A Genetic-Algorithm-Based Approach for Task Migration in Pervasive Clouds

Weishan Zhang,1 Shouchao Tan,1 Qinghua Lu,1,2 Xin Liu,1 and Wenjuan Gong1

1Department of Software Engineering, China University of Petroleum, No. 66 Changjiang West Road, Qingdao 266580, China
2Software Systems Research Group, National ICT Australia (NICTA), 13 Garden Street, Eveleigh, NSW 2015, Australia

Correspondence should be addressed to Weishan Zhang; zhangws@upc.edu.cn

Received 10 November 2014; Accepted 10 February 2015

Academic Editor: Xiuzhen Cheng

Copyright © 2015 Weishan Zhang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Pervasive computing is converging with cloud computing which becomes pervasive cloud computing as an emerging computing paradigm. Users can run their applications or tasks in pervasive cloud environment in order to gain better execution efficiency and performance leveraging powerful computing and storage capacities of pervasive clouds through task migration. During task migration, there are possibly a number of conflicting objectives to be considered when making migration decisions, such as less energy consumption and quick response, in order to find an optimal migration path. In this paper, we propose a genetic algorithms- (GAs-) based approach which is effective in addressing multiobjective optimization problems. We have performed some preliminary evaluations of the proposed approach which shows quite promising results, using one of the classical genetic algorithms. The conclusion is that GAs can be used for decision making in task migrations in pervasive clouds.

1. Introduction

Pervasive cloud computing is a converged computing paradigm, by leveraging powerful computing and storage capacities of cloud computing [1]. In order to realize the pervasive cloud computing functionalities to improve the processing and storage capabilities of small devices, task migration [2] is an important feature of pervasive clouds which migrates computing intensive tasks or high-volume-storage-demanding tasks to backend powerful nodes.

There are a number of task migration solutions proposed so far [3], such as OSGi-PC proposed in [4, 5] and CloneCloud [6]. These approaches focus on different aspects of application execution in pervasive cloud environments, including resource and service discovery mechanism, and application partition and tasks migration. But none of these proposed solutions provide a good solution for decision making on task migration in pervasive clouds. These solutions could not provide a scalable approach for obtaining optimal migration paths while tasks migrate, especially in a possibly large problem space like a smart transportation management application.

Genetic algorithms are one of the most successful computational intelligence branch, which can potentially work for a large-scale multiobjective optimization problems. Therefore, in this paper we propose a genetic algorithms-based approach for task migration in pervasive clouds which can be used to make decisions on how to assign tasks to cloud nodes. We have performed some preliminary evaluations of the proposed approach which shows quite promising results, using one of the classical genetic algorithms. The conclusion is that GAs can be used for decision making in task migrations in pervasive clouds.

The remainder of the paper is organized as follows. We introduce task migration in pervasive clouds and the design of our GAs-based approach in Section 2. Next, Section 3 presents how genetic algorithms are used in decision making for task migration problem. Section 4 evaluates the implementation that shows our approach can perform acceptably. Related work is described in Section 5. Section 6 concludes the paper and outlines the future work.
2. Task Migration and Design of GAs-Based Approach

2.1. Task Migration Overview. Tasks may need resources that are not available to them in their current location. Task migration in pervasive clouds is to reallocate these tasks to resource-rich computing nodes in the cloud environment before their execution. During the execution period, it may adjust running nodes of some tasks according to tasks' execution contexts. In task migration, one user application may be divided into some tasks following certain rules, so that the complex application can be executed in a distributed way. In order to ensure that the tasks can be executed correctly, task migration involves status, data and file transmission, and returning execution results back.

A further reason for migration may be that less power is used in mobile devices if tasks are offloaded. Although mobile device is becoming more powerful, the resource demanding surpasses this trend, for example, computing and resources of intensive applications including rich media applications such as photo and video editing and synthesis, scene recognition, object recognition, and image search [7]. This makes mobile applications becoming more and more complicated [8, 9], which requires enhancing user experiences and response time utilizing backend powerful devices.

2.2. Choosing Cloud Simulation Platform. Real pervasive clouds environment and tasks execution in it are very complex. Alternatively, we choose a simulation tool CloudSim [10] which makes it possible to evaluate migration strategies prior to software development. CloudSim is a generalized and extensible framework that enables seamless modeling, simulation, and experimentation of emerging Cloud computing infrastructures and application services. CloudSim framework models computing entities, cloud services, user tasks, and user tasks' executions are simulated by CloudSim.

Using CloudSim to do simulation, we focus on tasks' execution and obtain test data without taking into account low level details related to cloud-based infrastructures and services. There are certain computing nodes-VMs and user tasks-Cloudlets. After specific simulations, we can obtain each task's running time on all of the VMs.

2.3. Genetic Algorithms and GA Frameworks. Genetic algorithms [11] are used to obtain optimized solutions from a number of candidate solutions. GAs are inspired by evolutionary theory: weak and unfit species are faced with extinction by natural selection, and the strong ones have greater opportunity to pass their genes to future generations via reproduction [12]. Compared with other classic or optimization methods, GAs have it specific advantages in terms of its broad applicability, ease of use, and global perspective [13]. They are particularly useful to for single-objective and multiobjective problems which are difficult or impossible to get an exact solution [14]. So we choose it as our decision making algorithm.

In GA terminology, a solution is called an individual or a chromosome. A chromosome corresponds to a unique solution in the solution space. GAs operate with a collection of chromosomes, called a population, and use two operators to generate new solutions from existing ones: crossover and mutation. There are many GAs proposed, such as NSGA-II [15] and MOCell [16].

In our implementation, we choose to use Java based GA frameworks. There are some popular implementations, such as JGap [17, 18], ECJ [19] and JMetal [20]. When compared with JMetal's counter-parts, the design of JMetal has a good separation of concerns in terms of its easiness for applying different GAs after a problem is abstracted. Therefore, JMetal is chosen as the GA framework in our approach.

The overall decision-making process is as follows: firstly CloudSim is used to model the computing nodes of the testing cloud environment and user tasks. Then some simulations are conducted to collect tasks execution information, which is the time array which shows all the time taken by each task run on all VMs. And then this time information is used in JMetal where GAs will give optimized solutions for task allocation. Figure 1 shows the overall design process, and the working flow of GAs-based decision making for task migration is shown in Figure 2.

In the following section we will describe how the GAs-based approach finds optimized solutions to the task migration problem. The definition of task migration problem with GAs and the algorithm execution process are explained. And we will also discuss another two decision making policies used for task assignment.

3. GAs-Based Approach Implementation

3.1. Task Migration Scenario. Suppose we have the following hypothetical scenario: there are 28 user tasks and 12 computing devices which have specific processing ability. This will be considered as one of the examples which is discussed in detail in this paper. Considering the features of CloudSim, user tasks are modeled as cloudlets and computing devices are a series of VMs which belong to specific hosts in the cloud. We assumed there are two hosts and the former 6 VMs are allocated in Host1 and the others are in Host2 which has more powerful process capability and stable network. Cloudlet characteristics include the length of the instruction set which is related to computing process time and input files’
To test our decision making approach, we assume task characteristics and VM characteristics as shown in Tables 1 and 2. In Table 1, "CloudletID" is the identification of the cloudlet while "CLen" is how many instructions in the cloudlet, "CISize" and "COSize" are the size of input and output files for this cloudlet. In Table 2, "HostID" shows which host the VM is allocated. We can find how many million instructions in one second the VM can process with the "Mips" and "PesN" is the CPU number and "Size" indicates the size of the virtual machine.

Table 3 shows the two hosts information, in which "HostID" is the ID number identifying a specific host.

“Datacenter” means which data center the host belongs to, and there are only one data center “HostDatacenter” we created. “Bw” stands for the bandwidth characteristic of the host. "Pes" is how many CPUs the host has and "Mips" is the total process capability.

As illustrated previously, we have 28 cloudlets and 12 VMs. For each cloudlet, it can be allocated to any of the 12 VMs. So there will be \(12^{28}\) allocation schemes. And our goal for task migration is to choose the best allocation scheme from all of the possible schemes using GAs. Before we make this decision, we must know whether this cloudlet can be executed in the VM, if it can, how long it will take to finish.

### Table 1: Cloudlets characteristics.

| CID | CLen | CISize | COSize | CID | CLen | CISize | COSize |
|-----|------|--------|--------|-----|------|--------|--------|
| 0   | 1000 | 1000   | 1000   | 14  | 5000 | 1000   | 1000   |
| 1   | 1000 | 2000   | 1000   | 15  | 5010 | 1000   | 1000   |
| 2   | 1000 | 3000   | 16     | 16  | 5010 | 1010   | 1000   |
| 3   | 1000 | 4000   | 19     | 17  | 5010 | 1010   | 1010   |
| 4   | 1000 | 2000   | 1000   | 18  | 5000 | 1010   | 1010   |
| 5   | 1000 | 4000   | 1000   | 19  | 10000| 1000   | 1000   |
| 6   | 1000 | 6000   | 1000   | 20  | 10000| 1000   | 2000   |
| 7   | 1000 | 2000   | 2000   | 21  | 10000| 1000   | 5000   |
| 8   | 1000 | 4000   | 22     | 22  | 10000| 5000   | 5000   |
| 9   | 2000 | 1000   | 23     | 23  | 10000| 2000   | 2000   |
| 10  | 2000 | 2000   | 24     | 24  | 100000|100   | 100   |
| 11  | 2000 | 1000   | 25     | 25  | 100000|100   | 100   |
| 12  | 2000 | 2000   | 26     | 26  | 100000|100   | 100   |
| 13  | 5000 | 100    | 100    | 100 | 100000|1000  | 1000  |

### Table 2: VMs characteristics.

| VmID | HostID | Mips | PesN | Ram/MB | Bw | Size/MB |
|------|--------|------|------|--------|----|---------|
| 0    | 0      | 200  | 1    | 512    | 500| 500     |
| 1    | 0      | 200  | 1    | 512    | 1000|500 |
| 2    | 0      | 200  | 1    | 512    | 2000|500 |
| 3    | 0      | 400  | 1    | 512    | 500 | 500     |
| 4    | 0      | 400  | 1    | 768    | 500 | 500     |
| 5    | 0      | 600  | 1    | 1024   | 500 | 500     |
| 6    | 0      | 600  | 1    | 512    | 500 | 500     |
| 7    | 1      | 1000 | 1    | 512    | 1000|1000    |
| 8    | 1      | 1000 | 1    | 768    | 1000|1000    |
| 9    | 1      | 1000 | 1    | 768    | 1000|1000    |
| 10   | 1      | 2000 | 1    | 1024   | 2000|2000    |
| 11   | 1      | 2000 | 2    | 1024   | 2000|2000    |

### Table 3: Hosts characteristics.

| HostID | Datacenter | HostDatacenter | RAM/MB | Storage | Bw | PeS |
|--------|------------|----------------|--------|---------|----|-----|
| 0      | 32768      | 10000000       | 10000  | 10000   | 10000000 |
| 1      | 2048       | 10000000       | 10000  | 10000000 |

As illustrated previously, we have 28 cloudlets and 12 VMs. For each cloudlet, it can be allocated to any of the 12 VMs. So there will be \(12^{28}\) allocation schemes. And our goal for task migration is to choose the best allocation scheme from all of the possible schemes using GAs. Before we make this decision, we must know whether this cloudlet can be executed in the VM, if it can, how long it will take to finish.
this cloudlet, how many resources it will consume, and how much the energy consumption is. At present, we choose one important factor of execution performance: the time cost as a decision indicator. We simulate the execution of all the cloudlets in each VM of the 12 VMs. Then we get a two-dimensional time array which is shown in Table 4.

The decision making for task migration aims at getting the best allocation scheme which can make the system have the maximum processing efficiency, so that the cloud system can finish all the tasks with the minimum time consumption, in our case. This process may also involve the optimization for energy consumption, for security and privacy, and even all of them together. This combined optimization work will be done in our future work later on.

3.2. Problem and Optimization Objectives Formulation. In pervasive clouds environment, computing nodes are running in parallel. In other words, all the VMs in CloudSim start in the same time. And then they begin to process the tasks (cloudlets) assigned to themselves. But when more than one cloudlet is assigned to one VM, they will be executed one by one. Another important thing about task execution in pervasive clouds is that there are file transfers between VMs when task migrates which takes time. But when there are more than one cloudlet that are executed in the VM, only the first one’s file transfers consume time, because the following file transfer in tasks can be done during the execution of the first cloudlet. This process needs to calculate the predicted finish time of each VM. The time is the sum of execution time of all tasks assigned to the VM. In other words, it is the finish time of the last task allocated in the VM. For example, for a given VM \( i \), the predicted finish time \( V T_i \) is \( \sum_{j=0}^{m} C T_{ji} \times E(j, i) \). The meaning of each symbol in the formula is as follows.

(i) \( i \) represents the sequence number of a VM with the scope of \([0 - n]\).

(ii) \( j \in [0 - m] \) represents the sequence number of a cloudlet.

(iii) \( C T_{ji} \) is the time cloudlet \( j \) consumes when running in the VM \( i \) calculated by \( C T_{ji} = E T_{ji} + (C I S + COS)/B w_{ji} \). It consists of two parts: execution time in the choosing VM \( j \) and data transfer time which are decided by input and output size of the task file and the network bandwidth of the VM.

(iv) \( E(i, j) = 1 \) when cloudlet \( j \) is assigned to VM \( i \); otherwise, \( E(j, i) = 0 \).

If the start time of clouds system is 0, then at time \( T = \max VT_i \), the clouds system will finish the execution of all the cloudlets. In the scenario under consideration, \( n = 11 \) and \( m = 27 \).

In order to get optimized solutions as task migration decision, we choose these solutions which have the minimum finishing time \( T \). This is the objective of the task migration problem.

3.3. Chromosome Encoding and Fitness Evaluations. A chromosome corresponds to a unique solution in the solution space. GAs can typically make use of booleans, real numbers, and integers to encode a chromosome [20]. The representation of chromosome in our case is using integer (starting from 0) sharing the idea of [21]. That is to say, we are using a integer vector \( V = [V_0, V_1, \ldots, V_p, \ldots, V_{p-1}] \) (where \( p \) is the number of decision variables, and in our case, it is 28: the number of tasks) to represent a solution. \( V_i \) is a natural number and acts as a pointer to the sequence of the VM to which the \( i \)th cloudlet is assigned. For example, a chromosome \([11, 7, 2, 7, 4, 0, 5, 1, 7, 11, 5, 1, 3, 5, 4, 6, 11, 11, 3, 2, 4, 4, 5, 8, 11, 10, 9]\) represents that a solution chooses the twelfth VM whose sequence number is 11 to execute Cloudlet 0 and chooses the eighth VM for Cloudlet 1, the tenth VM as processing node of the last task, and so on. Based on the chosen allocation strategies, the GAs then decide its fitness using the objective equations as introduced in Section 3.2.

3.4. Testing Environments and Parameter Settings for Evaluations. In our evaluations, JMetal 4.5 is used and the classical GA NSGAII is chosen as a concrete algorithm. Algorithm parameters setting in our implementation includes the maximum evaluations, the cross over probability, and the mutation operator, as shown in Table 5.

As choosing integer chromosome, so Single point crossover, Bit flip mutation, and Binary tournament2 operators are selected drawing on the experience of [22]. Reference [14] talks about how the operators work to make a change to the chromosome in GAs.

4. Evaluation and Discussion

From Table 4 we can see that time consumption is different for different cloudlets running on the same VM, and the same task takes different time to finish its execution in different VMs. Analysing the execution of tasks in VMs, it is clear that the faster the VM is, the less execution time a task takes. The bigger a task (more instructs and bigger files) is, the more time it consumes. And with better network (high bandwidth), the data transfer time is reduced. This is consistent with our theoretical and actual situations.

4.1. Greedy and Simple Policies. In order to compare the performance of NSGA-II with other algorithms, we choose both greedy policy and simple allocation method for comparisons. Greedy policy is an extension to CloudSim and simple allocation method is included in the original CloudSim framework. Simple allocation method is a sequential allocation policy: all tasks are assigned to a group of virtual machine in order. The first task is assigned to the first VM, the second to the second, and so on. When all the virtual machine has been assigned one task, then the second round will start, and we assign next task to the first VM. In greedy policy, a matrix time will be defined to describe execution time for all tasks. Time \([i][j]\) shows the execution time of task \( i \) in VM \( j \). Before initializing the time matrix, firstly we should sort the tasks in descending order according to the instruction set length and sort all the VMs in ascending order according to the processing capacity. From the first task (after sorting),
Table 4: Cloudlet to VM execution time.

| time | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 |
|------|----|----|----|----|----|----|----|----|----|----|----|----|
| cid  |    |    |    |    |    |    |    |    |    |    |    |    |
| 0    | 5.4| 5.2| 5.1| 2.9| 2.9| 2.07| 2.07| 1.2| 1.2| 1.2| 0.6| 0.6 |
| 1    | 5.6| 5.3| 5.15| 3.1| 3.1| 2.27| 2.27| 1.3| 1.3| 1.3| 0.65| 0.65 |
| 2    | 5.8| 5.4| 5.2| 3.3| 3.3| 2.47| 2.47| 1.4| 1.4| 1.4| 0.7 | 0.7 |
| 3    | 7.2| 6.1| 5.55| 4.7| 4.7| 3.87| 3.87| 2.1| 2.1| 2.1| 1.05| 1.05 |
| 4    | 5.6| 5.3| 5.15| 3.1| 3.1| 2.27| 2.27| 1.3| 1.3| 1.3| 0.65| 0.65 |
| 5    | 6  | 5.5| 5.25| 3.5| 3.5| 2.67| 2.67| 1.5| 1.5| 1.5| 0.75| 0.75 |
| 6    | 6.4| 5.7| 5.35| 3.9| 3.9| 3.07| 3.07| 1.7| 1.7| 1.7| 0.85| 0.85 |
| 7    | 5.8| 5.4| 5.2| 3.3| 3.3| 2.47| 2.47| 1.4| 1.4| 1.4| 0.7 | 0.7 |
| 8    | 6.6| 5.8| 5.4| 4.1| 4.1| 3.27| 3.27| 1.8| 1.8| 1.8| 0.9 | 0.9 |
| 9    | 10.4| 10.2| 10.1| 5.4| 5.4| 3.73| 3.73| 2.2| 2.2| 2.2| 1.1 | 1.1 |
| 10   | 10.8| 10.4| 10.2| 5.8| 5.8| 4.13| 4.13| 2.4| 2.4| 2.4| 1.2 | 1.2 |
| 11   | 10.4| 10.2| 10.1| 5.4| 5.4| 3.73| 3.73| 2.2| 2.2| 2.2| 1.1 | 1.1 |
| 12   | 10.4| 10.2| 10.1| 5.4| 5.4| 3.73| 3.73| 2.2| 2.2| 2.2| 1.1 | 1.1 |
| 13   | 25.04| 25.02| 25.01| 12.54| 12.54| 8.37| 8.37| 5.02| 5.02| 5.02| 2.51| 2.51 |
| 14   | 25.4| 25.2| 25.1| 12.9| 12.9| 8.73| 8.73| 5.2| 5.2| 5.2| 2.6 | 2.6 |
| 15   | 25.45| 25.25| 25.15| 12.93| 12.93| 8.75| 8.75| 5.21| 5.21| 5.21| 2.6 | 2.6 |
| 16   | 25.45| 25.25| 25.15| 12.93| 12.93| 8.75| 8.75| 5.21| 5.21| 5.21| 2.61| 2.61 |
| 17   | 25.45| 25.25| 25.15| 12.93| 12.93| 8.75| 8.75| 5.21| 5.21| 5.21| 2.61| 2.61 |
| 18   | 25.4| 25.2| 25.1| 12.9| 12.9| 8.74| 8.74| 5.2| 5.2| 5.2| 2.6 | 2.6 |
| 19   | 50.4| 50.2| 50.1| 25.4| 25.4| 17.07| 17.07| 10.2| 10.2| 10.2| 5.1 | 5.1 |
| 20   | 50.6| 50.3| 50.15| 25.6| 25.6| 17.27| 17.27| 10.3| 10.3| 10.3| 5.15| 5.15 |
| 21   | 51.2| 50.6| 50.3| 26.2| 26.2| 17.87| 17.87| 10.6| 10.6| 10.6| 5.3 | 5.3 |
| 22   | 52 | 51| 50.5| 27| 27| 18.67| 18.67| 11| 11| 11| 5.5 | 5.5 |
| 23   | 51.3| 50.9| 50.7| 26.05| 26.05| 17.63| 17.63| 10.5| 10.5| 10.5| 5.25| 5.25 |
| 24   | 500.04| 500.02| 500.01| 250.04| 250.04| 166.71| 166.71| 100.02| 100.02| 100.02| 50.01| 50.01 |
| 25   | 500.22| 500.11| 500.06| 250.22| 250.22| 166.89| 166.89| 100.11| 100.11| 100.11| 50.05| 50.05 |
| 26   | 500.22| 500.11| 500.06| 250.22| 250.22| 166.89| 166.89| 100.11| 100.11| 100.11| 50.05| 50.05 |
| 27   | 500.22| 500.11| 500.06| 250.22| 250.22| 166.89| 166.89| 100.11| 100.11| 100.11| 50.05| 50.05 |

Table 5: Parameters for NSGA-II.

| Parameter               | Value       |
|-------------------------|-------------|
| Population size         | 50          |
| Max evaluations         | 2000        |
| Cross-over operator     | Single point cross-over |
| Cross-over probability  | 0.9         |
| Mutation operator       | Bit flip mutation |
| Mutation probability    | 1/number of variables |
| Selection operator      | Binary tournament 2 |

4.2 Result Evaluation. In order to verify and evaluate each allocation policy, we did simulations in CloudSim. The results are shown in Tables 6, 7, and 8, respectively.

From these three tables, we can find that the scheme obtained from GAs-based approach takes 100.21 time units to finish all 12 tasks while the other two schemes take 100.32 and 530.41 time units. So GAs-based approach can do the best decision and make the system have the fastest execution efficiency. According to the GAs-based scheme, user tasks can be finished as fast as possible and therefore users can have better experience and can be more satisfied.

In order to be more persuasive and trusted, we add another set of tasks which contains 12 cloudlet and another group of vms which consist of 7 VMs as test data as shown in Table 9 and in Table 10. So in total we can have four combinations. Except for the aforementioned scenario discussed in detail (28 cloudlets and 12 VMs), we have another three experiments: experiment one where 28 cloudlets are allocated to 7 VMs, experiment two where 12 cloudlets are
allocated to 12 VMs, and experiment three where 12 cloudlets are allocated to 7 VMs. The results with 3 different allocation policies are shown in Table 11. With the total four results, we can see that GA decision can obtain better allocation scheme. This indicates that choosing genetic algorithm as decision making algorithm for task migration is efficient to some extent.

5. Related Work

Task migration is becoming popular with the development of cloud computing. It aims at getting better execution of user applications or tasks. Reference [23] developed an offloading inference engine that can do efficient partitioning of applications. The offloading inference engine dynamically partitions the application, and offload part of the application execution with data to a powerful nearby surrogates. This allows the application to be delivered in a pervasive computing environment without significant fidelity degradation or expensive application rewriting.

MAUI (mobile assistance using infrastructure) [24] is an architecture where mobile program code can be offloaded to powerful infrastructure. MAUI decides at runtime whose methods should be remotely executed, driven by an optimization engine that achieves the best energy savings possible under the mobile device’s current connectivity constraints. It uses integer linear programming to do the optimization which may incur quite some overhead as pointed by [24]. Instead the GA-based approach can be quite efficient as evaluated in [22].

CloneCloud [6] is a system automatically migrating mobile applications dynamically to cloud. It uses remote cloud resources and partitions application at thread level to obtain fine granularity and flexibility. The basic idea of CloneCloud is rewriting the program. At runtime, it will choose threads to do migration, the rest will still be executed locally. Integer linear programming is also used to obtain optimization migration paths which may have high performance overhead.

There are also some GAs-based approaches proposed in task scheduling and migration in mobile cloud computing. In [25] the author proposes a new concept “universal mobile service cell” (UMSL) as the abstraction for mobile cloud computing. And then UMSL is chosen as scheduling unit instead of tasks or processes in order to have high level of abstraction to shield the heterogeneity and distribution of

---

**Table 6: NSGAII-based approach.**

| CID | VMID | Time | STime | FTime |
|-----|------|------|-------|-------|
| 6   | 10   | 0.85 | 0.1   | 0.95  |
| 1   | 11   | 1.3  | 0.1   | 1.4   |
| 10  | 5    | 2.07 | 0.1   | 2.17  |
| 7   | 8    | 1    | 1.4   | 2.4   |
| 2   | 6    | 2.47 | 0.1   | 2.57  |
| 4   | 3    | 3.1  | 0.1   | 3.2   |
| 5   | 5    | 1.67 | 2.17  | 3.83  |
| 3   | 6    | 1.67 | 2.57  | 4.23  |
| 14  | 10   | 2.5  | 1.95  | 4.45  |
| 16  | 9    | 5.21 | 0.1   | 5.31  |
| 8   | 6    | 1.67 | 4.23  | 5.9   |
| 13  | 8    | 5    | 2.4   | 7.4   |
| 9   | 3    | 5    | 3.2   | 8.2   |
| 10  | 6    | 3.33 | 5.9   | 9.23  |
| 12  | 2    | 10.1 | 0.1   | 10.2  |
| 17  | 5    | 8.35 | 3.83  | 12.18 |
| 15  | 3    | 12.52| 8.2   | 20.72 |
| 18  | 0    | 25.4 | 0.1   | 25.5  |
| 19  | 5    | 16.67| 12.18 | 28.85 |
| 20  | 5    | 16.67| 28.85 | 45.52 |
| 25  | 11   | 50.05| 0.1   | 50.15 |
| 27  | 11   | 50.16| 0.1   | 50.26 |
| 23  | 1    | 50.9 | 0.1   | 51    |
| 24  | 10   | 50   | 4.45  | 54.45 |
| 21  | 2    | 50   | 10.2  | 60.19 |
| 22  | 5    | 16.67| 45.52 | 62.18 |
| 26  | 7    | 100.11| 0.1 | 100.21|

**Table 7: Greedy policy based approach.**

| CID | VMID | Time | STime | FTime |
|-----|------|------|-------|-------|
| 23  | 7    | 10.5 | 0.1   | 10.6  |
| 15  | 3    | 12.92| 0.1   | 13.02 |
| 19  | 6    | 17.07| 0.1   | 17.17 |
| 20  | 5    | 17.27| 0.1   | 17.37 |
| 9   | 3    | 5    | 13.02 | 18.02 |
| 21  | 7    | 10   | 10.6  | 20.6  |
| 10  | 7    | 2    | 20.6  | 22.6  |
| 11  | 3    | 5    | 18.02 | 23.02 |
| 12  | 7    | 2    | 22.6  | 24.6  |
| 13  | 2    | 25.01| 0.1   | 25.11 |
| 14  | 1    | 25.2 | 0.1   | 25.3  |
| 18  | 0    | 25.4 | 0.1   | 25.5  |
| 0   | 3    | 2.59 | 23.02 | 25.61 |
| 1   | 7    | 1.01 | 24.6  | 25.61 |
| 16  | 6    | 8.44 | 17.17 | 25.61 |
| 17  | 5    | 8.35 | 17.37 | 25.72 |
| 2   | 7    | 1    | 25.61 | 26.61 |
| 22  | 4    | 27   | 0.1   | 27.1  |
| 3   | 6    | 1.67 | 25.61 | 27.28 |
| 4   | 5    | 1.67 | 25.72 | 27.39 |
| 5   | 7    | 1    | 26.61 | 27.61 |
| 7   | 3    | 2.5  | 25.61 | 28.11 |
| 8   | 7    | 1    | 27.61 | 28.61 |
| 6   | 4    | 2.5  | 27.1  | 29.6  |
| 24  | 11   | 50.01| 0.1   | 50.11 |
| 25  | 10   | 50.12| 0.1   | 50.22 |
| 26  | 9    | 100.11| 0.1  | 100.21|
| 27  | 8    | 100.22| 0.1  | 100.32|
Table 8: Simple allocation method.

| CID | VMID | Time | STime | FTime |
|-----|------|------|-------|-------|
| 11  | 11   | 1.1  | 0.1   | 1.2   |
| 10  | 10   | 1.21 | 0.1   | 1.31  |
| 7   | 7    | 1.4  | 0.1   | 1.5   |
| 8   | 8    | 1.8  | 0.1   | 1.9   |
| 9   | 9    | 2.2  | 0.1   | 2.3   |
| 5   | 5    | 2.67 | 0.1   | 2.77  |
| 6   | 6    | 3.07 | 0.1   | 3.17  |
| 4   | 4    | 3.18 | 0.1   | 3.28  |
| 3   | 3    | 4.7  | 0.1   | 4.8   |
| 2   | 2    | 5.2  | 0.1   | 5.3   |
| 1   | 1    | 5.3  | 0.1   | 5.4   |
| 23  | 11   | 5.3  | 0.1   | 5.4   |
| 0   | 0    | 5.4  | 0.1   | 5.5   |
| 17  | 10   | 5    | 1.31  | 6.31  |
| 19  | 7    | 10   | 1.5   | 11.5  |
| 18  | 6    | 8.33 | 3.17  | 11.5  |
| 20  | 8    | 10   | 1.9   | 11.9  |
| 21  | 9    | 10   | 2.3   | 12.3  |
| 12  | 0    | 10   | 5.5   | 15.5  |
| 16  | 4    | 12.52| 3.28  | 15.8  |
| 15  | 3    | 12.52| 4.8   | 17.32 |
| 14  | 2    | 25   | 5.3   | 30.3  |
| 13  | 1    | 25.01| 5.4   | 30.41 |
| 27  | 3    | 250  | 17.32 | 267.32|
| 24  | 0    | 500  | 15.5  | 515.5 |
| 26  | 2    | 500  | 30.3  | 530.3 |
| 25  | 1    | 500  | 30.41 | 530.41|

Table 9: Another group of Cloudlets.

| CloudletID | CLengt  | CISize | COSize |
|------------|---------|--------|--------|
| 0          | 300000  | 100    | 100    |
| 1          | 200000  | 100    | 100    |
| 2          | 200000  | 500    | 500    |
| 3          | 150000  | 1000   | 1000   |
| 4          | 150010  | 1000   | 1000   |
| 5          | 100000  | 300    | 300    |
| 6          | 100000  | 1000   | 1000   |
| 7          | 100000  | 2000   | 2000   |
| 8          | 50000   | 300    | 300    |
| 9          | 30000   | 100    | 100    |
| 10         | 30000   | 200    | 100    |
| 11         | 30000   | 1000   | 1000   |

Table 10: Another group of VMs.

| VmID | HostID | Mips | PesN | Ram/MB | Bw | Size/MB |
|------|--------|------|------|--------|----|---------|
| 0    | 0      | 2000 | 1    | 768    | 1000| 1000    |
| 1    | 0      | 1000 | 1    | 512    | 500 | 1000    |
| 2    | 0      | 2000 | 2    | 1024   | 1000| 1000    |
| 3    | 0      | 1500 | 1    | 768    | 500 | 1000    |
| 4    | 0      | 2500 | 1    | 1024   | 1000| 1000    |
| 5    | 1      | 1000 | 1    | 512    | 500 | 1000    |
| 6    | 1      | 500  | 1    | 512    | 1000| 1500    |
| 7    | 1      | 800  | 1    | 512    | 1000| 500     |

Table 11: Another 3 experiment results.

| Combination | NSGA-II | Greedy | Simple |
|-------------|---------|--------|--------|
| 28 Cl, 8 Vms| 150.5   | 150.5  | 230.14 |
| 12 Cl, 12 Vms| 200.11  | 200.2  | 1500.14|
| 12 Cl, 8 Vms| 67.17   | 67.17  | 108.73 |

6. Conclusion and Future Work

Decision making for task migration is one of most important issues in pervasive clouds computing. There is scarce reported work providing decision making policies, for example, considering the characteristics of user tasks and cloud environment. In this paper, we proposed a genetic algorithms based approach for obtaining optimized tasks allocation scheme. The optimized solutions can be used to enable effective allocation strategies, and then in the actual running, system can make use of the chosen allocation scheme to execute user tasks. The whole process is evaluated and it was shown that our approach is feasible with acceptable performance. Due to the limitations in CloudSim, currently the evaluation is simplified and in the future the approach will be extended with a real test bed which will be ready soon [5].

Currently we consider task execution time, input, and output data sizes of the tasks and the bandwidth of network, accordingly choose time consumption characteristic of task execution, and make tasks allocation decision on its basis. We are planning to consider other characteristics such as...
as energy consumption, resources requirement in decision making process, and test this approach with our pervasive cloud environment [4, 5]. Pervasive cloud environment is changing continually with time and tasks’ situations are also different during the execution. In the future, we hope to make the decision dynamically according to cloud environment and user tasks’ status with the help of a context-awareness framework.

Conflict of Interests
The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments
The research in this paper is jointly supported by the National Natural Science Foundation of China (Grant nos. 61309024 and 61402533) and also the Natural Science Foundation of Shandong Province (Grant no. ZR2014FM038), “Key Technologies Development Plan of Qingdao Technical Economic Development Area.” Weishan Zhang has been supported by the start-up funds for “Academic Top-Notch Professors in China University of Petroleum.”

References
[1] W. Zhang, K. M. Hansen, and P. Bellavista, “A research roadmap for context-awareness-based self-managed systems,” in Service-Oriented Computing—ICSOC 2012 Workshops, vol. 7759 of Lecture Notes in Computer Science, pp. 275–283, Springer, Berlin, Germany, 2013.
[2] L. Gkatzikis and I. Koutsopoulos, “Migrate or not? Exploiting dynamic task migration in mobile cloud computing systems,” IEEE Wireless Communications, vol. 20, no. 3, pp. 24–32, 2013.
[3] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, “A survey of computation offloading for mobile systems,” Mobile Networks and Applications, vol. 18, no. 1, pp. 129–140, 2013.
[4] W. Zhang, L. Chen, Q. Lu, Y. Rao, and J. Zhou, “Towards an OSGi based pervasive cloud infrastructure,” in Proceedings of the IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing (GreenCom-iThings-CPSCom ’13), pp. 418–425, IEEE, August 2013.
[5] W. Zhang, L. Chen, X. Liu, Q. Lu, P. Zhang, and S. Yang, “An OSGi-based flexible and adaptive pervasive cloud infrastructure,” Science China Information Sciences, vol. 57, no. 3, pp. 1–11, 2014.
[6] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “CloneCloud: elastic execution between mobile device and cloud,” in Proceedings of the 6th ACM EuroSys Conference on Computer Systems (EuroSys ’11), pp. 301–314, ACM, April 2011.
[7] N. Snavely, S. M. Seitz, and R. Szeliski, “Photo tourism: exploring photo collections in 3D,” ACM Transactions on Graphics, vol. 25, no. 3, pp. 835–846, 2006.
[8] D. Narayanan and M. Satyanarayanan, “Predictive resource management for wearable computing,” in Proceedings of the 1st International Conference on Mobile Systems, Applications and Services, pp. 113–128, ACM, San Francisco, Calif, USA, May 2003.
[9] C. Byung-Gon and P. Maniatis, “Augmented smartphone applications through clone cloud execution,” in Proceedings of the 12th Conference on Hot Topics in Operating Systems (HotOS ’09), vol. 9, pp. 8–11, 2009.
[10] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” Software—Practice and Experience, vol. 41, no. 1, pp. 23–50, 2011.
[11] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998.
[12] A. Konak, D. W. Coit, and A. E. Smith, “Multi-objective optimization using genetic algorithms: a tutorial,” Reliability Engineering & System Safety, vol. 91, no. 9, pp. 992–1007, 2006.
[13] K. Deb, “An introduction to genetic algorithms,” Sadhana, vol. 24, no. 4-5, pp. 293–315, 1999.
[14] S. M. Thede, “An introduction to genetic algorithms,” Journal of Computing Sciences in Colleges, vol. 20, no. 1, pp. 115–123, 2004.
[15] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182–197, 2002.
[16] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, “MOCell: a cellular genetic algorithm for multiobjective optimization,” International Journal of Intelligent Systems, vol. 24, no. 7, pp. 726–746, 2009.
[17] http://jgap.sourceforge.net/.
[18] D.-Y. Chen, T.-R. Chuang, and S.-C. Tsai, ”Jgap: a java-based graph algorithms platform,” Software: Practice and Experience, vol. 31, no. 7, pp. 615–635, 2001.
[19] http://cs.gmu.edu/~eclab/projects/ecj/.
[20] J. J. Durillo and A. J. Nebro, ”JMetal: a Java framework for multi-objective optimization,” Advances in Engineering Software, vol. 42, no. 10, pp. 760–771, 2011.
[21] W. Zhang, J. Schütte, M. Ingstrup, and K. M. Hansen, ”A genetic algorithms-based approach for optimized self-protection in a pervasive service middleware,” in Service-Oriented Computing, vol. 5900 of Lecture Notes in Computer Science, pp. 404–419, Springer, Berlin, Germany, 2009.
[22] W. Zhang and K. M. Hansen, ”An evaluation of the NSGA-II and MOCell genetic algorithms for self-management planning in a pervasive service middleware,” in Proceedings of the 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS ’09), pp. 192–201, June 2009.
[23] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, ”Adaptive offloading for pervasive computing,” IEEE Pervasive Computing, vol. 3, no. 3, pp. 66–73, 2004.
[24] E. Cuervo, A. Balasubramanian, D.-K. Cho et al., ”MAUI: making smartphones last longer with code offload,” in Proceedings of the 8th Annual International Conference on Mobile Systems, Applications and Services (MobiSys ’10), pp. 49–62, ACM, June 2010.
[25] L. Qingfeng, J. Xie, H. Jicheng, Z. Hongchen, and Z. Shanshan, ”An optimized solution for mobile environment using mobile cloud computing,” in Proceedings of the 5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM ’09), pp. 1–5, IEEE, September 2009.
[26] P. Kumar and A. Verma, ”Independent task scheduling in cloud computing by improved genetic algorithm,” International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, no. 5, pp. 111–114, 2012.
[27] Y. Ge and G. Wei, “GA-based task scheduler for the cloud computing systems,” in Proceedings of the International Conference on Web Information Systems and Mining (WISM ’10), vol. 2, pp. 181–186, IEEE, October 2010.