Abstract—Cloud computing has been emerging as powerful computational architecture which will be a major technology in the development of the future Internet of Services. In order to save energy consumption we need to apply new energy-efficiency mechanism and observe its effects on a datacenter. As it is difficult to test new mechanism in real cloud computing environment and researchers often cannot reach the real cloud environment, simulation to model the mechanism and evaluate the results is necessary. Simulating a datacenter avoids spending time and effort to configure a real testing environment. This paper propose an extensible simulation framework called CloudSim which supports simulation, modelling of cloud environment and creation of one or more virtual machines(VMs) on a simulated node of a Data Center, jobs, and their mapping to suitable VMs, power to manage services and modeling of cloud infrastructure. Also manage your own mechanism in cloud computing. It also allows simulation of multiple Data Centers.

IndexTerms—Cloud computing, Cloudsim, Datacenter, Simulation

INTRODUCTION

Cloud computing delivers infrastructure, platform, and software as services, which are made available as subscription-based services in a pay-as-you-go model to users and in recent years, it is the biggest issue in IT fields. Industries such as Amazon, Google, Microsoft, HP and IBM have heavily invested in it. Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services[5]. Citrix and VMware which provide Virtualization, core technology in Cloud computing, are growing fast. This technology aims to offer distributed, virtualized, and elastic resources as utilities to end users.

Simulation-based approaches offer significant benefits to Cloud customers by allowing them to: (i) test their services in repeatable and controllable environment free of cost; and (ii) tune the performance bottlenecks before deploying on real Clouds[1]. At the provider side, simulation environments allow evaluation of different kinds of resource leasing scenarios under varying load and pricing distributions. Such studies could aid providers in optimizing the resource access cost with focus on improving profits. In the absence of such simulation platforms, Cloud customers and providers have to rely either on theoretical and imprecise evaluations, or on try-and-error approaches that lead to inefficient service performance and revenue generation.

CloudSim is a simulating program from CLOUDS lab in University of Melbourne for cloud computing. It is developed in java platform including the pre developed modules such as SimJava and GridSim.

CloudSim offers the following novel features: (i) support for modeling and simulation of large scale Cloud computing infrastructure, including data centers on a single physical computing node; and (ii) a self-contained platform for modeling data centers, service brokers, scheduling, and allocations policies. Among the unique features of CloudSim, there are: (i) availability of virtualization engine, which aids in creation and management of multiple, independent, and co-hosted virtualized services on a datacenter node; and (ii) flexibility to switch between spaceshared and time-shared allocation of processing cores to virtualized services. These compelling features of CloudSim would speed up the development of new resource allocation policies and scheduling algorithms for Cloud computing.

It is customizable tool[2], it allows extension and definition of policies in all the components of the software stack, which makes it suitable as a research tool that can handle the complexities arising from simulated environments. It allows fast evaluation of scheduling and resource allocation mechanism within cloud data centers which are sometimes not easy to access. CloudSim performs system modeling for Cloud system components, such as datacenter and VM, and behavioral modeling as resource provisioning policy and can analyze performance of them. CloudSim, however, does not consider failure mode of Cloud resources such as VM, host, datacenter, etc.

CLOUDSIM ARCHITECTURE[1]

Figure 1 shows the layered implementation of the CloudSim software framework and architectural components. At the lowest layer is the SimJava discrete
event simulation engine that implements the core functionalities required for higher-level simulation frameworks such as queuing and processing of events, creation of system components (services, host, data center, broker, virtual machines), communication between components, and management of the simulation clock. Next follows the libraries implementing the GridSim toolkit that support: (i) high level software components for modeling multiple Grid infrastructures, including networks and associated traffic profiles; and (ii) fundamental Grid components such as the resources, datasets, workload traces, and information services.

The CloudSim is implemented at the next level by programmatically extending the core functionalities exposed by the GridSim layer. CloudSim provides novel support for modeling and simulation of virtualized Cloud-based data center environments such as dedicated management interfaces for VMs, memory, storage, and bandwidth. CloudSim layer manages the instantiation and execution of core entities (VMs, hosts, data centers, application) during the simulation period. This layer is capable of concurrently instantiating and transparently managing a large scale Cloud infrastructure consisting of thousands of system components. The fundamental issues such as provisioning of hosts to VMs based on user requests, managing application execution, and dynamic monitoring are handled by this layer. A Cloud provider, who wants to study the efficacy of different policies in allocating its hosts, would need to implement its strategies at this layer by programmatically extending the core VM provisioning functionality. There is a clear distinction at this layer on how a host is allocated to different competing VMs in the Cloud. A Cloud host can be concurrently shared among a number of VMs that execute applications based on user-defined QoS specifications.

The top-most layer in the simulation stack is the User Code that exposes configuration related functionalities for hosts (number of machines, their specification and so on), applications (number of tasks and their requirements), VMs, number of users and their application types, and broker scheduling policies. A Cloud application developer can generate: (i) a mix of user request distributions, application configurations; and (ii) Cloud availability scenarios at this layer and perform robust tests based on the custom configurations already supported within the CloudSim.

As Cloud computing is a rapidly evolving research area, there is a severe lack of defined standards, tools and methods that can efficiently tackle the infrastructure and application level complexities. Hence in the near future there would be a number of research efforts both in academia and industry towards defining core algorithms, policies, application benchmarking based on execution contexts. By extending the basic functionalities already exposed by CloudSim, researchers would be able to perform tests based on specific scenarios and configurations, hence allowing the development of best practices in all the critical aspects related to Cloud Computing.

by using SimJava, we can avoid reimplementation of event handling and message passing among components. This save time and cost of software engineering and testing. Similarly, the use of the GridSim framework allowe to reuse its implementation of networking, information services, files, users, and resources. Since SimJava and GridSim have been extensively utilized in conducting cutting edge research in Grid resource management by several researchers. Therefore, bugs that may compromise the validity of the simulation have been already detected and fixed. By reusing these long validated frameworks, it is possible to focus on critical aspects of the system that are relevant to Cloud computing. At the same time taking advantage of the reliability of components that are not directly related to Clouds.

Figure 1. Layered CloudSim architecture.

CLASSES IN CLOUDSIM[1]

The Class design diagram for the simulator is depicted in Figure 2. In this section, we provide finer details related to the fundamental classes of CloudSim, which are building blocks of the simulator.

DataCenter. This class models the core infrastructure level services (hardware, software) offered by resource providers in a Cloud computing environment. It encapsulates a set of compute hosts that can be either homogeneous or
heterogeneous as regards to their resource configurations (memory, cores, capacity, and storage). Furthermore, every DataCenter component instantiates a generalized resource provisioning component that implements a set of policies for allocating bandwidth, memory, and storage devices.

**Figure 2: CloudSim class design diagram**

**DatacenterBroker.** This class models a broker, which is responsible for mediating between users and service providers depending on users’ QoS requirements and deploys service tasks across Clouds. The broker acting on behalf of users identifies suitable Cloud service providers through the Cloud Information Service (CIS) and negotiates with them for an allocation of resources that meet QoS needs of users. The researchers and system developers must extend this class for conducting experiments with their custom developed application placement policies.

**SANStorage.** This class models a storage area network that is commonly available to Cloud-based data centers for storing large chunks of data. SANStorage implements a simple interface that can be used to simulate storage and retrieval of any amount of data, at any time subject to the availability of network bandwidth. Accessing files in a SAN at run time incurs additional delays for task unit execution, due to time elapsed for transferring the required data files through the data center internal network.

**VirtualMachine.** This class models an instance of a VM, whose management during its life cycle is the responsibility of the Host component. As discussed earlier, a host can simultaneously instantiate multiple Vms and allocate cores based on predefined processor sharing policies (space-shared, time-shared). Every VM component has access to a component that stores the characteristics related to a VM, such as memory, processor, storage, and the VM’s internal scheduling policy, which is extended from the abstract component called VMScheduling.

**Cloudlet.** This class models the Cloud-based application services (content delivery, social networking, business workflow), which are commonly deployed in the data centers. CloudSim represents the complexity of an application in terms of its computational requirements. Every application component has a pre-assigned instruction length (inherited from GridSim’s Gridlet component) and amount of data transfer (both pre and post fetches) that needs to be undertaken for successfully hosting the application.

**CloudCoordinator.** This abstract class provides federation capacity to a data center. This class is responsible for not only communicating with other peer CloudCoordinator services and Cloud Brokers (DataCenterBroker), but also for monitoring the internal state of a data center that plays integral role in loadbalancing/application scaling decision making. The monitoring occurs periodically in terms of simulation time. The specific event that triggers the load migration is implemented by CloudSim users through Sensor component. Each sensor may model one specific triggering procedure that may cause the CloudCoordinator to undertake dynamic load-shredding.

**BWProvisioner.** This is an abstract class that models the provisioning policy of bandwidth to VMs that are deployed on a Host component. The function of this component is to undertake the allocation of network bandwidths to set of competing VMs deployed across the data center. Cloud system developers and researchers can extend this class with their own
policies (priority, QoS) to reflect the needs of their applications.

**MemoryProvisioner.** This is an abstract class that represents the provisioning policy for allocating memory to VMs. This component models policies for allocating physical memory spaces to the competing VMs. The execution and deployment of VM on a host is feasible only if the MemoryProvisioner component determines that the host has the amount of free memory, which is requested for the new VM deployment.

**VMProvisioner.** This abstract class represents the provisioning policy that a VM Monitor utilizes for allocating VMs to Hosts. The chief functionality of the VMProvisioner is to select available host in a data center, which meets the memory, storage, and availability requirement for a VM deployment. The default SimpleVMProvisioner implementation provided with the CloudSim package allocates VMs to the first available Host that meets the aforementioned requirements. Hosts are considered for mapping in a sequential order. However, more complicated policies can be easily implemented within this component for achieving optimized allocations, for example, selection of hosts based on their ability to meet QoS requirements such as response time, budget.

**VMAllocationPolicy.** This is an abstract class implemented by a Host component that models the policies (space-shared, time-shared) required for allocating processing power to VMs. The functionalities of this class can easily be overridden to accommodate specific processor sharing policies.

**CLOUDSIM WORK STYLE**

CloudSim work style is shown as Fig. 3. In general, the tasks from different users are relatively independent; consider there are \( m \) users, as User1, User2, ……User\( m \), \( n \) independent tasks, as T1, T2 ……Tn, \( n \) VMs, as VM1, VM2 ……VM\( n \) and \( p \) datacenters, as Datacenter1, Datacenter2 ……Datacenter\( p \).

It can be seen from Fig. 3, in the cloud computing environment, we carry out the task scheduling in the virtual machines. But in the grid computing environment, the task scheduling is carried out in the idle hardware resources directly.

**COMMUNICATION AMONG ENTITIES**

Figure 4 depicts the flow of communication among core CloudSim entities. In the beginning of the simulation, each Datacenter entity registers itself with the CIS (Cloud Information Service) Registry. CIS provides database level match-making services for mapping user requests to suitable Cloud providers. Brokers acting on behalf of users consult the CIS service about the list of Clouds who offer infrastructure services matching user’s application requirements. In case the match occurs the broker deploys the application with the Cloud that was suggested by the CIS.

![Figure 3. CloudSim Work Style](image-url)

**Figure 3. CloudSim Work Style**

The communication flow described so far relates to the
basic flow in a simulated experiment. Some variations in this flow are possible depending on policies. For example, messages from Brokers to Datacenters may require a confirmation, from the part of the Datacenter, about the execution of the action, or the maximum number of VMs a user can create may be negotiated before VM creation.

**INTERNAL PROCESSING OF DATA CENTER**

Processing of task units is handled by respective VMs; therefore their progress must be continuously updated and monitored at every simulation step. For handling this, an internal event is generated to inform the DataCenter entity that a task unit completion is expected in the near future. Thus, at each simulation step, each DataCenter entity invokes a method called updateVMsProcessing() for every host that it manages. Following this, the contacted VMs update processing of currently active tasks with the host. The input parameter type for this method is the current simulation time and the return parameter type is the next expected completion time of a task currently running in one of the VMs on that host. The next internal event time is the least time among all the finish times, which are returned by the hosts.

![Sequence Diagram](image)

Figure 5 Cloudlet processing update process.[6]

At the host level, invocation of updateVMsProcessing() triggers an updateCloudletsProcessing() method that directs every VM to update its tasks unit status (finish, suspended, executing) with the Datacenter entity. This method implements a similar logic as described previously for updateVMsProcessing() but at the VM level. Once this method is called, VMs return the next expected completion time of the task units currently managed by them. The least completion time among all the computed values is sent to the Datacenter entity. As a result, completion times are kept in a queue that is queried by Datacenter after each event processing step. The completed tasks waiting in the finish queue that are directly returned concern CloudBroker or CloudCoordinator. This process is depicted in Figure 5 in the form of a sequence diagram.

**CLOUDSIM MODELING**[4]

**Modeling the cloud**

The infrastructure-level services (IaaS) related to the clouds can be simulated by extending the Datacenter entity of CloudSim. The datacenter entity manages a number of host entities. The hosts are assigned to one or more VMs based on a VM allocation policy that should be defined by the cloud service provider.

A Datacenter can manage several hosts that in turn manage VMs during their life cycles. Host is a CloudSim component that represents a physical computing server in a Cloud: it is assigned a pre-configured processing capability (expressed in millions of instructions per seconds – MIPS), memory, storage, and a provisioning policy for allocating processing cores to virtual machines. The host component implements interfaces that support modeling and simulation of both single-core and multi-core nodes. To simulate a datacenter, you can just create a class which extends from the Cloud Datacenter, and the same as Host. A Cloudlet can model the cloud-based application services(such as content delivery, social networking, and businessworkflow). CloudSim orchestrates the complexity of an application in terms of its computational requirements. Every application service has a pre-assigned instruction length and data transfer (both pre and post fetches) overhead that is needed to undertake during its life cycle.

**Modeling the VM Allocation**

VM allocation (provisioning) is the process of creating VM instances on hosts that match the specific characteristics (storage, memory, configurations) of virtual machines, and requirements (availability zone) of the SaaS provider. CloudSim supports the development of custom application service models that can be deployed within a VM instance and its users are required to extend the core cloudlet object for implementing their application services. Once an application service id defined and modeled, it is assigned to one or more pre-instantiated VMs through a service specific allocation policy. Allocation of application-specific VMs to Hosts in a Cloud-based data center is the responsibility of a Virtual Machine Allocation controller component (called VmAllocationPolicy). This component exposes a number of custom methods for researchers and developers that aid in implementation of new policies based on optimizations goals. By default, VmAllocationPolicy implements a straightforward policy that allocates VMs to the Host in First-Come-First-Serve (FCFS) basis. To implement your own VM Allocation policy, just extends the class VmAllocationPolicy.

**Modeling the Dynamic Workloads**

To make a simulation-based evaluation applicable, it is important to conduct experiments using workload traces form a real system. In CloudSim 3.0, the authors used data provided as a part of the CoMon project, a monitoring infrastructure for PlanetLab. They have used the data on the CPU utilization by more than a thousand VMs form servers...
located at more than 500 places around the world. The interval of utilization measurements is 5 minutes and each traced file have 288 lines, therefore, each one represent a VMs CPU utilization about 24 hours. In the example which contained in the CloudSim originally, these workloads file are used to instance cloudlet which models cloud-based application services. The CloudSim simulator assigns the cloudlet to VMs sequentially. The CloudSim also model the workloads randomly. The UtilizationModelStochastic class implements a model, according to which a Cloudlet generates random CPU utilization every time frame.

**Modeling the Data Center Power Consumption**

Power consumption by computing nodes in data centers is mostly determined by the CPU, memory, disk storage, power supplies and cooling systems. Recent studies have shown that the power consumption by servers can be accurately described by a linear relationship between the power consumption and CPU utilization, even when Dynamic Voltage and Frequency Scaling (DVFS) is applied.

CloudSim provide several Power models, such as PowerModelSqrt, PowerModelLinear, PowerModelSquare and PowerModelCubic. These power models are all based on the linear relationship between power consumption and CPU utilization. As to analysis and build the precise power consumption model is a complex research problem. Therefore, instead of using an analytical model of power consumption by a server, CloudSim also provide power model called PowerModelSpecPower, which utilize real data on power consumption provided by the results of SPECpower benchmark.

**POLICY AND ALGORITHMS**

**A. VM Scheduling Policy**

CloudSim models scheduling of CPU resources at two levels: Host and VM. At Host level, the host shares fractions of each processor element (PE) to each VM running on it. Because resources are shared among VMs, this scheduler is called VmScheduler. The scheduler a host uses is a parameter of the Host constructor. In the VM level, each virtual machine divides the resources received from the host among Cloudlets running on it. Because in this level resources are shared among Cloudlets, this scheduler is called CloudletScheduler. The scheduler a VM uses is a parameter of its constructor. In both levels, there are two default policies available: the first policy, xSpaceShared (x stands for VmScheduler or CloudletScheduler), required PEs by Cloudlets/VMs are exclusively allocated. It means that if there are more running elements (VMs or Cloudlets) than available PEs, the last elements to arrive wait on a queue until enough resources are free. In the second policy, xTimeShared, fraction of available PEs are shared among running elements, and all the elements run simultaneously. Policies for VM scheduling and Cloudlet scheduling can be used in any combination. For example, researchers can use VmSchedulerTimeShared and CloudletSchedulerSpace-Shared, or researchers can use VmSchedulerTimeShared and CloudletSchedulerTimeShared. It is possible even having a host running VMs with different Cloudlet scheduling policies, or a data center with hosts with different VM scheduling policies. To define your own policy, you have to extend either VmScheduler or CloudletScheduler, create the methods for deciding sharing of PEs and pass the new class during construction of the relevant object. For example, extend VmScheduler and pass the object to the host. Or extend CloudletScheduler and pass the object to the VM.

**B. VM Provisioning Policy**

The provisioning problem consists of defining, among the available hosts in the data center, which one should receive a new machine requested by a user. Provisioning of hosts to VMs in data centers follows a simple strategy where the host with less running VMs receives the next VM. This behavior is defined in the VmAllocationPolicySimple class. To change this behavior, extend VmAllocationPolicy to define the new provisioning behavior, and pass this object in the initialization of Datacenter. The provisioning policy in the CloudSim 3.0 can be described in figure 6. VmAllocationPolicy is an abstract class that represents the provisioning policy of hosts to virtual machines in a datacenter. PowerVmAllocationPolicyAbstract is an abstract class which defined a power-aware VM allocation policy. PowerVmAllocationPolicyMigrationAbstract is an abstract class which defined a power-aware VM allocation policy that dynamically optimizes the VM allocation using...
migration. The other classes, which defined some different policy of power-aware VM allocation policy, are all extended from PowerVmAllocationPolicyMigrationAbstract or PowerVmAllocationPolicyAbstract. Researchers can also implement dynamic VM reallocation algorithms by implementing the optimizeAllocation method of the PowerVmAllocation-PolicyAbstract class, which is called at every time frame and passed with the full set of current VMs in the data center.

C. VM Selection Policy
The VM selection problem consists of defining, among the VMs in a certain host, which one should migrate to a new machine requested by a user because of the power consideration. PowerVmSelectionPolicy is an abstract class that represents the VM selection policy. CloudSim 3.0 defines four VM (Figure 7) selection policies for the host to choose which VM should be migrated. The Minimum Migration Time (MMT) policy select a VM that requires the minimum time to complete a migration relatively to the other VMs allocated to the host. The Random Choice (RC) policy selects a VM to be migrated according to a uniformly distributed discrete random variable. The Maximum Correlation (MC) policy selects those VMs to be migrated that have the highest correlation of the CPU utilization with other VMs. The Minimum Utilization (MU) policy selects a VM to be migrated that requires the minimum CPU utilization relatively to the other VMs in the host. To define your own VM selection policy, extend PowerVmSelectionPolicy to define the new selection policy, and pass this object in the initialization of Host.

D. Implement Your Own Algorithms

There are several places in CloudSim where you can implement your algorithm according to what the algorithm is intended to do. Except Scheduling policy, VM provisioning policy and VM selection policy, which describe above, here are several other classes that you may need to modify or extend in order to make your own simulation:

- DatacenterBroker – this class only submit a list of VMs to be created and schedules Cloudlets sequentially on them. Modifying the way VM provisioning requests are submitted to datacenters and the way cloudlets are submitted and assigned to VMs, researchers can create a new Broker class which extends DatacenterBroker class.
- Datacenter – this class behaves like an IaaS provider: it receives requests for VMs from brokers and creates the VMs in hosts.

To change the default behavior of DatacenterBroker and Datacenter, researchers can either extend these classes to add the intended behavior, or implement new ones from scratch. In the latter case, these new entities have to extend SimEntity and implement the processEvent (SimEvent) method.

- PowerVmAllocationPolicyMigrationAbstract – this class implement a power-aware dynamic VM consolidation algorithms that use VM live migration to dynamically reallocate VMs at every time frame. Researchers can implement new power-aware dynamic VM consolidation algorithms by extending this class and overriding the optimizeAllocation method.

This paper introduced the CloudSim simulator including its architecture, and how to use it to model the cloud environment. Simulation in evaluating Cloud computing offer significant benefits, as they allow Cloud developers: (i) to test performance of their provisioning and service delivery policies in a repeatable and controllable environment free of cost; and (ii) to tune the performance bottlenecks before real-world deployment on commercial Clouds. It supports flexible, scalable, efficient and repeatable evaluation of provisioning policies for different applications. In the future, we will use CloudSim platform to evaluate algorithm which aim at improve the average resource utilization of the cloud datacenter and achieve energy efficiency.

REFERENCES
University of Melbourne, Australia, pp.1-9, 2009.


