Evaluation of Cloud Systems*

Mihaela-Andreea VASILE[1], George-Valentin IORDACHE[1], Alexandru TUDORICA[1], and Florin POP[1,2,*]

[1]University *Politehnica* of Bucharest, Computer Science Department, Faculty of Automatic Control and Computers, Romania
[2]*NationalInstituteforResearchandDevelopmentinInformatics(ICI), Bucharest, Romania*
florin.pop@cs.pub.ro, mihaela.vasile@hpc.pub.ro
george.iordache@cs.pub.ro, alexandru.tudorica@cti.pub.ro
[*]*Corresponding Author*

Abstract. Modelling and simulation represent suitable instruments for evaluation of distributed system. These essential tools in science are used in Cloud systems design and performance evaluation. The chapter covers the fundamental skills for a practitioner working in the field of Cloud Systems to have, for the development of a correct methodology for the evaluation using simulation of Cloud services and components. We concentrate on subjects related to tasks scheduling and resource allocation with the focus on scalability and elasticity, the constraints imposed by SLA and the use of CloudSim for performance evaluation of Cloud Systems. Several metrics used in modelling and simulation are presented in this chapter.

Keywords: Resource Management, Task Scheduling, Cloud Computing, Service Level Agreement, CloudSim

1 Introduction

Cloud services are classified by NIST into three categories [1]: *IaaS* - Infrastructure as a Service, *PaaS* - Platform as a Service, *SaaS* - Software as a Service.

IaaS offers hardware infrastructure like switches, routers, servers, load balancers, firewalls, storage. Usually these resources are virtual resources if these are bare metal versions, then the term used is Metal as a Service (MaaS). Notable examples of IaaS are Amazon Web Services, Google Compute Engine, IBM Softlayer (which has both IaaS and MaaS offerings).

PaaS goes beyond IaaS and offers a computing platform, which includes managed operating system, execution environment, storage, database and HTTP server. The platform is managed by the provider, this allows application developers to build applications without the complexity and inherent cost of managing the underlying stack.

* This work was presented during the event cHiPSet Training School "*New Trends in Modeling and Simulation in HPC Systems*" Bucharest, Romania, 21-23 September 2016, supported by cHiPSet ICT COST Action IC1406.

SaaS compared to PaaS or IaaS offers an application that is completely managed by the provider. Typical applications include databases, CRM software, Git repositories, etc. The pricing is usually pay-per-use or subscription based.

Modern day Cloud computing has adapted to the service oriented architecture by the means of microservices. Microservices are services that are isolated from each other and communicate over a network in order to fulfill a goal. The main difference between microservices and SOA is that the latter focuses on reusability and integrating larger business applications, while the former focuses on replacing an application with a set of services that can be replaced, updated and scaled independently. Each microservice can be implemented in different programming languages, databases and software environment thus increasing the development speed. Also in contrast with SOA each microservice defines its required resources. Microservices also pose the advantage that you can scale specific bottlenecks in your application, by assigning a different number of instances to each microservice. Architecturally speaking, microservices should be designed with fault in mind. If a microservice instance fails while processing a task, that task gets assigned to another instance of the same microservice.

For high availability application the placement of these instances is constrained by locality restrictions, for example cloud service providers like Amazon often provide multiple Availability Zones (AZs), especially designed for high availability application such that they have a small chance of failing simultaneously. A microservice scheduler must be able to balance the number of instances of a service across a number of AZs depending on the availability restrictions. Other restrictions imposed on microservices might be: data locality, specific machine requirements like virtualization mode used, presence of a certain generation of GPU or processor generation.

Scheduling microservices is very similar to the online multi-capacity bin packing problem, for which multiple algorithms exist. This problem was studied for scheduling Virtual Machines (VMs) and is sometimes combined with offline phases of the algorithm [2] for increased performance. For example Song et al. [3] presents an online algorithm for scheduling VMs with using as few servers as possible reaching a competitive ratio of 3/2. Another algorithm named HarmonicMix [4] improves on the previous work, reaching a competitive ratio of 4/3, meaning that the number of bins necessary is only 4/3 bigger than offline scheduling algorithms with infinite migration.

These algorithms make the assumption that all machines are equal, but in order to optimize for the smallest price, we cant make such an assumption. The nature of microservices, fault tolerance and scalability, make them able to be run on a cluster of Amazon Spot instances. Amazon Spot instances are spare virtual machine capacity auctioned off in real time by Amazon to the highest bidder. Amazon Spot prices are usually 1/4-1/6 lower than their OnDemand counterparts, thus bringing huge cost savings. Each type of instance in each availability zone has a dynamic price set by supply and demand. Thus some instance types might become unavailable for periods of time. Qu et al. [5] has shown how you can balance availability with price while using Amazon Spot Instances to run web applications on them, by over provisioning resources depending on the availability constraints of the application.

This chapter presents the general features of cloud systems and services in Section 2, the main evaluation metrics in Section 3, then in Section 4 address the SLA issue for Cloud Systems. Section 5 presents the modeling of Cloud Systems using CloudSim and the extension for it for scheduling algorithms.

2 General features of cloud systems and services

Cloud solutions allow users to access via Internet various types of resources such as existing applications in the Cloud, frameworks that can be used for development of custom built applications, access to Virtual Machines for installing operating systems and also storage and sharing solutions.

Table 1. General features of cloud systems and services.

Feature	**Description**
Availability	degree to which a system is in a specified state.
Reliability	power to remain functional with time without.
Efficiency	the ratio of the useful work performed by a system to the total energy expended or heat taken in.
Reusability	the level to which a component may be used in a number of systems or applications.
Interoperability	the capability to integrate with different standards and technologies.
Adaptability	the level of efficiency in adjusting a solution for the utilization in different context.
Usability	the quantity to which a Cloud service could be used by particular consumers to gain certain aims with usefulness.
Modifiability	the capability to make modifications to a component rapidly and cost effectively.
Sustainability	environmental effect of the Cloud service (usual carbon footprint or even energy capable of the Cloud services).
Scalability	the capability of a system to handle a growing amount of resources and workloads.
Elasticity	"the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible" [6].

The Cloud is now a significant choice for multiple types of users, common individuals, scientists or technical users so large datasets are generated, and have to be processed. The scheduling algorithms used in Clouds can be improved to fit the new patterns of jobs and big data sets using hybrid approaches that will consider independent tasks, tasks with dependencies, asymptotic scale requests or smaller rates of arriving jobs [7], [8], [9]. All this algorithms are designed considering the main features of Cloud systems and services, which are presented in Table 1.

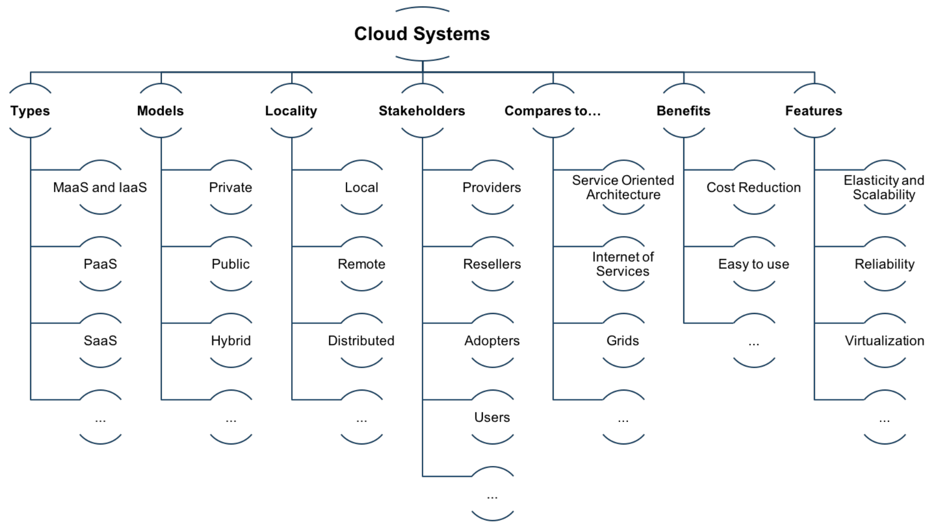


Fig. 1. Brief overview of Cloud Systems.

A brief overview of Cloud systems considering dimensions like type, model, locality, stakeholders, comparison with other models, benefits and future is presented in Figure 1.

The recent Cloud computing paradigm was designed in order to provide end users and businesses with various advantages such as: self-service provisioning, broad network access, resource pooling , elasticity, measured service, pay per use [1]. This approach is based on utility computing were we have infinite resources (as much as you need) and a concrete billing model (e.g. hourly).

The main benefits of Cloud systems are represent by the possibility to use high-scale / low-cost providers, by having any time / place access via a web browser, rapid scalability (incremental cost and load sharing), and a great focus on local IT systems.

We still have several concerns and open issues for Cloud systems, like performance evaluation, reliability and interoperability assurance, SLA negotiation, control of data and offered service parameters, no standard API (a mixture of SOAP and REST and other standards), and many open issued about privacy, security, and trust.

We classify the main characteristics and issues about Cloud system considering nonfunctional aspects, economic models and technological features. This synthetic approach is presented in Figure 2.

Nonfunctional	Elasticity (ex: Amazon EC2)
	Reliability (ex: Vmware ecosystem)
	Quality of Service (ex: Amazon S3)
	Agility and adaptability (ex: FlexNet)
	Availability (ex: MS Azure)
Economic	Cost reduction
	Pay per use
	Improved time to market
	Return of investment (ROI)
	Turning CAPEX (capital expenditure) into OPEX (operational expenditure)
	Going Green
Technological	Virtualization (ex: Virtual Box)
	Multi-tenancy (ex: MS SQL)
	Security, privacy and compliance
	Data Management (ex: WebSphere)
	APIs and / or Programming Enhancements (ex: Hadoop)
	Tools

Fig. 2. Cloud computing characteristics/issues.

3 Evaluation Metrics

The evaluation metrics are presented for all described features in the previous section. A comprehensive and well described taxonomy of evaluation metrics where presented in [10]. According with this evaluation we have *basic performance metrics* (execution time, speedup, efficiency, scalability, elasticity, etc.), *Cloud capabilities* (latency, throughput, bandwidth, recoverability, storage capacity, software tunning, etc.), and *Cloud productivity* (QoS, power demand, cost of services, availability, productivity, SLA, security, etc.). The **evaluations metrics** can be grouped by [11], [12], [13]:

– *Availability metrics*: "flexibility, accuracy, response time";
– *Reliability metrics*: "service constancy, accuracy of service, fault tolerance, maturity, recoverability";
– *Efficiency metrics*: "utilization of resource, ratio of waiting time, time behavior";
– *Reusability metrics*: "readability, coverage of variability, publicity";
– *Interoperability metrics*: "service Modularity, service interoperability, LISI (Level of Information System Interoperability)";
– *Adaptability metrics*: "coverage of Variability, other performance metrics";

– *Usability metrics*: "operability, attractiveness, learnability";
– *Modifiability metric*: "MTTC (Mean Time To Change)";
– *Sustainability metrics*: "DPPE (Data Centre Performance per Energy) parameter, PUE (Power Usage Efficiency)";
– *Scalability metric*: "average of assigned resources among the requested resources";
– *Elasticity metrics*: "boot time, suspend time, delete time, provision (or Deployment) time, total acquisition time";
– *Communication metrics*: "packet loss frequency, connection error rate, transfer bit/Byte speed, transfer delay";
– *Computation metrics*: "CPU Load, benchmark OP (FLOP) rate, instance efficiency (% CPU peak)";
– *Storage metrics*: "response time, latency, bandwidth, capacity";
– *Memory metrics*: "mean hit time, memory bit/Byte Speed, random memory update rate, response time (ms)";
– *Time metrics*: "computation time, communication time";
– *Data Security metrics*: "Is SSL applicable, communication latency over SSL, auditability, resistance to attacks";
– *Authentication metrics*: "meaning, sensitivity, effectiveness, confidentiality".

Other evaluation metrics can be defined to evaluate task scheduling and resource allocation systems [14], [15], [16]. We highlights here several performance evaluation metrics for a set of N jobs that is subject to a scheduling algorithm or policy in a Cloud system:

$$AverageWaitTime = \frac{1}{N} \sum_{j \in Jobs} (StartTime_j - SubmitTime_j).$$

$$AverageTurnaroundTime = \frac{1}{N} \sum_{j \in Jobs} (EndTime_j - SubmitTime_j).$$

$$FractionOfJobsTransferred = \frac{NumberOfJobsMigrated}{TotalNumberOfJobs}.$$

$$FractionDataVolumeTransferred = \frac{\sum_K (InputSize_K + OutputSize_K)}{\sum_J (InputSize_J + OutputSize_J)}.$$

$$DataMigrationOverhead = \frac{TotalDataMigrationTime}{\sum_J (EndTime_J - QueueTime_J)}.$$

These are several composed metrics defined for a task scheduling systems, but we can define many other evaluation metrics, according with the defined model and properties.

4 Performance and Service Level Agreement

One of the most important constraints of resource allocation techniques in the Cloud is the level of client satisfaction. This level is described by the Service Level Agreement (SLA) contract, which represents as a service level warranty between the provider and the customer of the service. Usually, some of the most important goals of the SLA contract are given by the necessity to have a common language between the customer and the services provider, and to verify the level of customer satisfaction during the use of the agreed services. An SLA contract is designed and planned based on the objective requests related to the cost reduction, efficiency increase and high performance, availability and highest level of security of the provider Cloudbased services.

Designing and implementing a SLA contract is a usual open discussion, because it often involves complex simulations or difficult results to analyze and implement. Our article has the purpose of presenting a survey of how Service Level Agreements (SLAs) are specified in Cloud computing environments. One of the methods for optimizing the resource allocation techniques is by satisfying the specifications of the SLA. The analysis of the level of satisfaction of the Service Level Agreement (SLA) and the improvement of the Quality of Service (QoS) is very important when studying those methods for optimizing the Cloud resource allocation.

When designing a Service Level Agreement contract in general we can discuss about the following phases [17], [18], [19], [20], [21] (see Figure 3):

- The first phase in the design of an SLA contract is the SLA creation. During this phase service providers propose a SLA contract based on their capabilities and the contract contains SLA Offers. When discussing about the service consumers we refer to the SLA requirements specifications.
- The second phase in the design of an SLA contract is the SLA discovery and selection. During this phase, there exists a discovery of the offered services from different service providers and the selection of the services that satisfy both functional and non-functional requirements.
- The third phase in the design of an SLA contract is the SLA negotiation and it represents the negotiation and renegotiation step between providers and consumers.
- The fourth phase in the design of an SLA contract is the SLA monitoring phase when the service is starting and is provided to the consumer. During this phase the consumer monitors and validates the service characteristics offered by the service provider.
- The fifth phase when discussing about an SLA contract is the SLA termination which occurs when the SLA contract expires or either the consumer or the provider decide to end the agreement.

Cloud computing systems (or hosting datacenters) represent one of the main research areas in the field of distributed systems. Utility computing, reliable data storage, and infrastructure-independent computing are example applications of such systems [22].

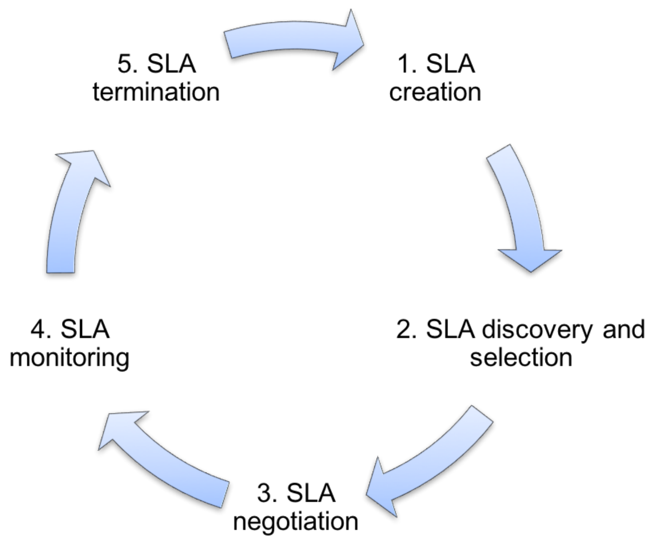


Fig. 3. SLA contract management phases.

Because adopting the Cloud services has various reasons such as lower costs because of the economy of resources (in comparison with when the client has to buy the necessary resources for example performant servers, etc.), transferring the responsibility when discussing the availability, maintenance, backup and lower licensing prices of the applications that are in place in the Cloud [23]. On the other hand, the fact that when adopting the Cloud computing paradigm with different purposes some of the service characteristics is transferred out of the customer control to the Cloud computing services providers. For this reason, there is a need for a contract (Service Level Agreement (SLA)) between the customer and the provider [24].

In addition, one of the characteristics of the contract is represented by the profit in the system, which depends on how the system can meet the SLA. (e.g. average response time, number of jobs completed per unit of time, availability of the resources in the system etc). The SLA contract usually specifies the constraints that need to be satisfied by the system in order to achieve the level of client satisfaction and system quality of service (QoS) agreed in this contract. Another way of thinking about a Service Level Agreement (SLA) contract is that it represents a complex document that describes the parameters that need to be satisfied in the time and at the values described by the range specified in the Service Level Agreement.

In the following section of this article we present int three Tables the Cloud SLAs which are drafted by Cloud providers. In this section we introduce Service Level Agreement (SLA) and Service level ranking criterion (Table 2) based on the JDN/CloudScreener/Cedexis U.S. Cloud Benchmark March 2016.

Table 2. Service Level Agreement and Service level ranking criterion.

	AWS	**Google**	**Microsoft**	**Rackspace**	**IBM Softlayer**
Announced SLA	99,95%	99,95%	99,95%	99,90%	99,73%
Service level ranking	90.83	90.83	79,38	73,33	73,21

The service level index is based on different qualitative criterion such as the geographical coverage (presence in the U.S. and outside of the U.S.), the number of certifications, the SLA, and the range of VM (the full ranking methodology is available on the JDN/CloudScreener/Cedexis U.S. Cloud Benchmark website: http://www.journaldunet.com/us-Cloud-benchmark/) [25]. This article presented a survey of the existing major Cloud providers and how Service Level Agreements (SLAs) have been defined by these providers.

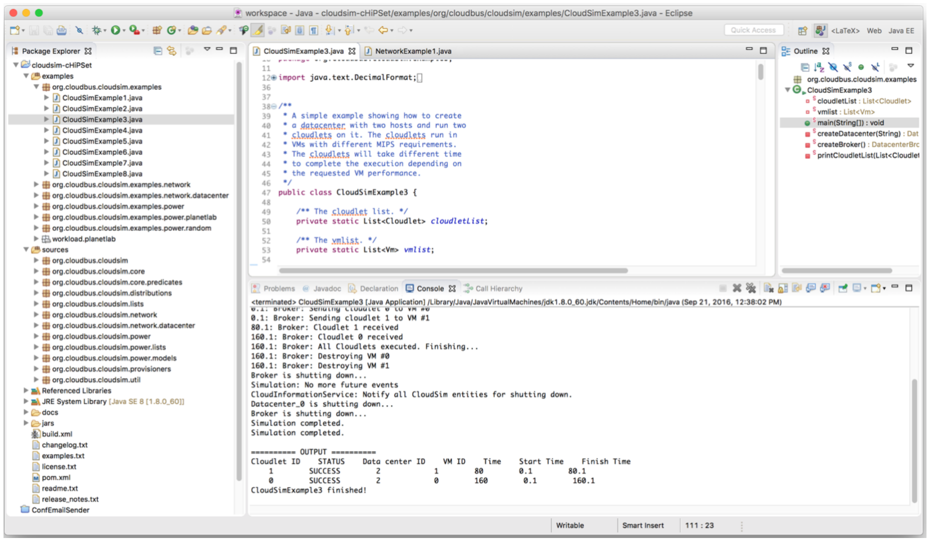
SLAs are very important in utility computing systems because they characterize the various interactions between the Cloud providers and the clients or consumers. The future research in SLA-oriented Cloud computing has to take into account the following goals:

- the service management has to be based on the requested levels of service characteristics; the characteristics that need to be taken into consideration when designing a Service Level Agreement (SLA) are related both to the computational risks and the service requirements;
- to identify the execution risks involved in the execution of applications, risks that might have an impact on the levels of performance specified in the Service Level Agreements (SLAs);
- there has to exist an equilibrium between the customer satisfaction and the level of provider profit;
- there might be a need to model the different resource management designs that are based both on the customers service demands and existing service properties;
- there are various operations that need to be taken into consideration when decinding to construct a Service Level Agreements contract such as: discover-service provider, define-SLA elements, establish-agreement, monitor SLA violation, terminate-SLA and SLA violation control [17].

Currently, the automatic negotiation of a Service Level Agreement in Cloud computing is still an open issue. Other open issues are scalability and heterogeneity of a service in Cloud computing, dynamic environmental changes, multiply QoS parameters and SLA suitable for cross domains [17]. Finally, we need to take into consideration the fact that the SLA needed in order to define the trust and quality of services has to be based on an agreed framework that represents a contract between consumer and provider about service terms such as: performance, availability and billing [26]. All these challenges are still open and can be explored in the future.

5 Modeling of Cloud systems using CloudSim

The CloudSim [27] Java toolkit allows the modeling of different entities in a Cloud environment and simulate various scenarios: evaluate the configuration of a Cloud System, resource allocation policies or scheduling algorithms. CloudSim is an extensible framework (developed in CLOUDS Laboratory, Computer Science and Software Engineering Department of the University of Melbourne) due to its high modularity. The Cloud entities: data centers, hosts, VMs, jobs, inter-host agreements or VM allocation policies are modeled as classes in different packages that can be interconnected or extended to enhance them with additional functionality. A quick look over running CloudSim environments in Eclipse is presented in Figure 4. CloudSim has the main benefit by having cloud resource provisioning modules, energy-efficient management of data center resources strategies and support for optimization.



```
import java.text.DecimalFormat;

/**
 * A simple example showing how to create
 * a datacenter with two hosts and run two
 * cloudlets on it. The cloudlets run in
 * VMs with different MIPS requirements.
 * The cloudlets will take different time
 * to complete the execution depending on
 * the requested VM performance.
 */
public class CloudSimExample3 {

    /** The cloudlet list. */
    private static List<Cloudlet> cloudletList;

    /** The vmlist. */
    private static List<Vm> vmlist;
```

```
<terminated> CloudSimExample3 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java (Sep 21, 2016, 12:38:02 PM)
0.1: Broker: Sending cloudlet 1 to VM #1
80.1: Broker: Cloudlet 1 received
160.1: Broker: Cloudlet 0 received
160.1: Broker: All Cloudlets executed. Finishing...
160.1: Broker: Destroying VM #0
160.1: Broker: Destroying VM #1
Broker is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Broker is shutting down...
Simulation completed.
Simulation completed.

========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    1        SUCCESS        2              1        80        0.1        80.1
    0        SUCCESS        2              0        160        0.1        160.1
CloudSimExample3 finished!
```

Fig. 4. Quick look over running CloudSim environment in Eclipse.

In this section we analyze the required steps to extend the CloudSim framework for implementing a custom scheduling algorithm and evaluate the simulation results and algorithm performance.

Cloudlet represents the abstraction of a Job/Task. Some of its properties are the length (computational requirements) and file size (IO), either for input or output. For statistics computation, it stores the VM that executed it, the start and finish execution times.

We will extend the Cloudlet by creating an additional object: Task that stores application specific information, and is connected to a Cloudlet object using the same id value.

```
public class Task {
  // rank the current task reported to the complete set of
     tasks
  public double processingRank;
  // rank the current task reported to the complete set of
     tasks
  public double ioRank;
  // connect the Task to a Cloudlet using this attribute
  public int id ;
  public long length;
  public long fileSize;
  public long outputSize;
  public int pesNumber;
  public long deadline;
  public long io;
```

Extend a Cloudlet using the Task class.

A *VM* object holds the properties of the underlying hardware, a link to the physical host and the policy for submitting tasks on PEs. We can extend the VM with the Resource object and add the specific attributes required by the scheduling algorithms, in our case, we added the resource load attribute.

```
public class Resource {
    // connect the Resource to a VM using this attribute
  private int id;
  // work load of the current VM
  private int load;
  public ArrayList<Task> schedTasks = new ArrayList<Task>();
  private int mips;
  private int ram;

  public Resource(int id, int load, int mips, int ram) {
```

Extend a VM using the Resource class.

The *DatacenterBroker* handles the allocation of Cloudlets on VMs using the function *submitCloudlets*. The implementation of a scheduling algorithm can be done by extending this class and overwriting the submitCloudlets function.

```
  protected void submitCloudlets() {
    int vmIndex = 0;
    for (Cloudlet cloudlet : getCloudletList()) {
      Vm vm;
      // if user didn't bind this cloudlet and it has not been
         executed yet
      if (cloudlet.getVmId() == -1) {
        vm = getVmsCreatedList().get(vmIndex);
      } else { // submit to the specific vm
        vm = VmList.getById(getVmsCreatedList(), cloudlet.
            getVmId());
        if (vm == null) { // vm was not created
          Log.printLine(CloudSim.clock() + ": " + getName() +
              ": Postponing execution of cloudlet "
              + cloudlet.getCloudletId() + ": bount VM not
                  available");
          continue;
        }
      }

      Log.printLine(CloudSim.clock() + ": " + getName() + ":
          Sending cloudlet "
          + cloudlet.getCloudletId() + " to VM #" + vm.getId()
              );
      cloudlet.setVmId(vm.getId());
      sendNow(getVmsToDatacentersMap().get(vm.getId()),
          CloudSimTags.CLOUDLET_SUBMIT, cloudlet);
      cloudletsSubmitted++;
      vmIndex = (vmIndex + 1) % getVmsCreatedList().size();
      getCloudletSubmittedList().add(cloudlet);
    }

    // remove submitted cloudlets from waiting list
    for (Cloudlet cloudlet : getCloudletSubmittedList()) {
      getCloudletList().remove(cloudlet);
    }
  }
```

Default scheduling in DatacenterBroker.

```
public class Scheduler  extends DatacenterBroker implements
    IScheduler{

  public static SchedulingMethods method;
  public static ArrayList<Task> tasks;
  public static ArrayList<Resource> resources;

  @Override
  protected void submitCloudlets() {
    switch(method){
    case Default: defaultSchedule();
      break;
    case SJF: sjf();
      break;
    case ClusteringSJF: clusteringSJF();
      break;
    }
  }
```

Extend a DatacenterBroker using the Scheduler class.

```
Collections.sort(tasks, new Comparator<Task>() {

  @Override
  public int compare(Task o1, Task o2) {
    int r = (int)(o1.length - o2.length);
    return r != 0 ? r : (int)(o1.io - o2.io);
  }
});

for (int i = 0; i < tasks.size(); i++) {
  int id = tasks.get(i).id;
  Cloudlet cloudlet = cloudletList.get(id);
  Vm vm;
  // if user didn't bind this cloudlet and it has not been
      executed yet
  if (cloudlet.getVmId() == -1) {
    vm = getVmsCreatedList().get(vmIndex);
  } else { // submit to the specific vm
    vm = VmList.getById(getVmsCreatedList(), cloudlet.
        getVmId());
    if (vm == null) { // vm was not created
      Log.printLine(CloudSim.clock() + ": " + getName() +
          ": Postponing execution of cloudlet "
          + cloudlet.getCloudletId() + ": bount VM not
              available");
      continue;
    }
  }

  cloudlet.setVmId(vm.getId());
  sendNow(getVmsToDatacentersMap().get(vm.getId()),
      CloudSimTags.CLOUDLET_SUBMIT, cloudlet);
  cloudletsSubmitted++;
  vmIndex = (vmIndex + 1) % getVmsCreatedList().size();
  getCloudletSubmittedList().add(cloudlet);
}

// remove submitted cloudlets from waiting list
for (Cloudlet cloudlet : getCloudletSubmittedList()) {
```

Implement the SJF algorithm.

6 Conclusion

The new trends in modeling and simulation of Cloud Systems require performance evaluation metrics with a high level of accuracy. We presented in this chapter several feature of Cloud systems and services ans a set of evaluation metrics. We included a practical example using CloudSim. which analyze the required steps to extend the CloudSim framework for implementing a custom scheduling algorithm and evaluate the simulation results.

Acknowledgment

The research presented in this paper is supported by the projects: *DataWay*: Real-time Data Processing Platform for Smart Cities: Making sense of Big Data - PN-II-RU-TE-2014-4-2731; *MobiWay*: Mobility Beyond Individualism: an Integrated Platform for Intelligent Transportation Systems of Tomorrow - PN-II-PT-PCCA-2013-4-0321; and *cHiPSet*: High-Performance Modelling and Simulation for Big Data Applications, ICT COST Action IC1406.

We would like to thank the reviewers for their time and expertise, constructive comments and valuable insight.

References

1. Mell, P., Grance, T.: The nist definition of cloud computing. Communications of the ACM **53**(6) (2010) 50
2. Leinberger, W., Karypis, G., Kumar, V.: Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In: Parallel Processing, 1999. Proceedings. 1999 International Conference on, IEEE (1999) 404–412
3. Song, W., Xiao, Z., Chen, Q., Luo, H.: Adaptive resource provisioning for the cloud using online bin packing. IEEE Transactions on Computers **63**(11) (2014) 2647–2660
4. Kamali, S.: Efficient bin packing algorithms for resource provisioning in the cloud. In: Algorithmic Aspects of Cloud Computing. Springer (2016) 84–98
5. Qu, C., Calheiros, R.N., Buyya, R.: A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances. Journal of Network and Computer Applications **65** (2016) 167–180
6. Herbst, N.R., Kounev, S., Reussner, R.: Elasticity in cloud computing: What it is, and what it is not. In: Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13). (2013) 23–27
7. Vasile, M.A., Pop, F., Tutueanu, R.I., Cristea, V., Kołodziej, J.: Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. Future Generation Computer Systems **51** (2015) 61–71
8. Vasile, M.A., Pop, F., Tutueanu, R.I., Cristea, V.: Hysarc2: hybrid scheduling algorithm based on resource clustering in cloud environments. In: International Conference on Algorithms and Architectures for Parallel Processing, Springer (2013) 416–425
9. Sfrent, A., Pop, F.: Asymptotic scheduling for many task computing in big data platforms. Information Sciences **319** (2015) 71–91
10. Hwang, K., Bai, X., Shi, Y., Li, M., Chen, W.G., Wu, Y.: Cloud performance modeling with benchmark evaluation of elastic scaling strategies. IEEE Transactions on Parallel and Distributed Systems **27**(1) (2016) 130–143
11. Bardsiri, A.K., Hashemi, S.M.: Qos metrics for cloud computing services evaluation. International Journal of Intelligent Systems and Applications **6**(12) (2014) 27

12. Kan, S.H.: Metrics and models in software quality engineering. Addison-Wesley Longman Publishing Co., Inc. (2002)
13. Iosup, A., Ostermann, S., Yigitbasi, M.N., Prodan, R., Fahringer, T., Epema, D.: Performance analysis of cloud computing services for many-tasks scientific computing. IEEE Transactions on Parallel and Distributed systems **22**(6) (2011) 931–945
14. Topcuoglu, H., Hariri, S., Wu, M.y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE transactions on parallel and distributed systems **13**(3) (2002) 260–274
15. Feitelson, D.G., Rudolph, L.: Metrics and benchmarking for parallel job scheduling. In: Workshop on Job Scheduling Strategies for Parallel Processing, Springer (1998) 1–24
16. Pop, F., Cristea, V., Bessis, N., Sotiriadis, S.: Reputation guided genetic scheduling algorithm for independent tasks in inter-clouds environments. In: Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on, IEEE (2013) 772–776
17. Wu, L., Buyya, R.: Service level agreement (sla) in utility computing systems. IGI Global (2012)
18. Debusmann, M., Keller, A.: Sla-driven management of distributed systems using the common information model. In: Integrated Network Management VIII. Springer (2003) 563–576
19. Alhamad, M., Dillon, T., Chang, E.: Sla-based trust model for cloud computing. In: Network-Based Information Systems (NBiS), 2010 13th International Conference on, IEEE (2010) 321–324
20. Venticinque, S., Aversa, R., Di Martino, B., Rak, M., Petcu, D.: A cloud agency for sla negotiation and management. In: European Conference on Parallel Processing, Springer (2010) 587–594
21. Sahai, A., Machiraju, V., Sayal, M., Van Moorsel, A., Casati, F.: Automated sla monitoring for web services. In: International Workshop on Distributed Systems: Operations and Management, Springer (2002) 28–41
22. Goudarzi, H., Ghasemazar, M., Pedram, M.: Sla-based optimization of power and migration cost in cloud computing. In: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). (May 2012) 172–179
23. Goudarzi, H., Pedram, M.: Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems. In: Cloud Computing (CLOUD), 2011 IEEE International Conference on, IEEE (2011) 324–331
24. Dastjerdi, A.V., Tabatabaei, S.G.H., Buyya, R.: A dependency-aware ontology-based approach for deploying service level agreement monitoring services in cloud. Software: Practice and Experience **42**(4) (2012) 501–518
25. CCMBenchmark: Jdn, cloudscreener, cedexis u.s. cloud benchmark website, 2016, http://www.journaldunet.com/us-cloud-benchmark
26. Alhamad, M., Dillon, T., Chang, E.: Conceptual sla framework for cloud computing. In: 4th IEEE International Conference on Digital Ecosystems and Technologies, IEEE (2010) 606–610
27. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience **41**(1) (2011) 23–50