



Multi-resource scheduling and power simulation for cloud computing



Weiwei Lin^{a,*}, Siyao Xu^a, Ligang He^b, Jin Li^c

^aSchool of Computer Science & Engineering, South China University of Technology, Guangzhou, China

^bDepartment of Computer Science, University of Warwick, Coventry, CV4 7AL, United Kingdom

^cSchool of Computer Science, Guangzhou University, Guangzhou, China

ARTICLE INFO

Article history:

Received 18 November 2016

Revised 23 December 2016

Accepted 24 February 2017

Available online 27 February 2017

Keywords:

Cloud computing

Multi-resource scheduling

Power modeling

Power simulation

CloudSim

ABSTRACT

Resource scheduling and energy consumption are the two of most significant problems in cloud computing. Owing to the scale and complexity of various resources, it is often difficult to conduct the theoretical analysis of the performance and power consumption of scheduling and resource provisioning algorithms on Cloud testbeds. Thus, simulation frameworks are becoming important ways to complete evaluation. CloudSim is one of the most popular and powerful simulation platforms for cloud computing. However, it requires much improvement to enable CloudSim to perform multi-resource or energy-aware simulations. To overcome this problem, we have extended CloudSim with a multi-resource scheduling and power consumption model, which allows more accurate valuation of power consumption in dynamic multi-resource scheduling. Extensive experiments on six combinations of task assignment algorithms and resource allocation algorithms demonstrate the powerful functionality and superior convenience of the extended CloudSim, MultiRE-CloudSim. Different task assignment and resource scheduling policies will bring about very different energy cost. We could easily repeat the experiment to find out the efficiency and the power consumption of the algorithms under diverse arguments with MultiRECloudSim.

© 2017 Elsevier Inc. All rights reserved.

1. Introduction

Cloud computing [2,6,11,31] has rapidly attracted more and more attention in both academic and industry community. In cloud computing, server consolidation is an approach to the efficient usage of server resources in order to reduce the total number of servers that user requires [34]. The growth of server consolidation is owing to virtualization technology which enables multiple VMs to share the physical resources of a computer. The total resources of VMs shared the same server must not exceed that of the server while the number of servers is required to be as small as possible. Server virtualization provides technical ways to consolidate multiple servers bringing about increased utilization and energy saving. As for resource scheduling for tasks, Resource provisioning consists of two provisioning plan for allocating resources in cloud. These are long term Reservation plan and short term On-demand plan [9]. On-demand plan is a scheme where the users can obtain resources when they need. Reservation plan is a scheme where the resources could be reserved earlier. The on-demand plan could satisfy user's need but it usually charges higher fees compared to Reservation plan. The drawback of Reservation plan is obvious as well. One is the under provisioning problem in which the resources could not fully meet the need due

* Corresponding author.

E-mail address: linww@scut.edu.cn (W. Lin).

to dynamic varying workload. Some other problem is over provisioning, where the reserved resources is provisioned more than what actually needed. Thus, the resources reserved will not be fully used and results in energy waste. It remains an important and difficult issue to take full advantage of various resources and reduce power consumption. Resource utilization and power consumption in cloud computing are tightly coupled. A host with low resource utilization typically still consumes much power in comparison with the power consumption when it is being fully utilized. For example, recent studies reported in [4,13] indicate that average resource utilization of the hosts in most datacenters can be as low as 20% and that an idle host still consumes as much as 60% of the power consumed by a fully utilized host. On the other hand, a lot of researches [3,15–17,32,35] focus on multi-resource allocation and they usually perform simulations on a cluster instead of simulators. We think one of the reasons may be most of the cloud simulators do not support multiple resources. Furthermore, a few papers discuss about multi-resource allocation and power consumption at the same time, it is a non-trivial task to evaluate a candidate solution on real Cloud platforms. Firstly, demands and supply patterns, system scale and platform infrastructure vary from one cloud to another. Moreover, background workloads in a cloud change dynamically. It is very difficult, if not impossible, to repeat the experiments with the exactly same settings to compare two candidate solutions. Secondly, many factors may influence the application performance and the power consumption of the supporting data center, such as users' QoS (Quality of Service) requirements, various dynamic workloads, complex scheduling strategies on multiple resources and different power consumption pattern of diverse hardware. Thirdly, the real experiments are time-consuming and sometimes impossible because it is typically required to perform a number of test runs under a variety of experimental settings. Thus, performing simulation experiments through simulation tools become an alternative, viable experimentation methodology. The simulation experiments are controllable and the experimental results can be reproduced, which enables the researchers to compare candidate solutions in fair experimental environments.

There is one more important problem to be discussed. The implementation of power simulation of CloudSim assumes that there is just one cloulet which is working as an online service. It only models a simple scenario and cannot meet a variety of complex need of energy simulation. To carry out power simulation, CloudSim uses dynamic workload model. This model works well under the above assumption, but it's not applicable for the multiple-task scenario because the dynamic workload model of CloudSim works if and only if we know the exact time when the task starts and ends. However, the multiple task scenario we discuss in this paper, tasks are scheduled by different allocation policies. We don't know the exact time when they start or end during the scheduling process. We find out the limitations of CloudSim after we read through the source code. Only after a task ends, we know its finished time. They are listed as follows.

1. CloudSim to date only supports single-resource tasks [25,26]. The current version of CloudSim appears to support various types of resources on the surface, including CPU, memory and bandwidth. In the process of task submission and scheduling, however, it does not consider memory and bandwidth as constraints. No matter how much memory or bandwidth the tasks demand (even if the demand is more than the physical capacity of the host), CloudSim does not consider it as an error. Moreover, in power consumption simulation, CloudSim only considers the power consumed by CPU, and it does not take into account the power by memory accessing or communication.
2. CloudSim only supports the single-task running in power simulation. The following comment is added in the source code of the "ClouletSchedulerDynamicWorkload" component in CloudSim: "*ClouletSchedulerDynamicWorkload implements a policy of scheduling performed by a virtual machine assuming that there is just one cloulet which is working as an online service.*" This comment explicitly tells its users that CloudSim does not support the case where multiple tasks are running simultaneously. Consequently, CloudSim is not able to perform power simulation of multiple task scheduling. Due to this exact limitation, researchers seldom use CloudSim to conduct research on power consumption. Instead, they usually do experiments on physical hosts [5,12,20,22].
3. There is a problem concerning about the implementation of dynamic workload in CloudSim. The dynamic workload is implemented in the following way. It obtains the utilization data by reading the workload file and calculates the resource utilization using the simulated time. Here comes the problem of the implementation. If a task starts at 10 s and ends at 30 s, then the data before 10 s and after 30 s in the workload file are not utilized. However, we generally do not know the exact time when the task starts and ends. Actually, we don't want to know when every cloulet start or end before the simulation because if we do, we have to match the running time of every cloulet with corresponding part of data in workload file.

The motivation of this paper is to provide a fast and convenient approach to evaluate the multi-resource scheduling and power-aware algorithms. It provides a multi-resource multiple task scheduling model and it is easy to design different task assignment policy for different scenarios, for instance, CPU intensive, ram intensive and comprehensive scheduling concerning multi-resource shows various patterns. In addition, MultiRECloudSim has the multi-resource power simulation mechanism and power model interface, which provides a far more accurate power consumption result about multiple resources compared with CloudSim. Therefore, we extend CloudSim 3.03 by overcoming the limitations of CloudSim discussed above. The enhanced CloudSim is called **MultiRECloudSim**. The contribution of us is the five new features of MultiRECloudSim. They are summarized as follows.

1. **Multi-resource cloulet.** Cloud application have diverse resource demands. For instance, machine learning tasks are CPU-intensive while sort tasks are memory-intensive. Tasks are constrained on multiple resources, e.g., reduce tasks that are both memory and network-intensive. The extended cloulet supports CPU, memory, IO and bandwidth. A VM

is able to allocate the resources to cloudlets within its resource capacity and then runs the cloudlets, which makes it possible to design a variety of more realistic resource allocation policies and task scheduling algorithms. We design and implemented a class called SimCloudlet for this purpose. The mechanism of multiple resources is supported by a series of related classes, including SimPowerHost, SimPowerVm, CloudletAssignmentPolicy, SimCloudletSchedulerDynamicWorkload, etc. The SimCloudlets are assigned to SimPowerVms placed on the SimPowerHosts. In running process, SimCloudlets are updated and scheduled by SimCloudletSchedulerDynamicWorkload. With these extensions, we could easily implement algorithms like dominant resource fairness (DRF) algorithm [3] and compare it with other algorithms.

2. **Cloudlet assignment policy.** Application workloads in the cloud are dynamic and change over time, which means workloads become imbalanced among VMs as well as diverse resources. It's necessary to balance the multiple resources of VMs to raise the resource utilization and reduce power consumption. The task assignment policy is a key algorithm that relates to throughput, resource utilization and power consumption. We implement a new class called CloudletAssignmentPolicy to complete this task independently, so as to loosen the coupling with the DatacenterBroker. By doing so, the task of defining various cloudlet assignment policies becomes very simple, especially in the case of multiple resource scheduling. The Main Resource Task Balance Algorithm we proposed is a multiple resource scheduling policy implemented by the subclass of CloudletAssignmentPolicy.
3. **Multi-resource cloudlet scheduling and resource allocation scheme.** After a task is assigned to VM, it enters to the task waiting queue waiting for scheduling. The task scheduler of VM will schedule the task according to the task's priority, the demand for multiple types of resources, the arrival time, etc. In CloudSim, the class CloudletScheduler is used to schedule the cloudlets submitted to VMs and we extend its subclass CloudletSchedulerDynamicWorkload to support the scheduling of multiple multi-resource cloudlets. Specifically, we implement a cloudlet resource allocator for each type of resource (CPU, memory, IO and bandwidth). The role of the resource allocator is to allocate and manage resource independently. Moreover, this extension makes it easy to define different resource allocation policies. In MultiRECloudSim, we implement two types of modes: reservation scheme and non-reservation scheme. Reservation scheme refers to the case where the maximum amount of CPU resource is reserved for a cloudlet when it starts and therefore there is certainly no shortage of CPU resource for this cloudlet. It's the same as long term Reservation plan mentioned above. The CPU demand of a cloudlet under this mode is satisfied all the time. However, the disadvantage of this scheme is that the CPU resource will be wasted on idle time. In contrast, the non-reservation scheme refers to the case where the reservation is not supported. Currently, CPU supports these two schemes while the other three resources only support reservation scheme. We implemented two algorithms: First-come-first-served and Max-Min fairness for non-reservation scheme.
4. **Multi-resource power consumption simulation.** The increasing costs of power delivery and cooling and the trend toward higher-density server cloud systems, have created a growing demand for better power management for cloud computing. Scholars have done a lot of work on power models with respect to different resources. [1,5,17]. These models may be evaluated with MultiRECloudSim. The power model in MultiRECloudSim is a function of power consumption over resource utilization. Power models vary from resource to resource. With the power models, we can perform the power simulations reflecting the real situations.
5. **Progress-based multi-resource cloudlet.** Progress-based Cloudlet aims to overcome the disadvantage of CloudSim that the utilization data in the workload file are not fully utilized in dynamic workload simulation. It is a further extension to SimCloudlet. The principle of the progress-based cloudlet is that the utilization data are read by the progress percentage of the cloudlet instead of by simulated time. This feature enables the utilization data in a workload file to be all utilized for various workloads, regardless of the running time of the cloudlet. Similarly, the progress-based multi-resource mechanism are supported by a series of progress-based related classes.

The paper is organized as follows. [Section 2](#) discusses the related cloud simulators and some improved simulators based on CloudSim. The third section focuses on the models we design and in [Section 4](#), we introduce the implementation of MultiRECloudSim. An evaluation including static workload and dynamic workload is provided in [Section 5](#) and the paper is concluded in [Section 6](#).

2. Related work

Although cloud computing has been advancing rapidly in recent years, there are only a few existing simulation frameworks for Cloud computing nowadays due to the complexity of the nature. Main cloud computing simulators are CloudSim [8], GreenCloud [21] and MDCCSim [23]. Green-Cloud is an extension to the NS2 network simulator for the evaluation of power-aware data centers in cloud environments. It is a packet-level simulator, focusing mainly on cloud communication power consumption, providing a fine-grained data center power consumption model. MDCCSim is a commercial discrete event simulator which can help model diverse components of a data center such as servers, communication links and switches with specific hardware characteristics from different vendors. CloudSim is most advanced and comprehensive among the three simulation frameworks. CloudSim is a generalized and extensible simulation framework that enables seamless modeling and simulation of scalable cloud computing infrastructures and various application services. CloudSim (i) supports modeling and simulation of large scale cloud computing systems, including data centers, physical hosts and virtual machines,

(ii) supports modeling service brokers, cloudlet schedulers, vm schedulers, vm allocation policies and resource provision policies, (iii) supports network simulation among the simulated system nodes and (iv) supports power simulation with one application on each vm.

Due to its powerful modeling and simulation capabilities, many studies are based on CloudSim. Network CloudSim [14] extended CloudSim with a scalable network flow model and a parallel application model. The network flow model utilizes bandwidth sharing and latency to enable scalable and fast simulations, while the parallel application model enables the simulation of parallel and distributed applications. With Network CloudSim, scheduling and resource provisioning policies can be evaluated more accurately, helping optimize the performance of a datacenter. DynamicCloudSim [7] extends CloudSim by modeling instability in cloud environments, including inhomogeneity, dynamic changes of performance at runtime and failures during task execution. Experiments in simulation and on real cloud infrastructure are both conducted to simulate the influence of instability on scientific workflow scheduling. Results indicate that the model adequately reflects the major aspects of cloud performance instability. Tom et al. [18] added a new package called DVFStoCloudSim for conducting energy-aware simulation with Dynamic Voltage and Frequency Scaling. WorkflowSim [10] is a toolkit for simulating scientific workflows in distributed environments, which also extends from CloudSim. The authors indicate that ignoring system failures and overheads in simulating scientific workflows may bring significant inaccuracy during the predicted workflow runtime. Xiang Li et al. [24] designed a CloudSim-based simulator called DartCSim+, which supports the power-aware network simulation and network-aware live migration. Moreover, to address transmission failure caused by migration or network failure and capture more realistic network behaviors a resubmit mechanism for packets transmission is implemented. [38] presents a cloud computing adoption framework (CCAF) security suitable for business clouds. CCAF multilayered security is based on three major security technologies: firewall, identity management, and encryption.

Even though so many simulators are extended from CloudSim, none of them consider multi-resource tasks and power consumption in multi-resource scheduling. Efficient multi-resource allocation policies increase resource utilization and therefore save cost. Power-aware allocation policies reduce the power consumption, preventing a blind pursuit of efficiency. On the other hand, our previous work [25–30,36] proposed the models for distributed resource and task scheduling, more specifically to improve the resource utilization and the performance of cloud datacenter. However, these methods cannot be directly applied to multi-resource task scheduling and power simulation. In order to address this problem, we develop MultiRECloudSim in this work.

3. Design of MultiRECloudSim

With MultiRECloudSim, we investigate the effective policies on assigning multi-resource cloudlets to VMs, allocating multiple resources to VMs and cloudlets, scheduling multi-resource cloudlets to save energy, etc. Next, we present the models in MultiRECloudSim in detail.

3.1. Multi-resource cloudlet

SimCloudlet, an extension to Cloudlet, supports CPU, memory, input-output (io) and bandwidth resources. The attributes of the class SimCloudlet, mips, ram, io and bw represent the demand for CPU, memory, IO, bandwidth of the task respectively. Among them, mips can be a fixed or varying value while other three can only be of fixed values. In other words, SimCloudlet supports static and dynamic CPU workload, but only supports static workload for memory, IO, bandwidth. The reason for this is because the model takes the following assumptions:

1. To simplify the model, we set ram, io, bw to be of fixed values, which means that we assume the demands of tasks for these three resources are fixed and that once the task starts, it will occupy these resources until its completion.
2. As for the IO resource, we abstractly treat IO as a type of resource that can be allocated. The allocation is represented by the average speed of read and write operations. Its unit is MB/s.
3. For memory, IO and bandwidth, we assume that when the demands of a task for these three resources are satisfied, the task can be executed. It is not the case for the CPU demand. The amount of CPU (measured by mips) allocated to a task only influence the speed of execution. If the demands for memory, IO or bandwidth are not satisfied, the task will be put in the waiting queue.

In addition, we also modify the original source of dynamic demand for mips.

The current requested mips of a vm is originally calculated as the total requested utilization of mips over all running tasks times the mips allocated to the vm. In our revision, the current requested mips of a vm is calculated as the sum of the requested mips utilization of a task times the standard mips of the task. The standard mips refers to the initial mips assigned to a task, i.e., the mips when the requested utilization is 100%. The change we made can help us change the mips workload in a more flexible way. Moreover, it is easier to simulate and more realistic.

3.2. Cloudlet assignment policy

By inheriting the abstract class CloudletAssignmentPolicy, we could define our own cloudlet assignment policies. The original Cloudlet allocation policy in CloudSim is called sequential allocation algorithm and its implementation class is

CloudletAssignmentPolicySimple. We propose another policy, which we call Main Resource Task Balance (MRTB) assignment policy.

Before explaining the new policy, we first introduce two definitions: normal resource load of a task and normal resource load of a vm.

The Normal Resource Load of a Task (NRLT) is a metric that measures how much workload of a task for the resource. It is calculated by the product of the amount of resource and the occupied time. The resources considered here include cpu, memory, io and bandwidth. NRLT is calculated by (1), where x denotes the cloudlet, c_j denotes the j th resource, $Load(x, c_j)$ denotes the normal load of cloudlet x for resource c_j , $time(x)$ denotes the estimated execution time of cloudlet x , $Normal(x, c_j)$ denotes the normalization of the average demand of cloudlet x for resource c_j .

$$Load(x, c_j) = time(x)Normal(x, c_j) \quad (1)$$

$time(x)$ is the length of cloudlet x divided by the average mips allocated to cloudlet x , shown as in Eq. (2), where $length(x)$ denotes the length of cloudlet x , $c_{CPU}(x)$ denotes the average demand for CPU. But we may not know the average demand in dynamic workload circumstance. In this case, we can use the max demand as the estimation. Normalization is used to make the data dimensionless, so that the demand for different resources can be compared to each other. The normalization is performed by dividing the demand by a reference value for the concerning resource, shown as in Eq. (3), where $c_j(x)$ denotes the average demand for resource c_j , c_j^* denotes the reference value of resource c_j , which we also call the normal value of resource c_j . The reference value is selected by the users empirically. For example, the reference value of CPU may take the initial value of the resource allocated to vm.

$$time(x) = \frac{length(x)}{c_{CPU}(x)} \quad (2)$$

$$Normal(x, c_j) = \frac{c_j(x)}{c_j^*} \quad (3)$$

The Normal Resource Load of a VM(NRLV) is a metric that measures how much workload of a vm for resources. It is the sum of normal resource load of all running tasks and the tasks to be executed in the vm, as shown in as Eq. (4), where $Load(vm_i, c_j)$ denotes the NRLV of vm_i for resource c_j , $Load(x, c_j)$ denotes the NRLT of cloudlet x for resource c_j , the set $E(i)$ includes all running tasks and the tasks to be run in the vm.

$$Load(vm_i, c_j) = \sum_{x \in E(i)} Load(x, c_j) \quad (4)$$

The main workings of main resource task balance (MRTB) algorithm is as follows. Given a reference value to each resource, the maximum NRLT of a task is calculated over all resources. The resource corresponding to the maximum NRLT is regarded as the main resource of the task. Then the task is assigned to the vm whose NRLV best fit the demand of the main resource of the task. The NRLV of the assigned task is added to the total NRLV of the vm. Only when a task is completed will the task's NRLT is subtracted from the total NRLV of the vm.

The algorithm outline is as follows:

3.3. Multi-resource cloudlet scheduling and resource allocation scheme

Task scheduling in vm is mainly implemented by class SimCloudletSchedulerDynamicWorkload, the major functions of which are cloudlet submission, cloudlet queue, resource allocation and resource allocator.

1. Cloudlet submission. When a cloudlet is assigned to vm, whether the cloudlet can start running depends on whether there are sufficient resources in the vm. In the non-reservation scheme, as long as memory, io and bandwidth are sufficient, cloudlet start running. In the reservation scheme, all four types of resources have to be sufficient for the cloudlet to start.
2. Cloudlet queue. If the resources are insufficient, then the cloudlet will enter the cloudlet waiting queue.
3. Resource allocation. During the execution of cloudlet, all resources are allocated in each round. A host allocates the resources to the VMs in the host and a vm in turn allocates resources to the Cloudlets in the vm.
4. Resource allocator. In order to facilitate the resource management for cloudlets, we designed the task resource allocator class, called Allocator. Allocator is used to allocate, manage and recycle resources for a task managed by SimCloudletSchedulerDynamicWorkload. The Allocator is described in more detail below.

According to the resource type, the allocator is divided into four types of allocator: CloudletCpuAllocator, CloudletRamAllocator, CloudletIoAllocator, CloudletBwAllocator. These four classes are abstract classes. The basic resource allocation is implemented by their simple classes. Essentially, the implementation logic of Allocator is the same as that of the Provisioner class of CloudSim. In addition, CloudletCpuAllocatorSimple also implements the interface of appending mips. In CloudSim, every interval, the demands for resources will be recalculated because of the dynamic workload for resources. Appending mips is to append extra mips to vm for starting new cloudlets. Because it is not necessary to fully meet CPU demand, we can design a variety of mips allocation algorithms. In this paper, we implement three mips allocation algorithms:

First-come-first-served, Max-Min fairness [30] and CPU Reservation algorithms [37]. They are implemented by the classes, `CloudletCpuAllocatorSimple`, `CloudletCpuAllocatorMaxMinFairness` and `CloudletCpuAllocatorReservation`, respectively.

The resource reservation scheme and the non-reservation scheme are two schemes that are only related to CPU resource and do not including other types of resources because other resources must be completely satisfied. In the reservation scheme, the cpu allocator allocates the maximum mips the cloudlet demands to the cloudlet. This scheme can guarantee to meet the QoS of the tasks, but may cause resource waste. In the non-reservation scheme, mips are allocated to cloudlets using one of the three algorithms, which cannot ensure the satisfaction of the tasks' QoS.

3.4. Multi-resource power simulation

We extend `PowerDatacenter` to support power simulation for these four resources: CPU, memory, io and bandwidth. The main methodology for power simulation is to calculate power consumption every slot time, sum of which is the total power consumption. Based on the resource utilization measured at the beginning and end of a slot as well as the model reflecting the relation between power and resource utilization (the utilization-power model), we apply the linear fit method to estimate the power consumption. The utilization-power models of different resources we used in the experiment are as follows:

1. CPU power model. We use `PowerModelSpecPowerIbmX3550XeonX5675()` of `CloudSim`, which represents the IBM server x3550 (2 x [Xeon X5675 3067 MHz, 6 cores], 16GB) [19]. We can use the benchmark tests presented in [19] to implement CPU power models for the hosts different from IBM server x3550.
2. Memory power model. We design the following simple memory power model (implemented in the class called `PowerModelRamSimple`), where P denotes power (the unit is Watt), u denotes memory utilization, P_{\max} denotes the power when memory utilization is 100% and r denotes the total host memory. The unit of r is MB and (6) means 1024 MB memory brings about 1 W energy consumption.

$$P = u * P_{\max} \quad (5)$$

$$P_{\max} = r / 1024 \quad (6)$$

3. IO power model. We design a simple IO power model (implemented in a class called `PowerModelIoSimple`) as follows, where P is power, u is IO utilization, P_{\max} is the power when IO utilization is 100%, io denotes the total host IO resource, 0.0314573 is the training parameter, which is obtained through linear fitting based on the data measured by ourselves on the server Dell PowerEdge T110 with a 1 TB Seagate enterprise disk ST31000340NS. We measure the energy with Joulemeter [16], using atto benchmark [33].

$$P = u * P_{\max} \quad (7)$$

$$P_{\max} = 0.0314573 * io / 1024 \quad (8)$$

Since there is little research to establish the relation between power and bandwidth utilization, there is no commonly accepted power model for bandwidth. Therefore, the power model for bandwidth is not provided in this work. If more works on bandwidth are put forward, it is easy to implement such models with `MultiRECloudSim`. The power consumption experiments in this paper do not cover the bandwidth resource.

3.5. Progress-based multi-resource cloudlet

Progress-based multi-resource cloudlet is an extension to multi-resource cloudlet, implemented by the class `SimProgressCloudlet`. To explain the motivation of developing the progress-based multi-resource cloudlet, we need to understand the current limitation of `CloudSim` in the way of obtaining resource utilization. There is an interface class called `UtilizationModel` in `CloudSim`, the key method of which is `getUtilization(double time)`, where the parameter `time` is the simulated time. Its subclass `UtilizationModelPlanetLabInMemory` is a class that implements reading utilization data from the workload file to generate varying, dynamic workload. Its attribute `schedulingInterval` denotes the time interval of two utilization data. One can lengthen or shorten the time span of the data by varying the value of `schedulingInterval`. However, there exist significant limitations of time-base utilization model as follows:

1. The data is not fully utilized. If the time of the utilization data spans 1 h, but the cloudlet only executes for 6 minutes, then 90% of the utilization data is not utilized. Even though we can reduce the value of `schedulingInterval`, we have no idea about how long the cloudlet will last. Besides, the cloudlet cannot pause because it will cause part of utilization data in the middle of the workload file not to be used.
2. The span of workload running is uncertain. We have to know when the cloudlet starts, when it ends in order to know which part of the utilization data should be used, which, however, is nearly impossible in a complex scheduling scenario of multiple cloudlets. The cloudlet pause will also break the workload span. What is desired is that all utilization data of a workload file is fully used for a cloudlet. Then we know the characteristic of the dynamic workload.

The progress-based utilization model can address the issues mentioned above. We now present an example to explain the calculation method. Suppose there are 11 utilization data points, the resource utilization increases from 0% at the first point to 100% at the last point with increment of 10%, i.e., the first point corresponds to 0% of utilization, the second to 10%, ..., and the last to 100%. To order to achieve progress-based cloudlet, what we need primarily to overwrite the method *getUtilization(double time)* to *getUtilization(double progress)*. If the progress of the task is 50%, then the method *getUtilization(double progress)* returns the value of the 6th point. If the progress of the task is 66%, then the utilization is calculated as $u_7 + \frac{u_8 - u_7}{10} (66 - 60)$, where u_7 , u_8 denote the values of the 7th and the 8th points, respectively. That is to say, if the task progress does not fall on an exact utilization point, the linear fitting is used to estimate the utilization.

Using the progress-based utilization model, *SimProgressCloudlet* can fully utilize the utilization data, no matter how long the cloudlet is, when the cloudlet starts or ends, whether it pauses or not. The attribute *cloudletFinishedSoFar* of the class *SimProgressCloudlet* records the length that cloudlet has completed. The method *getProgress()* returns the progress of the cloudlet. The progress-based utilization model, called *UtilizationProgressModelByFile*, uses the method *getUtilization(double progress)* to obtain the requested utilization of the cloudlet based on the cloudlet progress. To perform progress-based simulation, additional progress-based classes are required, such as *SimProgressCloudlet*, *UtilizationProgressModelByFile*, *SimProgressCloudletSchedulerDynamicWorkload*, *SimProgressCloudletSchedulerDynamicWorkloadReservation*.

4. Implementation of MultiRECloudSim

There are lots of classes in *MultiRECloudSim*, we could mainly divide them into three categories. First are the fundamental elements, including *SimDatacenter*, *SimDatacenterBroker*, *SimPowerHost*, *SimPowerVm*, *SimCloudlet*, *UtilizationModel* and their subclasses. Second are the policy classes that decide all kinds of allocation scheduling algorithms, such as *CloudletAssignmentPolicy*, *SimCloudletSchedulerDynamicWorkload*. The last are the resource manager classes that allocate and record resources, for example, *SimRamProvisioner*, *CloudletIoAllocator*. Multi-resource scheduling involves most of the classes. When a *SimCloudlet* representing a multi-resource cloudlet is submitted to *SimDatacenter* by *SimDatacenterBroker*, a specific vm will be assigned to run it, then the *SimCloudletScheduler* of the vm will accept the *SimCloudlet* and schedule it in a cloudlet queue. When the four kinds of resources for the *SimCloudlet* are adequate, the *SimCloudlet* starts to run. During the running process, resources are repeating allocation and recovery every interval. Every interval, *SimDatacenter* will calculate the energy consumption of each host. Finally, all *SimCloudlets* finish, we'll get the running states of *SimCloudlets* and the power consumption of *SimPowerHosts*. Next, we introduce the simulation flow of *MultiRECloudSim* in detail, i.e., how the simulation runs and how the classes interact with each other.

1. *SimDatacenterBroker* assigns every *SimPowerVm* (extended Vm) to certain *SimPowerHost*. The outcome of creating *SimPowerVm* (success or failure) depends on whether the resources of *SimPowerHost* are sufficient.
2. *DatacenterBroker* assigns each *SimCloudlet* to the created VMs according to *CloudletAssignmentPolicy*.
3. When *SimCloudlet* is submitted to *SimPowerVm*, it will be scheduled by *SimCloudletSchedulerDynamicWorkload*, which checks whether cloudlet can start according to the resources of *CloudletIoAllocator*, *CloudletCpuAllocator*, *CloudletRamAllocator*, *CloudletIoAllocator* and *CloudletBwAllocator*. If the resources are sufficient, then the cloudlet starts. Otherwise it enters the waiting queue.
4. Cycle State. *MultiRECloudSim* updates cloudlets, reallocates resources to VMs, calculates power consumption every fixed time interval:
 - (i) *SimCloudletSchedulerDynamicWorkload* updates the cloudlets and checks whether there are appending resources for waiting cloudlets. If there are, it allows the waiting cloudlet to start. If any cloudlet finishes, it checks whether the remaining resources are sufficient for any waiting cloudlets.
 - (ii) *SimPowerHosts* reallocates all resources according to the demand of *SimPowerVms*. Reallocation consists of two sections: allocating resources to the executing cloudlets and appending resources to the waiting cloudlets. If there are some resources left after allocating resources to the executing cloudlets, it will check whether the remaining resources are sufficient for any waiting cloudlets. If they are, it appends the resources to the waiting cloudlets. The process repeats until the remaining resources of the host are not adequate for any waiting cloudlet.
 - (iii) *SimPowerDatacenter* calculates the power consumption according to the resource utilization of *SimPowerHosts* and the resource-power models.
5. If all cloudlets of a host finish, the host shuts down. All objects shut down after all cloudlets finish.

These classes mentioned above are implemented in *MultiRECloudSim*. Fig. 2 is the diagram reflecting the class inheritance relation in *MultiRECloudSim*. White color classes indicate the original classes of *CloudSim* while the color of light gray and dark gray indicates the newly added class in *MultiRECloudSim*. In addition, dark gray color indicates the class is related to *SimProgressCloudlet*.

Symbol \blacktriangledown in Fig. 2 represents the inherent relation, whose start point is the parent class and end point is the child class. Symbol \blacktriangleup in Fig. 3 represents the affiliation relation. The class on its end point includes the class on its start point. The including relation also indicates superior-subordinate connections between classes. For example, as *SimPowerDatacenter* possesses many *SimPowerHosts*, *SimPowerDatacenter* is the superior of *SimPowerHost*. *SimPowerHost* is also the superior of *SimPowerVm* for the same reason. There are two types of relationship between the inclusion classes: one-to-one and one-to-many, represented by the number next to the symbol \blacktriangleup .

```

1 Input: CloudletList, VmList,  $c_1^*, c_2^*, \dots, c_m^*$ 
2 Output: Assignment of CloudletList
3 Init  $Load(vm_i, c_j) = 0, i = 1, \dots, n, j = 1, \dots, m;$ 
4 For each cloudlet  $x$  in CloudletList
5 Calculate  $Normal(x, c_1), \dots, Normal(x, c_j)$ , find the minimum  $Normal(x, c_k);$ 
6 Find the  $VM_{tar}$  with the minimum  $Load(VM_{tar}, c_k);$ 
7 Assign cloudlet  $x$  to  $VM_{tar}$ ,  $Load(VM_{tar}, c_j) = Load(VM_{tar}, c_j) + Load(x, c_j), j = 1, 2, \dots, m;$ 
8 Return Assignment of CloudletList.

```

Fig. 1. The main resource task balance algorithm.

Symbol \Downarrow in Fig. 4 shows the flow of SimCloudlet during multi resource scheduling. SimCloudlets are submitted to SimDatacenterBroker and assigned to vm according to the CloudletAssignmentPolicy, and then the scheduling and update are conducted by SimCloudletSchedulerDynamicWorkload. SimCloudlet, SimPowerHost, SimPowerVm, SimCloudletSchedulerDynamicWorkload all demand multi resources. SimPowerHost manages multi resources by a set of classes Provisioner while SimPowerVm or SimCloudletSchedulerDynamicWorkload is administered by a series of classes Allocator.

The classes in MultiRECloudSim are described in detail as follows.

SimCloudlet: This class models the cloudlet demands for multiple types of resources, including CPU, memory, IO and bandwidth. Currently, CPU supports static and dynamic demands. While memory, IO and bandwidth support static demand. Cloudlets can be small tasks that require fast execution and response, or long-running applications that offer non-stop services. We plan to investigate different scheduling algorithms and conduct power simulations under various scenarios.

SimCloudletSchedulerDynamicWorkload: This class is an extension to the class CloudletSchedulerDynamicWorkload, implementing multi-resource allocation and multi-resource cloudlet scheduling. Essentially, it is a time-shared cloudlet scheduling policy. It allocates a certain amount of MIPS to each cloudlet in each slot time to update the cloudlet. It supports the cloudlet waiting queue and uses the cloudlet resource allocators to manage and allocate resources, including CPU allocator, memory allocator, IO allocator and bandwidth allocator.

SimCloudletSchedulerDynamicWorkloadReservation: This class is a further extension to the class SimCloudletSchedulerDynamicWorkload that supports the CPU reservation scheme.

UtilizationModelByFile: This class varies the mips demand of SimCloudlet according to the utilization data read from the workload file, which is used to model the dynamic workload.

SimCloudletStateHistoryEntry: This is an auxiliary class that is responsible for recording the situation of resource allocation for cloudlets and the state of cloudlets at different time to facilitate statistical analysis of cloudlet execution.

SimPowerVm: Except the newly added IO resource, there is no other difference with PowerVm.

SimPowerHost: This class models the hosts that require CPU, memory, IO, bandwidth resources and implements allocating and appending resources to VMs in every slot time. The SimPowerHost also records the utilization of all resources at different times and calculates the host power consumption according to resource utilization.

SimPowerHostReservation: It is SimPowerHost that supports the resource reservation scheme. It implements appending resources of reservation scheme.

SimPowerDatacenter: It models the power-aware datacenter that is responsible for calculating the multi-resource power consumption of all hosts every slot time.

SimDatacenterBroker: It models the datacenter broker with a new member variable CloudletAssignmentPolicy for determining which vm the cloudlet should be assigned to.

CloudletAssignmentPolicy: The abstract class of cloudlet assignment policy and the subclasses are only required to fulfill one method, i.e., how to assign cloudlets to VMs.

CloudletAssignmentPolicySimple: It is the sequential cloudlet assignment policy that is the original policy of CloudSim.

CloudletAssignmentPolicyBalance: It's the main resource balance cloudlet assignment policy.

SimRamProvisionerSimple: It's an extension to RamProvisionerSimple with new methods of checking whether memory is sufficient and of appending memory to vm.

SimBwProvisionerSimple: It is similar to SimRamProvisionerSimple.

SimIoProvisioner: It's the abstract class of IO provisioner. The logic of implementation is the same as that of RamProvisioner, BwProvisioner.

SimIoProvisionerSimple : It is similar to SimRamProvisionerSimple.

CloudletCpuAllocator: It is the abstract class responsible for the allocating and managing of the tasks' CPU resources. The implementation comes from the idea of the Provisioner class. The basic property is the total mips and the remaining mips. Core functions include checking whether the remaining CPU resource meets the demand of the tasks and assigning the CPU resource to a single task or multiple tasks. It can implement different CPU allocation algorithms such as priority-based

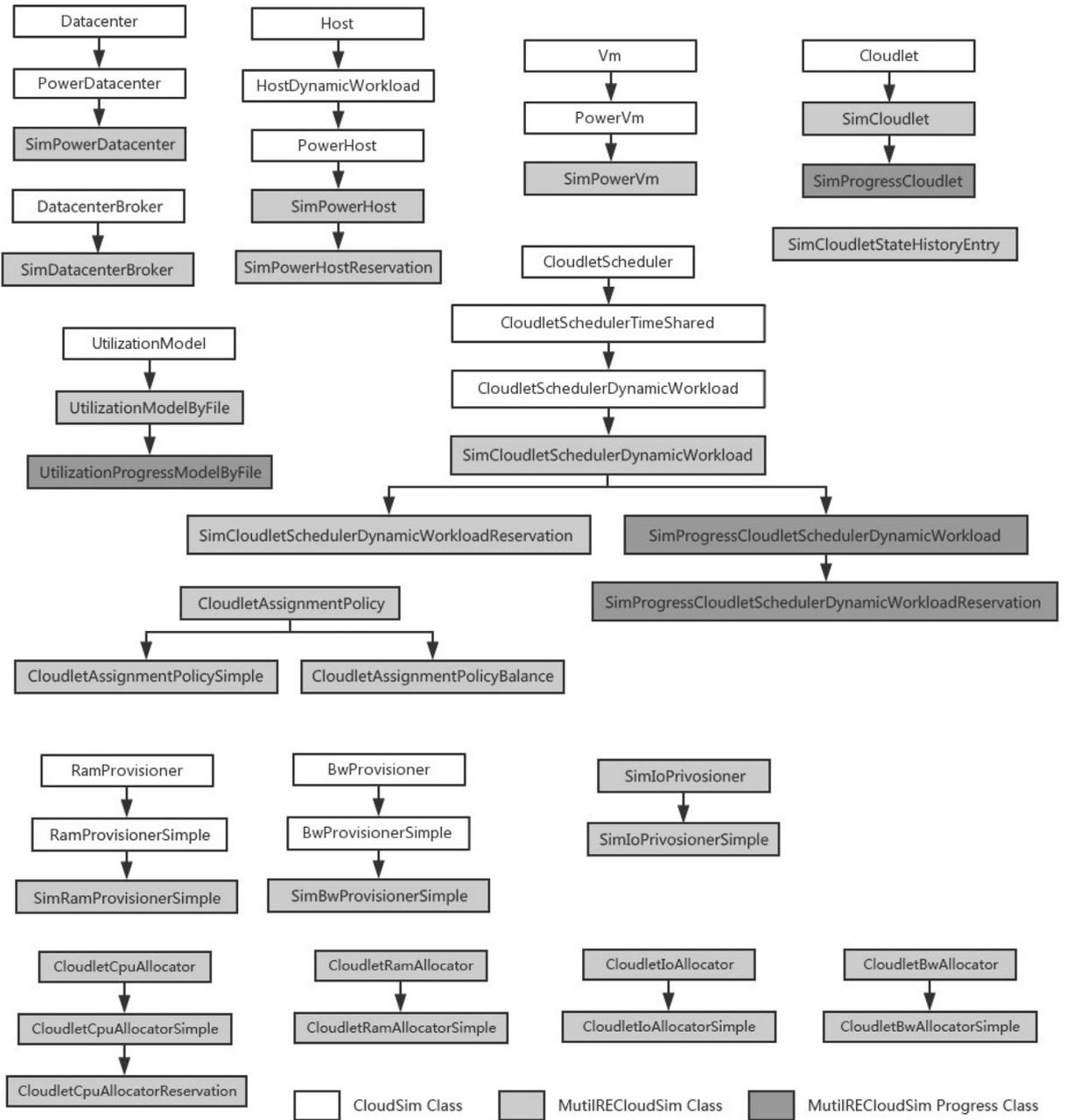


Fig. 2. Inheritance relation diagram of MultiRECloudSim.

allocation or fairness allocation. The Provisioner class is responsible for helping SimPowerHost allocate resources while the Allocator class is to allocate resources for SimPowerVm.

CloudletCpuAllocatorSimple: It is the implementation of CloudletCpuAllocator, which implements all basic methods. The method that reflects the feature of the algorithm is how to allocate CPU resources to multi-tasks. The allocation algorithm tries to satisfy the tasks at the head of the queue and the rest of the tasks lack CPU resources or even do not have resources.

CloudletCpuAllocatorMaxMinFairness: It is the implementation of the max-min fairness CPU resource allocation algorithm.

CloudletCpuAllocatorReservation: It is the CPU reservation resource allocation algorithm. It will allocate enough resources to the task before the task starts running. The knowledge of max CPU demand of the task is required.

CloudletRamAllocator, CloudletIoAllocator, CloudletBwAllocator: They are the allocator abstract classes of memory, IO and bandwidth, which are responsible for the allocation, management and recycle of resources, the difference of which from CloudletCpuAllocator is that it does not have method to allocate the resources to multi-tasks.

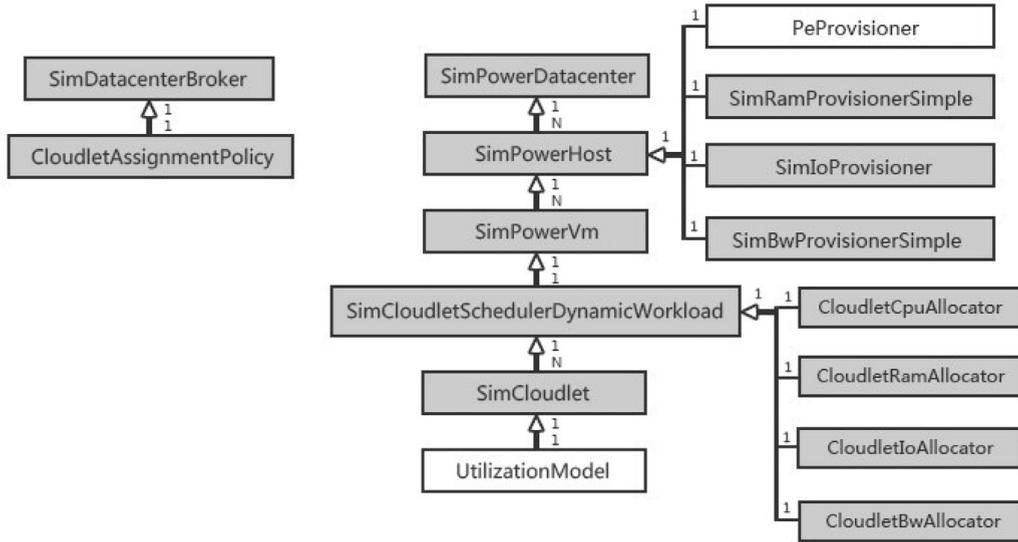


Fig. 3. Affiliation relation diagram of MultiRECloudSim.

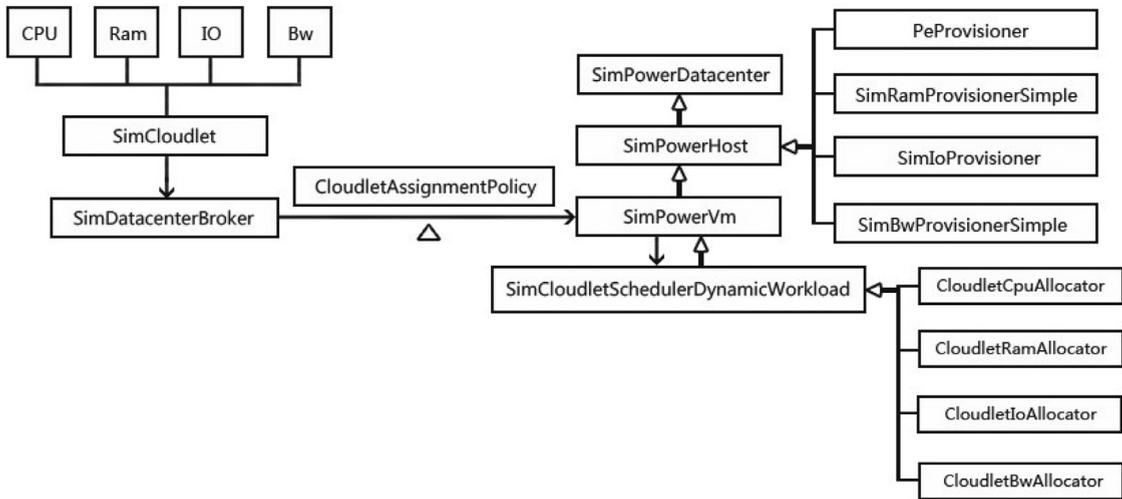


Fig. 4. Workflow of multi-resource scheduling.

CloudletRamAllocatorSimple, CloudletIoAllocatorSimple, CloudletBwAllocatorSimple: The implementation classes of task resources allocator. They implement all the basic methods.

The classes that related to the progress-based cloudlet are as follows.

SimProgressCloudlet: It is the progress-based cloudlet. It reads the utilization data according to the progress of the tasks, not according to the time.

UtilizationProgressModelByFile: It is the utilization model based on progress. The currently requested resource utilization is calculated based on the progress of the tasks and the data from the workload file.

SimProgressCloudletSchedulerDynamicWorkload: Compared with SimCloudletSchedulerDynamicWorkload, it adds the update of SimProgressCloudlet progress as well as the support to SimProgressCloudlet.

SimProgressCloudletSchedulerDynamicWorkloadReservation: Compared with SimProgressCloudlet-SchedulerDynamicWorkload, it adds the support to CPU resource reservation algorithm.

Helper: Helper is a general util class. We can use the methods of this class to create SimPowerDatacenter, SimPowerHost, SimPowerVm and SimCloudlet. Or we can print the cloudlet running result and write the result to file with Helper.

In addition, Some examples can be found in the appendix.

Table 1
Experiment parameters.

Parameters	Value
Simulation interval	1s
Number of hosts	50
Number of tasks	2000, 4000, 6000, 8000

Table 2
Host machine parameters.

Host parameters	Value
The number of host cores	6
Mips of each core	3067 MHz
Ram	16 GB
IO	500 MB/s
CPU power model	PowerModelSpecPowerIbmX3550XeonX5675()
Memory power model	PowerModelRamSimple
IO power model	PowerModelIoSimple

Table 3
Initial parameters of virtual machine.

Initial parameters of virtual machine	Value
Virtual machine number of each host	6
Virtual machine MIPS	3067 MHz
Virtual machine RAM	2 GB
Virtual machine IO	83 MB/s

Table 4
The parameters of cloudlet.

	Length	CPU (MHz)	RAM (MB)	IO (MB/s)
CPU Intensive	7500, 8181, 9000, 10,000, 11 250, 12 857, 15 000, 18 000, 22 500, 30,000, 45,000, 90,000	1500	256	5
RAM Intensive		600	1024	5
IO Intensive		600	256	40

5. Experiments and evaluation

According to whether the demand of CPU resource changes dynamically, there are two types of workload: static workload and dynamic workload. We conduct experiments to evaluate MultiRECloudSim with both types of workload. In static load, the tasks' CPU demand is fixed throughout the execution, while in dynamic workload the CPU demand varies as the task execution progresses (progress-based cloud task). The change model (strategy) is determined by the utilization data in workload file.

The result is evaluated with respect to the three aspects: time, power cost and host SLA violation rate. Host SLA violation rate is the CloudSim's original metric. The SLA violation indicates that the demand for CPU resource of the running task exceeds the allocated CPU resource in a certain slot time. The host SLA violation rate is calculated by dividing the host SLA violated time by the host total executing time.

There are two parts of the comparing algorithm, first part is the task assigning algorithm CloudletAssignmentPolicy, the second is CPU resource allocation algorithm CloudletCpuAllocator. Task assigning algorithm includes sequential assignment algorithm (Simple) and main resource task balance algorithm (MRTB). The CPU resource allocating algorithm includes first-come-first-served algorithm (FCFS), max-min fairness algorithm (MMF) and resource reservation algorithm (RR). We also call sequential assignment algorithm Simple assignment algorithm, and call first-come-first-served allocation algorithm Simple allocation algorithm. There are 6 cases when combining the two parts: SimpleSimple (SS), SimpleMaxMinFairness (SM), Simple-Reservation (SR), Balance-Simple (BS), BalanceMaxMinFairness (BM) and BalanceReservation (BR).

The parameters of experiment are in Table 1. Interval indicates the time slice in loop stage, which affects the accuracy and running time of experiment result. Generally, the smaller the interval, the more precise the result and the longer the running time. Host machine is 50 IBM server \times 3550, the detailed configuration is in host configuration table. The amount of tasks in experiments is 2000, 4000, 6000 and 8000.

The configuration of IBM server \times 3550 is shown in Table 3. As the lack of IO information, we use our test data to fit the power model and assume that the same kind of hard disk have the same power model in different servers.

Table 5

The comparison between static tasks' makespan.

	SS	SM	SR	BS	BM	BR
2000	234	206	240	109	105	124
4000	420	412	420	177	164	186
6000	600	589	600	254	223	243
8000	818	797	840	316	289	309

Table 6

The comparison between static tasks' power consumption.

	SS	SM	SR	BS	BM	BR
2000	256	246	253	214	208	218
4000	504	501	492	417	395	402
6000	751	761	721	613	591	580
8000	1015	1030	975	811	784	761

Table 7

The comparison between static tasks' host SLA violation.

	SS	SM	SR	BS	BM	BR
2000	72%	97%	0.0%	39%	41%	0.0%
4000	85%	98%	0.0%	62%	70%	0.0%
6000	89%	99%	0.0%	70%	74%	0.0%
8000	92%	99%	0.0%	77%	78%	0.0%

Table 8

Comparison of dynamical task's makespan.

	SS	SM	SR	BS	BM	BR
2000	254	237	896	242	241	254
4000	474	436	1568	369	371	376
6000	680	625	2240	464	460	478
8000	901	866	3136	575	569	595

Table 9

Comparison of dynamical task's power consumption.

	SS	SM	SR	BS	BM	BR
2000	309	304	759	385	384	387
4000	591	559	1518	588	585	577
6000	880	864	2273	847	841	832
8000	1169	1167	3033	1102	1093	1069

Table 10

Comparison of dynamical task's host SLA violation.

	SS	SM	SR	BS	BM	BR
2000	35%	43%	0.0%	7%	8%	0.0%
4000	49%	85%	0.0%	14%	17%	0.0%
6000	63%	83%	0.0%	22%	24%	0.0%
8000	71%	85%	0.0%	29%	33%	0.0%

We define 3 kinds of cloudlets: CPU intensive, RAM intensive, IO intensive. CPU intensive tasks need 1500 MHz MIPS, 256 MB memory and 5 MB/s IO resources. RAM intensive tasks need 600 MHz, 1024 MB RAM and 5 MB/s IO resources. IO intensive task will need 600 MHz MIPS, 256 MB RAM and 40 MB/s IO resources. The proportion of the 3 kind of task is 1:1:1, each task has 12 different types of length which represent the different size of tasks. To dynamic load, we choose 2 files from the load files in CloudSim. The utilization data of one covers between 20% and 40%, and that of the other covers between 60% and 80%.

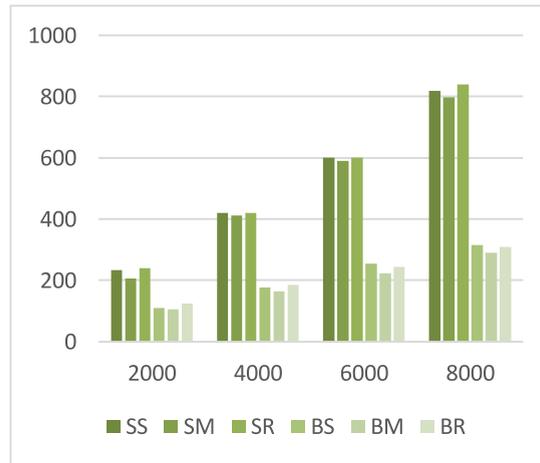


Fig. 5. Comparison of static task's makespan.



Fig. 6. Comparison of static task's power consumption.

5.1. Static workload

As shown in Figs. 5, 6 and 7, MRTB algorithm averagely reduce 57.7% time cost, 19.1% power consumption, 30.4% SLA violation in comparison with the sequential assignment algorithm. Compared with FCFS algorithm, max-min fairness algorithm reduces 6.3% time cost, 2.1% power consumption, however increases 11.7% SLA violation in average. To ensure the quality of service, resource reservation CPU allocation algorithm inevitably leads to longer running time. Compared with FCFS CPU allocation algorithm, the data doesn't have much difference with the increase of time cost by 2.2% and reduction of power consumption 3.2% in average.

5.2. Dynamic workload

From the Figs. 8, 9 and 10 sequential assignment algorithm combined with resource reservation CPU allocation algorithm costs a lot more time and energy than other two algorithms. Except for combination SR and BR, MRTB algorithm averagely reduces 21.1% time cost and SLA violation, at the same time, it increases 4.6% power consumption. Compared with FCFS algorithm, max-min fairness CPU allocation algorithm reduces 3.6% time cost and 1.5% power consumption, increases 25.8% SLA violation. Resource reservation CPU allocation algorithm increase 257% time cost, 155% power consumption along with sequential assignment algorithm, but increases 3.4% time cost and 1.5% power consumption along with MRTB algorithm.

In dynamic load experiment, the distribution of utilization data in load file has significant effect on algorithm. Extremely, if it is under high load only for a short time but under low load for a long time, then host resources will be wasted a lot.

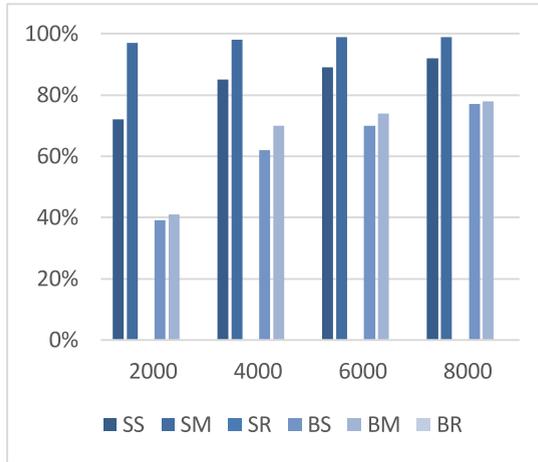


Fig. 7. Comparison of static task's host SLA violation.



Fig. 8. Comparison of dynamic task's makespan.

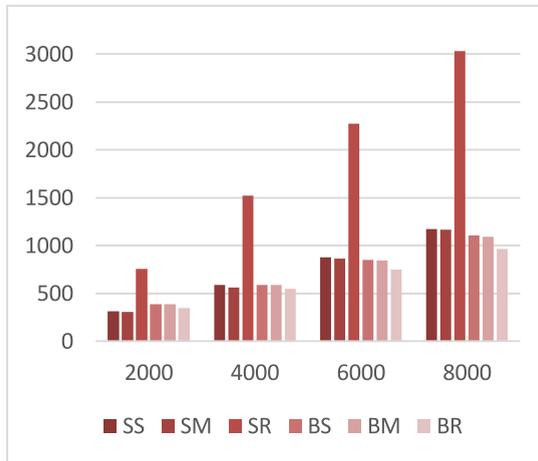


Fig. 9. Comparison of dynamic task's power consumption.

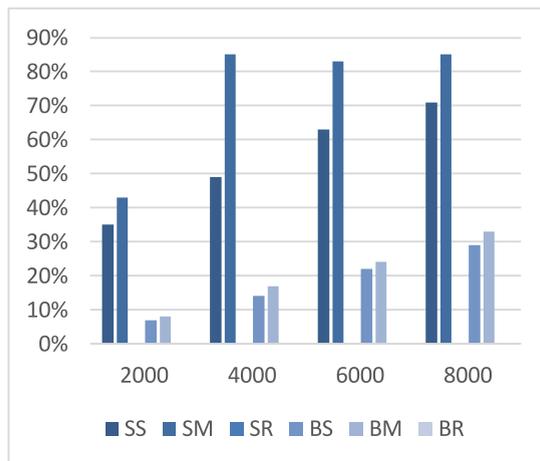


Fig. 10. Comparison of dynamic task's host SLA violation.

In our work, the utilization data of the selected load file's cover in a small interval, so that it won't cause much waste of resource.

Conclusion and analysis of experiment:

1. MRTB algorithm is able to schedule tasks with different intensive types effectively. It raises the host resource utilization, reduces the waiting time of tasks, increases the throughput as well as service quality.
2. Compared with FCFS CPU allocation algorithm, max-min fairness CPU allocation algorithm can reduce task's time cost and power consumption, however increases SLA violation. That's because each task can be allocated a little CPU resource and the demand mips can't be satisfied.
3. Resource reservation CPU allocation algorithm is able to 100% satisfy SLA, with cost of longer running time. In dynamic load situation, the combination of resource reservation CPU allocation and sequential assignment algorithm costs far longer time because of the imbalance of resource allocation. With MRTB algorithm, both static and dynamic situation, resource reservation CPU allocation algorithm cost more time but the power consumption is lower compared with FCFS algorithm. The reason for lower power consumption may be that the resource reservation CPU allocation algorithm reserves the host's resource causing lower utilization. Also the effect of resource reservation CPU allocation algorithm is closely related to the distribution of utilization data from load file in dynamic load situation.
4. After calculation, CPU power consumption is the main cost in the 3 types of resources. In experiment, CPU, RAM, IO's power consumption proportion is 17:1:1.
5. In experiment, power simulation can perform the situation that when host is under low load, it still costs much energy. Calculated by the power model, even the utilization of CPU is 5%, it costs 35% of the full load power consumption.

6. Discussion

MultiRECloudSim presented in this paper is extended to CloudSim. It provides us a convenient way to evaluate the multi-resource scheduling power-aware algorithms and test the experiment parameters. We concentrate on the source code of CloudSim and find out some incomplete designs of CloudSim, especially on multiple resources and power simulation. We overcome the drawbacks and make our contributions as follows.

1. **Multi-resource cloudlet and progress-based multi-resource cloudlet.** CloudSim to date only supports single resource: CPU. This is a large limitation, obviously, so we propose a multi-resource cloudlet model. The CPU resource of cloudlet may change dynamically while memory, IO and bandwidth are regarded as stable. It is a simplified model that could be further extended. The other three resources could be implemented like CPU, the workload of whom change dynamically. It is an improvement direction we consider. The progress-based multi-resource cloudlet is the further extension to the multi-resource cloudlet. To explain the motivation of the progress-based multi-resource cloudlet, we need to talk about the limitation of the implementation of dynamic workload in CloudSim. The dynamic workload of cloudlet is implemented by reading the utilization data from a workload file. A variable *interval* represents the time interval between the utilization data. For example, if *interval* equals 60 (second) and there are 10 utilization data, then the period of workload is $(10 - 1) * 60 = 540$ (second). However, if a task starts at 10 s and ends at 30 s, then the

data before 10 s and after 30 s in the workload file cannot be utilized. Moreover, we generally do not know the exact time when the task starts and ends. Actually, we don't want to know when every cloudlet start or end before the simulation because if we do, we have to match the running time of every cloudlet with corresponding part of data in workload file. A pleasant way is that a cloudlet can utilize all the utilization data in a workload file no matter when the cloudlet starts or ends, no matter how long the cloudlet runs. The solution is the progress-based multi-resource cloudlet. When reading the utilization data, it is based on the running progress of the cloudlet instead of the time. That is to say, when the process of the cloudlet is 0%, it will utilize the first utilization data and when the process is 100%, it will utilize the last utilization data. The workload pattern will be in keeping with the data in the workload file. We don't need to worry about when the cloudlet start or ends.

2. **Multi-resource cloudlet scheduling and resource allocation scheme.** To fulfill multi-resource scheduling, besides multi-resource cloudlet, we also extend PowerDatacenter, DatacenterBroker, PowerHost, PowerVm, Cloudlet, CloudletSchedulerDynamicWorkload, etc., classes of CloudSim. In MultiRECloudSim, multi-resource cloudlet scheduling is consist of two parts: cloudlet assignment and resource allocation scheme. Cloudlet assignment refers to assigning cloudlets to VMs. It is a key part of multi-resource scheduling. We proposed the Main Resource Task Balance algorithm (MRTA), which can balance different resource workload thereby raising the utilization of resource and reduce power consumption. The main idea of Main Resource Task Balance Algorithm is the resource workload. First, we calculate the Normal Resource Load of a Task (NRLT) for the four resources. NRLT is a metric we design for comparing the demand for different types of resources because the unit of different resources are not the same. The resource with the largest NRLT is regarded as the main resource of the task. Then we will assign the task to the VM whose Vm Normal Resource Load (NRLV) of the main resource of the task. Vm Normal Resource Load (NRLV) is a similar metric as NRLT. It is only the sum of the NRLT of the tasks assigned to the VM, which aims to measure the workload of resources for a VM. Both the static and dynamic workload experiments have demonstrated the effectiveness of the MRTA algorithm. On the other hand, how to allocate the resources of VM to its cloudlets is another significant problem. In MultiRECloudSim, we implement the non-reservation scheme and reservation scheme of CPU allocation. For the non-reservation scheme, we have First-come-first-served and Max-Min fairness [30] algorithms. For the reservation scheme, we have CPU reservation algorithm. CPU reservation algorithm refers to the case where the maximum amount of CPU resource is reserved for a cloudlet when it starts and therefore there is certainly no shortage of CPU resource for this cloudlet. With this multi-resource scheduling mechanism, a lot of multi-resource scheduling algorithms can be easily researched.
3. **Multi-resource power consumption simulation.** On the basis of the multi-resource cloudlet scheduling mechanism, we continue to enable power-aware simulation. The principle is that according to the resource utilization and the power-utilization model, we can calculate the power consumption of a host. We can easily implement other CPU power model using the benchmark tests presented in [19]. In this paper, we also implement a simple memory power model and an IO power model. In CloudSim, it provides us the linear power model, the square power model and so on. We can design different power model for different types of resources and evaluate them with resource scheduling algorithms in MultiRECloudSim. Reducing power consumption is a critical issue in cloud computing. It will be a great help to possess the feature of multi-resource power-aware simulation.

Both multi-resource scheduling and power consumption are significant issue in cloud computing. With the new capabilities, MultiRECloudSim will show comprehensive functionality and superior convenience on multi-resource scheduling and power-aware simulation.

7. Conclusions

Owing to the support for flexible, scalable, efficient, and repeatable evaluation of resource scheduling and allocation policies for different applications, using simulation tools such as CloudSim is becoming more and more popular. Fast evaluation of scheduling and resource allocation algorithms within data centers becomes available. Therefore, we present a novel CloudSim-based simulation framework which supports the modeling of multi-resource scheduling and power consumption to make up the shortcomings of CloudSim in these aspects. Cloud simulation experiment with MultiRECloudSim has obvious priorities. (1) We can change configuration of host and power models of resources easily, and test the effect of algorithm under different parameters. (2) We simply simulate tasks that demand multi kinds of resources and define different resource allocation algorithms with fine-grained evaluation. (3) We could seamlessly switch static load and dynamical load experiment, which makes it able to simulate more actual scenes. (4) We support the power simulation of multi-resources. It is more accurate compared with single resource CPU power simulation. Additionally, in our work, we compare multiple combinations of task assignment algorithms and CPU allocation algorithms with each other from the aspects of time, power consumption and SLA violation. The result helps us to know more about the efficiency, power consumption and service quality's performance of the scheduling algorithms. Our proposed main resource task balance assignment algorithm raise the data center's resource utilization effectively and improve the throughput as well as service quality.

Although model can simulate most kinds of multi resource, it's still not perfect. Our further work includes supporting the dynamical load to memory, IO, and bandwidth. Besides, because there are lots of factors affecting the power consumption of vm, host and data center in realistic environment, the accuracy of our model is not so high. How to simulate more accurate power consumption is a problem we need to further study. Data storage is another important problem for cloud computing [39,40], and no simulation framework supports this kind of simulation. The data storage distribution is uneven, how to place the data blocks efficiently remains to be solved. It is a direction for our improvement work.

Acknowledgements

This work is partially supported by the [National Natural Science Foundation of China](#) (Grant No. 61402183), [Guangdong Natural Science Foundation](#) (Grant No. S2012030006242), [Guangdong Provincial Scientific and Technological Projects](#) (Grant Nos. 2016A010101007, 2016B090918021, 2014B010117001, 2014A010103022 and 2014A010103008), [Guangzhou Science and Technology Projects](#) (Grant Nos. 201607010048 and 201604010040) and the [Fundamental Research Funds for the Central Universities, SCUT](#) (No. 2015ZZ0098).

Appendix

```

public static List<SimPowerVm> createVms(int userId, int vms) { // create Vms
    List<SimPowerVm> list = new ArrayList<SimPowerVm>();
    long size = 10000; // image size (MB)
    int ram = 2048;
    int mips = 3067;
    long bw = 10000;
    long io = 500;
    int pesNumber = 1; // number of cpus
    String vmm = "Xen"; // VMM name
    SimPowerVm vm = null;
    CloudletCpuAllocator cpuAllocator = new CloudletCpuAllocatorSimple(mips);
    for (int i = 0; i < vms; i++) {
        vm = new SimPowerVm(i, userId, mips, pesNumber, ram, io, bw, size, 1, vmm, new
        SimProgressCloudletSchedulerDynamicWorkload(mips, pesNumber, cpuAllocator, new CloudletRamAllocatorSimple
        (ram), new CloudletIoAllocatorSimple(io), new CloudletBwAllocatorSimple(bw)), 5);
        list.add(vm);
    }
    return list;
}

private static SimDatacenterBroker createBroker() {
    SimDatacenterBroker broker = null;
    try {
        broker = new SimDatacenterBroker("Broker", new CloudletAssignmentPolicySimple());
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
    return broker;
}

```

```

private static SimPowerDatacenter createDatacenter(String name, double interval, int hostNumber) {
    List<SimPowerHost> hostList = new ArrayList<SimPowerHost>();
    int mips = 3067;
    int ram = 16 * 1024;
    long storage = 1000000; // host storage
    long bw = 10000;
    long io = 500;
    double staticPowerPercent = 0.01;
    PowerModel powerModelCpu = new PowerModelSpecPowerIbmX3550XeonX5675();
    PowerModel powerModelRam = new PowerModelRamSimple(ram);
    PowerModel powerModelIo = new PowerModelIoSimple(io);
    PowerModel powerModelBw = new PowerModelCubic(100, staticPowerPercent);
    for (int i = 0; i < hostNumber; i++) { //
        List<Pe> peList = new ArrayList<Pe>();
        peList.add(new Pe(0, new PeProvisionerSimple(mips)));
        hostList.add(new SimPowerHost(i, new SimRamProvisionerSimple(ram), new SimBwProvi-
sionerSimple(bw), new SimIoProvisionerSimple(io), storage, peList, new VmSchedulerTimeShared(peList),
powerModelCpu, powerModelRam, powerModelIo, powerModelBw));
    }
    String arch = "x86"; // system architecture
    String os = "Linux"; // operating system
    String vmm = "Xen";
    double time_zone = 10.0; // time zone this resource located
    double cost = 3.0; // the cost of using processing in this resource
    double costPerMem = 0.05; // the cost of using memory in this resource
    double costPerStorage = 0.001; // the cost of using storage in this resource
    double costPerBw = 0.02; // the cost of using bw in this resource
    LinkedList<Storage> storageList = new LinkedList<Storage>();
    DatacenterCharacteristics characteristics = new DatacenterCharacteristics(arch, os, vmm,
hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);
    SimPowerDatacenter datacenter = null;
    try {
        datacenter = new SimPowerDatacenter(name, characteristics, new VmAllocationPoli-
cySimple(hostList), storageList, interval);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return datacenter;
}

public static List<SimProgressCloudlet> createCloudlets(int userId, int cloudlets, boolean loadDynamic)
throws URISyntaxException, NumberFormatException, IOException { // create cloudlets
    List<SimProgressCloudlet> cloudletList = new ArrayList<SimProgressCloudlet>();
    String inputFolder = Example1.class.getClassLoader().getResource("utilization").
toURI().getPath();
    File[] files = new File(inputFolder).listFiles();
    String workloadPath = null;
    SimProgressCloudlet cloudlet = null;
    long length = 10000;
    long fileSize = 0;
    long outputSize = 0;
    int pesNumber = 1;
    int mips = 1000;
    int ram = 256;
    long io = 5;
    long bw = 0;
    double interval = 100;
    UtilizationModel utilizationModelCpu = new UtilizationModelFull();
    UtilizationModel utilizationModelRam = new UtilizationModelFull();
    UtilizationModel utilizationModelIo = new UtilizationModelFull();
    UtilizationModel utilizationModelBw = new UtilizationModelFull();
    for (int i = 0; i < cloudlets; i++) {
        if (loadDynamic) {
            workloadPath = files[i % 2].getPath();
            if (workloadPath != null)
                utilizationModelCpu = new UtilizationProgressModelByFile (workload-
Path, interval, false);
        }
        cloudlet = new SimProgressCloudlet(i, length, pesNumber, fileSize, outputSize,
mips, ram, io, bw, utilizationModelCpu, utilizationModelRam, utilizationModelIo, utilizationModelBw);
        cloudlet.setUserId(userId);
        cloudletList.add(cloudlet);
    }
    return cloudletList;
}

```

References

- [1] M. Allalouf, Y. Arbitman, M. Factor, R.I. Kat, K. Meth, D. Naor, Storage modeling for power estimation, in: Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference, 2009, p. 3.
- [2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, *Commun. ACM* 53 (4) (2009) 50–58.

- [3] Y. Awano, S. Kuribayashi, A joint multiple resource allocation method for cloud computing environments with different QoS to users at multiple locations, communications, computers and signal processing (PACRIM), in: IEEE Pacific Rim Conference, 2013, pp. 1–5.
- [4] L.A. Barroso, U. Holzle, The case for energy-proportional computing, *IEEE Comput.* 40 (12) (2007) 33–37.
- [5] R. Basmađjan, N. Ali, F. Niedermeier, H.d. Meer, G. Giuiani, A methodology to predict the power consumption of servers in data centres, in: Proceedings of the 2nd international conference on energy-efficient computing and networking, 2011, pp. 1–10.
- [6] S. Bera, S. Misra, J. Crodrigues, Cloud computing applications for smart grid: a survey, *IEEE Trans. Parallel Distrib. Syst.* 26 (2015) 1477–1494.
- [7] M. Bux, U. Leser, DynamicCloudSim: simulating heterogeneity in computational clouds, *Fut. Gener. Comp. Syst.* 46 (2015) 85–99.
- [8] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F.D. Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, *Softw. Pract. Exp.* 41 (1) (2011) 23–50.
- [9] S. Chaisiri, B. Lee, D. Niyato, Optimization of resource provisioning cost in cloud computing, *IEEE Trans. Serv. Comput.* 5 (2) (2012) 164–177.
- [10] W. Chen, E. Deelman, WorkflowSim: a toolkit for simulating scientific workflows in distributed environments in: E-Science (e-Science), in: IEEE 8th International Conference, 2012, pp. 1–8.
- [11] M. Dabbagh, B. Hamdaoui, M. Guizani, A. Rayes, An energy-efficient VM prediction and migration framework for overcommitted clouds. *IEEE Trans. Cloud Comput.* 2016. DOI 10.1109/TCC.2016.2564403.
- [12] H. David, C. Fallin, E. Gorbato, U.R. Hanebutte, O. Mutlu, Memory power management via dynamic voltage/frequency scaling, in: Proceedings of the 8th ACM international conference on Autonomic computing, 2011, pp. 31–40.
- [13] X. Fan, W. Weber, L. Andrebarroso, Power provisioning for a warehouse-sized computer, *ACM Sigarch Comput. Archit. News* 35 (2) (2007) 13–23.
- [14] S.K. Garg, R. Buyya, Networkcloudsim: Modelling parallel applications in cloud simulations, in: Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference, 2011, pp. 105–113.
- [15] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica, Dominant resource fairness: fair allocation of multiple resource types, *NSDL* 11 (2011) 24–24.
- [16] H. Goudarzi, M. Pedram, Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems, in: IEEE International Conference on Cloud Computing (CLOUD), 2011, pp. 324–331.
- [17] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, A. Akella, Multi-resource packing for cluster schedulers, *ACM SIGCOMM Comp. Commun.* 44 (4) (2014) 455–466.
- [18] T. Guérout, T. Monteil, G.G. Costa, R.N. Calheiros, R. Buyya, M. Alexandru, Energy-aware simulation with DVFS, *Simul. Model. Pract. Theory* 39 (2013) 76–91.
- [19] C.H. Hsu, S.W. Poole, Power signature analysis of the SPECpower_ssj2008 benchmark, in: IEEE International Symposium on Performance Analysis of Systems and Software, 2011, pp. 227–236.
- [20] A. Kansal, F. Zhao, J. Liu, N. Kothari, A.A. Bhattacharya, Virtual machine power metering and provisioning, in: Proceedings of the 1st ACM symposium on Cloud computing, 2010, pp. 39–50.
- [21] D. Kliazovich, P. Bouvry, S.U. Khan, GreenCloud: a packet-level simulator of energy-aware cloud computing data centers, *J. Supercomput.* 62 (3) (2012) 1263–1283.
- [22] C. Lefurgy, X. Wang, M. Sware, Server-level power control, International Conference on Autonomic Computing, 2007 4–4.
- [23] S.H. Lim, B. Sharma, G. Nam, E.K. Kim, C.R. Das, MDCSim: A multi-tier data center simulation, platform, in: IEEE International Conference on Cluster Computing and Workshops, 2009, pp. 1–9.
- [24] X. Li, X. Jiang, K. Ye, P. Huang, DartCSim+: Enhanced cloudsim with the power and network models integrated, in: IEEE Sixth International Conference on Cloud Computing, 2013, pp. 644–651.
- [25] W. Lin, C. Yang, C. Zhu, Energy efficiency oriented scheduling for heterogeneous cloud systems, *Int. J. Grid High Perform. Comput.* 6 (4) (2014) 1–14.
- [26] W. Lin, C. Liang, J.Z. wang, Bandwidth-aware divisible task scheduling for cloud computing, *Softw. Pract. Exp.* 44 (2) (2014) 163–174.
- [27] W. Lin, L. Tan, J.Z. wang, Novel resource allocation algorithm for energy-efficient cloud computing in heterogeneous environment, *Int. J. Grid High Perform. Comput.* 6 (1) (2014) 63–76.
- [28] W. Lin, C. Zhu, J. Li, Novel algorithms and equivalence optimisation for resource allocation in cloud computing, *Int. J. Web Grid Serv* 11 (2) (2015) 193–210.
- [29] W. Lin, S. Xu, J. Li, L. Xu, Z. Peng, Design and theoretical analysis of virtual machine placement algorithm based on peak workload characteristics, *Soft Comput.* (2015) 1–14.
- [30] W. Lin, W. Wu, J.Z. Wang, A heuristic task scheduling algorithm for heterogeneous virtual clusters, *Sci. Program.* (2016).
- [31] P. Mell, T. Grance, The NIST definition of cloud computing, 2011.
- [32] K. Shen, X. Zheng, Y. Song, Y. Bai, Fair multi-node multi-resource allocation and task scheduling in datacenter, in: IEEE Asia Pacific Cloud Computing Congress (APCloudCC), 2012, pp. 59–63.
- [33] A. Skendžić, B. Kovačić, E. Tijan, Effectiveness analysis of using solid state disk technology, *MIPRO*, 2016.
- [34] B. Speitkamp, M. Bichler, B. Martin, A mathematical programming approach for server consolidation problems in virtualized data centers, *IEEE Trans. Serv. Comput.* 3 (4) (2010) 266–278.
- [35] W. Wang, B. Liang, B. Li, Multi-resource fair allocation in heterogeneous cloud computing systems, *IEEE Trans. Parallel Distrib. Syst.* 26 (10) (2015) 2822–2835.
- [36] W. Wu, W. Lin, Z. Peng, An intelligent power consumption model for virtual machines under CPU-intensive workload in cloud environment, *Soft Comput.* (2016) 1–10.
- [37] K. Ye, X. Jiang, D. Huang, J. Chen, B. Wang, Live migration of multiple virtual machines with resource reservation in cloud computing environments, in: IEEE International Conference, 2011, pp. 267–274.
- [38] V. Chang, M. Ramachandran, Towards achieving data security with the cloud computing adoption framework, *IEEE Trans. Serv. Comput.* 9 (1) (2016) 138–151.
- [39] V. Chang, Towards a big data system disaster recovery in a private cloud, *Ad Hoc Netw.* 35 (2015) 65–82.
- [40] V. Chang, G. Wills, A model to compare cloud and non-cloud storage of big data, *Fut. Gener. Comp. Syst.* 57 (2016) 56–76.