CTaG: An Innovative Approach for Optimizing Recovery Time in Cloud Environment

Pham Phuoc Hung[1], Mohammad Aazam[2] and Eui-Nam Huh[1]
Department of Computer Engineering, Kyung Hee University
Yongin-si, South Korea
[e-mail: {hungpham, johnhuh}@khu.ac.kr[1], aazam@ieee.org[2]]
*Corresponding author: Eui-Nam Huh

Received August 4, 2014; revised November 25, 2014; revised January 29, 2015; accepted March 12, 2015; published April 30, 2015

Abstract

Traditional infrastructure has been superseded by cloud computing, due to its cost-effective and ubiquitous computing model. Cloud computing not only brings multitude of opportunities, but it also bears some challenges. One of the key challenges it faces is recovery of computing nodes, when an Information Technology (IT) failure occurs. Since cloud computing mainly depends upon its nodes, physical servers, that makes it very crucial to recover a failed node in time and seamlessly, so that the customer gets an expected level of service. Work has already been done in this regard, but it has still proved to be trivial. In this study, we present a Cost-Time aware Genetic scheduling algorithm, referred to as CTaG, not only to globally optimize the performance of the cloud system, but also perform recovery of failed nodes efficiently. While modeling our work, we have particularly taken into account the factors of network bandwidth and customer's monetary cost. We have implemented our algorithm and justify it through extensive simulations and comparison with similar existing studies. The results show performance gain of our work over the others, in some particular scenarios.

Keywords: Cloud computing, task scheduling, big data, recovery time, offloading

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning (2010-0020725). The corresponding author is professor Eui-Nam Huh.

http://dx.doi.org/10.3837/tiis.2015.04.001 **ISSN : 1976-7277**

1. Introduction

Cloud computing (CC) has emerged as a promising as well as inevitable technology [9], deemed as an ideal solution to handle the requirements of today's increasing demands, more specifically, in terms of availability, total cost of ownership (TCO), and time to value (TTV) [20, 21]. Number of enterprises, specially large-scale businesses, have been adopting CC, resulting in popularity of cloud-based services. For highly computationally complex business processes, CC is becoming a necessity.

To achieve high performance, reliability, and availability of cloud services, thousands of servers coordinate with each other for efficient task scheduling and in the end, user satisfaction [4]. While achieving this, at times, computing nodes in a datacenter may fail, causing degradation of performance. Although, it is not very common for a server to face such situation, but across all devices in the datacenter, the effect can be very diverse [5]. Failure of nodes can not only adversely affect the performance, but also, reliability of services being provided [2]. Financial damage may be up to millions of dollars, if the system is not recovered in time [1]. On the other hand, customers may lose their trust, resulting in huge financial loss for the service provider. Service providers would always like their customers to be loyal with them and this depends mainly upon the quality and integrity of services.

Other than the major facilities, like affordable cost, quick deployment, scalability, etc., cloud computing comes with, resiliency is considered to be a vital feature of it. It is very important to bring back the infrastructure into its original working state as soon as possible. Availability of the system and quick recovery from a failed state determines the level of trust of customers. That is why, in case of any failure, the system is ought to be brought back to normal seamlessly. In respect of recovery from failure, research activities have been focusing on different methodologies to minimize recovery time in high performance computing. Methodologies are mainly categorized into two: stochastic optimization [22, 23] and heuristic-based optimization [8, 10]. Heuristic approach solves the problem efficiently, but compromising on the performance. On the other hand, stochastic approach has a comparatively better performance, keeping in view the reliability of solution [6]. Both these approaches have their pros and cons, but overall, none of these approaches are preferable, since they do not take into account the network condition and monetary cost customers have to pay. In our work, we have tried our best to keep the advantages of both of these approaches and overcome their limitations. We present an efficient and cost-effective method, based on genetic algorithm, called CTaG, for task scheduling to globally optimize cloud-based system performance and reduce recovery time for physical server falures in cloud computing. Besides overall efficiency and reliability, we have considered network condition and monetary cost while designing our system. In designing our system, we foresee that the overall processing time of the cloud system can be significantly reduced and user experience and quality of service can be improved.

We have evaluated our methodology through extensive simulations to justify its efficiency and performance. Comparison with the existing approaches proves the edge our approach has over the others. Our approach minimized both recovery time of failed nodes and monetary cost of customers, with a lower overhead, as compared to other approaches. As a result, a better efficiency per cost (EC) ratio is achieved. We believe that our approach can be very useful for the discussed situations, especially for small and medium sized business models, working on cloud based services.

The organization of this paper is in such a way that section 2 discusses already done work in this regard. Section 3 presents motivating scenarios and importance of our approach. We present our methodology in section 4. Implementation and evaluation are presented in section 5. Section 6 concludes our paper.

2. Related Work

There is a survey of 584 individuals in U.S. organizations who have responsibility for data center operations. Among those organizations which experienced a loss of primary utility power, 91 percent reported unplanned outages [13]. Unplanned data center outages present a difficult and costly challenge for organizations. Therefore, to deliver better services to customers, cloud-based infrastructures are expected to come up with failure-tolerance and resiliency, which implies that those systems can be quickly recovered from failure because of the outages and restored to normal working state, especially for distributed systems. In those heterogeneous systems, scheduling algorithms play a key role in obtaining high performance. There have been numerous studies which attempt to solve task scheduling problems, where the sequence of the tasks (workflow) is popularly presented by a directed acyclic graph (DAG), as shown in **Fig. 4**. In [7], authors propose task scheduling approaches for assigning processors to task graph templates prepared in advance. The limitation of these methods is that they do not consider the network contention. They assume that the network has a perfect communication, and there is no limited bandwidth. Sinnen et al. [8] present an efficient task scheduling method based on network contention. This proposal has two steps: firstly, each task is set a priority based on the upward value of this task in the workflow; secondly, choose the most appropiate processor that minimizes the completion time of this priority based task. Anyhow, the method does not look attentively at monetary cost paid by cloud service customers (CSCs), for use of the cloud resources.

In heterogeneous CC environment, despite numerous efforts, task scheduling remains one of the most challenging problems [6]. Good performance of workflow and satisfaction of a budget constraint are typical criteria for multitask scheduling. Authors in [28] introduce a cost-efficient approach to select the most appropriate system (private or public cloud) to execute the workflow. Selection also depends on the possibility of meeting the deadline of each workflow as well as the cost savings**.** L.Zeng et al. in [3] propose budget conscious scheduling Comparative Advantage (CA) function to satisfy the strict budget constraint. In its first phase, each of tasks is assigned to the best VM whose CA1 value is maximum to get a worthy tradeoff between cost saving and efficiency. In the second phase, budget constraint is used in the CA2 function to evaluate the improvement of task reassignment to another VM on cost and performance. Notwithstanding CA is hard to be applied to the large-scale workflows. In the meantime, J. Li et al. in [11] present a scheduling algorithm CCSH to schedule the application of large graph processing with considering both cost and schedule length. The input cost-concious factor of this method is the cloud cost and used as a weight to calculate the earliest finish time EFT of each task. However, global optimality and tradeoff between cost and schedule length are not properly addressed in these approaches.

Recently, some GAs have been developed for solving the globlal optimal problem in task scheduling. The authors in [22**]** propose approaches using genetic processes to find multiple solutions faster and ensure global optimal usage of the processing system. Sachi et al. [23] develope a method based on genetic algorithm (GA) to find an optimal scheduling, which shows to be efficient to discover optimal solutions more than Heterogeneous Earliest Finish Time (HEFT) with same length of problem size, focusing on the quality of solution and effect

of muation probability on the performance of GA. Mohammad A. et al. [26] propose a GA based method which not only considers the deadline time in sorting the tasks in the first population, but also includes the maximum computational time of individuals in the population to define the priority level of these tasks. Hadis H. et al. [25] exhibit a node duplication genetic algorithm based technique to avoid some unnecessarily replicated node without any negative affect on its schedule length for minimizing inter processor traffic communication. Yujia et al. [27] show a new scheduling algorithm according to a fitness adaptive algorithm-job spanning time adaptive genetic algorithm, so as to enhance the overall performance of the cloud computing environment. Nonetheless, in these proposals, monetory cost and failure of computing devices are not considered.

Zhang et al. in [19], and Y.Gu et al. in [14] introduce a solution to recover from failure according to check-pointing in stream processing system. As such, when the system has failure, it finds the closest ancestor node which is not impacted by failure to resume the results saved in that node. The authors in [10] come up with a scheme to reduce the recovery time in case of failure, but they do not contemplate the cost paid by users. Similar to our approach, Huynh T.T.B in [24] proposes a genetic based method that miminizes the processing against a failure in a network system. But monetory cost is not considered in this solution.

For ease of understanding, we present an overview of common scheduling approaches along with ours in **Table 1**.

Table 1. Overview of the existing scheduling approaches and ours

Algorithm	Target System	Minimum Schedule Length	Minimum Cost	Trade off	Minimum Recovery Time	Global Optimality
H. Topcuoglu et al. [16]	Heterogeneous	Yes	No	No	No	No
Oliver Sinnen et al. [8]	Cloud	Yes	No	No	No	No
Gonzalo et al. [15]	Cloud	Yes	No	No	No	No
Louis C. et al. [17]	Heterogeneous	Yes	No	No	No	No
L.Zeng et al. [3]	Cloud	Yes	Yes	Yes	No	No
Ruben et al. [28]	Cloud	Yes	Yes	Yes	No	No
J. Li et al. [11]	Heterogeneous	Yes	Yes	Yes	No	No
Shohei G. et al. [10]	Multicore processor	Yes	No	No	Yes	No
Zhang et al. [19]	Stream processing	Yes	No	No	Yes	No
Tashniba K. et al. [22]	Multicore processor	Yes	No	No	No	Yes
Sachi et al. [23]	Heterogeneous	Yes	No	No	No	Yes
Hadis H. et al. [25]	Multicore processor	Yes	No	No	No	Yes
Mohammad A. et al. [26]	Multicore processor	Yes	No	No	No	Yes
Yujia et al. [27]	Cloud	Yes	No	No	No	Yes
Huynh T.T.B [24]	Homogeneous	Yes	No	No	Yes	Yes
Our approach	Cloud	Yes	Yes	Yes	Yes	Yes

As shown in **Table 1**, a desirable scheduling approach should consider all of the four factors including globally optimal execution time, network contention, cost of using cloud services and recovery time in the case of physical failure of servers. That can satisfy QoS as well as increase reliability and the reputation of the cloud provider. In this paper, we aim to provide a scheduling scheme which takes these factors into account and develop an algorithm to reduce the recovery time in the case of failure of a physical server in a data center. The main contribution of this work is the development of a genetic algorithm to determine the global optimal schedule.

3. Motivation Scenario

In this section, we discuss the importance and applicability of our work, by presenting relevant scenario. **Fig. 1** illustrates cloud datacenter, having several physical servers (PSs**),** represented by bounded rectangles. Each of which may not have enough capacity to host all processors (rectangles), functioning as virtual machines (VMs), to fulfill user requirements in solving some computationally intensive tasks. These processors can communicate with each other via a network interface (gray circle). In one physical server, even if there is enough capacity, it is generally not recommended to have all the processors hosted in the same PS in order to avoid complete service unavailability in the case of power outage. Therefore, it is suggested, that tasks should be divided into smaller subtasks which are executed by processors (or VMs), located on different physical servers at the cloud provider side. Each of those processors can execute tasks independently.

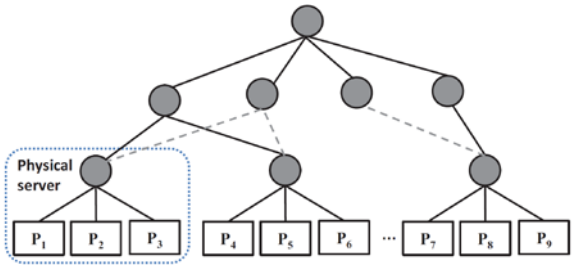


Fig. 1. A typical data center

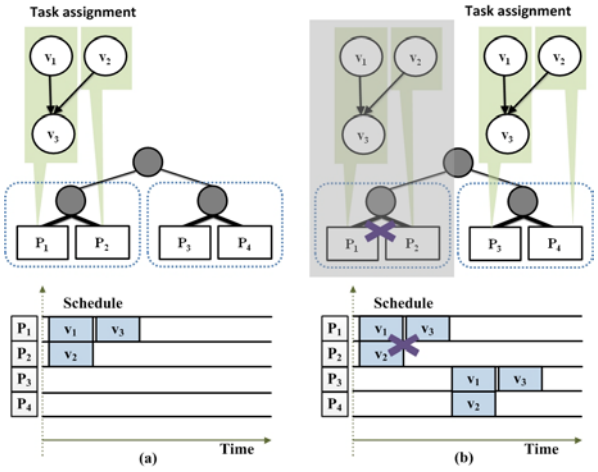


Fig. 2. An example of existing task scheduling (a) and recovery solution (b)

One network interface is shared among the scheduled processors. Due to the high stability of the network among processors in the same local PS compared with the Internet, the average data transfer rate of internal communication is always higher than that of external one. In this regard, existing schedulers try to utilize this high speed link, resulting in many dependent tasks being assigned to processors hosted in the same PS. When a failure occurs, dependent tasks tend to be destroyed at a time, requiring a more time consuming process for recovery.

In **Fig. 2a** an example is illustrated, regarding task scheduling created by an existing approach which does not consider the possibility of the system failure. In this scheduler, task v_1, v_2, and v_3 are assigned to processors P_1 and P_2 that are located in the same PS in order to utilize the high speed communication between processors. After task v_1 and v_2 are executed in parallel, task v_3 can be processed on processor P_1. However, if the PS on which P_1 and P_2 reside fails while task v_3 is being executed, the results of all task nodes, are lost. If we want to recover the results, all the tasks v_1, v_2, and v_3 have to be performed again on another physical server. This may double the execution time (**Fig. 2b**).

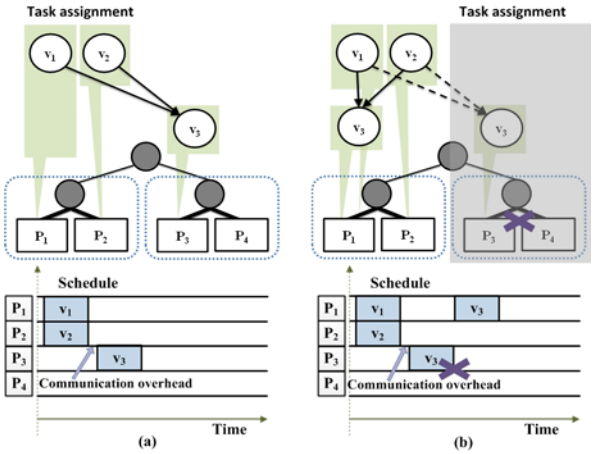


Fig. 3. An example of our scheduling proposal (a) and its recovery solution (b)

Let us now discuss another situation where all tasks in the critical path are assigned to processors in the same physical server. In this regard, the worst case scenario is that a failure of a physical server occurs during the time when the last tasks are being executed and the entire set of tasks have to be executed again. Here, the critical path refers to the longest execution path between the initial task (entry task) and the final task (end task) of the workflow, and this greatly influences the completion time of the DAG [10]. A one-hour delay in one of these tasks may immediately imply a one-hour delay on the workflow. Therefore, it is recommended that the tasks in the critical path should be distributed among processors residing in different PSs in order to improve the recovery time of a failure. As a trade-off, however, the communication overhead increases if no failure happens due to the extra communication between processors (**Fig. 3a**).

In case, if a failure occurs in our proposed system, only the result of task v_3 is lost since the results of task v_1 and v_2 are still stored in the other PS. Hence, only task v_3 needs executing again on the non-failure PS. This can reduce recovery time (**Fig. 3b**). However, it will become counter-effective, if tasks are distributed to multiple processors located on a high number of different PSs, the overhead will be significantly increased. Keeping this in mind, our paper tries to reduce that overhead as much as possible to address the following issues: Scheduling

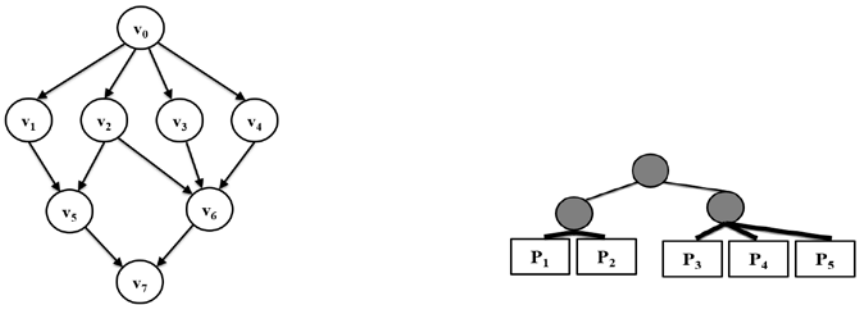
tasks to globbally minimize the execution time in regular operation and, in case of a failure, to reduce recovery time while considering the network contention and the cost paid by CSCs.

4. Problem Formulation and Solution

Task scheduling [18] on a target system having a network topology is defined as the problem of allocating the tasks of an application to a set of processors that have diverse processing capabilities in order to minimize total execution time. Thus, the input of task scheduling includes a task graph and a process graph. The output is a schedule representing the assignment of a processor to each task node.

4.1 Problem Formulation

In this section, we first define the terms used and then formulate the problem. Eventually, a genetic method for task scheduling is presented to solve the above mentioned problems.



(a) (b)

Fig. 4. A sample DAG and a processor graph

Definition 1. A task graph (e.g. as in **Fig. 4a**) is represented by a Directed Acyclic Graph (DAG), $G=(V, E, w, c)$, where the set of vertices $V=\{v_1,v_2,...,v_k\}$ represents the set of parallel subtasks, and the directed edge $e_{ij}=(v_i,v_j)\in E$ describes the communication between subtasks v_i and v_j, $w(v_i)$ associated with task $v_i \in V$ represents its computation time and $c(e_{ij})$ represents the communication time between task v_i and task v_j with corresponding transferred data $d(e_{ij})$. We presume that a task v_i without any predecessors, $prec(v_i)=0$, is an entry task v_{entry}, and a task that does not have any successors, $succ(v_i)=0$, is an end task v_{end}. The task consists of workload wl_i, which delimits the amount of work processed with the computing resources. Besides, it also contains a set of preceding subtasks $prec(v_i)$ and a set of successive subtasks $succ(v_i)$ of task v_i, $t_s(v_i,P_j)$ denotes Start Time and $w(v_i, P_j)$ refers to the Execution Time of task $v_i \in V$ on processor P_j. Hence, the finish time of that task is given by $t_f(v_i, P_j)= t_s(v_i, P_j)+ w(v_i, P_j)$.

Suppose that the following conditions are satisfied:

Condition 1. A task cannot begin its execution until all of its inputs have been gathered sufficiently. Each task appears only once in the schedule.

Condition 2. The ready time $t_{ready}(v_i, P_j)$ is the time that processor P_j completes its last assigned task and be ready to execute task v_i. Therefore,

$$t_{ready}(v_i, P_j) = \max\{\max_{v_y \in exec(j)}(t_f(v_y, P_j)), \max_{e_{zi}\in E, v_z \in prec(v_i)}(t_f(e_{zi}))\}, \quad (1)$$

where *exec(j)* is a set of tasks executed at processor P_j, $t_f(e_{zi}) = t_f(v_z)+c(e_{zi})$ and *prec*(v_i) is a set of preceding tasks of v_i.

Condition 3. Let $[t_A,t_B]\in[0,\infty]$ be an idle time interval on processor P_j in which no task is executed. A free task $v_i \in V$ can be scheduled on processor P_j within $[t_A,t_B]$ if

$$\max\{t_A, t_{ready}(v_i, P_j)\} + w(v_i, P_j) \le t_B. \quad (2)$$

Definition 2. A processor graph $TG=(N,D)$ demonstrated in **Fig. 4b** is a graph that describes the topology of a network between vertices (processors) that are cloud virtual machines (VMs). In this model, N is the finite set of vertices, and a directed edge $d_{ij} \in D$ denotes a directed link from vertex P_i to vertex P_j with $P_i,P_j\in N$. Each processor P_i controls the processing rate p_i and bandwidth bw_i on the link connecting it to other processors.

Problem Definition

In this section, we make some assumptions about the proposed method. Only output data from each task node is stored as a saved state. The state containing the data will be transferred to the processors that execute the child nodes of the completed task nodes. When a PS failure occurs, the saved state can be found in the ancestor task node. We suppose only a single stop failure of a physical server. When it occurs, all processors in the server simultaneously stop their execution. Let $tt(S_1+S_2, v_i)$ be the total time of an entire schedule S_1+S_2 with a failure happening at this task v_i, S_1 be a schedule before the failure, S_2 be a schedule after the failure. Our purpose is trying to find and reallocate the last tasks in the critical path to multiple machines to minimize the entire schedule time and avoid the worst case. That means finding the schedule $(S_1 + S_2)$ to $\text{minimize} \max_{v_i \in V} tt(S_1 + S_2, v_i))$.

4.2 Proposed approach

Given a task graph $G = (V, E, w, c)$ and a processor graph with network topology $TG = (N,D)$, our method uses a genetic algorithm to choose the most appropriate schedule to execute the tasks. Among the various guided random techniques, genetic algorithms (GAs) are the most widely used for the task scheduling problem [12].

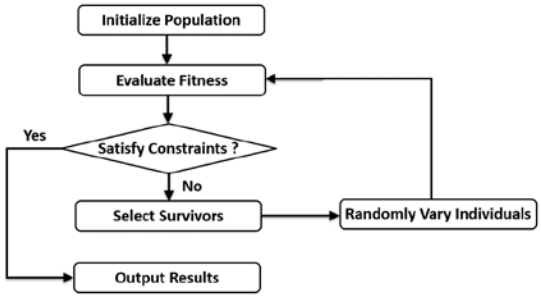


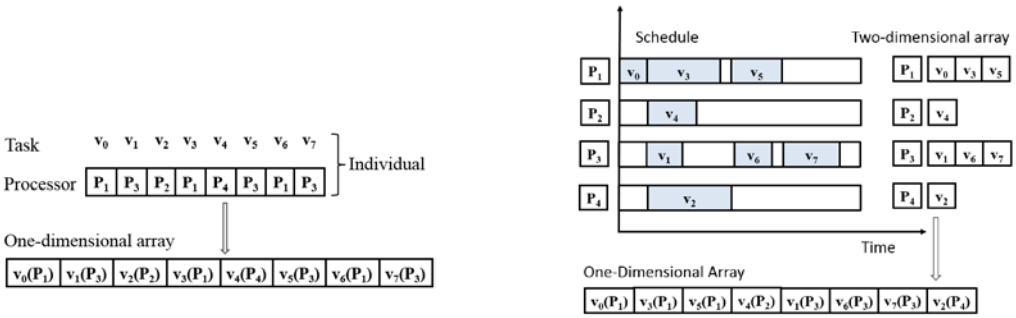
Fig. 5. Genetic algorithm

A genetic algorithm [7], illustrated in **Fig. 5**, is inspired by natural evolution. It is a robust search technique that allows a global high-quality solution to be derived from a large search space in polynomial time. This is in contrast to other algorithms that find only local optimal results. GA combines the best solutions from past searches with exploration of new regions of the solution space. In this algorithm, a feasible solution is represented using an individual (chromosome), that is a set of task assignments. The algorithm keeps a population of these

randomly generated individuals that evolves over generations. The quality of an individual in the population can be characterized by a fitness function whose value specifies the fitness of an individual compared to others in the population. Higher fitness level present better solutions. Based on fitness, parents are selected to produce offspring for a new generation. The fitter individual has a better chance to reproduce. A new generation has the same number of individuals as the previous generation, which dies off once it is replaced with the new generation. By applying genetic operators, namely selection, crossover and mutation to a population of chromosomes, the quality of the chromosomes can be improved. As a result, a new population of individuals is produced. If well designed, this new population will converge to optimal solution. The following section describes in detail each operator of the genetic algorithm:

- *Representation*

Here, we choose each individual (as shown in **Fig. 6a**) in the population to illustrate a feasible solution to the problem and contain an array of task assignments. Each of the assignments consists of a task and a corresponding assigned processor. The time frames of each task in each individual, such as Earliest Start Time, Earliest Finish Time, and so on, can be changed to adjust those of its successive tasks. These changes can lead to a very complex state during the genetic algorithm. Hence, our solution is to ignore the time frame while conducting genetic manipulation and assign a time slot to each assignment in order to obtain a feasible schedule later.



(a) (b)

Fig. 6. A one-dimensional array and two-dimensional array

A one-dimensional array (**Fig. 6a**) may not be suitable for representing the workflow because it only defines which processor is allocated to each task and cannot show the order of task assignments on each processor. However, the execution order is very important since it significantly impacts the workflow execution [6]. We use a two-dimensional array to represent a schedule, as demonstrated in **Fig. 6b**. In this two-dimensional array, the order of tasks on each processor is shown. During genetic manipulation, the two-dimensional array is transformed into a one-dimensional array.

- *Establishing the Initial Population*

The initial population is a set of individuals generated through a random heuristic. Each individual consists of pairs of tasks and processors on which the tasks are allocated.

- *Constructing a Fitness Function*

A fitness function can be used to characterize the quality of each individual in a population based on its optimization value. According to fitness value, parents are selected to generate new offspring. Since the purpose of our method is to minimize the schedule length while considering the network contention and the cost for cloud users, the fitness function has to rely

on *EFT* and cloud costs paid by CSCs. The following section illustrates establishment of *EFT* and the cost of task v_i on a processor from its start time as well as the ingredient costs.

The start time of a task is defined when the last preceding task is completed. Thence, to determine that start time, the earliest idle interval $[t_A,t_B]$ on processor P_j has to be searched and found to satisfy *condition 2 and condition 3*. As a result, the start time t_s of task v_i on processor P_j is set as:

$$t_s(v_i, P_j) = \begin{cases} \max(t_A, t_{ready}(v_i, P_j)), & \text{if } v_i \neq v_{entry} \\ 0, & \text{otherwise} \end{cases}, \tag{3}$$

Thus, the Earliest Start Time (*EST*) of a task v_i executed on a processors P_j is computed as follows:

$$EST(v_i, P_j) = \max_{v_z \in prec(v_i), P_k \in N} (t_f(v_z, P_k)) + \max_{P_k \in N} (c(e_i^{kj})), \tag{4}$$

where $c(e_i^{kj})$ is the communication time between processors P_k and P_j to execute task v_i and is defined as:

$$c(e_i^{kj}) = (di_i^k + \sum_{v_z \in (prec(v_i) \cap exec(k))} do_{zi}) * (\frac{1}{bw_j} + \frac{1}{bw_k}) \tag{5}$$

Here, di_i^k is the amount of input data stored at processor P_k and used for executing task v_i and do_{zi} is amount of outgoing data executed from P_k then transferred to P_j. Therefore, Earliest Finish Time (*EFT*) of the task v_i is calculated as:

$$EFT(v_i, P_j) = w(v_i, P_j) + EST(v_i, P_j). \tag{6}$$

In addition, the algorithm also considers the cost paid by cloud customers for using cloud resources to execute the tasks. The cost $C(v_i,P_j)$ for task v_i executed at a VM P_j is defined by:

$$C(v_i, P_j) = C_{proc}^{(v_i,P_j)} + C_{wait}^{(v_i,P_j)} + C_{comm}^{(v_i,P_j)} + C_{disc}^{(v_i,P_j)} + C_{str}^{(v_i,P_j)} + C_{mem}^{(v_i,P_j)} \tag{7}$$

In equation (7), each cost is calculated as follows:
Cost of processing is expressed as:

$$C_{proc}^{(v_i,P_j)} = c_1 * wl_i / p_j, \tag{8}$$

where c_1 is the processing cost per time unit of workflow execution on processor P_j with processing rate p_j.

Let t_{min} be the finish time of the task which is completed first out of the parallel tasks and there is no available task after this one, c_2 be the waiting cost per time unit and t_i be the finish time of the task v_i. Then the cost of waiting time is as:

$$C_{wait}^{(v_i,P_j)} = c_2 * (t_i - t_{\min}). \tag{9}$$

Suppose that the amount of money per time unit for transferring outgoing data from processor P_j is c_3, then the cost of communication time is defined as follows:

$$C_{comm}^{(v_i,P_j)} = c_3 * (di_i^j + \sum_{v_z \in (prec(v_i) \cap exec(j))} do_{zi}) / bw_j. \quad (10)$$

We assume that the distribution of disconnection events between a cloud and clients is a Poisson distribution with parameter μ_T, which represents the stability of the network. The expected number of arrivals over an interval of length τ is $E[N_T]= \mu_T*\tau$. Let L be a random variable for the length of an offline event, μ_L be the mean length and c_4 be the disconnection cost per unit time. Therefore, the expected duration of a disconnection event, which can affect the processing time of task v_i, is $\mu_T*\tau*\mu_L$. Hence, the cost of disconnection can be derived as:

$$C_{disc}^{(v_i,P_j)} = c_4 * (\mu_T * \tau * \mu_L). \quad (11)$$

Let c_5 be the storage cost per data unit and st_i be the storage size of task v_i on processor P_j . Then the storage cost of task v_i on processor P_j is calculated as:

$$C_{str}^{(v_i,P_j)} = c_5 * st_i. \quad (12)$$

Further, we compute the cost of using the memory of processor P_j for task v_i as follows:

$$C_{mem}^{(v_i,P_j)} = c_6 * s_{mem}, \quad (13)$$

where s_{mem} is the size of the memory used and c_6 is the memory cost per data unit.

Using this cost, we can calculate a fitness function that computes the tradeoff $U(v_i,P_j)$ between the cost and EFT as:

$$U(v_i,P_j) = Min \sum_{v_i \in E, P_k \in N} (\frac{cost(v_i,P_j)}{Max[cost(v_i,P_k)]} * \frac{EFT(v_i,P_j)}{Max[EFT(v_i,P_k)]}). \quad (14)$$

By considering the above fitness function that combines $cost(v_i,P_j)$ and $EFT(v_i,P_j)$, we can determine which individual in a population is the most appropriate to satisfy the function. This indicates that its combination of $cost(v_i,P_j)$ and $EFT(v_i,P_j)$ should demonstrate the minimum value of the tradeoff $U(v_i,P_j)$.

- *Genetic Operators*

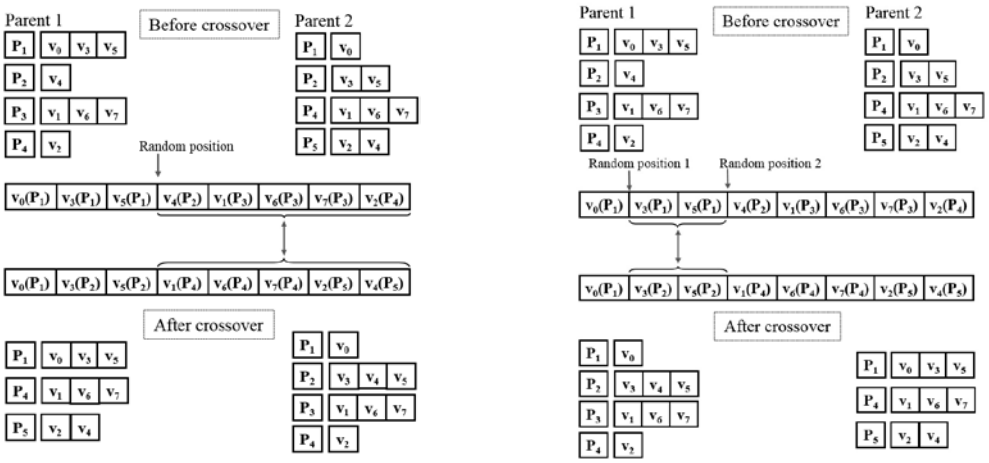
❖ *Selection*

New individuals are selected according to their fitness described by the utility function's tradeoff value after being compared to others in the population. The chance of being selected as a parent is proportional to fitness, and is in inverse ratio to the tradeoff value. An individual whose tradeoff value is lower, is better than one with a higher tradeoff value. The fittest is considered as successive generation evolves. An excessively strong fitness selection bias can lead to sub-optimal solution.

❖ *Crossover*

Crossover operates at an individual level and is used to generate new offspring from two randomly selected individuals (parents) in the current population in order to result in an even better individual in the subsequent generation. There are three methods of crossover: single-point crossover, two-point crossover, and uniform crossover, for all of which the chance of crossover is between 0.6 and 1 in general. As shown in **Fig. 7a**, **7b** and **8**, the crossover operator used is determined by the following rules:

- One, two (or multiple) points are randomly chosen from selected parents.
- These random points divide each individual into left and right sections.
- Crossover then swaps the left (or the right) sections of the two individuals.
- Two new offspring are created by recombining sections taken from two parents.



(a) (b)

Fig. 7. Single-point crossover and two-point crossover

Specially, a random mask containing bits (as illustrated in **Fig. 8**) is generated in uniform crossover. The mask determines which bits are copied from each parent. The bit represents the position of the elements in each individual, and the bit density in the mask determines how much materials is taken from each parent.

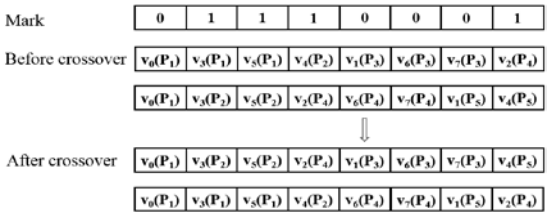
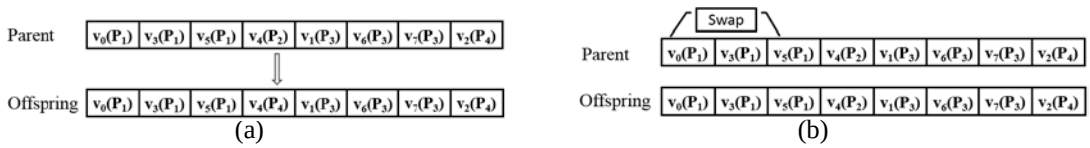


Fig. 8. Uniform crossover

❖ *Mutation*

In genetic algorithms, a mutation generates new offspring from a single parent in the current population. Mutation maintains the diversity of individuals by exploring new and better genes than were previously considered in order to prevent a combination of all solutions in the population converging into a local optimum of solved problems as crossover can only explore the current combinations in the gene pool. However, mutation rates are low as the chance of mutation in a specific individual is low (approximately 0.001). There are two types of mutations: a *replacing mutation* and a *swapping mutation*.



(a) (b)

Fig. 9. Replacing mutation and swapping mutation

The purpose of the replacing mutation is to reallocate a substitute processor to a random task in an individual. The selected processor is also randomly chosen and has enough capacity to execute the task. **Fig. 9a** illustrates the replacing mutation. In this figure, processor P_2 allocated to task v_4 is replaced by processor P_4. In contrast, the swapping mutation changes the execution order of independent tasks on the same processor in an individual for the same time slot. The example of swapping mutation in **Fig. 9b** shows that task v_5 occupies the initial time slot of task v_0.

To verify the performance of our proposed genetic algorithm based approach, we have also designed several other task scheduling algorithms. These algorithms minimize the schedule length of the workflow or lessen the cloud cost paid by CSCs. Algorithm 1 is Greedy for Cost algorithm, where each task of the workflow is assigned to a processor which minimizes cost greedily for Cloud resources to execute that task. In algorithm 2, Contention aware Scheduling [11] aims to create a schedule based on *EFT* pondering on network contention. Interleaved method in algorithm 3 spreads tasks to professors of all PMs as much as possible. Meanwhile, algorithms 4,5 show that our approach takes into account both network contention and the cloud cost as well as the tradeoff between them. Moreover, our method also tries to get a global optimal scheduling of the workflow to reduce recovery time in case of failure.

Algorithm 1. Greedy for cost algorithm

```
Input    : Task graph G = (V, E, w, c) , processor graph TG = (N,D)
Output : A new task scheduling
Function greedyForCostScheduling(G, TG) {
{
    Sort task v_n ∈V into list L according to priority
    for each v_n ∈L
    {
        Find the best processor P_j which minimize the execution cost of task v_n
        Assign v_n on P_j
    }
    return a new task scheduling
}
```

Algorithm 2. Network contention aware scheduling algorithm

```
Input    : Task graph G = (V, E, w, c) , processor graph TG = (N,D)
Output : A new task scheduling
Function contentionAwareScheduling (G, TG) {
{
    Sort task v_n ∈V into list L according to priority
    for each v_n ∈L
    {
        Find the best processor P_j which allows EFT of v_n, taking account of network bandwidth usage;
        Assign v_n on P_j;
    }
    return a new task scheduling;
}
```

Algorithm 3. Interleaved scheduling algorithm

Input : Task graph $G = (V, E, w, c)$, processor graph $TG = (N,D)$
Output : A new task scheduling
Function interleavedScheduling (G, TG) {
{
Sort task $v_n \in$V into list L according to priority
Spread tasks to professors of all PSs as much as possible and processor P_j executing task v_n task has to allow EFT of v_n
return a new task scheduling
}

Algorithm 4. Cost-Time aware Genetic scheduling algorithm

Input : Task graph $G = (V, E, w, c)$, processor graph $TG = (N,D)$
Output : A new task scheduling
Function geneticScheduling (G, TG) {
{
Generate initial population
Compute fitness of each individual according to the equation (14)
repeat // New generation
for population_size
{
Select two parents from old generation
Recombine parents for two offspring
Compute fitness of offspring
Insert offspring in new generation
}
until population has converged
return a new task scheduling
}

Algorithm 5. Minimum recovery time approach

Input : Task graph $G = (V, E, w, c)$, processor graph $TG = (N,D)$
Output : A new task scheduling
Function minimizeRecoveryTime (G, TG)
{
for (i =1; i<=number of tasks in critical path)
{
$S = \varnothing$
$S1$ = schedule generated by Cost-Time aware Genetic scheduling algorithm (Algorithm 4)
for (*failtask* = 1 to number of all tasks)
{
//Assuming failure happens at *failtask*
Find set T of all tasks executed after recovery if task *failtask* fail
$S2$ = schedule is generated by Algorithm 4 with input is T and available processors
Set $S \supseteq S \cup (S1 \cup S2)$
}
int index = 0
Criticalpath[index++] = $\max_{v \in V} scheduleLength(S)$
}
return the shortest element of the Criticalpath
}

5. Implementation and Analysis

In this section, we present experiments that analyze many aspects of our approach. To justify the efficiency of the proposed approach, the Cost-Time aware Genetic scheduling algorithm (CTaG), numerical simulations are used to evaluate it and compare its performance with those of other methods, in terms of monetary cost or network bandwidth. The compared methods include the well-known Contention-aware Scheduling (CaS) algorithm [11], the Greedy for Cost algorithm (GfC), the Interleaved Scheduling method (IS) and a Time aware Genetic scheduling (TaG) algorithm [23] keeping in view only processing time of the system.

5.1 Experimental Settings

All the parameters are different task graphs $G=(V, E, w, c)$ with the increase of the matrix sizes (10-60) and heterogeneous processor graphs $TG=(N,D)$ which is a combination between 30 VMs with the different configurations located at the local system of CSCs for the above algorithms as list in **Table 2**. We developed the simulations in Java with JDK-7u7-i586 and Netbeans-7.2 using CloudSim [16]. It is a framework for modeling and simulation of cloud computing infrastructures and services. In our simulation, we describe MI as Millions of Instructions and denote MIPS as Million Instructions per Second to represent the processing capacity of processors. Moreover, we define a sample of processing cost in **Table 3**, data transmission cost in **Table 4**, waiting cost and disconnection one in **Table 5**. It is obvious that the processing cost and transmission cost are inversely proportional to processing time and transmission time correspondingly. The I/O data of the task has a size from 100 to 500 MB.

Table 2. Characteristics of the target system

Parameter	Value
Network	LAN
Topology model	Fully Connected
Operating system	Windows 7 professional
Number of processors	[5, 30]
Number of tasks	[10, 90]
Processing rate	[10, 750] MIPS
Bandwidth	10, 100, 512, 1024 Mbps

Table 3. Processing rate and corresponding cost for executing a task at processor P_j

Processing rate (MIPS)	Cost per time unit
[10, 50)	0.6
[50, 125)	1.7
[125, 250)	3.6
[250, 500)	7.5
[500, 750)	11

Table 4. Cost of data transmission

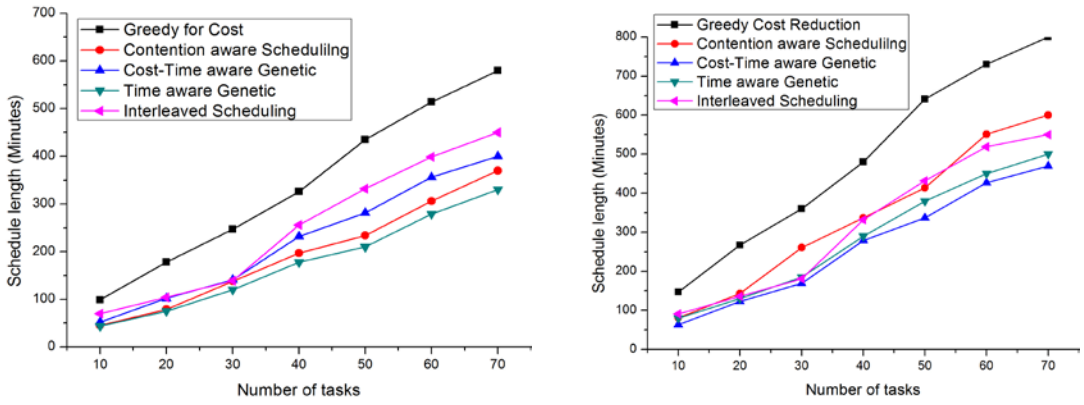
Bandwidth (Mbps)	Cost per time unit
10	0.01
100	0.1
512	0.52
1024	1.024

Table 5. Other costs

Cost	Cost per time unit
Cost of waiting time	[0.1, 0.4]
Cost of a disconnection time	[0,02, 0.2]

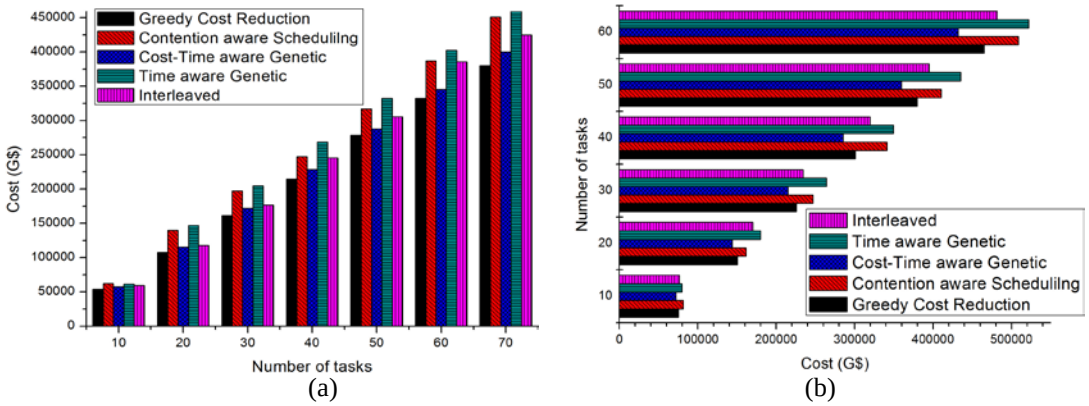
5.2 Experimental Results

The following figures show the simulation results of our proposed genetic method for task scheduling compared against other scheduling techniques. In the following figures, it is obvious to see that there are some differences between the simulated results. **Fig. 10a** shows that, in terms of schedule length, in an environment with no failure, as the number of tasks increases, our method is 27% better than GfC and 15% better than IS due to the extra communication between processors. This is because the proposed method can determine optimal schedules while considering network contention. Additionally, in **Fig. 10b**, when the system has a failure that increases the recovery time, some physical machines have to restart, which increases processing time in the workflow because some task nodes' results are lost and must be reacquired. Our proposal produces a workflow schedule with better performance than others regardless of the number of tasks. Particularly, it achieves a greater than **13%** increase in speed compared with the **TaG** and more than **38%**, **18%** increase compared with **GfC**, **CaS**, respectively.



(a) (b)

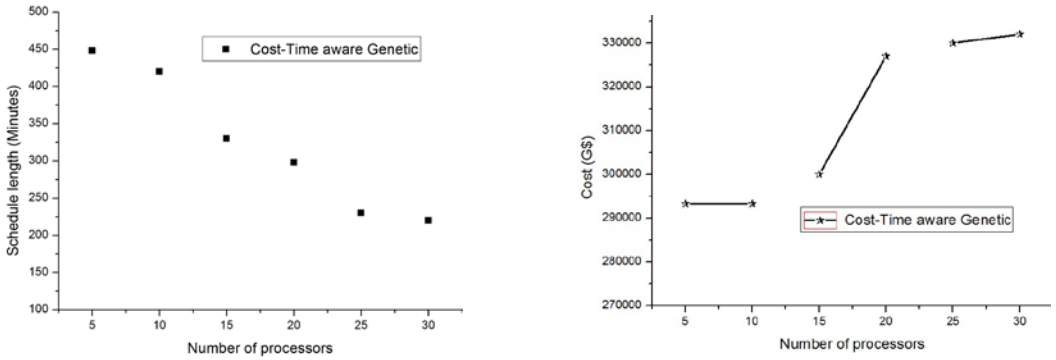
Fig. 10. Schedule length comparison without failure (a) and with failure (b)



(a) (b)

Fig. 11. Cost comparison without failure (a) and with failure (b)

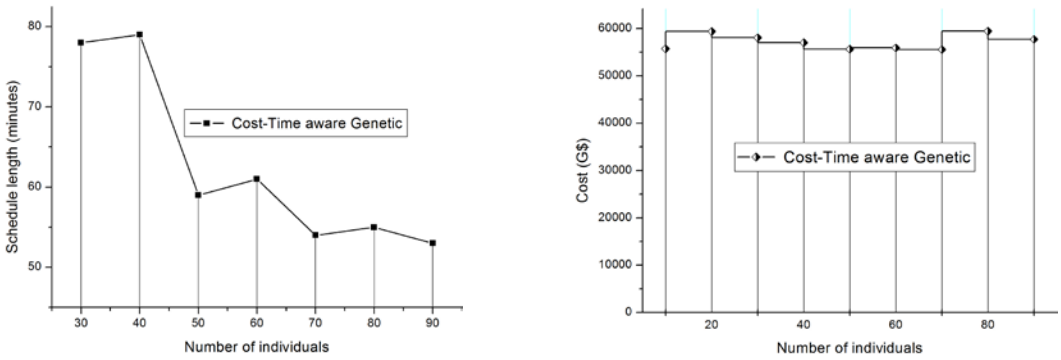
Regarding the monetary cost paid by CSCs (as illustrated in **Fig. 11a**), it has been observed that in case of no failure, **TaG and** CaS have the highest cost, while GfC has the lowest cost but its performance is not good comparatively. The CTaG has a economic advantage compared with IS, which means that its effectiveness increases together with monetary cost. In the meantime, our solution is balanced between schedule length and cloud cost. As a matter of fact, when compared with TaG, our method can save about 19% cost for CSCs while performance reduction is not greater than 17%. Nonetheless, when a failure occurs, the graph in **Fig. 11b** shows that, when the number of tasks increases, **TaG and** CaS require the highest monetary cost, the GfC is intermediate and the proposed method has the lowest cost. Notably, the cost of our approach is **24%, 20%** less than the cost of the **TaG**, **CaS**, respectively, and saves 17% when compared with the GfC cost.



(a) (b)

Fig. 12. Schedule length (a) and cost (b) with numbers of processors

We next measured the effect of increasing number of processors on the cloud cost and the schedule length only in CTaG with a fixed number of tasks. The results reflected in **Fig. 12a** and **12b** indicate that more processors result in better system performance but higher cost. It is highly noticeable to find that the cost goes up from 300500 G$ to 325000 G$ as the number of processors increases from 15 to 20.

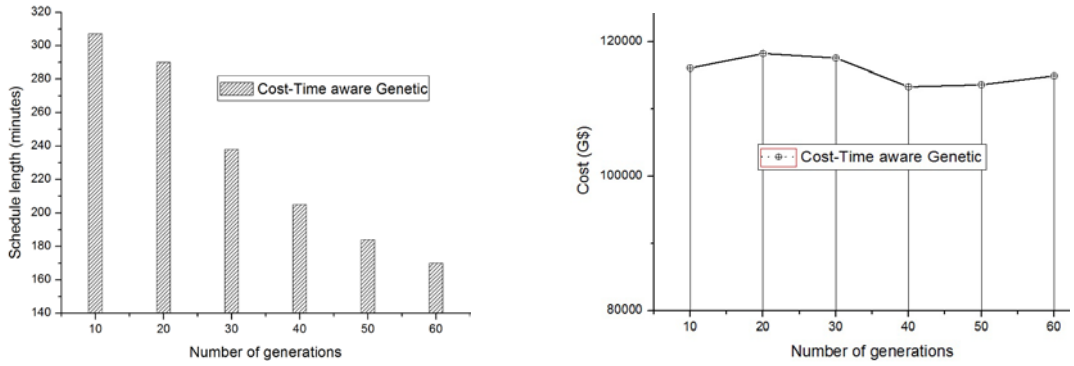


(a) (b)

Fig. 13. Schedule length (a) and cost (b) with numbers of individuals

Further, when the number of individuals is altered from 20 to 90 (**Fig. 13a, 13b**), we witness that the increase in the population size does not significantly affect the execution cost of the workflow while probability of finding a faster solution is higher. The cost just fluctuates

between 55000 and 60000 G$. On the other hand, scheduling time exhibits a downward trend from approximately 80 minutes to around 50 minutes. Finally, we observe the performance of the CTaG with different number of generations.



(a) (b)

Fig. 14. Schedule length (a) and cost (b) with numbers of generations

Similar to the above simulation that regards the number of individuals, results from the **Fig. 14a** and **14b** show that the schedule length of the workflow is reduced with slight decrease in the execution cost when the number of the generations increases. In particular, the schedule length drops dramatically from more than 290 minutes to 200 minutes when the number of generations increases from 20 to 40. This is because each individual selected considers the tradeoff between cloud cost and execution time.

5. Conclusion

In our study, we have presented an optimization and node recovery model for cloud computing, to improve the reliability of cloud services. We modeled our work through genetic task scheduling algorithm. The presented work can be very useful for large amount of data. The proposed model works in such a way that it distributes the tasks among the computing nodes in a datacenter on the basis of minimal scheduling length, hence, globally optimizing the overall process. In case of failure, the system is returned to its previous state in minimum possible time. Our model is cost-effective, since it considers the network bandwidth and the amount of money user has to pay for the services and the tradeoff between them. The presented simulation results and comparison with existing works justifies our model's performance and efficiency. We intend to extend our work with more diverse and challenging scenarios to further extensively check the reliability of the system.

References

[1] Bartholomy, E., "The need to move toward virtualized and more resilient disaster-recovery architectures," *IBM Journal of Research and Development*, 2013. Article (CrossRef Link).

[2] Web Startups Crumble under Amazon S3 Outage http://www.theregister.co.uk/2008/02/15/amazon_s3_outage_feb_2008/.

[3] L. Zeng, "Budget Conscious Scheduling Precedence-Constrained Many-task Workflow Applications in Cloud," in *Proc. of Conf. on Advanced Information Networking and Applications*, 2012. Article (CrossRef Link).

[4] O. Sinnen, "Task Scheduling for Parallel Systems," *Wiley Series on Parallel and Distributed Computting*, Wiley-Interscience, 2007. Article (CrossRef Link).
[5] Kashi Venkatesh V., "Characterizing cloud computing hardware reliability," in *Proc. of the 1st ACM symposium on Cloud computing, Microsoft Research*, 2010. Article (CrossRef Link).
[6] Y.-C. Lee and A. Zomaya, "A novel state transition method for metaheuristic-based scheduling in heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 9, pp. 1215–1223, 2008. Article (CrossRef Link).
[7] Joel Wolf, "SODA, An Optimizing Scheduler for Large-Scale Stream-Based Distributed Computer Systems," in *Proc. of International Conf. on Middleware*, pp. 306-325, 2008. Article (CrossRef Link).
[8] Oliver Sinnen and Leonel A., "Communication Contention in Task Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 6, 2005. Article (CrossRef Link).
[9] Chia-Lee Yang, Bang-Ning Hwang, "Key consideration factors of adopting cloud computing for science," *CloudCom*, 2012. Article (CrossRef Link).
[10] Shohei Gotoda, Naoki Shibata, "Task scheduling algorithm for multicore processor system for minimizing recovery time in case of single node fault," in *Proc. of IEEE Conf. on Cluster, Cloud and Grid Computing (CCGrid)*, 2012. Article (CrossRef Link).
[11] J. Li, Sen Su, Xiang Cheng, Qingjia Huang, Zhongbao Zhang, "Cost-Conscious Scheduling for Large Graph Processing in the Cloud," in *Proc. of IEEE Conf. on High Performance Computing and Communications*, 2011. Article (CrossRef Link).
[12] Cloudsim, "A Framework for Modeling And Simulation of Cloud Computing Infrastrutures And Services," URL https://code.google.com /p/cloudsim/downloads/list.
[13] Ponemon institute, "2013 Study on Data Center Outages," 2013. Article (CrossRef Link).
[14] Y. Gu, Z. Zhang, "An empirical study of high availability in stream processing systems," in *Proc. of 10th ACM Conf. on Middleware*, 2009. Article (CrossRef Link).
[15] Gonzalo H., "A Virtual Cloud Computing Provider for Mobile Devices," in *Proc. of ACM Int. Workshop on Mobile Cloud Computing and Services: Social Networks and Beyond*, 2010. Article (CrossRef Link).
[16] H. Topcuoglu, Min-You Wu, "Performance-Effective and Low-Complexity Task Schduling for Heterogeneous Computing," *IEEE Trans.on Parallel and Distributed Systems*, 2002. Article (CrossRef Link).
[17] Louis C. Canon and E. Jeannot, "Evaluation and Optimization of the Robustness of DAG Schedules in Heterogeneous Environments," *IEEE Trans. on Parallel and Distributed Systems*, vol.21, no.4, pp. 532-546, 2010. Article (CrossRef Link).
[18] Pham Phuoc Hung, Mohammad Aazam, Eui-Nam Huh, "A solution for optimizing recovery time in cloud computing," in *Proc. of Int. Conf. on Ubiquitous Information Management and Communication*, 2014. Article (CrossRef Link).
[19] Zhe Zhang, Hui Lei, Zhen Liu, "A Hybrid Approach to High Availability in Stream Processing Systems," in *Proc. of IEEE Int. Conf. on Distributed Computing Systems*, 2010. Article (CrossRef Link).
[20] B. Narasimhan, R. Nichols, "State of cloud applications and platforms: The cloud adopters view, Computer", vol. 3, pp. 24 -28, 2011. Article (CrossRef Link).
[21] A. Iosup, S. Ostermann, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed Systems*, 2011. Article (CrossRef Link).
[22] T. Kaiser, O. Jegede, "A Genetic Algorithm for Multiprocessor Task scheduling," in *Proc. of World Congress. in Computer Science, Computer Engineering, and Applied Computing, 2013*. Article (CrossRef Link).
[23] Sachi G., Gaurav A., "Task Scheduling in Multi Processor System Using Genetic Algorithm," in *Proc. of IEEE Conf. on Machine Learning and Computing*, Karnataka, 2010. Article (CrossRef Link).

[24] Huynh T.T.B, "Multi-objective Genetic Algorithm for Solving the Multilayer Survivable Optical Network Design Problem," *Journal of Convergence*, vol. 5, no. 1, 2014. Article (CrossRef Link).
[25] Hadis H., Abdolah C., "Scheduling in Multiprocessor System Using Genetic Algorithm," *International Journal of Advanced Science and Technology*, vol. 43, 2012. Article (CrossRef Link).
[26] Mohammad A., Salama H., Sulieman B., "On Static Scheduling of Tasks in Real Time Multiprocessor Systems: An Improved GA-Based Approach," *The International Arab Journal of e-Technology (IAJeT)*, vol. 11, no. 6, 2013. Article (CrossRef Link).
[27] Yujia Ge, Guiyi Wei, "GA-Based Task Scheduler for the Cloud Computing Systems," in *Proc. of IEEE Conf. on Web Information Systems and Mining*, 2010. Article (CrossRef Link).
[28] Ruben V. den Bossche, "Cost-Efficient Scheduling Heuristics for Deadline Constrained Workloads on Hybrid Clouds", in *Proc. of IEEE Conf. on Cloud Computing Technology and Science (CloudCom)*, pp. 320-327, 2011. Article (CrossRef Link).

Pham Phuoc Hung received the B.S. degree in Computer Engineering from Ho Chi Minh National University, University of Sciences, Vietnam, Master's degree in Computer Science from Dongguk University, Korea. He used to be a project manager in some software companies. He has been a PhD scholar in Computer Engineering Department at Kyung Hee University, Korea, since 2012. At present, he is also working as Research Engineer at Real-time Mobile Cloud Research Center (RmCRC), Kyung Hee University, where he has been working on several large-scale R&D funded projects, including their proposals. His research interests include Resource Allocation, Parallel and Distributing Computing, High Performance Computing, Cluster and Grid Computing, Cloud Computing, Sensor Network.

Mohammad Aazam has been a PhD scholar in the School of Computer Engineering, Kyung Hee University, since 2012. He did his MS from Mohammad Ali Jinnah University, in 2011 and BS from Gomal University, in 2006. He has served as Assistant Professor at Federal Urdu University and Bahria University. He has also been Adjunct Faculty at SZABIST and Research Engineer at Mohammad Ali Jinnah University. Currently, he is serving as Research Engineer at Real-time Mobile Cloud Research Center (RmCRC), Kyung Hee University, where he has been working on several large-scale R&D funded projects, including their proposals. He is currently serving Sensors & Transducers Journal as Editor and IEEE Communications Magazine as Associate Editor. Besides that, he is serving several IEEE Transactions as Reviewer Board Member. He has, so far, published more than 70 journal and conference papers. He is a Member IEEE, Member IEEE Communications Society, Member IEEE Cloud Computing, Member IEEE Internet of Things, Member IEEE Smart Cities, Member ISOC, Member ISOC-APAN (Asia Pacific Advanced Network), and Member KIPS (Korea Information Processing Society).

Eui-Nam Huh has earned B.S. degree from Busan National University in Korea, Master's degree in Computer Science from University of Texas, USA in 1995 and Ph.D degree from the Ohio University, USA in 2002. He was a director of Computer Information Center and Assistant Professor in Sahmyook University, South Korea during the academic year 2001 and 2002. He has also served for the WPDRTS/IPDPS community as program chair in 2003. He has been an editor of Journal of Korean Society for Internet Information and Korea Grid Standard group chair since 2002. He was also an Assistant Professor in SeoulWomen's University, South Korea. Now he is with Kyung Hee University, South Korea as Professor in Dept. of Computer Engineering. His interesting research areas are: High Performance Network, Sensor Network, Distributed Real Time System, Grid, Cloud Computing, and Network Security.