symmetry

Article

Adaptive Incremental Genetic Algorithm for Task Scheduling in Cloud Environments

Kairong Duan [1,*], Simon Fong [1,*], Shirley W. I. Siu [1], Wei Song [2] and Steven Sheng-Uei Guan [3]

[1] Department of Computer and Information Science, University of Macau, Taipa 999078, Macau; shirleysiu@umac.mo
[2] School of Computer Science, North China University of Technology, Beijing 100144, China; sw@ncut.edu.cn
[3] Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, Suzhou 215123, China; steven.guan@xjtlu.edu.cn
* Correspondence: yb67408@umac.mo (K.D.); ccfong@umac.mo (S.F.)

Received: 24 April 2018; Accepted: 15 May 2018; Published: 17 May 2018

Abstract: Cloud computing is a new commercial model that enables customers to acquire large amounts of virtual resources on demand. Resources including hardware and software can be delivered as services and measured by specific usage of storage, processing, bandwidth, etc. In Cloud computing, task scheduling is a process of mapping cloud tasks to Virtual Machines (VMs). When binding the tasks to VMs, the scheduling strategy has an important influence on the efficiency of datacenter and related energy consumption. Although many traditional scheduling algorithms have been applied in various platforms, they may not work efficiently due to the large number of user requests, the variety of computation resources and complexity of Cloud environment. In this paper, we tackle the task scheduling problem which aims to minimize makespan by Genetic Algorithm (GA). We propose an incremental GA which has adaptive probabilities of crossover and mutation. The mutation and crossover rates change according to generations and also vary between individuals. Large numbers of tasks are randomly generated to simulate various scales of task scheduling problem in Cloud environment. Based on the instance types of Amazon EC2, we implemented virtual machines with different computing capacity on CloudSim. We compared the performance of the adaptive incremental GA with that of Standard GA, Min-Min, Max-Min , Simulated Annealing and Artificial Bee Colony Algorithm in finding the optimal scheme. Experimental results show that the proposed algorithm can achieve feasible solutions which have acceptable makespan with less computation time.

Keywords: cloud computing; Infrastructure as a Service; genetic algorithm; task scheduling

1. Introduction

As the National Institute of Standards and Technology (NIST) defines, cloud computing [1] is a new paradigm that provides configurable resource pool by network access. Resources such as networks, servers and storage can be rapidly provided and released without service provider getting heavily involved. Since Cloud computing technologies offer scalability, are reliable and trustworthy, and offer high performance at relatively low cost, they have a variety of application domains [2], such as virtual computing labs [3], linguistic group decision-making [4], three-dimensional reconstruction [5], etc. In the Cloud model, services including Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) are available to consumers with various demands. Among these three service models, IaaS has the capability to provide Virtual Machines (VMs) and enables customers to control the computational resources or networking components without caring about the location, maintenance or management of the physical devices. In general, IaaS can meet the requirement of performing complex tasks or large applications and measure the service according to the Cloud pricing schemes. However, the process of mapping tasks to VMs, namely task scheduling, becomes more complex due to the large number of physical servers, the complex Cloud environment, the variety of consumer requirements and different Service Level Agreements (SLA) compared with traditional distributed or heterogeneous computing environment.

The task Scheduling problem in Cloud, which is known to be NP-hard, is assigning different tasks to corresponding resource node under the Quality of Services (QoS) constraints. Cloud task scheduling is an NP-hard problem. For an NP-hard algorithm, it is generally believed that no algorithm exists that solves each instance in polynomial time [6]. Some traditional task scheduling algorithms have been applied in heterogeneous computing environments such as Min-Min [7], Max-Min [8], etc. Min-Min algorithm allocates tasks with shortest completion time to the corresponding machine. By contrast, Max-Min algorithm prefers to choose larger tasks which can cause smaller task delays for long time. Although these algorithms are simple and can be easily transplanted to the Cloud environment, they may lead to lower efficiency when the number of tasks is very large.

In recent years, bio-inspired algorithms (also called swarm intelligence algorithms), such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Simulated Annealing (SA), and Artificial Bee Colony (ABC), have proven able to solve NP problems effectively. These algorithms originate from the behavior of nature biological group or physical phenomena, and have features such as intelligence, self-organization, parallelism, etc. The nature group can perform complex intelligent behavior, even though the capacity of each individual is very limited. All the individuals can cooperate with each other without any centralized control and spontaneously move toward the optimal direction. For example, ants, bee or birds can always be efficient in finding food sources despite the limited ability of communication and intelligence. Inspired by group intelligence, swarm intelligence algorithms can provide solutions for those complex optimization problems which do not have priori knowledge or global model and have the following characteristics: (1) The ability of each individual in the system is relatively simple and their active period is short. (2) There are no special requirements on the problems to be optimized, such as continuity, linearity, constraint, etc. (3) Individuals have the chance to change the environment and the whole system is self-regulated. (4) Centralized control constraints so human interventions or adjustments are not necessary during the problem solving process. (5) Members in the group are independent and distributed, so they can deal with the problem in parallel. (6) Each individual can perceive the environment and cooperate with others, and the cost of increasing or decreasing the scale of population is low. (7) The complex behavior manifested by the group is acquired through the interaction process of simple individuals. Due to the merits of swarm intelligence algorithms mentioned, researchers have introduced many of them to obtain feasible solutions for non-linear, complex, strong constraint or large-scale problems.

Zhan et al. [9] presented an improved Particle Swarm Optimization algorithm to solve the problem of cloud task scheduling. In the iterative process, PSO is combined with simulated annealing algorithm to achieve global fast convergence while avoiding running into local optimal. Zuo et al. [10] proposed a resource allocation framework in which an IaaS provider can outsource its tasks to External Clouds (ECs) when its own resources are not sufficient to meet the demand. They focused on how to allocate users' tasks to maximize the profit of IaaS provider while guaranteeing QoS. The task scheduling problem is formulated as an integer programming (IP) model, and solved by a self-adaptive learning particle swarm optimization (SLPSO)-based scheduling approach. Chen et al. [11] presented a parallelized genetic ant colony system (PGACS), which consists of the genetic algorithm, including the new crossover operations and the hybrid mutation operations, and the ant colony systems with communication strategies. Tsai et al. [12] proposed parallel cat swarm optimization method, and, in [13], they used the enhanced parallel cat swarm optimization (EPCSO) to solve the numerical optimization and aircraft schedule recovery problem.

Wu et al. [14] implemented genetic algorithm and Chemical Reaction Optimization (CRO) algorithm to optimize task scheduling problem in Cloud environment. They compared the performance the two algorithms in terms of makespan and energy consumption. Mahmood et al. [15] proposed a greedy and a genetic algorithm with an adaptive selection of suitable crossover and mutation operations to allocate and schedule real-time tasks with precedence constraint on heterogamous virtual machines.

Artificial Bee Colony (ABC) [16] algorithm is an optimization algorithm based on the intelligent behaviour of honey bee swarm, which has the ability to get out of a local minimum and can be efficiently used for multivariable, multimodal function optimization. Babu et al. [17] presented a load balancing algorithm based on Artificial Bee Colony (ABC), in which tasks on overloaded virtual machines are selected and migrated to other VMs. Navimipour et al. [18] applied ABC to the task scheduling problem in the cloud environment. A food source represents a feasible task scheduling scheme and is evaluated by the pre-defined fitness. The population is randomly generated in the initialization phase, after which employed, onlooker and scout bees are sent to mine the food sources until the iteration reach its max value.

Simulated annealing is a heuristic method that has been implemented to obtain good solutions of an objective function defined on a number of discrete optimization problems [19]. It has proven to be a flexible local search method and can be successfully applied to many real-life problems. Mandal et al. [20] implemented three meta-heuristic algorithms to solve Cloud task scheduling, including Simulated Annealing, Firefly Algorithm and Cuckoo Search Algorithm. The main goal of these algorithms is to minimize the overall processing time of the VMs which execute a set of tasks.

With the prevalence of Cloud Computing, more users have obtained benefits from Cloud Computing service. For example, one of the most promising cloud computing applications is online data sharing [21], such as photo sharing in Online Social Networks among more than one billion users [22], and online health record system [23]. Massive data centers and high-speed networks enable processing multiple tasks in parallel, and there is a need to schedule or allocate those tasks to appropriate virtual machines efficiently. However, most of the works mentioned above were simulated or tested in a small scale. The number of tasks is up to decades in [10,17,18,20], hundreds in [9,15], or a little more than one thousand in [14]. Those algorithms may not work effectively under high pressure in Cloud environment. Moreover, the pay-per-use payment method and sustainable development policy also have strong correlations with time consumption and the energy consumed. Therefore, there is a need to research on the task scheduling problem in large scale. Motivated by these considerations and the works mentioned above, we present an incremental genetic algorithm to task scheduling problem in Cloud environment. The main contributions of this paper include: (i) proposing an incremental genetic algorithm to tackle large scale task scheduling problem in Cloud environment; (ii) designing an adaptive mutation and crossover rate for the proposed algorithm; and (iii) experimental results obtained using the proposed method compared with the results of applying Min-Min, Max-Min, Standard GA, Min-Min, Max-Min, Simulated Annealing and Artificial Bee Colony Algorithm, showing that our approach can reduce the computation time. In this paper, we investigate some traditional and bio-inspired algorithms that can be applied in the distributed computing environments and propose an adaptive incremental Genetic Algorithm (AIGA) to address the Cloud task scheduling problem. We model the problem based on IaaS instance and set the minimum makespan as the objective.

The remainder of this paper is organized as follows. In Section 2, we describe the task scheduling problem and optimization objective by mathematic model. We also represent two best-known heuristics, Min-Min and Max-Min algorithms, which are widely used to address task scheduling in many platforms. Section 3 provides details of the proposed adaptive incremental genetic algorithm on IaaS. In Section 4, we compare the performance of Min-Min, Max-Min, standard GA and the adaptive incremental GA and discuss the experimental result. Section 5 concludes the paper.

2. Task Scheduling Problem

In a Cloud model, datacenter consists of many physical servers on which VMs are running and user tasks are bound to different types of VMs according to the specific demand. There are two scheduling problems based on different levels. One is deciding to which host the VM should be deployed. The VM scheduling method directly affects resource utilization, energy consumption and efficiency of datacenter in Cloud. Some scheduling algorithms, such as first-come-first-serve or greedy algorithm, may result in an increase in the number of active physical servers, especially when the virtual machine requests are greatly different. Another one is task scheduling problem that determines to which VM the task should be allocated. In the Cloud environment, large-scale tasks are firstly divided into several small tasks by Map/Reduce model, and then distributed to the appropriate Cloud servers in the resource pool. Based on the status of each virtual machine and the quality of service requirement, tasks should be assigned to appropriate VM. Task scheduling strategy affects task completion time and associated execution costs. It is important for both providers and users to improve efficiency and reduce costs by applying proper scheduling methods in Cloud computing.

Cloud Information Service (CIS) and datacenter broker are responsible for scheduling VM and tasks based on user requests and quality requirements. Users are concerned about the cost of services, while service providers expect to maximize efficiency, while minimizing energy consumption and costs. To some extent, these factors relate to the task execution time, so we consider makespan as the optimization objective in this paper. We model the task scheduling problem as follows.

Suppose that there are m tasks $T = \{t_1, t_2, ..., t_m\}$, and n virtual machines $VM = \{vm_1, vm_2, ..., vm_n\}$. It is a NP-complete problem which has n^m ways to allocate these tasks to VMs.

In general, a VM can be described by CPU, memory bandwidth and storage. Some Cloud service providers such as Amazon use Compute Unit (CU) to describe the CPU capacities. One EC2 Compute Unit (ECU) provides the equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor [24]. Since Cloud service is used on demand and paid according to the used hours, CU can be used to estimate the required execution time before applying for the cloud resources. When task t_i is bound to virtual machine vm_j, the occupation time of vm_j depends on the length of task and the CU of VM. The execution time for t_i on vm_j is given by t_{ij},

$$t_{ij} = L_i / C_j \tag{1}$$

where $i \in \{1, 2, ..., m\}$, $j \in \{1, 2, ..., n\}$, L_i represents the length of task t_i, and C_j denotes the CU of virtual machine vm_j.

The occupation time for vm_j can be calculated by,

$$T_j = \sum_{i=1}^{m} x_{ij} \cdot t_{ij} \tag{2}$$

where $x_{ij} \in \{0, 1\}$ represents the mapping relationship between a task and a virtual machine. When $x_{ij} = 1$, t_i is allocated on vm_j.

It is relevant to point out that the execution time required to complete all the tasks equals the maximum value of all the virtual machines' occupation time, because tasks are processing in parallel on VM. For the purpose of minimizing makespan, the objective function can be formulated as follows,

$$\min(\max(T_j)) = \min(\max(\sum_{i=1}^{m} x_{ij} \cdot t_{ij})) \tag{3}$$

where x_{ij} denotes the allocation status and t_{ij} represents corresponding execution time.

3. Adaptive Incremental Genetic Algorithm

Genetic Algorithm (GA) is a bio-inspired algorithm which belongs to Evolutionary Algorithms (EA). Genetic algorithm is based on the idea of Darwin's Theory of Evolution and Mendelian Genetics. Darwin's Theory of Evolution indicates that, during the process of biological evolution, individuals who are more able to adapt to the environment have a higher probability of survival; Mendel's genetic theory demonstrates that genes are kept by chromosome in the form of genes, and gene mutation or chromosome crossover can generate new features for individuals. Those genetic structures with high adaptability are easier to be retained.

The main operators of GA include selection, crossover and mutation. In GA, select operator imitates the natural selection. The purpose of selection is saving "good" individuals. The larger the fitness value is, the higher the probability that allows the individual to enter into the next generation. On the contrary, individuals with smaller fitness value will be less likely to be kept. Selection strategies affect the direction of evolution and convergence rate of the population. Crossover is the process of gene recombination. Ideally, the offspring produced by the crossover of two parents should inherit the excellent features of their parents. Mutation operation produces new individual by changing a few genes. It contributes to the diversity of population and also plays an important role in approaching the optimal solution.

When using GA to optimize different problems, each chromosome represents a candidate solution. Fitness is defined to evaluate the solution. A series of genetic operations (crossover, recombination, mutation, etc.) are used to generate new chromosomes, while select operators will choose better solution to retain. After a number of generations, the group converges to the individuals who are most adaptive to the environment, that is, the optimal solution.

To address task scheduling problem, the chromosome can be encoded with real number. The chromosome represents a feasible solution. Considering the complex cloud environment, the datacenter schedules a great number of tasks in the request list. Standard GA can cost a long time to search for the optimal solution because evaluation, crossover and mutation operators involve more computing. In this paper, an incremental method is proposed to tackle this problem. The tasks are divided into some independent sets, and then every time some tasks are chosen to schedule. Details about the proposed adaptive incremental GA are as follows.

3.1. Encoding

Encoding strategies such as binary encoding, real number encoding and symbol encoding have been applied on different problems. Binary coding maps solution space to a bit string, which is a simple and common encoding method. Similarly, a real-valued chromosome is represented by a real number in the decision space. In the task scheduling problem, the chromosome is encoded by real number. The length of a chromosome equals the number of tasks. The gene's value represents to which VM the task is allocated.

3.2. Fitness Function

The fitness function is used to evaluate the quality of the candidate solution, which is an important index for selecting the best individual. Usually, the value of objective function can be directly used for fitness measurement, or define the fitness function according to the specific problem. As given in Equation (3), the fitness is calculated by the makespan of a solution. The maximum finish time of among all the VMs is the fitness value.

3.3. Select

There are several selection strategies such as roulette, tournament and sorting selection. The roulette method firstly computes the probability of selection according to each individual's fitness (the proportion of the individual's fitness to the sum of all individual fitness value). Then, the disk is divided into N copies, each with a central angle which is proportional to the probability of selection. Choosing an individual depends on a randomly generated number falling into which region. Tournament selects a certain number of individuals, and the individual with highest fitness will be chosen. This process continues until the number of individuals to reach a pre-set size. Sorting selection firstly calculates each individual's fitness value. After sorting by fitness, the solution will be assigned a probability of selection. Then, the selection is determined by the order of distribution.

3.4. Crossover

There are some crossover strategies such as single-point crossover, two-point crossover, uniform crossover and so on. An example of single-point crossover is shown in Figure 1 .

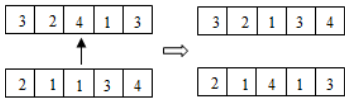


Figure 1. Crossover operator.

3.5. Mutation

Mutation operators include uniform mutation, Gaussian mutation, etc. In this paper, we use a new mutation strategy state as follows. First, choose a mutation position, and then exchange its value with a randomly selected gene. The values of the two positions should not be the same, otherwise select another position. The mutation operator is shown in Figure 2.

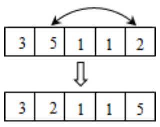


Figure 2. Mutation operator.

We define adaptive mutation and crossover operator as follows.

$$m_i = (1 - \alpha \cdot (i/mg)^2)^\lambda, c_i = \beta - m_i \tag{4}$$

where $m_i, c_i \in (0,1)$, mg is the maximum iteration, i is the current iteration, and λ, α, β are the control parameters.

For each generation, sort the population by fitness in ascending order. For each individual, its mutation rate and crossover rate is calculated by,

$$\begin{aligned} mr_{ij} &= m_i + (1 - m_i)/s \cdot p_j,\ cr_{ij} = c_i - c_i/s \cdot p_j, \texttt{if } l \geq t \\ mr_{ij} &= m_i - m_i/s \cdot p_j,\ cr_{ij} = c_i + (1 - c_i)/s \cdot p_j, \texttt{others} \end{aligned} \tag{5}$$

where $l = lb_{i-1} - lb_i$, p_j is the position of individual j in the population, s is the population size, lb_i is the fitness of local best solution in iteration i, l represents the evolutionary trend of the population, and t is the threshold. The main steps of GA-based task scheduling include,

(1) Initialize GA parameters: Initialize the population size, crossover rate, mutation rate, max generation, etc.
(2) Initialize job list: Each job is a set of tasks and the number of tasks is equal to the predefined interval size. Define the maximum iteration as the value that max generation divided by the number of jobs (also other finite bound can be used).

(3) Optimization stage: For each job in the job list, use adaptive GA to optimize the scheduling problem. After randomly generating or initializing the population based on Max-Min, do select, crossover, mutation and record the best individual until the iteration reach the maximum value. Notice that, before handling the next job, the virtual machines' occupation times should be updated by the optimal solution of the previous job.

The pseudo code of Adaptive Incremental Genetic Algorithm is shown in Algorithm 1.

Algorithm 1 Incremental Genetic Algorithm

```
Input: Population size $Np$, max iteration, objective function, control parameters, task list $T$, VM types,
    interval size;
Output: The global best solution;
 1: Choose tasks from $T$, the number of tasks equals to the interval size;
 2: Initialize the population $P$
 3: Evaluate each individual in $P$
 4: while iteration < max generation do
 5:   for each individual in $P$ do
 6:     Calculate the $probability$ to be chosen;
 7:     if $rand(0,1) < probability$ then
 8:       Select the individual to the new population $P'$;
 9:     end if
10:   end for
11:   Sort $P'$ according to fitness
12:   Calculate the mutation and crossover rate of each individual;
13:   for each individual in $P'$ do
14:     if $rand(0,1) < crossoverrate$ then
15:       Perform crossover;
16:     end if
17:     if $rand(0,1) < mutationrate$ then
18:       Perform mutation;
19:     end if
20:   end for
21:   Update the global best solution; iteration++;
22: end while
23: Update the occupation time of each virtual machine
```

4. Experimental Result

In this section, we give details about the experimental work. Experiments were set to compare the performance of the proposed Adaptive Incremental Genetic Algorithm (AIGA) with that of two traditional scheduling methods, Min-Min and Max-Min, and three meta-heuristic algorithms, Standard Genetic Algorithm (SGA) [14], Artificial Bee Colony (ABC) [18] and Simulated Annealing (SA) [20] algorithm.

Min-Min is based on the idea that shortest unassigned tasks will be scheduled with highest priority to the machine which can complete the task in shortest time. Min-Min begins with a set of unallocated tasks in the waiting list. First, it computes the Minimum Completion Time (MCT) for all tasks on all available machines. Then, it selects the minimum MCT and allocates the task to the corresponding resource. After removing the task from the list, the available time of the machine should be updated. This process repeats until all the tasks in the waiting list have been scheduled. Max-Min is similar to the Min-Min algorithm. In the choosing stage, Max-Min selects the largest task first and allocates it to the machine which has the MCT.

The virtual machine (instance) specifications refer to the scheme of Amazon Elastic Compute Cloud [25,26] (Table 1). The length of tasks is randomly generated ranging from 10 to 1000.

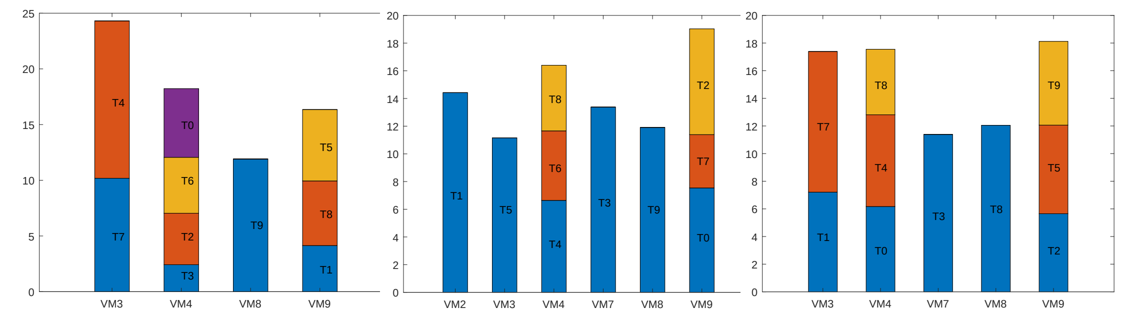
We first compare the performance of AIGA with that of two traditional scheduling methods, Min-Min and Max-Min. Suppose there are ten randomly generated tasks, the length of each tasks is in [814.0, 447.0, 610.0, 319.0, 876.0, 692.0, 663.0, 631.0, 626.0, 655.0]. Then, task execution time on each VM can be calculated as shown in Table 2. Figure 3 represents the makespan and assignment of tasks by Min-Min, Max-Min and AIGA. Note that AIGA can achieve different result as random factors are involved, so Figure 3c shows only one possible solution of AIGA. Min-Min and Max-Min allocate the tasks in the order [3, 1, 2, 8, 7, 9, 6, 5, 0, 4] and [4, 0, 5, 6, 9, 7, 8, 2, 1, 3], respectively. As makespan is the maximum execution time among all the VMs after all the tasks are scheduled, the makespan obtained by each algorithm is 24.31, 19.5 and 18.12, respectively. AIGA has the minimum fitness of optimal solution.

Table 1. VM types.

	Instance Type	CU
1	c3:large	7
2	c3:xlarge	14
3	c3:2xlarge	28
4	c3:3xlarge	55
5	c3:4xlarge	108
6	c4:large	8
7	c4:xlarge	16
8	c4:2xlarge	31
9	c4:4xlarge	62
10	c4:8xlarge	132

Table 2. Execution time of each virtual machine.

	VM0	VM1	VM2	VM3	VM4	VM5	VM6	VM7	VM8	VM9
0	101.75	50.88	26.26	13.13	6.17	116.29	58.14	29.07	14.8	7.54
1	55.88	27.94	14.42	7.21	3.39	63.86	31.93	15.96	8.13	4.14
2	76.25	38.13	19.68	9.84	4.62	87.14	43.57	21.79	11.09	5.65
3	39.88	19.94	10.29	5.15	2.42	45.57	22.79	11.39	5.8	2.95
4	109.5	54.75	28.26	14.13	6.64	125.14	62.57	31.29	15.93	8.11
5	86.5	43.25	22.32	11.16	5.24	98.86	49.43	24.71	12.58	6.41
6	82.88	41.44	21.39	10.69	5.02	94.71	47.36	23.68	12.05	6.14
7	78.88	39.44	20.35	10.18	4.78	90.14	45.07	22.54	11.47	5.84
8	78.25	39.13	20.19	10.1	4.74	89.43	44.71	22.36	11.38	5.8
9	81.88	40.94	21.13	10.56	4.96	93.57	46.79	23.39	11.91	6.06



(**a**) Min-Min (**b**) Max-Min (**c**) AIGA

Figure 3. Tasks assignment by: (**a**) Min-Min; (**b**) Max-Min; and (**c**) AIGA.

Then, we mainly evaluate the flexibility of SGA and AIGA according to different parameters. It has been long known that crossover rate and mutation rate have an important impact on the performance of GA. Typical values of crossover rate are in the range 0.5–1.0, and mutation rate is chosen in range 0.005–0.05 [27]. Actually, features of the optimization problem should be considered when deciding these parameters. We set up two groups of experiments, one changes mutation rate (MR) and the other alters crossover rate (CR). Both set the number of tasks as 300. Each condition was run 10 times, giving the average fitness value.

Table 3 shows the result when the mutation rate ranges from 0.1 to 0.9 (crossover rate is 0.6). Overall, the fitness of global best solution obtained by AIGA is better than that of SGA; Table 4 shows the result when the crossover rate ranges from 0.1 to 0.9 (mutation rate is 0.3). The fitness of global best solution obtained by AIGA also outperforms that of SGA. Figure 4a,b gives the makespan chart according to different MR and CR. When mutation rate is low, SGA converges slowly which may run into local optimal. On the other hand, AIGA steadily outmatches SGA in the experiments.

Table 3. Fitness of optimal solution.

MR	SGA	AIGA
0.1	594.933	355.407
0.2	373.862	350.565
0.3	372.02	365.267
0.4	384.447	374.111
0.5	376.61	365.998
0.6	370.013	366.195
0.7	365.656	364.522
0.8	367.732	366.922
0.9	353.147	352.397

Table 4. Fitness of optimal solution.

CR	SGA	AIGA
0.1	353.558	338.128
0.2	372.904	363.381
0.3	390.524	371.02
0.4	363.018	345.526
0.5	360.534	354.471
0.6	369.683	365.282
0.7	360.473	343.74
0.8	369.95	357.186
0.9	364.27	352.722

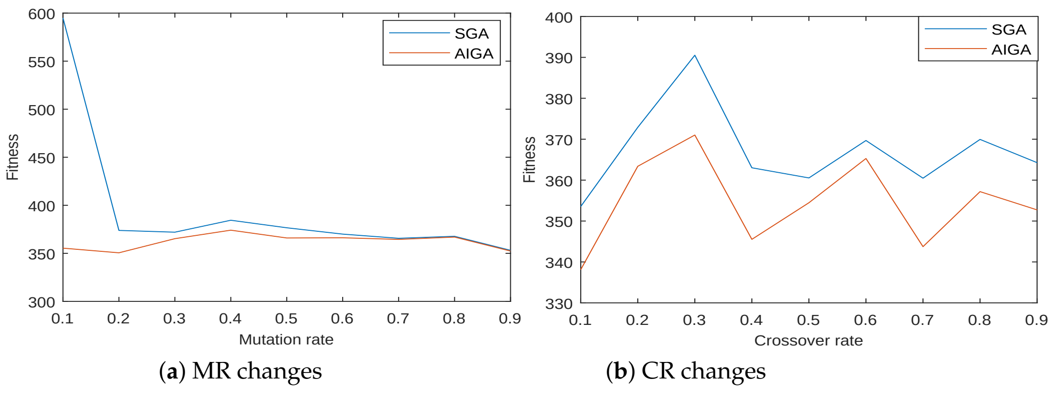


Figure 4. Makespan by SGA and AIGA. (**a**) MR changes; (**b**) CR changes.

We also compare AIGA with Simulated Annealing (SA) and Artificial Bee Colony (ABC) algorithm in the task scheduling problem. As mentioned above, SGA may not find the optimal solution when mutation rate is too small, so we set MR 0.3 and CR 0.8. Then, 300 tasks were generate and each algorithm was run 10 times. Figure 5 shows the boxplot of optimal fitness obtained by SA, ABC, SGA and AIGA. The distribution of fitness acquired by AIGA is more concentrated in a small range in comparison with others. Figure 6 gives convergence curve of the four algorithms. SA and ABC converge relatively slowly compared to SGA and AIGA. AIGA starts from lowest point and gradually increases until to it meets the stop criteria. The optimal fitness of SGA and AIGA are quite close at the end of iteration.

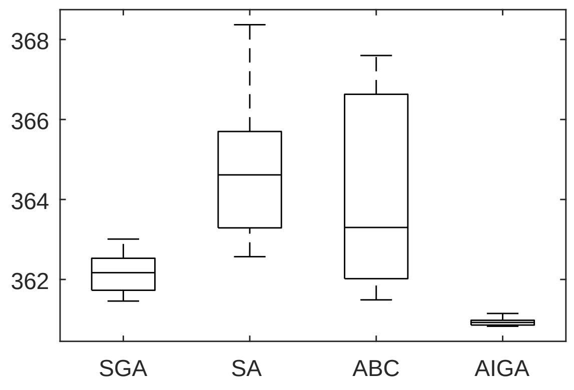


Figure 5. Optimal fitness of SA, ABC, SGA and AIGA.

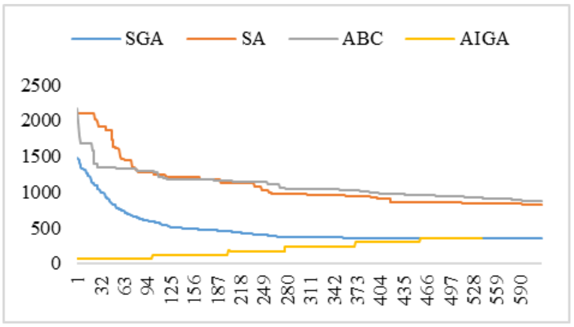


Figure 6. Convergence of SA, ABC, SGA and AIGA.

Table 5 gives the optimal fitness when the number of tasks increases from 100 to 90,000 (mutation rate is 0.5 and crossover rate is 0.8). Figure 7 depicts the execution time of SGA, Min-Min, Max-Min and AIGA. Overall, AIGA has the minimum value of makespan. The execution time of these algorithms extend as the number of tasks increases. When there is a small number of tasks, Min-Min and Max-Min perform more efficiently. When the number of tasks exceeds 20,000, AIGA costs least computation time.

Table 5. Fitness of optimal solution.

Number of Tasks	Min-Min	Max-Min	SGA	AIGA
100	128.23	118.2	118.756	117.592
5000	5993.54	5986.27	6242.097	5984.747
10,000	11,978.1	11,973.29	12,408.91	11,971.14
15,000	17,979.93	17,974.44	22,354.83	17,971.24
20,000	23,691.6	23,689.81	35,008.51	23,686.19
25,000	29,812.08	29,808.85	50,104.01	29,804.54
30,000	35,720.45	35,722.94	66,403.67	35,717.09
35,000	41,669.28	41,669.37	82,303.76	41,663.41
40,000	47,685.03	47,689.28	101,092.9	47,682.35
45,000	53,759.32	53,770.26	122,641.1	53,761.93
50,000	59,652.83	59,661.76	144,997.2	59,653.09
60,000	71,799.67	71,809.81	195,167.3	71,796.99
70,000	83,595.83	83,607.86	241,417.4	83,590.55
80,000	95,387.53	95,401.51	292,828.4	95,380.96
90,000	107,296.4	107,312.2	342,214.8	107,287.4

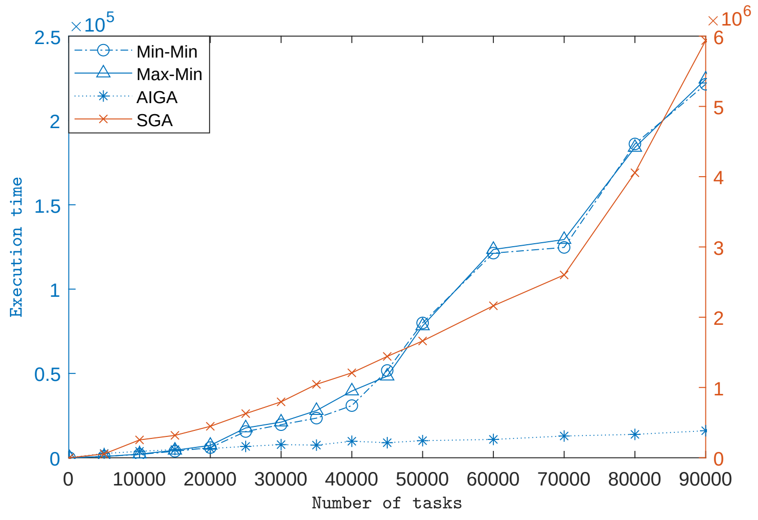


Figure 7. Execution time of Min-Min, Max-Min, AIGA and SGA.

5. Conclusions and Future Work

For the task scheduling problem in Cloud environment, the time consumption has impact on the users' payment and the energy consumed. It is important to optimize allocation of resources to reduce the makespan of processing a batch of tasks. Although GA or other bio-inspired algorithms are effective to solve NP-complete problems, it converges slowly when high-dimensional data are involved. To optimize large-scale task scheduling problem in Cloud environment with less makespan and computation time, we proposed an adaptive incremental Genetic algorithm. Our method called AIGA based on genetic algorithm which has adaptive probability of mutation rate and crossover rate can provide feasible solutions for the allocation of large numbers of tasks with less computation time.

We model the task scheduling as an objective optimization problem where the number of tasks corresponds to the dimensions of the problem. Extensive experiments had been designed to evaluate the performance of AIGA. Compared with two traditional scheduling methods, Min-Min and Max-Min, AIGA can find better solutions and costs less computation time as the number of tasks is very large. We also contrast different parameters of Standard GA by changing the mutation rate and crossover rate. AIGA steadily outperforms SGA in optimizing makespan despite the various control parameters. Two other meta-heuristic algorithms, Simulated Annealing (SA) and Artificial Bee Colony (ABC) algorithm, had been implemented. Experimental results show that the proposed AIGA performs best in terms of minimum objective function value, and has faster convergence speed than SA and ABC.

As part of our future work, we intend to establish a sophisticated mathematical model to describe the task scheduling problem on IaaS, such as introducing data or control dependences between tasks. Furthermore, we will consider evaluating AIGA for other objective functions such as energy consumption, resource utilization or optimize multiple objectives simultaneously.

Author Contributions: K.D., S.F. and S.S.-U.G. conceived and designed the experiments; K.D. performed the experiments and wrote the paper; and S.W.I.S. and W.S. provided guidance for this paper.

Acknowledgments: The authors are grateful for financial support from the research grants: (1) "Nature-Inspired Computing and Metaheuristics Algorithms for Optimizing Data Mining Performance" from the University of Macau (grant No. MYRG2016-00069-FST); (2) "Improving the Protein-Ligand Scoring Function for Molecular Docking by Fuzzy Rule-based Machine Learning Approaches" from the University of Macau (grant No. MYRG2016-00217-FST); and (3) "A Scalable Data Stream Mining Methodology: Stream-based Holistic Analytics and Reasoning in Parallel" from FDCT, Macau government (grant No. FDCT/126/2014/A3).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mell, P.; Grance, T. The NIST Definition of Cloud Computing. Available online: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf (accessed on 15 March 2018).
2. Radu, L.D. Green Cloud Computing: A Literature Survey. *Symmetry* **2017**, *9*, 295. [CrossRef]
3. Encalada, W.L.; Sequera, J.L.C. Model to Implement Virtual Computing Labs via Cloud Computing Services. *Symmetry* **2017**, *9*, 117. [CrossRef]
4. Gao, J.; Yi, R. Cloud generalized power ordered weighted average operator and its application to linguistic group decision-making. *Symmetry* **2017**, *9*, 156. [CrossRef]
5. Chu, P.M.; Cho, S.; Fong, S.; Park, Y.W.; Cho, K. 3D Reconstruction Framework for Multiple Remote Robots on Cloud System. *Symmetry* **2017**, *9*, 55. [CrossRef]
6. Aarts, E.; Korst, J.; Michiels, W. Simulated annealing. In *Search Methodologies*; Springer: Boston, MA, USA, 2005; pp. 187–210.
7. He, X.; Sun, X.; Von Laszewski, G. QoS guided min-min heuristic for grid task scheduling. *J. Comput. Sci. Tech.* **2003**, *18*, 442–451. [CrossRef]
8. Mao, Y.; Chen, X.; Li, X. Max-min task scheduling algorithm for load balance in cloud computing. In Proceedings of the International Conference on Computer Science and Information Technology, Barcelona, Spain, 22–24 December 2014; Springer: New Delhi, India, 2014; pp. 457–465.
9. Zhan, S.; Huo, H. Improved PSO-based task scheduling algorithm in cloud computing. *J. Inf. Comput. Sci.* **2012**, *9*, 3821–3829.
10. Zuo, X.; Zhang, G.; Tan, W. Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud. *IEEE Trans. Autom. Sci. Eng.* **2014**, *11*, 564–573. [CrossRef]
11. Chen, S.M.; Chien, C.Y. Parallelized genetic ant colony systems for solving the traveling salesman problem. *Expert Syst. Appl.* **2011**, *38*, 3873–3883. [CrossRef]
12. Tsai, P.W.; Pan, J.S.; Chen, S.M.; Liao, B.Y.; Hao, S.P. Parallel cat swarm optimization. In Proceedings of the Seventh International Conference on Machine Learning and Cybernetics, Kunming, China, 12–15 July 2008; pp. 3328–3333.
13. Tsai, P.W.; Pan, J.S.; Chen, S.M.; Liao, B.Y. Enhanced parallel cat swarm optimization based on the Taguchi method. *Expert Syst. Appl.* **2012**, *39*, 6309–6319. [CrossRef]
14. Wu, L.; Wang, Y.J.; Yan, C.K. Performance Comparison of Energy-aware task scheduling with GA and CRO algorithms in Cloud Environment. *Appl. Mech. Mater.* **2014**, *596*, 204–208. [CrossRef]
15. Mahmood, A.; Khan, S.A. Hard Real-Time Task Scheduling in Cloud Computing Using an Adaptive Genetic Algorithm. *Computers* **2017**, *6*, 15. [CrossRef]
16. Karaboga, D.; Basturk, B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *J. Glob. Optim.* **2007**, *39*, 459–471. [CrossRef]
17. Babu, K.R.; Samuel, P. Enhanced bee colony algorithm for efficient load balancing and scheduling in cloud. In *Innovations in Bio-Inspired Computing and Applications*; Springer: Cham, Switzerland, 2016; pp. 67–78.

18. Navimipour, N.J. Task scheduling in the cloud environments based on an artificial bee colony algorithm. In Proceedings of the 2015 International Conference on Image Processing, Production and Computer Science, Istanbul, Turkey, 3–4 June 2015; pp. 38–44.
19. Fidanova, S. Simulated annealing for grid scheduling problem. In Proceedings of the IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing, Sofia, Bulgaria, 3–6 October 2006; pp. 41–45.
20. Mandal, T.; Acharyya, S. Optimal task scheduling in cloud computing environment: Meta heuristic approaches. In Proceedings of the 2015 2nd International Conference on Electrical Information and Communication Technology (EICT), Khulna, Bangladesh, 10–12 December 2015; pp. 24–28.
21. Wang, S.; Liang, K.; Liu, J.K.; Chen, J.; Yu, J.; Xie, W. Attribute-based data sharing scheme revisited in cloud computing. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 1661–1673. [CrossRef]
22. Liang, K.; Liu, J.K.; Lu, R.; Wong, D.S. Privacy concerns for photo sharing in online social networks. *IEEE Internet Comput.* **2015**, *19*, 58–63. [CrossRef]
23. Liu, J.; Huang, X.; Liu, J.K. Secure sharing of personal health records in cloud computing: Ciphertext-policy attribute-based signcryption. *Future Gener. Comput. Syst.* **2015**, *52*, 67–76. [CrossRef]
24. Wen, J.; Lu, L.; Casale, G.; Smirni, E. Less can be more: Micro-managing vms in amazon EC2. In Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing, New York, NY, USA, 27 June–2 July 2015; pp. 317–324.
25. Amazon EC2 Pricing. Available online: https://aws.amazon.com/ec2/pricing/on-demand/?nc1=h_ls (accessed on 1 April 2018).
26. Previous Generation Instances. Available online: https://aws.amazon.com/ec2/previous-generation/?nc1=h_ls (accessed on 1 April 2018).
27. Srinivas, M.; Patnaik, L.M. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Trans. Syst. Man Cybern.* **1994**, *24*, 656–667. [CrossRef]