. Section 4 details our proposed approach, and Sect. 5 presents the corresponding experimental results on well-known benchmarks. Finally, Sect. 6 concludes the paper.

2 Related Work

Despite all the advantages of cloud computing, task scheduling in cloud workflows with minimum completion time and reduced cost while maintaining high reliability have become a major challenge, which have attracted great attention from researchers and industry. For instance, Topcuoglu et al. [5] proposed a Heterogeneous Earliest Finish Time (HEFT) algorithm which assigns the task with the highest priority to the VM, in order to achieve the earliest finish time. Panday et al. [6] presented a scheduling heuristic based on Particle Swarm Optimization (PSO) to minimize the total execution cost of application workflows on cloud computing environments while balancing the task load on the available resources. Since, the faster cloud services are normally more expensive, therefore, users face a time-cost trade-off in selecting services. As any delay in completion time can produce negative impacts on cost optimization of workflow scheduling. A general way to address this trade-off is to minimize monetary cost under a deadline constraint. Nonetheless, only a few approaches have been presented to address this issue in the literature [7–10], which solve the workflow scheduling problem on the Infrastructure as a Service (IaaS) platform. Aforementioned literatures can effectively minimize the makespan or cost but, none of them considered reliability during task scheduling.

In order to achieve the reliability, Wang et al. [11] proposed a LAGA (Look-Ahead Genetic Algorithm) to optimize the reliability and makespan of a workflow application simultaneously. An algorithm was designed and implemented in [12] by Wen et al. to solve the problem of deploying workflow applications over federated clouds while meeting the reliability, security and monetary requirements. Although the above work can guarantee the reliability but, they did not consider the soft error occurrences in cloud data centers. Wu et al. [3] proposed a soft error-aware energy-efficient task scheduling for workflow applications in DVFS-enabled cloud infrastructures under reliability and completion time constraints. However, the above work did not consider the cost optimization.

To our best knowledge, our work is the first attempt to minimize the execution cost of cloud workflows under makespan, reliability and memory constraints while considering soft errors in cloud data centers.

3 Scheduling Model and Problem Definition

In this section, we present VM model, workflow model and fault tolerance. Finally, the problem of cost optimization workflow scheduling in the cloud environment is defined.

3.1 Modeling of VM

IaaS cloud provider offers a set of VM configurations $C = \{C_0, C_1, ..., C_n\}$ to tenants by renting VMs on demand. The VM configuration C_i is characterized by a four-tuple $(vn, bw, ram, price)$, where $vn(C_i)$, $bw(C_i)$, $ram(C_i)$ and $price(C_i)$ denote the number of vCPUs, the network bandwidth, the memory and the rental price per unit time of C_i, respectively. A running VM with certain configuration is treated as an instance and customers can purchase unlimited number of VM instances according to their requirements. The set of VM instances is denoted by $S = \{S_0, S_1, ..., S_i\}$, where S_i is a VM instance with a certain configuration $\varphi(S_i)$ of C. We assume that all the tasks are parallelizable so that all vCPUs can be fully used and have same processing capacities. It is noteworthy that although cloud service providers have massive computing and memory resources, there are upper limits on the number of vCPU and the amount of memory for a single VM instance. In addition, the allocation of memory source of VM is usually discrete, i.e., $ram(S_i) = \alpha \cdot M$, where M is the unit of memory which depends on cloud service providers and α is an integer.

3.2 Modeling of Workflow

A workflow $W = (T, E)$ as shown in Fig. 1 with dependent tasks is represented as the Directed Acyclic Graph (DAG), where $T = \{t_0, t_1, ..., t_n\}$ represents the task set and E denotes the set of dependencies between tasks. For instance, $e_{uv} \in E$ indicates the dependency between task t_u and t_v, where t_u is the immediate predecessor of t_v, and t_v is the immediate successor of t_u. We use a four-tuple $(referload, mem, pred, succ)$ to represent a task, where $referload(t_u)$, $mem(t_u)$, $pred(t_u)$ and $succ(t_u)$ denote the reference workload, the maximum memory required for task execution on a VM instance, the immediate predecessors and the immediate successors of task t_u, respectively. If $pred(t_u) = \emptyset$, then t_u is an entry task and if $succ(t_u) = \emptyset$, then t_u is an exit task. This article allows single entry and exit task, this can be assured by adding a pseudo entry task and a pseudo exit task. We assume the reference workload is task execution time on a VM instance whose vCPU number equals 1.

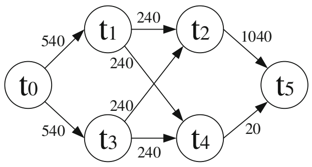

Fig. 1. A workflow example with 6 tasks

As shown in Fig. 1, each edge e_{uv} have weight $wt_{u,v}$, which represents the amount of data that needs to be transmitted from t_u to t_v. A task cannot start its execution until the input data has been received from all of its predecessors. If task t_u and t_v are assigned to VM S_i and S_j, the communication cost $comm(t_u, t_v)$ can be calculated as follows:

$$comm(t_u, t_v) = \begin{cases} 0 & if\ S_i = S_j \\ \frac{wt_{uv}}{bw_{i,j}} & if\ S_i \neq S_j \end{cases} \tag{1}$$

We consider that the communication bandwidth $bw_{i,j}$ between S_i and S_j is the lower bandwidth, i.e., $bw_{i,j} = min(bw(\varphi(S_i)), bw(\varphi(S_j)))$.

3.3 Modeling of Tasks with Fault Tolerance

To ensure the reliability of workflow execution in a cloud environment, we use an equidistant checkpointing technique [4], where the lengths of checkpoint intervals are same. The execution state of task is stored in a secure device [3,13], guaranteeing that the task can read the latest correct state to re-execute when a soft error occurs.

Suppose that task t_u is assigned to VM S_i, so the best case execution time of task t_u without any soft error can be formulated as

$$ET_{best}(t_u, S_i) = \frac{referload(t_u)}{vn(\varphi(S_i))} + N(t_u, S_i) \cdot O_i, \tag{2}$$

where $N(t_u, S_i)$ is the number of checkpoint of task t_u on VM S_i and O_i is the time overhead of checkpointing. Checkpoint interval length of task t_u assigned to VM S_i is formulated as

$$Seg(t_u, S_i) = \frac{referload(t_u)}{vn(\varphi(S_i))} \cdot \frac{1}{N(t_u, S_i) + 1}. \tag{3}$$

Let $Fmax$ denotes the maximum number of fault occurrences during task execution. Therefore, with F_{max} soft error occurrences, the worst case execution time of task t_u on VM S_i can be expressed as

$$ET_w(t_u, S_i) = \frac{referload(t_u)}{vn(\varphi(S_i))} + 2 \cdot N(t_u, S_i) \cdot O_i + Seg(t_u, S_i) \cdot F_{max}, \tag{4}$$

where $2 \cdot N(t_u, S_i)$ indicates the accumulative overhead of $Fmax$ checkpoint saving and retrieval operations, and $Seg(t_u, S_i) \cdot F_{max}$ represents the fault tolerance overhead.

In order to minimize the worst case execution time $ET_w(t_u, S_i)$, we use the optimal number of checkpoint $N_{opt}(t_u, S_i)$ [4], which can be calculated as

$$N_{opt}(t_u, S_i) = \sqrt{\frac{F_{max}}{O_i} \cdot \frac{referload(t_u)}{vn(\varphi(S_i))}} - 1. \tag{5}$$

We assume that the average arrival rate of soft error λ_i of the VM instance S_i is consistent with Poisson distribution [3]. Therefore, the probability of F soft error occurrences on VM S_i can be formulated as

$$Pr(t_u, S_i, F) = \frac{e^{-\lambda_i \cdot ET_w(t_u, S_i)} \cdot (\lambda_i \cdot ET_w(t_u, S_i))^F}{F!}. \tag{6}$$

Task reliability is defined as the probability that a task can be successfully executed in the presence of soft errors. The probability of successful recovery of F faults can be calculated as

$$Pr_{succeed}(F, S_i) = e^{-\lambda_i \cdot F \cdot Seg(t_u, S_i)}. \tag{7}$$

Hence, the reliability of task t_u on VM S_i can be calculated as

$$R(t_u, S_i) = \sum_{F=0}^{F_{max}} Pr(t_u, S_i, F) \cdot Pr_{succeed}(F, S_i). \tag{8}$$

3.4 Problem Definition

A binary tuple (*Task_VM, VM_VMC*) is used to represent a workflow scheduling scheme P, where *Task_VM* represents the mapping of tasks to VM instances, and *VM_VMC* represents the VM instances to VM configurations mapping. VM_VMC_i is used to represent the configurations of VM instance S_i, i.e., $\varphi(S_i) = VM\_VMC_i$. $Task\_VM_u$ indicates the VM instance to which task t_u is assigned. Let $ST_{i,u}$ and $FT_{i,u}$ denote the start time and finish time of task t_u on VM S_i, respectively. We get the start time of the task t_u on VM S_i as follows:

$$ST_{i,u} = \begin{cases} 0 & \text{if } pred(t_u) = \emptyset \\ \max\limits_{t_w \in previous(t_u)} \{ \max\limits_{t_v \in pred(t_u)} \{FT_{i,u} + \\ \qquad comm(t_u, t_v)\}, FT_{i,w}\} & \text{if } pred(t_u) \neq \emptyset \end{cases} \tag{9}$$

$FT_{i,w}$ is the finish time of the task t_w executed before task t_u on the same VM instance S_i and $previous(t_u)$ represents the tasks executed before t_u on S_i. Note that if there are multiple tasks that can be executed at the same time on the same VM instance, we select a task according to the order in which the tasks are scheduled to the VM instance. Therefore, $FT_{i,u}$ is formulated as

$$FT_{i,u} = ST_{i,u} + ET_w(t_u, S_i). \tag{10}$$

The makespan of workflow W can be obtained as,

$$makespan(W, P) = \max_{t_u \in T(W) \& S_i \in S(P)} \{FT_{i,u}\}, \tag{11}$$

where $T(W)$ is the task set of workflow W, $S(P)$ is the set of VM instances obtained by scheduling scheme P. In Sect. 3.3 we have obtained the reliability of a task, to calculate the workflow reliability we find the cumulative product of the reliability $R(W, P)$ of all the workflow tasks [3,14], such as

$$R(W, P) = \prod_{t_u \in T(W) \& S_i \in S(P)} R(t_u, S_i). \tag{12}$$

Let t_i denote the tasks assigned to the VM S_i, i.e., $t_i = \{t_u | Task\_VM_u = S_i\}$. The start time VM_{ST_i} and end time VM_{FT_i} of VM S_i can be formulated as

$$VM_{ST_i} = \min_{t_u \in t_i} \{ST_{i,u}\}, \tag{13}$$

$$VM_{FT_i} = \max_{t_u \in t_i} \{FT_{i,u}\}. \tag{14}$$

Finally, the cost of workflow W scheduling can be formulated as

$$Cost(P, W) = \sum_{S_i \in S(P)} price(VM\_VMC_i(P)) \cdot (VM_{ST_i} - VM_{FT_i}). \tag{15}$$

Considering a workflow W and a set of VM configurations C, we need to find a suitable scheduling scheme P to minimize the cost of workflow scheduling while satisfying the makespan constraint D_{goal}, memory constraint, and reliability constraint R_{goal}. Therefore, the problem to be solved in this paper can be formally defined as the minimization problem:

$$Minimize : Cost(W, P) \tag{16}$$

$$Subject\ to : R(W, P) \geq R_{goal}, \tag{17}$$

$$makespan(W, P) \leq D_{goal}, \tag{18}$$

$$mem(t_u) < ram(\varphi(S_i)), \quad t_u \in T(W)\ \&\ S_i \in S(P). \tag{19}$$

Equation (19) describes the memory constraints of workflow scheduling. The memory required for a task should not exceed the RAM of the VM instance on which the task it assigned. The problem presented in this paper is a typical combinatorial optimization problem. It is worth noting that although assigning tasks to the powerful VM instances can reduce the makespan of a workflow, but the idle time caused by data dependencies on the powerful VM instances will increase the cost and excessive memory resources can also impose costly penalties. Meanwhile, the reliability of task can also be influenced by the processing capability of VM instances, which makes the scheduling problem more complex.

4 Our Evolutionary Approach

Genetic algorithm (GA) has the characteristics of powerful global search ability, excellent concurrency and strong robustness, which is easy to combine with other methods and has become a universal optimization method [15]. It is widely used in workflow scheduling and cloud computing [8,16,17]. Since the existing GA is difficult to apply directly to the workflow scheduling problem, we explore the two-segment group genetic algorithm (TSG-GA) for cost optimized workflow scheduling with makespan, reliability and memory constraints. The algorithm consists of encoding, initial population generation, selection, crossover, mutation, elitism, fitness function and chromosome modification.

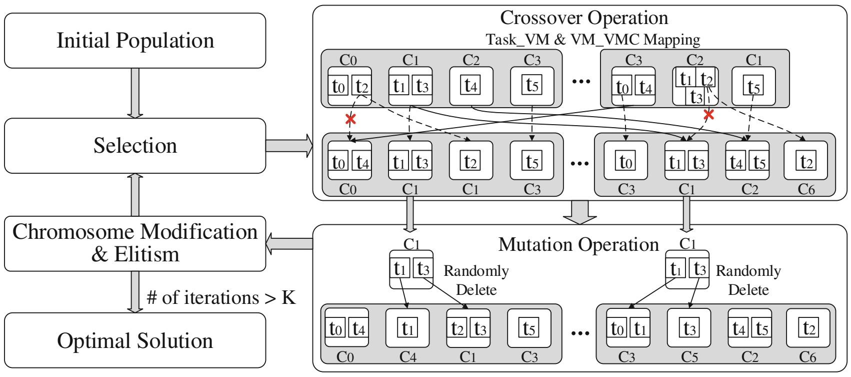

Fig. 2. The execution process of TSG-GA

The overall execution process of TSG-GA is shown in Fig. 2. We first randomly generate the initial population according to the target encoding, and select individuals with better fitness from initial population for crossover, mutation, and modification operations. Here the modification operation is used to satisfy the memory constraint for each chromosome. Then, we use the elitism strategy to preserve the best individual generated during the process of evolution. After a certain number of iterations (K), the final best individual (global optimal solution) is returned as the workflow scheduling solution.

4.1 Encoding

As discussed in Sect. 3.4, the encoding of our approach is a two segment integer encoding based on task grouping. For example, we use *ind* to represent a chromosome, i.e., an individual in the population, and *ind* consists of two segment: *ind.Task_VM* and *ind.VM_VMC*.

The encoding example is shown in Fig. 3. *Task_VM* is a group-based integer encoding, grouping tasks according to their corresponding VM instances. The encoding length of *Task_VM* is equal to the number of tasks. Gene index in *Task_VM* encoding represents the task, and the value represents the corresponding VM instance, For example, $Task\_VM(1) = 1$ indicates that task t_1 is assigned to VM instance S_1. *VM_VMC* is encoded as a variable length integer encoding, the length of which is the maximum index value of the VM instances in *Task_VM*.

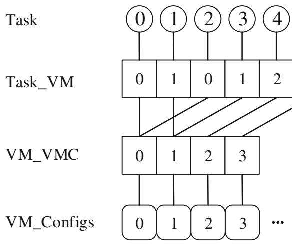

Fig. 3. An example of encoding

Similar to *Task_VM* encoding, $VM\_VMC(1) = 2$ indicates that the configuration of VM Instance S_1 is C_2. The two-segment integer encoding method designed in this paper is simple and intuitive.

4.2 Fitness Function

Fitness function is used to evaluate the fitness of solution in the evolution process. In this paper, we use fitness function to minimize the cost $Cost(W, P)$ as described in Eq. (15). In order to satisfy makespan and reliability constraints, penalty parameters γ and δ are introduced. How to satisfy memory constraints will be described in Sect. 4.4. The fitness of a chromosome deteriorates if it does not meet the makespan or reliability constraint. Let ξ be the set of constraints $\{D_{goal}, R_{goal}\}$, the *Fitness* function can be formulated as

$$Fitness(W, P) = \begin{cases} Cost(W, P), & \xi \ is\ satisfied \\ \gamma \cdot \delta \cdot Cost(W, P), & otherwise \end{cases} \tag{20}$$

γ and δ are the real numbers greater than 1 if D_{goal} and R_{goal} are not satisfied, respectively. The goal of TSG-GA is to minimize the $Fitness(W, P)$.

4.3 Crossover and Mutation

The designed crossover operator can ensure that the original task grouping and VM configurations information of chromosomes will not be lost, which uses the tasks in the same VM instance as the crossover unit to avoid the problem that the direct crossover for *Task_VM* may destroy the task grouping information.

Figure 4 shows the execution process of the crossover operator. Suppose $ind1$ and $ind2$ are parents. Firstly, an empty chromosome $L1$ is created as the offspring, and then a segment of genes of $ind1.Task\_VM$ are randomly selected. As shown in Fig. 4, $ind1.Task\_VM(3)$ to $ind1.Task\_VM(4)$ are selected. Genes in the same groups as the selected genes are copied into the offspring $L1$ with the associated VM configurations, i.e., $ind1.Task\_VM(1)$, $ind1.Task\_VM(3)$, $ind1.Task\_VM(4)$, $ind1.VM\_VMC(1)$, and $ind1.VM\_VMC(2)$. The length of the selected genes is limited to length of $ind1.VM\_VMC$ to avoid duplicating genes

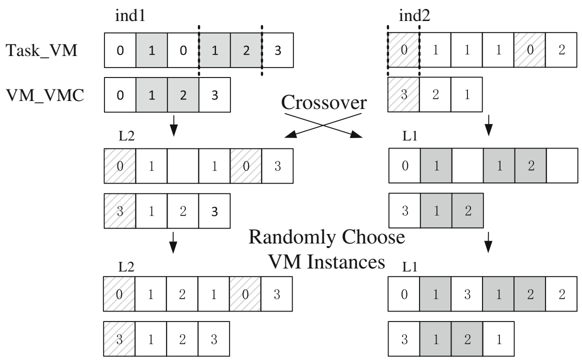

Fig. 4. An example of *Task_VM* and *VM_VMC* crossover

from *ind*1 too much which destroy the grouping information of *ind*2. Then copy the groups and corresponding VM configurations of *ind*2 that do not overlap with the previously copied genes of *ind*1 into *L*1. Here *ind*2.*Task_VM*(2) and *ind*2.*Task_VM*(5) cannot be copied because the corresponding VM instances overlap with the previously copied VM instances. At this time, there are some fragments of *L*1.*Task_VM* are not filled. We simply assign the tasks in these fragments to the existing or new VM instances and the crossover operation is completed. Same operation is performed by swapping the roles of *ind*1 and *ind*2, and we get the offspring individual *L*2.

For mutation operator, as shown in Fig. 2, it marks a chromosome as a mutated individual according to mutate rate, where we randomly delete one of the VM instance and assign the tasks of the VM instance to the existing or new VM instances. This paper argues that splitting and reorganizing the tasks in the VM instance with the most tasks is beneficial to jump out of the local optimum.

4.4 Chromosome Modification

The cloud workflow scheduling problem in this paper includes memory constraint that genetic algorithm does not have ability to deal with. In the process of evolution, if memory constraint is not satisfied, chromosome modification algorithm is called to satisfy the memory constraint.

As shown in Algorithm 1, for the chromosome that does not satisfy memory constraint, lines 3–15 search for an alternative VM configuration for each VM instance (traversing from $k = 0$) that does not satisfy memory constraint. Line 8 uses *GetAvailVmConfig* function to get an available VM configuration for the VM instance which need to increment the RAM from VM configurations C. The available configuration should satisfy memory constraint and have the same or similar processing capability with the original VM instance (the number of vCPUs is close to the original VM instance). Then line 9 uses the available VM configuration to replace the configuration of original VM instance, and line 12

Algorithm 1: Chromosome Modification

```
Input: i) $ind$, the chromosome to be modified;
       ii) $vmConfigs$, available VM configuration set;
Output: $new\_ind$, modified chromosome
1  ReformIndividual($ind, vmConfigs$) begin
2      $new\_ind = ind$;
3      for $k = 0$ to $ind.VM\_VMC.size()$ do
4          $v = ind.VM\_VMC[k]$;
5          $candidate = []$;
6          $tasks = GetTasks(ind, k)$;
7          if $FindMaxMem(tasks) > vmConfigs[v].ram$ then
8              for $vmConfig \in GetAvailVmConfig(vmConfigs[v], tasks)$ do
9                  $new\_ind.VM\_VMC[k] = vmConfig$;
10                 $candidate\_ind = new\_ind$;
11                 $candidate.add(candidate\_ind)$;
12             end
13             $new\_ind = FindElitis(candidate)$;
14         end
15     end
16     return $new\_ind$;
17 end
```

adds the modified chromosome to the candidate individual set. Line 13 selects the best individual from the candidate individual set. In the end, the entire chromosome is modified and the memory constraint is satisfied.

Table 1. Price of custom machine types provided by Google Cloud

Charge items	Cost
vCPU	\$0.033174/vCPU hour
Memory	\$0.004446/GB hour

5 Performance Evaluation

The effectiveness of the proposed method was evaluated through thorough experiments based on WorkflowSim [18] using well-known workflows [19] such as CyberShake, Inspiral and Epigenomics. Three workflows with two sets of tasks for each workflow were used in the experiment, which were generated by the toolkit Workflow-Generator [20] based on its default configurations. To reflect memory constraints, we randomly add memory attributes (1–8 GB) to the tasks of the generated workflows. Furthermore, HEFT and PSO algorithms with makespan and VM idle time minimization were compared with the proposed method. The HEFT algorithm assigns a task to the VM instance to achieve the earliest finish time according to task priority, yielding shortest workflow makespan. The PSO is an evolutionary computational algorithm which is widely used in the research of task scheduling for workflow application in the cloud. For comparison with our approach, the HEFT and PSO algorithm were modified to make sure an unlimited number of VM instances can be created with the same configuration. We also added memory constraints in both HEFT and PSO algorithm, and minimized the size of the memory of VM instance to reduce its execution cost. In addition, we calculated the reliability and cost of workflow scheduling according to Eqs. (12) and (15), respectively. All the experiments were performed on a desktop with 3.10 GHz Intel Core i5 CPU and 8 GB RAM.

5.1 Experimental Setting

The VM configuration and price were set by referring to custom machine types of Google Cloud. According to the characteristics of workflows and the VM configurations, a total of 40 VM configurations are selected with vCPU 1–4 and memory of 1–10 GB (1 GB, 2 GB, ..., 10 GB). Moreover, the network bandwidth between VM instances is 10 Mbps. The price of custom machine types provided by Google Cloud is shown in Table 1:

Note that although the price shown in Table 1 is in hour, Google Cloud can charge each VM instance in seconds. We assume that the soft error occurs independently in each VM instance and it is in accordance with the Poisson distribution. Supposing soft error occurrence rate λ_i of each VM instance is the same for a workflow, we have taken different soft error rates for different workflows, and the value of λ_i ranges from 10^{-6} to 10^{-3}. We set the maximum soft error tolerance number F_{max} to 1, and the overhead of each checkpoint O_i to 0.1 s. The size of the population of TSG-GA is 100, the number of generation, the crossover rate, and the mutation rate are 100, 0.8 and 0.1, respectively. For PSO, the size of the population is 100 and the number of generation is 100. While the learning factors c1 = 2, c2 = 2, and inertia weight is 0.9.

To evaluate the effectiveness of the proposed TSG-GA under makespan, reliability and memory constraints, we set makespan constraint to $D_{goal} = \theta \cdot MH$, where MH is the workflow scheduling makespan obtained by HEFT algorithm and θ is a constant real number. In the experiment, we let θ take different values ($1 \leq \theta \leq 2$), which means that our approach should not make the makespan of a workflow scheduling θ times longer than the makespan obtained by HEFT algorithm, meanwhile, we set the workflow scheduling reliability constraint to $R_{goal} = RH - \beta$, where RH is the workflow scheduling reliability obtained by HEFT algorithm and β is the reliability margin. The number of soft error tolerances F_{max} is fixed and assumed that the soft error rate of each VM instance is the same. Since the reliability of the task is directly related to the task execution time, it is more likely to encounter a soft error when the execution time becomes longer, which results in lower reliability. The HEFT algorithm can achieve approximate shortest makespan and highest task reliability as it finishes the task in the earliest time. Therefore, we used the reliability and makespan obtained by HEFT as references of R_{goal} and D_{goal} for each workflow.

5.2 Results and Analysis

In the experiment, the workflow scheduling generated by our approach always satisfies the memory constraint, because chromosome modification guarantees the memory constraints. We performed experiments five times on each workflow and finally took the average as the final result.

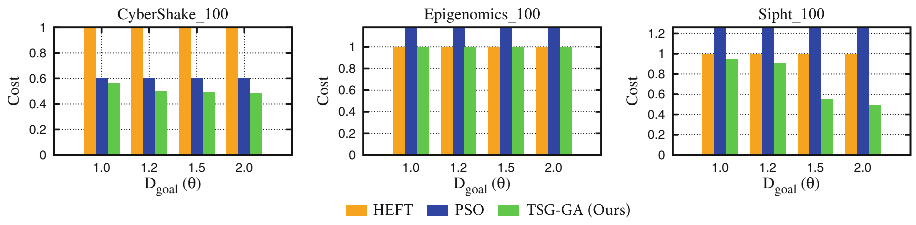

Fig. 5. Cost of large workflows with fixed reliability ($\beta = 0$) and varying makespan

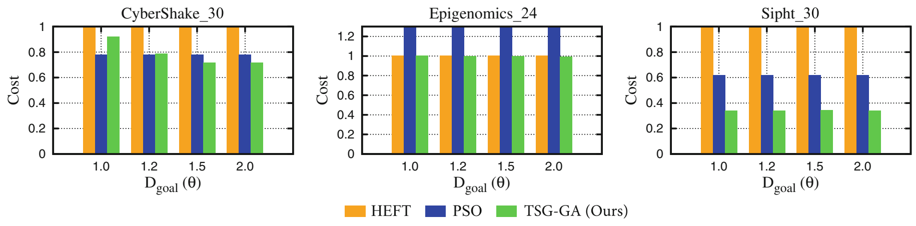

Fig. 6. Cost of small workflows with fixed reliability ($\beta = 0$) and varying makespan

Results of Workflows with Fixed Reliability Constraint. Firstly, we performed experiments using three workflows with two sets of task for each workflow under different makespan constraints, fixed reliability constraint, and memory constraint. Each set of tasks were defined as small and large workflows with 30 (or 24) and 100 tasks, respectively. We set θ to 1, 1.2, 1.5 and 2, and set β to 0 (i.e., RH). Figures 5 and 6 show the cost results of the proposed approach in comparison with HEFT and PSO on large workflows (i.e., CyberShake_100, Sipht_100 and Epigenomics_100) and small workflows (i.e., CyberShake_30, Sipht_30 and Epigenomics_24). Note that we did not set any constraint for PSO method, and just get results of HEFT and PSO once for one workflow in the case of $\theta = 1$. Our approach spent around 11.20 s on average to generate one schedule on large workflows and 1.41 s on small workflows.

To facilitate performance comparison, we took HEFT method as baseline, and took scheduling costs divided by the cost of HEFT as the final costs for each workflow. Our approach always satisfied the constraints both on large and small workflows. From Figs. 5 and 6, it can be observed that our approach outperforms the HEFT and PSO algorithms. For example, compared to the HEFT algorithm, PSO can achieve 40.0% cost reduction on the CyberShake_100 while our approach can achieve 44.1% cost reduction when $\theta = 1$. When $\theta = 2$, PSO can achieve 39.1% cost reduction on Sipht_30 while our approach can achieve 66.0% cost reduction. If we compare the worse performance cases, the proposed approach TSG-GA only performs worse than PSO on CyberShake_30 in the case of $\theta = 1$. However, PSO performance is even worse than HEFT on half of the workflows. This is mainly because it tends to converge prematurely and falls into local optimum due to the lack of diversity of the population in the search space. The processing capability of the VM instance using custom machine type provided by Google Cloud is linear to the price in the experiment. Therefore, cost reduction lies in reducing the idle time of VM instances, while HEFT just finishes tasks as quickly as possible. Complex dependencies between tasks make tasks to wait for execution on the VM instances, making it impossible to guarantee a minimum idle time. Our approach can create an appropriate number of VM instances with appropriate configurations and schedule tasks reasonably according to the dependencies, while reducing idle time of instances to reduce costs. We can see that the cost optimization achieved on Epigenomics is only 2%. This is because Epigenomics workflow transmits less data and its data dependency is relatively simple, so its main cost comes from the vCPU usage time but not idle time. It can be seen that as θ increases, our approach can achieve better results due to the vast search space in genetic algorithm.

Results of Workflows with Fixed Makespan Constraint. We conducted experiments with the three workflows discussed in the previous section under fixed makespan constraint, different reliability constraints and memory constraint. We set θ to 1 and β to 0.0000, 0.0001, 0.0002 and 0.0003. Figures 7 and 8 show the comparisons of workflow scheduling results obtained by our approach on those three workflows with HEFT and PSO algorithm.

Similarly, we used the HEFT as the benchmark. It is found that our approach always satisfies the constraints on these three workflows and outperforms the HEFT and PSO algorithms as depicted in Figs. 7 and 8. When $\beta = 0$, the reliability constraint of CyberShake_30 is 0.9987 and our approach can achieve 6.5% cost reduction compared to HEFT method. When $\beta = 0.0003$, the reliability constraint of CyberShake_30 is 0.9984, our approach can achieve 12% cost reduction compared to HEFT method. Reliability constraint make VM configurations to have strong processing capability to allow tasks to be completed as quickly as possible, which improves the reliability of the tasks. It can be seen from Figs. 7 and 8, when reliability constraint become loose, our approach can search for a better scheduling scheme.

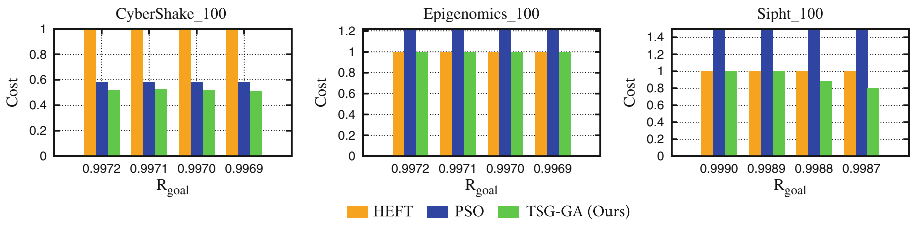

Fig. 7. Cost of large workflows with fixed makespan ($\theta = 1$) and varying reliability

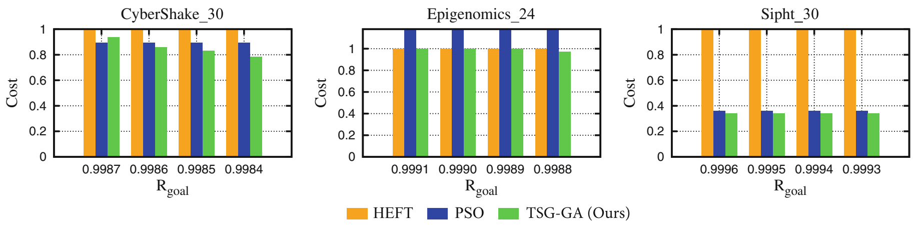

Fig. 8. Cost of small workflows with fixed makespan ($\theta = 1$) and varying reliability

6 Conclusions

Due to the increasing number of transistors on modern processors, the servers in data center is more susceptible to the notorious transient faults (i.e., soft errors). Although checkpointing with rollback-recovery mechanism is promising in tackling this problem to improve the reliability of cloud workflows, its overhead is too large to be neglected. The inevitable overhead will strongly affect the overall cost of workflow execution on cloud with a pay-as-you-go manner. To address this problem, this paper proposed a genetic algorithm based approach, known as TSG-GA, that can quickly figure out a cost-optimal schedule by considering both the overhead of checkpointing with rollback-recovery and the resource constraints (i.e., maximum number of vCPUs and available memory within a VM, network bandwidth) given by cloud workflow tenants. Comprehensive experimental results on well-known complex scientific benchmarks shows the effectiveness of our proposed approach.

References

1. Liu, X., et al.: The Design of Cloud Workflow Systems. Springer, New York (2012). https://doi.org/10.1007/978-1-4614-1933-4
2. Vishwanath, K.V., Nagappan, N.: Characterizing cloud computing hardware reliability. In: Proceedings of ACM Symposium on Cloud Computing (SoCC), pp. 193–204 (2010)
3. Wu, T., Gu, H., Zhou, J., Wei, T., Liu, X., Chen, M.: Soft error-aware energy-efficient task scheduling for workflow applications in DVFS-enabled cloud. J. Syst. Archit. **84**, 12–27 (2018)

4. Wei, T., Chen, X., Hu, S.: Reliability-driven energy-efficient task scheduling for multiprocessor real-time systems. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (TCAD) **30**(10), 1569–1573 (2011)
5. Topcuoglu, H., Hariri, S., Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. (TPDS) **13**(3), 260–274 (2002)
6. Pandey, S., Wu, L., Guru, S.M., Buyya, R.: A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: Proceedings of International Conference on Advanced Information Networking and Applications, pp. 400–407 (2010)
7. Qiu, M., Sha, E.H.M.: Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. ACM Trans. Des. Autom. Electron. Syst. (TODAES) **14**(2), 1–30 (2009)
8. Zhang, M., Li, H., Liu, L., Buyya, R.: An adaptive multi-objective evolutionary algorithm for constrained workflow scheduling in Clouds. Distrib. Parallel Databases **36**(2), 339–368 (2018)
9. Sahni, J., Vidyarthi, D.P.: A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Trans. Cloud Comput. **6**(1), 2–18 (2015)
10. Chen, M., Huang, S., Fu, X., Liu, X., He, J.: Statistical model checking-based evaluation and optimization for cloud workflow resource allocation. IEEE Trans. Cloud Comput. 1 (2016)
11. Wang, X., Yeo, C.S., Buyya, R., Su, J.: Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm. Future Gener. Comput. Syst. **27**(8), 1–18 (2011)
12. Wen, Z., Cala, J., Watson, P., Romanovsky, A.: Cost effective, reliable, and secure workflow deployment over federated clouds. In: Proceedings of IEEE International Conference on Cloud Computing, pp. 604–612 (2015)
13. Han, L., Canon, L., Casanova, H., Robert, Y., Vivien, F.: Checkpointing workflows for fail-stop errors. IEEE Trans. Comput. **67**(8), 1105–1120 (2018)
14. Zhang, L., Li, K., Li, C., Li, K.: Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. Inf. Sci. **379**, 241–256 (2016)
15. Whitley, D.: A genetic algorithm tutorial. Stat. Comput. **4**(2), 65–85 (1994)
16. Zhang, X., Wu, T., Chen, M., Wei, T., Zhou, J., Hu, S., Buyya, R.: Energy-aware virtual machine allocation for cloud with resource reservation. J. Syst. Softw. **147**, 147–161 (2019)
17. Gai, K., Qiu, M., Zhao, H.: Cost-aware multimedia data allocation for heterogeneous memory using genetic algorithm in cloud computing. IEEE Trans. Cloud Comput. 1 (2016)
18. Chen, W., Deelman, E.: WorkflowSim: a toolkit for simulating scientific workflows in distributed environments. In: Proceedings of International Conference on E-Science, pp. 1–8 (2012)
19. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M., Vahi, K.: Characterization of scientific workflows. In: Proceedings of International Workshop on Workflows in Support of Large-Scale Science, pp. 1–10 (2008)
20. Da Silva, R.F., Chen, W., Juve, G., Vahi, K., Deelman, E.: Community resources for enabling research in distributed scientific workflows. In: Proceedings of International Conference on e-Science, pp. 177–184 (2014)