ATAL BIHARI VAJPAYEE- INDIAN INSTITUTE OF INFORMATION TECHNOLOGY AND MANAGEMENT GWALIOR-474015

CC LAB ASSIGNMENT

CASE STUDY

ON

CloudSim

Submitted To:
Dr. Neetesh Kumar

Submitted By:-
Amal Shaji (2017BCS-010)
Sandarbh Yadav (2017BCS-027)
Vaibhav Garg (2017BCS-038)

1. What is the simulator?

Ans:

It is not always possible as well as feasible to give each student or person a real system including hardware to understand these systems and their working practically. Subjects like computer networks, cloud computing, digital circuits and many more requires person to understand the subject practically since what we study theoretically considers thing to be ideal. And to understand the situation and the working software helps which is known as simulator. Simulator gives advantage of less risk, less cost as well as the working is easier to understand because visually we can see. Due to this we can study the working in different alternate conditions.

2. What is the need of CloudSim?

Ans:

CloudSim is one of the simulation software available. CloudSim simulates the working of the clouds. Clouds have revolutionised the computing it can fulfil nearly any type of requirement of the user. With that they don't really need to invest for the costly hardware requirements and their maintenance. They only need to pay for the pay-per-use basis. CloudSim allows simulating the working for the Cloud service provider to simulate the working of their system as well as to design their system accordingly. This provides great helps to the cloud developers as they can also innovate new thing and check if it works well and use it instead of the traditional one.

3. Why CloudSim is implemented in Java?

Ans:

Java is much more than just a high-level programming language that is widely known and most demanded object-oriented programming language for enterprise-grade application development. With Java, developers have everything on their fingertips that they need to build web applications and software solutions. All these advantages of Java are the reason why CloudSim is implemented in Java. Further, classes in Java add to the advantages and applications of Java.

4. **What are the different versions of it? How are they different?**

Ans:

CloudSim is available in different versions. The different versions of CloudSim along with their features are mentioned below in chronological order:

- CloudSim 1.0 beta
- CloudSim 2.0: CloudSim 2.0 does not rely on SimJava to process
  information. A new simulation core was introduced which increased scalability and performance. Schedulers were improved to increase accuracy. New features like power-aware simulation, network simulation were introduced.
- CloudSim 2.1: The project was migrated to Apache Maven, which simplified java project management. The directory structure was also changed.
- CloudSim 3.0: A new VM scheduler was introduced which allowed unbounded number of VMS to be deployed in a single VM. An internal datacenter model was added, which supports switching hosts in arbitrary network topologies. Support for external workload and user-defined end of simulation was added.
- CloudSim 3.0.1: Some of the methods of the scheduler were protected. Default behaviour and inconsistencies in the cloudlet were changed for smooth functioning.
- CloudSim 4.0: Support for container virtualization was added. Dependency on Apache Math was increased. The minimal time between events was made configurable.
- CloudSim 5.0: Cloudsim 5.0 toolkit combines various releases including containers, VM extensions with performance monitoring features and
  modelling of Web applications on multi-clouds. This will also work with other simulation models such as Software-defined Networks (SDN) / Service Function Chaining (SFC).

5. **Install, run and prepare a summary of all the functions used in 8 examples of the CloudSim simulator (version 4.0/5.0).**

Ans: The examples in the CloudSim simulator v4.0 makes use of following inbuilt functions in the 8 examples:

- Calendar.getInstance(): The getInstance() method in Calendar class is used to get a calendar using the current time zone and locale of the system.
- CloudSim.init(): It takes 3 parameters - number of users, calendar instance and trace flag. It initializes the simulation, provided with the current time, number of users and trace flag.
- createDatacenter(): It creates a datacenter and initializes the various data center characteristics along with the host list.
- createBroker(): It creates a Datacenter broker and initializes the entity object from DatacenterBroker class.
- Vm(): It is a constructor function for creating a new virtual machine.
- UtilizationModelFull(): It is a constructor function according to which a Cloudlet always utilizes all available CPU capacity.
- Cloudlet(): It is a constructor function for class Cloudlet, and takes a number of inputs for creation of a cloudlet.
- DatacenterCharacteristics(): It allocates a new DatacenterCharacteristics object, which represents static properties of a resource.
- bindCloudletToVm(): It takes the id of cloudlet and vm and specifies that a given cloudlet must run in a specific virtual machine.
- CloudSim.startSimulation(): It sends the call to start the simulation.
- CloudSim.stopSimulation(): It sends the call to stop simulation.
- CloudSim.pauseSimulation(): This method is called if one wants to pause the simulation.
- CloudSim.resumeSimulation(): This method is called if one wants to resume the simulation that has previously been paused.

6. **Write a program to create multiple data centres with multiple VMs to schedule N number of jobs using simple best effort Scheduler. Assign a workload to the machine having the lowest load.**

Ans:

```
package org.cloudbus.cloudsim.allocationpolicies;

import org.cloudbus.cloudsim.hosts.Host;
import org.cloudbus.cloudsim.vms.Vm;

import java.util.Comparator;
import java.util.Optional;
import java.util.stream.Stream;

public class VmAllocationPolicyBestFit extends VmAllocationPolicyAbstract {
        @Override
    protected Optional<Host> defaultFindHostForVm(final Vm vm) {
```

```
        final Comparator<Host> activeComparator =
Comparator.comparing(Host::isActive).reversed();
        final Comparator<Host> comparator =
activeComparator.thenComparingLong(Host::getFreePesNumber);

        final Stream<Host> stream = isParallelHostSearchEnabled() ?
getHostList().stream().parallel() : getHostList().stream();
        return stream
                .filter(host -> host.isSuitableForVm(vm))
                .min(comparator);
    }

}
```

7. **What are schedulers used in CloudSim?**

Ans:

There are two types of schedulers in CloudSim. They are broadly classified into:

A. **VmScheduler**

- **VmSchedulerTimeShared**: This class implements the VM scheduling policy that allocates one or more processing elements to a single Virtual machine and allows the sharing of processing elements by multiple virtual machines with a specified time slice. Here, the VM allocation will fail if the number of processing elements requested is not available. Eg. if the VM
  request for quad-core processor whereas the allocated host has an only dual-core the allocation will fail.
- **VmSchedulerSpaceShared**: This class implements the VM scheduling policy that allocates one or more processing elements to a single virtual machine, but this policy implementation does not support sharing of processing elements (i.e.) all the requested resources will be used by the allocated VM till the time the VM is not destroyed.
- **VmSchedulerTimeSharedOverSubscription**: This is an extended
  implementation of VMSchedulerTimeShared VM scheduling policy, which allows over-subscription of processing elements by the virtual machine(s) (i.e.) the scheduler still allows the

allocation of VMs that require more CPU capacity that is available. And this oversubscription results in performance degradation.

B. **CloudletScheduler**

- **CloudletSchedulerSpaceShared**: This class implements a policy of scheduling for Virtual machine to execute cloudlet(s) in space shared environment (i.e.) only one cloudlet will be executed on a virtual machine at a time. It means cloudlets share the same queue and requests are processed one at a time per computing core. Space-sharing is similar to batch processing.
- **CloudletSchedulerTimeShared**: This class implements a policy of cloudlet scheduling for Virtual machines to execute cloudlets in a time-shared environment (i.e.) more than one cloudlet will be submitted to the virtual machine and each will get its specified share of time. It means several requests (cloudlets) are processed at once but they must share the computing power of that virtual machine(by simulating context switching), so they will affect each other's processing time. It basically influences the completion time of a cloudlet in CloudSim. Time-sharing is probably referring to the concept of sharing executing power (such as CPU, logical processor, GPU) and is commonly known as the round-robin scheduling.
- **CloudletSchedulerDynamicWorkload**: This implements a special policy of scheduling for virtual machine assuming that there is just one cloudlet which is working as an online service with a different requirement of workload as per the need of peak/offpeak user load at a specified period of time.

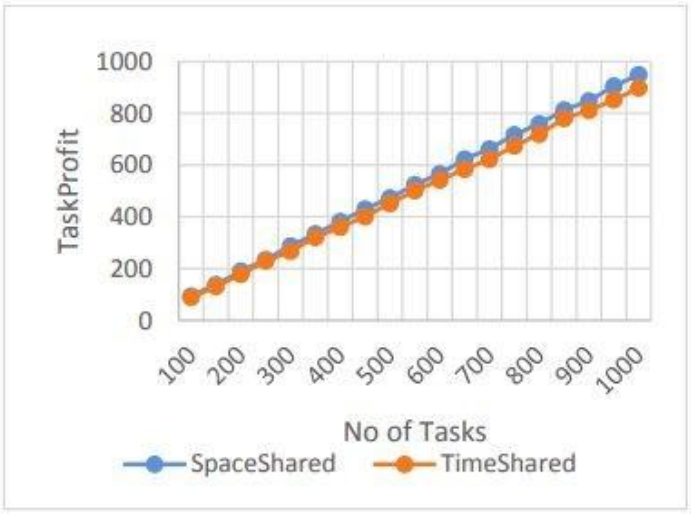
8. **What are the limitations of each version?**

Ans:

Scenario:

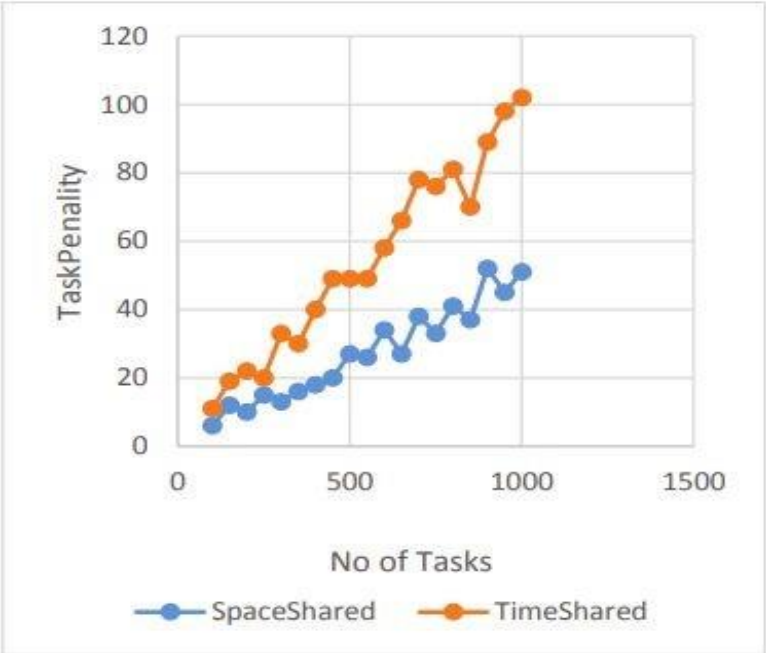
The performance measurement has been gathered by using the CloudSim framework. The goal of this study is to show the ability of the space-shared approach to meet deadlines in conditions where the Time Shared could not able to meet deadlines.
Speed of Processing Element: 1000
MIPS Bandwidth: 10 Gbps
Storage: 1 TB.
RAM: 2048 MB.
No of CloudIets: 1000

Task Profit:
The space-shared approach is more efficient to finish tasks before the deadline arrives than Time Shared scheduling algorithm. We compare the number of tasks completed by each approach for the given set of cloudlets



On the x-axis it shows the number of cloudlets and y-axis indicates the number of tasks completed successfully before the deadline reaches. By examining the figure we can say that the Time-shared algorithm is inefficient to meet deadline before it comes.
Task Penalty:

Here, we found that the task penalty of cloudlets i.e. the number of cloudlets misses their deadline.

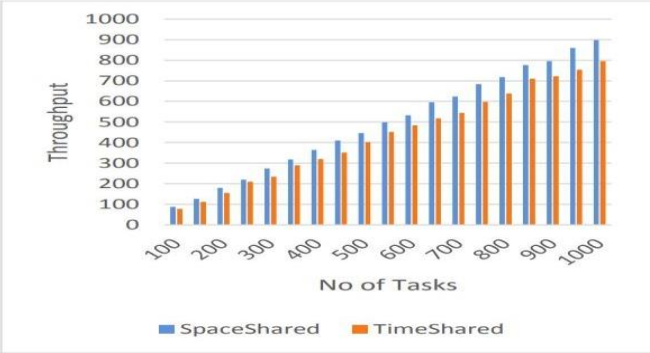


The x-axis represents the number of cloudlets and y-axis defines missed deadlines. Once again, it is noticed that space shared outperforms other

approaches. It missed a negligible number of deadlines compared to Time-shared policies.

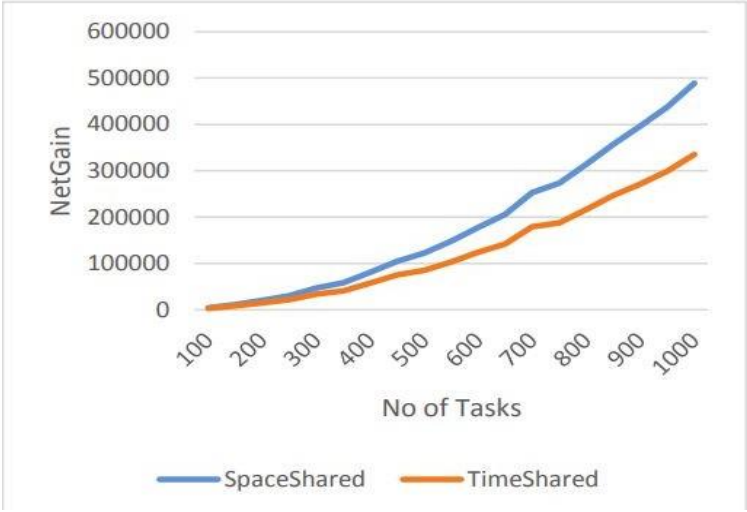
Throughput:

Another parameter which shows the space-shared approach is best among both policies. Throughput of tasks is the difference between earlier defined parameters.



Net Gain:

Space-shared is capable of processing soft deadline cloudlets. Such cloudlets has penalties when they misses their deadline. To calculate the cost Net gain parameter is introduced which depends on the lateness of the cloudlets. So it is important to analyze the cost in order to measure the performance policies.



In figure, x-axis shows number of cloudlets and y-axis shows the net gain of the cloudlets. Time-shared polices shows very less profit but for space-shared approach net gain increases as the number of cloudlets increases.

9. What is the future scope of CloudSim?

Ans:

The future plans of CloudSim are to incorporate new pricing and provisioning policies, in order to offer built-in support to simulate the currently available Public clouds. Other future directions of includes:

(i) workload models
(ii) models for database services such as blob, SQL etc.
(iii) QoS monitoring capability at VM and Cloud level
(iv) pricing models for public clouds to support economy-oriented resource provisioning studies.

Further, recent studies have revealed that data centres consume an unprecedented amount of electrical power; hence, they incur massive capital expenditure for day-to-day operation and management. For example, a Google data centre consumes power equivalent to that used by a city like San Francisco.

The socio-economic factors and environmental conditions of the geographical region where a data centre is hosted directly influence the total power bills
incurred. For instance, a data centre hosted in a location where power cost is low and has less hostile weather conditions would incur comparatively lesser expenditure in power bills. To achieve simulation of the aforementioned Cloud computing environments, much of our future work will investigate new models and techniques for allocation of services to applications depending on energy efficiency and expenditure of service providers.

10. How Dynamic Voltage and Frequency Scaling(DVFS) module works in CloudSim? Run its examples and make the summary of the same.

Ans:

Dynamic Voltage and Frequency Scaling (DVFS) is commonly used to reduce energy consumption in data centres. Nowadays, data centres consume about 2% of worldwide energy production. Also, the proliferation of urban data centres is responsible for the increasing

power demand of up to 70% in metropolitan areas, where the power density is becoming too high for the power grid. Besides the economic impact, the heat and the carbon footprint generated by cooling systems are dramatically increasing and they are expected to overtake the emissions of the airline industry by 2020. Therefore, the implementation of DVFS-aware consolidation policies has the potential to optimize the energy consumption of highly variable workloads in Cloud data centres.The code snippet for Dynamic Voltage and Frequency Scaling module is given below.

```
package org.cloudbus.cloudsim.examples.power.random;

import java.io.IOException;

public class Dvfs {
      public static void main(String[] args) throws IOException {
            boolean enableOutput = true;
            boolean outputToFile = false;
            String inputFolder = "";
            String outputFolder = "";
            String workload = "random";
            String vmAllocationPolicy = "dvfs";
            String vmSelectionPolicy = "";
            String parameter = "";

            new RandomRunner(
                        enableOutput,
                        outputToFile,
                        inputFolder,
                        outputFolder,
                        workload,
                        vmAllocationPolicy,
                        vmSelectionPolicy,
                        parameter);
      }

}
```

On analysing the code snippet, it can be said that the arguments of the functionRandomRunner are enableOutput, outputToFile, inputFolder, outputFolder, workload, vmAllocationPolicy, vmSelectionPolicy, parameter.

Summarising the output of above code:

Number of hosts:
50 Number of
VMs: 50
Total simulation time:
86400.00 sec Energy
consumption: 52.98 kWh
Number of VM migrations: 0

SLA: 0.00000%
SLA perf degradation due to migration:
0.00% SLA time per active host: 0.00%
Overall SLA violation:
0.00% Average SLA
violation: 0.00% Number
of host shutdowns: 29
Mean time before a host shutdown:
300.10 sec StDev time before a host
shutdown: 0.00 sec Mean time before a
VM migration: NaN sec StDev time
before a VM migration: NaN sec