A Dynamic Scheduling Method for Collaborated Cloud with Thick Clients

Pham Phuoc Hung[1], Golam Alam[2], Nguyen Hai[3], Quan Tho[3], and Eui-Nam Huh[4]
[1]Department of Computer Science, Kent State University, USA
[2]Department of Computer Science and Engineering, BRAC University, Bangladesh
[3]Ho Chi Minh City University of Technology, Vietnam National University, Vietnam
[4]Department of Computer Engineering, Kyung Hee University, Korea

Abstract: *Nowadays, the emergence of computation-intensive applications brings benefits to individuals and the commercial organization. However, it still faces many challenges due to the limited processing capacity of the local computing resources. Besides, the local computing resources require a lot of finance and human forces. This problem, fortunately, has been made less severe, thanks to the recent adoption of Cloud Computing (CC) platform. CC enables offloading heavy processing tasks up to the "cloud", leaving only simple jobs to the user-end capacity-limited clients. Conversely, as CC is a pay-as-you-go model, it is necessary to find out an approach that guarantees the highly efficient execution time of cloud systems as well as the monetary cost for cloud resource use. Heretofore, a lot of research studies have been carried out, trying to eradicate problems, but they have still proved to be trivial. In this paper, we present a novel architecture, which is a collaboration of the computing resources on cloud provider side and the local computing resources (thick clients) on client side. In addition, the main factor of this framework is the dynamic genetic task scheduling to globally minimize the completion time in cloud service, while taking into account network condition and cloud cost paid by customers. Our simulation and comparison with other scheduling approaches show that the proposal produces a reasonable performance together with a noteworthy cost saving for cloud customers.*

Keywords: *Genetic, cloud computing, task scheduling, thick client, distributed system.*

Received September 10, 2014; accepted January 20, 2016

1. Introduction

Despite recent technology advancements that manufacture a new generation of devices with generous resources, they can offer only limited processing capacity because of the complex properties e.g., large volume, high frequency and higher complexity, of business workflows. One of the ways to address these shortcomings is by applying Cloud Computing (CC) [22]. CC has recently become a rising paradigm in the information and communication technology industry, drawing a lot of attentions to professionals and researchers. It is an inevitable trend in the future computing development of technology [30]. The main idea is to move, or offload, heavy data processing and storage to powerful, centralized server computers resided in data centres, while leaving local devices with only little part, if not all, of the work. Hence, the efficient utilization of cloud resources for achieving high performance is one of the key factors for Cloud Service Customers (CSCs).

Each large-scale workflow contains a set of tasks with varied volumes of workloads. Thus, the effective workflow processing time depends on delivering a good-quality schedule to map tasks of workflow onto multiple processing systems so that task-precedence requirements are satisfied and the overall completion time is minimized. Due to its importance on performance, the task scheduling problem in general have been extensively carried out and many heuristics approaches were introduced in the research literature in high performance computing. There are two main categories, namely stochastic optimization [11, 26] and heuristic-based approaches [6, 13]. The heuristic-based solutions are efficient but could not ensure the optimal performance [1] whereas stochastic optimization ensures the optimal solution with the cost of higher time complexity [30].

Particularly, in the heterogeneous environments such as CC, it is difficult to produce a best solution without an exhaustive search since workflow scheduling problem is NP-Complete [22]. Most of previous attempts in task scheduling have focused on minimizing workflow completion time (makespan/schedule length) [12, 16, 17, 27, 29, 31] without considering the monetary cost paid by CSCs. Recently, some researchers have investigated to find the ways to improve this weakness by trying to reduce cost for the use of external resources such as Virtual Machines (VMs) rented from different cloud providers. A cost-effective method of cloud service provisioning with deadline constraint is presented in [4]. A budget conscious scheduling algorithms is proposed in [15, 16, 19] to satisfy the strictly budget constraint. One of

the reasons is that, the CSCs have to be charged for using their rented resources to execute workflow. For example, Amazon EC2 [2] charges CSCs according to the number of virtual machines initialized by CSCs and how long they have been used. Among those scheduling methods, there are some that may produce a good performance but not efficient enough, because of big monetary cost for using external cloud resources being required. This prevents CSCs from fulfilling the increasing demands while processing the sophisticated applications. Therefore, it is necessary to find an efficient task scheduling to balance the execution time and the cloud cost in order to positively affect the performance, Quality of Service (QoS) [3], and user satisfaction.

To eradicate the shortcoming of above methods while keeping their benefits, in this paper, the main technical contribution is a novel genetic algorithm in order to globally optimize task scheduling to minimize the completion time of the dynamic processing system, allowing for tasks to arrive for processing continuously. Moreover, our proposal takes the network contention and charged monetary cost to CSCs into account since these two factors play important roles in satisfying user's expectations [14]. Furthermore, we introduce a solution to exploit the collaboration between cloud network and the local computing resources, Thick Clients (TCs), residing on the system of CSCs, accordingly improve QoS and user experience. Traditionally TCs have been used for providing rich user interface functionalities and also for mobile users who would like to execute functions without connecting to the network or firewall [25]. In our architecture, thick clients are the traditional computer, such as personal computer or laptop, with powerful processing capacity and available to execute tasks that are dispatched to them as well as store an amount of user data for workflow execution if needed. Some latest generations of smartphones, though, can also be considered thick clients as they are resourceful enough, thanks to their multi-core CPU, large (built-in and extended) memory and especially the LTE connection. Additionally, there is a broker functioning as a centralized management node between thick clients and cloud resources.

By evaluating our approach using extensive simulations, we have demonstrated its efficiency and effectiveness, compared with other existing scheduling techniques. The simulation result shows improved performance of the proposed approach, and especially presented a viable trade-off between workflow performance and cost-effectiveness.

The organization of rest of the sections of this paper is as follows: The state-of-the-art methods related to this research are presented in section 2. System architecture is demonstrated in section 3. Section 4 details the problem formulation. The implementation details with performance evaluation are illustrated in section 5. Section 6 concludes the paper with future directions.

2. Related Work

A number of research proposals have been anticipated for efficient task scheduling in heterogeneous as well as homogeneous systems. Several graph template based task scheduling methods are presented in [4, 5, 8, 29]. The obliviousness of network contention is the major shortcoming of those proposals. Although a network contention based method of task scheduling is presented in [24], but it ignores the monitory cost of Cloud Customers (CCs) in utilizing cloud resources. Despite of numerous research proposals, task scheduling still remains a challenging open problem in heterogeneous CC environment [13, 18]. Authors in [4] introduce a cost-efficient approach to select the most proper system (private or public cloud) to execute the workflow according to a deadline constraint as well as cost savings. Zeng *et al.* [31] propose budget conscious scheduling algorithms to satisfy strictly the budget constrain. Li *et al.* [13] proposed an algorithm of scheduling in to schedule applications of massive graph processing. The schedule length and cost are considered in that proposal. Su *et al.* [26] and Pawar and Wagh [20] present methods to minimize the schedule length or cost in dynamic cloud computing but not study global optimality. Jegede *et al.* [10] and Zhu *et al.* [32] demonstrate scheduling algorithms to solve the global problems however they do not study the monetary cost. Table 1 presents the overview of the state-of-the-art scheduling approaches including our proposed approach.

Table 1. Current scheduling methods and ours.

	Target System	Minimum Completion Time	Minimum Cost	Global Optimality	Dynamic
Topcuoglu [28]	Heterogeneous	Yes	No	No	No
Gotoda *et al*. [9]	Cloud	Yes	No	No	No
Sinnen [24]	Cloud	Yes	No	No	No
Canon and Jeannot [5]	Heterogeneous	Yes	No	No	No
Sakellariou and Zhao [23]	Public & private cloud	Yes	Yes	No	No
Gopalakrishnan [8]	Cloud	Yes	No	Yes	No
Maguluri and Srikant [15]	Multicore processor	Yes	No	Yes	No
Srinivasan [25]	Cloud	Yes	No	Yes	No
Su *et al.* [26]	Cloud	Yes	Yes	No	Yes
Yu and Buyya[30]	Cloud	Yes	Yes	Yes	No
Pawar and Wagh [20]	Cloud	Yes	No	No	Yes
Our approach	Thick clients & cloud	Yes	Yes	Yes	Yes

According to the presented Table 1, the envisioned proposal should take into account all the three efficiency factors:

1. Completion time.
2. Network contention.

3. Cost of cloud resources with the intervention of thick clients as well as the global optimality in a dynamic cloud environment.

This motivates us to provide a Genetic Algorithm (GA) based task scheduling for global optimality in dynamic cloud environment while taking account those factors.

3. System Architecture

The following section gives an insight of our system architecture to address issues discussed above:

The proposed architecture as depicted in Figure 1, consists of two layers:

1. Cloud Provider (CP) layer: composed of pool of VMs.
2. Cloud Customer (CC) layer: resides the thick clients. In the CC layer, the thick client is considered as broker, which acts as a centralized node for management.

The functions of broker in our proposal are as follows:

1. It accepts users' request of computation.
2. It manages processing resources (e.g., capacity, cost and bandwidth) and returning query data.
3. It prepares the efficient and effective and optimal schedule in respect to input workflow.

The broker uploads the data to CP layer through a single connection but VMs of CP layer send data towards the CC layer and then broker divides those data into non-uniform chunks. Afterwards, broker delivers those chunks to thick clients through multi-connection links [21].

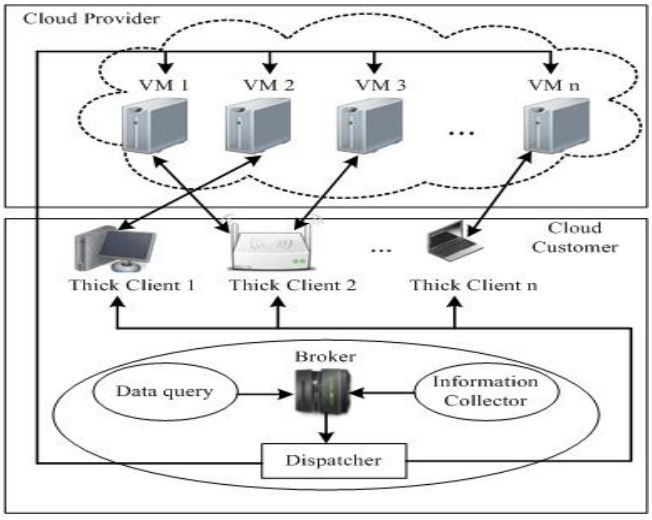

Figure 1. Layering architecture of the proposal.

We assume the following system requirements:

- A high bandwidth link between cloud VMs and thick clients for low latency and faster data communication.
- Among these P2P and thick clients, the communication library is shared.

4. Problem Domain

Task scheduling of complex systems having hierarchical topology is analogous to distribution of jobs or tasks of an application to a group of processors with heterogeneous processing capabilities for fulfilling the optimization goal of minimization of completion time. Therefore, a task graph and a process graph are feed as the inputs of task scheduling. The output is a schedule representing the assignment of tasks to processors.

4.1. Problem Formulation

Before problem formulation we state here the used terms in the problem formulation. Then a genetic algorithm based task scheduling method is presented elaborately.

- *Definition* 1: A processor graph $PG=(N,D)$ demonstrated in Figure 2 is a graph that represents the network topology between vertices (heterogeneous processors) that are Virtual Machines (VMs) on cloud servers and thick clients. In this graph, N is the finite set of vertices, and a directed edge $d_{kl} \in D$ means a directed link from vertex P_k to vertex P_l with $P_k,P_l \in N$. Each processor P_k controls the processing rate μ_k and bandwidth on the link connecting it to other processors. Due to the high stability of Local Area Network (LAN) compared with the Internet, the data transfer rate of internal communication among thick clients is always better than that of external communications between cloud VMs.

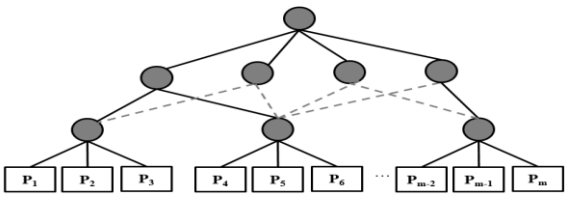

Figure 2. A processor graph.

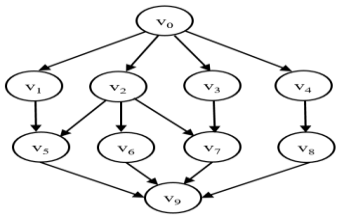

Figure 3. A sample DAG.

- *Definition* 2: A task graph, as shown in Figure 3, is denoted by a Directed Acyclic Graph (DAG) $G=(V, E, W, C)$ where the vertices set $V=\{v_1,v_2,...,v_k\}$ presents the parallel operators (subtasks). The edge $e_{ij}=(v_i,v_j) \in E$ of the DAG symbolized as

communication link between the task v_i and task v_j. The communication time from task v_i to task v_j is denoted by $C(e_{ij})$, whereas $d(e_{ij})$ represents the transferred data. The weight $W(v_i, P_k)$ of task v_i is its computation time on processor P_k. It is supposed here that a task v_i contains a set of preceding subtasks $prec(v_i)$ and a set of successive subtasks $succ(v_i)$. A task without any predecessors, $prec(v_i)=0$, is an start-up task v_{entry}, and a task that does not have any successors, $succ(v_i) =0$, is an end task v_{end}. The workload belongs to task v_i is represented as $l(v_i,P_l)$, , which delimits amount of work (e.g the number of instructions) processed with certain computing resource P_l. Let $t_s(v_i,P_l)$ denote Start Time of task v_i on processor P_l. Hence, the finish time of that task is given by $t_f(v_i, P_l)= t_s(v_i, P_l)+ w(v_i, P_l)$.

Assume that the conditions mentioned below are satisfied:

- *Condition* 1. Application made up of task v_1, task v_2 …task v_n, the broker knows the sequencing of tasks, e.g. task v_1 first, task v_2 second.
- *Condition* 2. When there are several different tasks are running, if new tasks need to be started anytime, the broker will check available processors and allocate the most appropriate processors to the tasks in order to result in a shortest completion time.
- *Condition* 3. Some tasks of the task graph are already scheduled, we may reschedule them upon the arrival new task.
- *Condition* 4. A task can start its execution after all of its parent tasks have already been executed. Each task appears only once in the schedule.
- *Condition* 5. The ready time $t_{ready}(v_i,P_l)$ is the time that processor P_l completes its last running assigned task and be ready to execute task v_i. Therefore,

$$t_{ready}(v_i,P_l) = \max\{t_f(v_y,P_l), \max_{e_{zi}\in E, v_z\in prec(v_i), k\in N}(t_f(v_z,P_k)+c(e_i^{kl}))\} \quad (1)$$

Where v_y is a task being executed at processor and $prec(v_i)$ is a set of preceding tasks of v_i. $c(e_i^{kl})$ is the communication time of connection between processors and It can be defined as:

$$c(e_i^{kl}) = \begin{cases} (do_{zi})\frac{1}{bw_{kl}} & \text{if } k \neq j \\ 0 & otherwise. \end{cases} \quad (2)$$

Here do_{zi} is volume of outgoing data transferred from P_k to P_l.

- *Condition* 6. Let $[t_A,t_B]\in[0,\infty]$ be an idle time interval on processor P_l in which no task is executed. A free task $v_i \in V$ can be scheduled on processor P_l within $[t_A,t_B]$ if

$$\max\{t_A, t_{ready}(v_i,P_l)\}+w(v_i,P_l)\leq t_B. \quad (3)$$

4.2. GA Task Scheduling

Given a task graph $G = (V, E, W, C)$ and a processor graph $PG=(N,D)$, our approach uses a genetic algorithm to produce the most appropriate scheduled list of tasks. Among the numerous random techniques, GAs are the most broadly used for the task scheduling problem [10]. Table 2 demonstrates that GA, motivated by the natural evolution, is a strong search procedure allowing a global high-quality result to be derived from a huge search space in polynomial time. Meanwhile, most of other algorithms discovery only local optimal effects. GA combines the best solutions from past searches with exploration of new regions of the solution space. In this algorithm, a feasible solution, represented an individual (chromosomes), containing a set of processor-task assignments (genes in chromosomes).

Table 2. Genetic algorithm.

Step	Action
1	Generate a random initial population of individuals.
2	Evaluate the fitness of each individual in the initial population if they satisfy their constraints
3	If yes, return output results. If no, generate new populations using procedures in steps 4-6
4	Selects two random individuals among the current population
5	Crossover the two selected individuals considering the crossover probability, to produce the individuals for the next generation
6	Mutate the one of the selected individuals at each defined mutation point, considering the mutation probability and place it in the new population
7	Evaluate the fitness of each of the individuals in the new population
8	Repeat steps 3-7 until the stopping criteria have been met.

A fitness function is used to measure the quality of each individual in the population. A higher fitness level shows a fitter individual, which has a greater rate to reproduce for a new generation. 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 spreading on genetic operators, namely selection, crossover and mutation to a population, quality of individuals can be improved. If well designed, these new individuals will converge to an optimal solution. The genetic algorithm is described in detail in the following section:

- *Producing the Initial Population*: The initial population contains individuals produced through a random heuristic. Each individual (as shown in Figure 4) is a set of tasks and corresponding assigned processors. The time frames of each task in each individual, such as Earliest Start Time, Earliest Finish Time, and so on, can be modified to amend those of its successive tasks. These modifications can generate to an actual multifaceted state during genetic operators. Thus, our way is to disregard the time frame through genetic manipulation and allot a time slot to each assignment so as to achieve a reasonable schedule well ahead.

$v_0(P_2) \mid v_1(P_1) \mid v_2(P_3) \mid v_3(P_3) \mid v_4(P_4) \mid v_5(P_3) \mid v_6(P_2) \mid v_7(P_3) \mid v_8(P_1) \mid v_9(P_3)$

Figure 4. An individual.

- *Constructing a Fitness Function*: Based on fitness value, a fitness function can be used to represent the quality of each individual in a population. The fitness function has to depend on Earliest Finish Time (EFT) and cloud costs paid by CCs because our method tries to minimize the completion time while considering the network contention and cloud cost. The following section exemplifies formation of *EFT* and the cost of task v_i on a processor from its Earliest Start Time (*EST*) as well as the element costs.

EST of a task v_i executed on a processors P_l can be calculated as follows:

$$EST(v_i, P_l) = \begin{cases} t_{ready}(v_i, P_l), \text{ if } v_i \neq v_{entry} \\ 0, \quad \text{otherwise} \end{cases} \tag{4}$$

Suppose that p_{il} is probability of task v_i processed at processor P_l, then the overall task arrival rate at processor P_l can be calculated as: $\sum p_{il}\lambda_{il}$

Thus, computation time to execute task v_i on processor P_l is defined as:

$$w(v_i, P_l) = \frac{l(v_i)}{\mu_l - \sum p_{il}\lambda_{il}} \tag{5}$$

Consequently, *EFT* of the task v_i is designed as:

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

Additionally, the algorithm also takes into account the cloud cost for using cloud resources to finish the tasks. The cost $C(v_i, P_l)$ to execute task v_i at VM P_l or local thick client P_l is defined by:

$$C(v_i, P_l) = \begin{cases} C_{proc}^{(v_i,P_l)} + C_{queue}^{(v_i,P_l)} + C_{comm}^{(v_i,P_l)} + C_{disc}^{(v_i,P_l)} + C_{str}^{(v_i,P_l)} + C_{mem}^{(v_i,P_l)}, & \text{VMs} \\ C_{comm}^{(v_i,P_l)} + C_{disc}^{(v_i,P_l)}, & \text{thick clients} \end{cases} \tag{7}$$

In Equation (7), each cost is calculated as follows:

Processing cost is expressed as:

$$C_{proc}^{(v_i,P_l)} = c_1 * w(v_i, P_l), \tag{8}$$

Where c_1 is the processing cost per time unit of workflow execution on processor P_l.

Let t_{assign} be the time point when task v_i is assigned to processor P_l. c_2 be the queuing cost per time unit of the task v_i until it can be executed. Then the queuing cost is as:

$$C_{queue}^{(v_i,P_l)} = c_5 * (EST(v_i, P_l) - t_{assign}(v_i, P_l)) \tag{9}$$

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

$$C_{comm}^{(v_i,P_l)} = c_3 * \Big(\sum_{v_z \in (prec(v_i) \cap exec(k))} do_{zi} \Big) \frac{1}{bw_{kl}} \tag{10}$$

We presume 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 completion time of task v_i, is $\mu_T * \tau * \mu_L$. Hence, the cost of disconnection can be derived as:

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

Let c_5 be the storage cost per data unit and st_i be the storage size of task v_i on processor P_l. Then we can present the storage cost of task v_i on processor P_l as:

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

In addition, the memory cost of processor P_l for task v_i is computed as follows:

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

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

Lastly, a fitness function that calculates the tradeoff U can be formulated as a convex combination of *EFT* and monetary cost for each individual of the population where task $v_i \in E$ running on processor $P_I \in N$ as follows:

$$U = Min \sum_{v_i, v_j \in V, P_k \in N} \left(\partial \frac{EFT(v_i, P_l) - Min[EFT(v_j, P_k)]}{Max[EFT(v_j, P_k)] - Min[EFT(v_j, P_k)]} + (1-\partial) \frac{C(v_i, P_l) - Min[C(v_j, P_k)]}{Max[C(v_j, P_k)] - Min[C(v_j, P_k)]} \right), \tag{14}$$

$$\partial \in [0,1],$$

Where ∂ a is a cost-conscious factor that represents a user's preference for the completion time and the monetary cost.

By considering the above fitness function that combines *cost* (v_i, P_l) and *EFT*(v_i, P_l), 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_l) and *EFT*(v_i, P_l) should demonstrate the minimum value of the tradeoff U.

- *Genetic Operators*
- *Selection:* An individual with highest fitness value is chosen from the population. Here, the fitness value of an individual is determined through trade-off values utility function U. Probability of selection is higher for those individuals which have lower trade-off utility value and vice versa. However, the fittest individual propagates its strength to the evolve generations strongly. Conversely, sub-optimal solution may returns because of the biasness of selection operators with overly strong fitness.

- *Crossover:* Crossover is used to generate new offspring from two randomly selected individuals in the current population in order to result in an even better individual in the subsequent generation. Usually, the selected two individuals (called parents) with strong fitness produce fittest offspring. Crossover can be single-point, multi-point or uniform. The crossover rate usually in the range between 0.6 to1. As shown in Figures 5, 6, and 7, crossover operators used is determined through the following procedure:
- Single or multiple points are randomly chosen from selected parents.
- Each of the selected points divides the parent in left and right segments.
- Through crossover, the segments are swapped between the selected feasible two parents.
- Therefore, two new offspring are generated by recombining swapped segments.

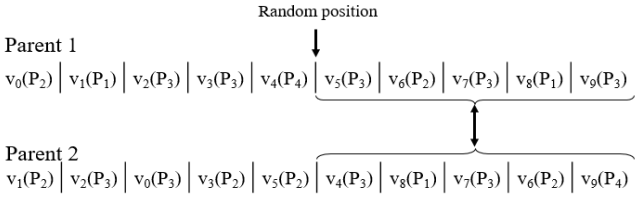

Figure 5. Single-point crossover.

The single-point crossover is shown in Figure 5. The crossover point is determined randomly. Afterwards, swapping is performed. Therefore, the tail of the first parent is added as the tail of the second parent and the tail of the second parent is added as the tail of the first parent. In the meantime, in a multi-point (e.g., two points) crossover operator, two positions in the individuals are determined randomly as shown in Figure 6. The benefit of multi-point crossover is to avoid the inherent problem of single-point crossover, wherein the segment at the head and the segment at the tail of a certain individual are always split when recombined.

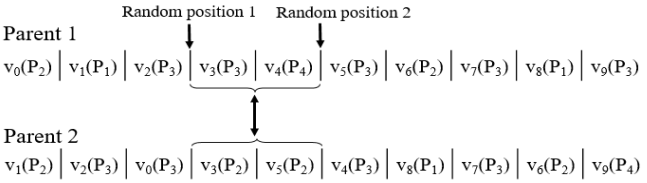

Figure 6. Multi-point crossover.

The uniform crossover is shown in Figure 7. Firstly, a mask of individuals is generated with random binary values. Afterwards, crossover is performed between the individuals only at the non-zero bit positions of the mask. The density or sparsity of uniform crossover is maintained through the masking bits.

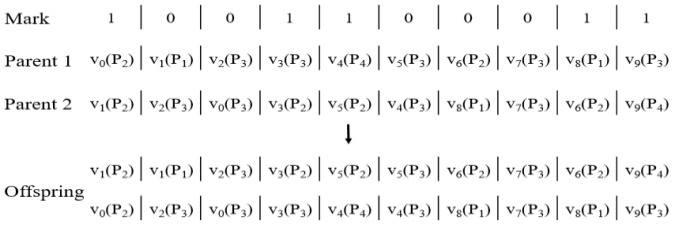

Figure 7. Uniform crossover.

- *Mutation*: Mutation is the self–replicating process in genetic algorithm. Therefore, offspring's are reproduced through the single parent. It is a way of exploring diversity of individuals for better generations. In algorithmic aspect, mutation resists local optima which may be caused by the crossover operation. However, the rate of mutation in a particular individual is little (approximately 0.001). The mutation can be *replacing* or *swapping*.

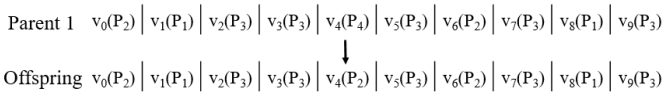

Figure 8. Replacing mutation.

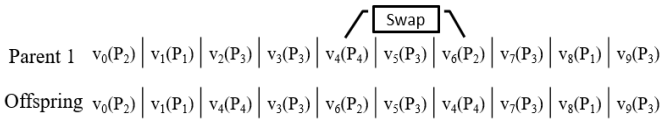

Figure 9. Swapping mutation.

The replacing mutation is shown in Figure 8. In replacing mutation, other than task swapping, the assigned processor is replaced with new processor. Therefore, the task is reallocated to a new processor. The processor replaced is chosen randomly (e.g., task v_4 is previously assigned to P_4 but after applying replacing mutation task v_4 is reallocated to processor P_2). On the other hand, in the case of swapping mutation as Figure 9 shows, the initial position of task v_4 on processor P_4 in parent individual is occupied by the task v_6 on processor P_2 and vice versa. A task is selected at random from a processor and if it is smaller than a task in the most heavily loaded processor, a swap is implemented.

Due to evaluate the performance of the proposal, we have compared our genetic algorithm based scheduling with state-of-the-art task scheduling algorithms. These minimize the total completion time of the workflow or reduce the cloud's task processing cost. Algorithm 1 is Greedy approach for cost reduction, which allocates tasks to processors following greedy principal that minimizes cloud cost. In algorithm 2, network contention aware task scheduling [24] is presented. And a dynamic Time aware Genetic method [26] has a fitness function based on only EFT. In the meantime, algorithm 4 shows that our approach, a dynamic Cost-Time aware Genetic method, stands on both network contention and cloud cost. Therefore, it can justify the

tradeoff of cloud cost and completion time. Moreover, it also ensures a global optimal scheduled list of tasks.

Algorithm 1: Greedy approach for cost reduction

Input: Task graph G, processor graph PG
Output : Scheduled list of tasks
Function greedyForCostScheduling(G, PG)
1. Sort task v_n in the descending order by its priority;
2. for each $v_n \in V$ do
3. Find the most appropriate processor P_l that minimizes the cost for accomplishing the task v_n;
4. Assign v_n to P_l;
5. end
6. return scheduled list of tasks;

Algorithm 2: Contention aware task scheduling

Input: Task graph G, processor graph PG
Output : Scheduled list of tasks
Function networkCostScheduling (G, PG)
1. Sort task v_n in the descending order by its priority;
2. for each $v_n \in V$ do
3. Find the most appropriate processor P_l that allows EFT of v_n, considering network bandwidth usage;
4. Assign v_n to P_l;
5. end
6. return scheduled list of tasks;

Algorithm 3. Dynamic Cost-Time aware Genetic

Input : Task graph G, processor graph PG
Output : Scheduled list of tasks
Function DCTGScheduling (G, PG)
1. Generate random population and determine the population size as p_size;
2. Evaluate the fitness for each individual based on the fitness function (14)
3. repeat //create a new population
4. for p_size do
5. From current generation, select two parents;
6. // better fitness has a bigger chance to be selected
7. Recombine parents for two offspring //with operators;
8. Evaluate fitness of offspring;
9. Insert offspring into new generation
10. end
11. until population has converged

5. Implementation and Analysis

5.1. Experimental Settings

A number of different experiments have been performed to evaluate the efficiency of our approach with varying communication cost and upcoming tasks: the Dynamic Cost-Time aware Genetic algorithm (DCTaG), and compare its performance with the existing ones: Contention aware Task Scheduling (CaTS) [8] just taking account of network contention, Greedy approach for Cost Reduction (RCR) concerning the monetary cost, and a dynamic Time aware Genetic method DTaG [26] having a fitness function based on only EFT.

Task graphs are created with number of tasks in the graph ranging from 10 to 90. The individual size is in range from 20 to 50. The task graphs are scheduled on a multiprocessor system a combination between 22 heterogeneous VMs with the different configurations and 8 thick clients located at the local system of CCs for the above algorithms as shown in Table 2. The parameters for genetic algorithm chosen are population size of 30, crossover rate of 0.7 and the mutation rate of 0.001, number of generations of 60. The metrics used for comparison are the completion time and monetary cost. We assume the communication network is fully connected. Each communication link has its own randomly generated cost. And simulation settings are developed in Java with JDK-7u7-i586 and Netbeans-7.2 using CloudSim [7]. CloudSim is a framework for modeling and simulation of cloud computing infrastructures and services. In our simulation, we use Millions of Instructions (MI) and Million Instructions per Second (MIPS) to represent the processing capacity of processors.

Table 2. Features of the simulated system.

Parameter	Value
Topology model	LAN, fully connected
Operating system	Windows 10 professional
Number of processors	[4, 32]
Number of tasks	[10, 100]
Processing rate	[15, 800]
Bandwidth	[20, 100, 512, 1024] Mbps
Cost per a time unit executed on processor P_l	[0.2, 0.6]
Cost per outgoing data unit from processor P_l	[0.1, 0.5]
Cost of waiting time	[0.2, 0.5]
Cost of a disconnection time	[0,03, 0.3]

5.2. Experimental Results

In the following figures, we present the experimental results to demonstrate that our method, DCTaG, can be more efficiency in term of schedule length and cloud cost. Figures 10 and 11 shows that although GCR obtains the worst result regarding the completion time, it provides the highest cost saving for CCs. In contrast, CaTS delivers the best performance but maximum cost. In the meantime, our solution conduces to the benefits of balance between acceptable completion time for workflow and the corresponding cost for utilizing cloud resources. In particular, compared with CaTS, our method can save nearly 23% cost for CCs and it is 22% faster than GCR.

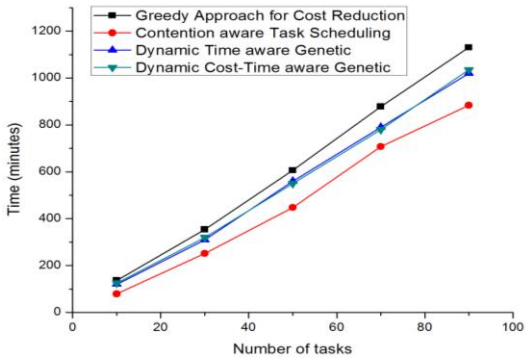

Figure 10. Schedule length comparison.

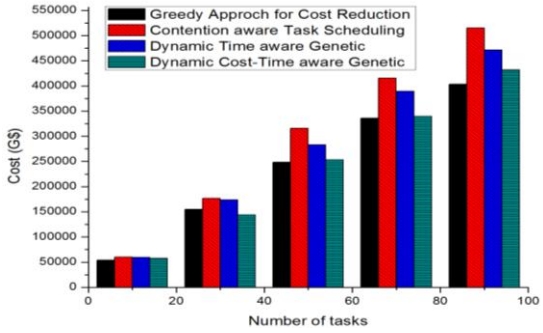

Figure 11. Cost comparison.

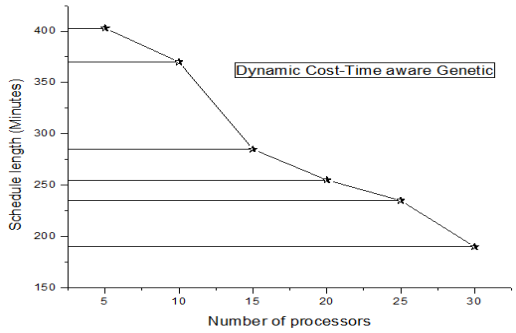

Figure 12. Schedule length with numbers of processors.

The evaluation of the effect on varying number of processors on the cloud cost and the schedule length only in DCTaG with a stable number of tasks is aslo made and given in Figures 12 and 13, respectively. It is clear to see that there is a great improvement in the speed obtained through DCTaG. This improvement rises as the number of processors increases. However, the cost is higher. It is conspicuous to see that the monetary cost increases from 272500 G$ to 297500 G$ as the number of processors goes up from 15 to 20.

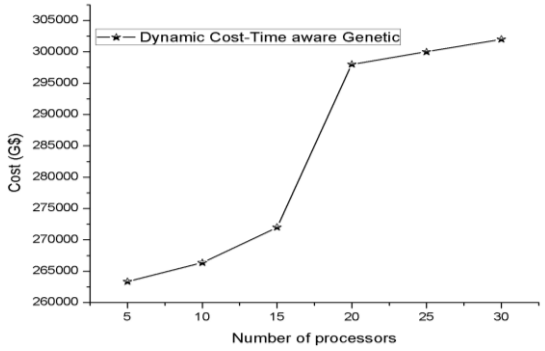

Figure 13. Cost with numbers of processors.

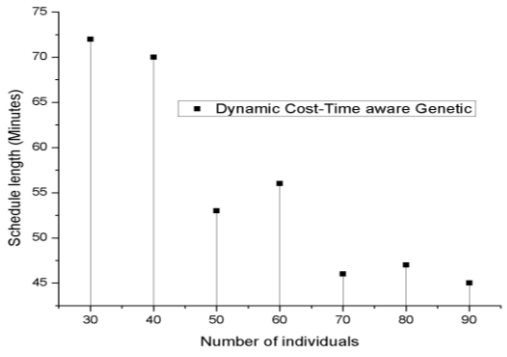

Figure 14. Schedule length with numbers of individuals.

Figures 14 and 15 show the case where the number of individuals is changed from 30 to 90. We observer that the increasing of the population size does not meaningfully have emotional impact in the cloud cost of the schedule but probability of producing a better performance is greater. The cost just varies from 50000 to 55000 G$. Conversely, completion time displays a descending movement between 74 minutes and 45 minutes.

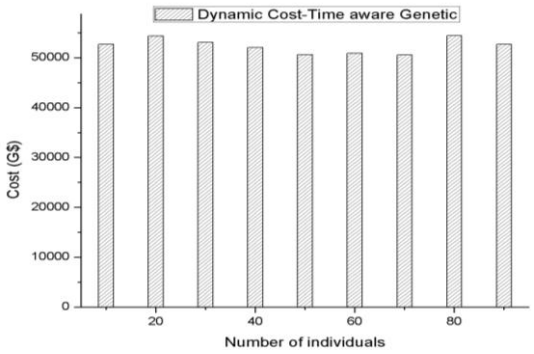

Figure 15. Cost with numbers of individuals.

As a final point, we measure the performance of the DCTaG when the number of generations changes. Similar to the above simulation regarding the amount of individuals, results from the Figures 16 and 17 show that the completion time of the schedule is decreased with the slightly reduction of the execution cost when the number of the generations rises. This is for the reason that each individual selected has to reflect the tradeoff of completion time and cloud cost.

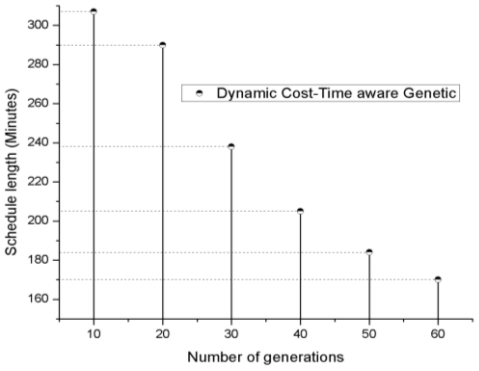

Figure 16. Schedule length with numbers of generations.

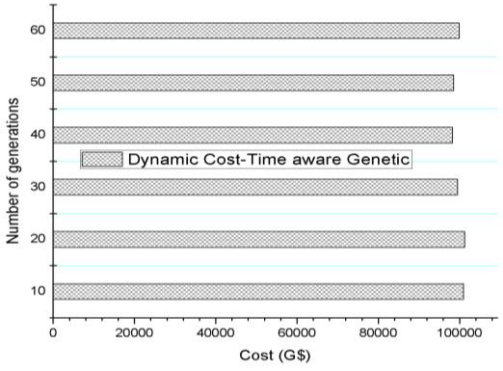

Figure 17. Cost with numbers of generations.

6. Conclusions

This paper proposes a co-operation of local thick clients and cloud resources in cloud platform to take advantage of the total computing power from both internal and external infrastructure. Furthermore, we presented a novel genetic method to expand the dynamic task scheduling in order to achieve desired completion time while balancing the system performance and cloud service cost. Moreover, we conducted simulations to evaluate our approach and compare with other methods. The experimental results demonstrate that the proposed scheduling approach can bring a better performance whilst spending less monetary cost. In future, we will enhance our scheduling method in numerous circumstances such as energy consumption to achieve higher trustworthiness and effectiveness with maximum agreement.

Acknowledgement

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.01-2015.16. This research was also supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the Information Technology Research Center (ITRC) support program (IITP-2016-H8501-16-1015) supervised by the Institute for Information & communications Technology Promotion (IITP). The corresponding author is professor Eui-Nam Huh.

References

[1] Ababneh M., Hassan S., and Bani-Ahmad S., "On Static Scheduling of Tasks in Real Time Multiprocessor Systems: An Improved GA-Based Approach," *The International Arab Journal of Information Technology*, vol. 11, no. 6, pp. 560-572, 2013.

[2] Amazon Web Services. http://aws.amazon.com/ec2/, Last Visited, 2015.

[3] Bhattacharya A., Wu W., and Yang Z., "Quality of Experience Evaluation of Voice Communication: An Affect-Based Approach," *Human-centric Computing and Information Sciences*, vol. 2, no. 7, pp. 1-18, 2012.

[4] Binh H., "Multi-objective Genetic Algorithm for Solving the Multilayer Survivable Optical Network Design Problem," *Journal of Convergence*, vol. 5, no. 1, pp. 20-25, 2014.

[5] Bossche R., Vanmechelen K., and Broeckhove J., "Cost-Efficient Scheduling Heuristics for Deadline Constrained Workloads on Hybrid Clouds," *in Proceedings of IEEE 3rd International Conference on Cloud Computing Technology and Science*, Athens, pp. 320-327, 2011.

[6] Canon L. and Jeannot E., "Evaluation and Optimization of the Robustness of DAG Schedules in Heterogeneous Environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 4, pp. 532-546, 2010.

[7] Cloudsim, https://code.google.com/p/cloudsim/downloads/list, Last Visited, 2014.

[8] Gopalakrishnan A., "A Subjective Job Scheduler Based on A Backpropagation Neural Network," *Human-centric Computing and Information Sciences*, vol. 3, no. 17, 2013.

[9] Gotoda S., Ito M., and Shibata N., "Task Scheduling Algorithm For Multicore Processor System for Minimizing Recovery Time in Case of Single Node Fault," *in Proceedings of 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Ottawa, pp. 260-267, 2012.

[10] Jegede O., Kaiser T., Ferens K., and Ferens K., "A Genetic Algorithm for Multiprocessor Task Scheduling," *in Proceedings of International Conference on Genetic and Evolutionary Methods*, Las Vegas, 2013.

[11] Kim B., Youn C., Park Y., Lee Y., and Choi W., "An Adaptive Workflow Scheduling Scheme Based on an Estimated Data Processing Rate for Next Generation Sequencing in Cloud Computing," *Journal of Information Processing Systems*, vol. 8, no. 4, pp. 555-566, 2012.

[12] Lee Y. and Zamaga A., "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.

[13] Li J., Su S., Cheng X., Huang Q., and Zhang Z., "Cost-Conscious Scheduling for Large Graph Processing in the Cloud," *in Proceedings of IEEE International Conference on High Performance Computing and Communications*, Banff, pp. 808-813, 2011.

[14] Li Q. and Guo Y., "Optimization of Resource Scheduling in Cloud Computing," *in Proceedings of 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Timisoara, pp. 315-320, 2010.

[15] Maguluri S., Srikant R., and Ying L., "Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters," *IEEE Infocom*, Orlando, pp. 702-710, 2012.

[16] Man N. and Huh E., "Cost and Efficiency-based Scheduling on a General Framework Combining between Cloud Computing and Local Thick Clients," *in Proceedings of International Conference on Computing, Management and Telecommunications*, Ho Chi Minh City, pp. 258-263, 2013.

[17] Munir E., Ijaz S., Anjum S., Khan A., Anwar W., and Nisar W., "Novel Approaches for Scheduling Task Graphs in Heterogeneous Distributed Computing Environment," *The International Arab Journal of Information Technology*, vol. 12, no. 3, pp. 270-277, 2014.

[18] Omara F. and Arafa M., "Genetic Algorithms for Task Scheduling Problem," *Journal of Parallel and Distributed Computing*, vol. 70, no. 1, pp. 13-22, 2010.

[19] Oprescu A. and Kielmann T., "Bag-of-tasks Scheduling under Budget Constraints," *in Proceedings of IEEE 2nd International Conference on Cloud Computing Technology and Science*, Indianapolis, pp. 351-359, 2010.

[20] Pawar C. and Wagh R., "Priority Based Dynamic Resource Allocation in Cloud Computing," *in Proceedings of International Symposium on Cloud and Services Computing*, Mangalore, pp. 1-6, 2012.

[21] Phuoc Hung P., Bui T., Morales M., Nguyen M., and Huh E., "Optimal Collaboration of Thin-Thick Clients and Resource Allocation in Cloud Computing," *Personal and Ubiquitous Computing*, vol. 18, no. 3, pp. 563-572, 2014.

[22] Qi H. and Abdullah G., "Research on Mobile Cloud Computing: Review, Trend and Perspectives," *in Proceedings of International Conference on Digital Information and Communication Technology and its Applications*, Bangkok, pp. 195-202, 2012.

[23] Sakellariou R. and Zhao H., "Scheduling Workflows with Budget Constraints," *in Proceedings of Integrated Research in Grid Computing*, Boston, pp. 189-202, 2007.

[24] Sinnen O. and Sousa L., "Communication Contention in Task Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 6, pp. 503-515, 2005.

[25] Srinivasan S., "Why Thick Clients Are Relevant in Cloud Computing," http://cloudcomputing.sys-con.com/node/ 1694221, Last Visited, 2014.

[26] Su S., Li J., Huang Q., Huang X., Shuang K., and Wang J., "Cost-Efficient Task Scheduling for Executing Large Programs in the Cloud," *Parallel Computing Journal*, vol. 39, no. 4-5, pp. 177-188, 2013.

[27] Tawfeek M., El-Sisi A., Keshk A., and Torkey F., "Cloud Task Scheduling Based on Ant Colony Optimization," *The International Arab Journal of Information Technology*, vol. 12, no. 2, pp. 129-137, 2014.

[28] Topcuoglu H., Hariri S., and Wu M., "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," I*EEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, 2002.

[29] Wolf J., Bansal N., Hildrum K., Parekh S., Rajan D., Wagle R., Wu K., and Fleischer L., "SODA: An Optimizing Scheduler for Large-Scale Stream-Based Distributed Computer Systems," *in Proceedings of International Conference on Middleware*, Berlin, pp. 306-325, 2008.

[30] Yu J. and Buyya R., "A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms," *in Proceedings of Workshop on Workflows in Support of Large-Scale Science*, Paris, pp. 1-10, 2006.

[31] Zeng L., Veeravalli B., and Xiaorong L., "ScaleStar: Budget Conscious Scheduling Precedence-Constrained Many-task Workflow Applications in Cloud," *in Proceedings of IEEE 26th International Conference on Advanced Information Networking and Applications*, Fukuoka, pp. 534-541, 2012.

[32] Zhu K., Song H., Liu L., Gao J., and Cheng G., "Hybrid Genetic Algorithm for Cloud Computing Applications," *in Proceedings of IEEE Asia-Pacific Services Computing Conference*, Jeju Island, pp. 182-187, 2011.

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, Ph.D degree in Computer Engineering from KyungHee University, Korea. He used to be a director, a project manager in some software companies. At present, he is also working as a Postdoctoral Researcher in Department of Computer Science at Kent State University, USA 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, Data Analysis, Cluster and Grid Computing, Cloud Computing, Fog Computing, Sensor Network.

Golam Alam received his B.S., M.S and Ph.D. degrees in Computer Science and Engineering, Information Technology, and Computer Engineering respectively. He is currently working as an Assistant Professor in Computer Science and Engineering department at BRAC University, Bangladesh. His research interest includes health informatics, mobile cloud computing, ambient intelligence and persuasive technology.

Nguyen Hai is a PhD Student in the Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology (HCMUT), Vietnam. He received his B.Eng. degree in Information Technology from HCMUT in 2007 and received his Master degree in 2010 from Bordeaux I University, France. His current research areas include formal methods, program analysis/verification, malware analysis, security and dynamic scheduling.

Quan Tho is an Associate Professor in the Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology (HCMUT), Vietnam. He received his B.Eng. degree in Information Technology from HCMUT in 1998 and received Ph.D degree in 2006 from Nanyang Technological University, Singapore. His current research interests include formal methods, program analysis/verification, the Semantic Web, machine learning/data mining and intelligent systems. Currently, he heads the Department of Software Engineering of the Faculty. He is also serving as the Chair of Computer Science Program (undergraduate level).

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 Seoul Women'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.