Webology (ISSN: 1735-188X)
Volume 18, Number 3, 2021

Optimizing Resource Allocation In Cloud Computing

R. MANIMEGALAI[1] , Dr. U. DURAI[2]

1. Research Scholar, Dept. Of Computer Science, Marudupandiyar College of Arts and Science, Thanjavur.

2. Research Advisor, Dept. Of Computer Science, Marudupandiyar College of Arts and Science, Thanjavur.

(Affiliated to Bharathidasan University, Tiruchirappalli)

Abstract

Before diving into the specifics of the suggested model's architecture, it's crucial to identify any design restrictions that must be overcome. In this paradigm, there are n data centres, each with its own set of resources. Only one data centre should be chosen to handle each job throughout the bargaining process. It's important to find a VM that can handle the workload without breaking the bank in terms of CPU speed, memory size, storage space, or any of the other criteria that may come into play. The tasks are non-pre-emptive, meaning that once they are allocated to a VM, it is expected that they will run entirely within that VM. Because it is assumed that each VM has a single core, conflict between running processes is avoided and the VM may focus on a single job at a time. All of the processes being run on a given resource shouldn't need more processing power than the resource can provide. All of the jobs being run on a given resource shouldn't use up more memory than is allowed for that resource. When there is no longer any work for the hosts to do, some of them will be turned off in order to conserve energy, therefore the total number of hosts and VMs is not fixed. The maximum number of VMs that may be hosted on a given host at once is limited by its processing power, which is itself determined by the number of available CPU cores and the assumption that each core can only ever host a single virtual machine. The scheduling method treats each job as if it were completely autonomous and believes there would be no coordination between them.

Keywords: Resource Allocation, Optimization, Cloud Computing.

Introduction

Four primary actors—the Consumer Agent, Broker Agent, Manager Module, and Provider Modules—form the proposed model's architecture. Each of these primary components represents a different tier in the cloud computing stack and is in charge of a distinct task related to the management of available resources. In the SaaS layer, Consumer Agents are responsible for user interaction. The PaaS layer's Manager Module is responsible for the model's aforementioned negotiating and management functions for both end users and the underlying provider

infrastructure. In the IaaS layer, the Provider Module controls both the cloud's virtual and physical hardware. Figure 3.2 depicts the overall design of the suggested paradigm. Detailed descriptions of each module in this design, along with their respective roles, follow.

The Consumer Agent is the agent on the consumer's side that serves as the interface through which the consumer submits the tasks. When customers make requests, they specify parameters including how much money they can spend, how fast their computers need to be, how much space they need to store data, and when they need it by. The Buyer's Agent then notifies the Broker's Agent of the pending duties and any prerequisites.

To facilitate communication between the Consumer Agent and the Manager Module, there is the Broker Agent. The Consumer Agent provides tasks to the Broker, and the Broker then converts those tasks into proposals before sending them on to the Manager Module. The Broker Agent takes in consumer-submitted tasks and uses those requests to determine what level of Quality of Service (QoS) is required. The Consumer Agent, who cares about the progress of each work, receives the outcomes of the Agent's comments and acknowledgements.

A crucial part of the suggested design is the Manager Module. It receives data from the DC Manager Agent in the Cloud Provider module and distributes those resources accordingly. To achieve the goals of the proposed paradigm, interaction between the three submodules is required. The Manager's ancillary components include a Service Level Agreement (SLA) Monitor Agent, a Negotiation Agent, and a Task Dispatcher Agent. The Manager takes bids from the DC Manager Agent in each data center and negotiates service level agreements with the Broker Agent through the SLA Negotiator Agent. At this point, the module's focus shifts to the actual negotiating process, which is grounded on the providers' and customers' respective predetermined goals. It also oversees the process of carrying out the consumer's job at the cloud provider's data centers. From the DC Manager Agents, it learns how each work is progressing. More information about the Manager's specialized modules is provided below.

The Service Level Agreement Negotiator (SLA Negotiator) Agent is in charge of taking all the information it gathers from the Broker Agent and the DC Manager Agents and negotiating the terms of service for all of the data centers. It kicks off negotiations by picking the best possible offer for each proposal given the available time and money. It determines which data center is capable of performing an operation according to the criteria set forth. The mapping process between end-users and data centers is handled by using Parallel PSO negotiation. After that, its draughts the contract and has it signed by both the client and the service provider.

After the SLA Negotiator Agent determines the data center ID for each work, it is the Task Dispatcher Agent's job to distribute those jobs to the appropriate facility.

The Service Level Agreement (SLA) Monitor Agent's job is to gather data on the SLA and keep an eye on it in case of a breach. It keeps tabs on the status of all submitted tasks and flags any potential service level agreement (SLA) violations. In this concept, missing the deadline is considered a violation.

Module of the cloud service provider that stands in for the infrastructure as a service layer of the cloud, which comprises the virtual machines and the hosts. In order to test our model with a high number of data centers, we suppose that it is composed of several dispersed data centers—here, 10. DC Manager Agent, Host Monitor Agent, Task Scheduler, Load Balancer Agent, and VM Manager Agent are the five parts that make up a data center, which is a centralized repository for both physical and virtual resources.

The Data Center Manager Agent communicates between the DC's hardware and software and the Manager Module. Each data center has its own "local manager." Each data center's DC Manager Agent will communicate with the Manager Agent to provide offers that meet the required QoS of the consumer. This agent has several purposes; for example, it relays information about the cloud's current state from the Host Monitor Agent to the Manager Agent at regular intervals and it gets a list of activities to be carried out from the Manager Module.

HM Agent, or the Host Monitor: It keeps track of how busy each host is and alerts virtual machine migration if the workload distribution isn't even. Data center hosts and virtual machines (VMs) are tracked, along with metrics like resource use and energy costs. The VM Scheduler relies on the data provided by this module to properly assign hosts to virtual machines.

Related Work

Algorithms for optimizing a problem's solution are a sophisticated tool for reaching the best possible outcome. An algorithm is a mathematical procedure for producing results from inputs subject to certain conditions (Blum and Roli, 2003). An objective function is a metric used in optimization methodologies to determine how well goals are satisfied within certain boundaries. Algorithms for optimizing a problem set out to either maximize or minimize an objective function (Yang, 2010). The programme employs the necessary mathematical methods to explore the space for optimum solutions after an optimization issue has been specified. It's possible to choose between finding the best answer, which can be time-consuming, and a near-optimal solution, which can be found in less time.

There is no universally applicable, efficient method for usage in distributed systems due to the wide variety of parameters that must be considered. As an alternative, the optimization algorithm is designed to achieve a set of predetermined goals. Optimization methods may be broken down into a few broad groups according to their defining characteristics. Based on the algorithm's nature and the solution-finding approach, they are often classified as deterministic or stochastic (Yang, 2010). Stochastic algorithms are based on unpredictability in the path and the variables, whereas deterministic algorithms follow a predetermined path and set of variables. In Genetic algorithms, for instance, the population of solutions varies from run to run since the search for the best solution is dependent on chance. Stochastic algorithms, on the other hand, provide similar outcomes despite the fact that the same pathways are used for each population. Methods that mix deterministic and stochastic algorithms aim to avoid the drawbacks of both types of algorithms while reaping the benefits of both.

As a class, stochastic algorithms can be further subdivided into heuristic and meta-heuristic subclasses. It is not guaranteed that heuristic approaches will discover an optimum solution (Madni et al., 2016), but they do find a good optimal solution with low computing cost. Meta-heuristic algorithms combine randomization and local search to provide better results than standard heuristics (Yang, 2010). The line between heuristic and meta-heuristic algorithms is not clearly drawn.

Methodology

The primary aspects of the proposed model's implementation and the primary configuration of the simulation components that will be utilized to create the necessary modules will be covered in this section. It also specifies how the resource allocation stages will be assessed and what criteria will be employed. However, the assessment and unique setup of each algorithm will be covered in later sections.

The benefits of simulation are discussed first. Hardware, software, and the underlying network are all present in cloud computing settings. Additionally, customers have varying and perhaps conflicting QoS expectations. Designing and evaluating the model's performance metrics under varying setups and settings might be challenging in a real cloud environment like Amazon EC2 or Microsoft Azure. This is necessary because the suggested approach necessitates the testing of many data center architectures with varying requirements. In addition, the evaluation process is constrained when real-world settings are used due to infrastructure constraints, and re-evaluating trials becomes very challenging in terms of measurement. This makes retesting incredibly challenging and necessitates extensive modifications to the testing environment and infrastructure. However, reconfiguring benchmarking parameters to alter the applications and workloads in order to conduct more tests is time-consuming, money-consuming, and tedious. Since real-world environments are sometimes too complex for benchmark studies and evaluations, developers and researchers often resort to using simulators instead (Sakellari and Loukas, 2013). When it comes to testing out new infrastructure setups, developers may save time and effort by adopting tools and environments designed for simulation. The usage of simulators can also increase the models' adaptability since it enables the developer to build a structure that is user- and requirement-friendly. In light of these considerations, the model will be tested with a simulator.

CloudSim, GreenCloud, and MDCSim are only a few examples of the simulation tools available for cloud systems (Malhotra and Jain, 2013). In this study, a CloudSim platform will be used to implement and assess the three stages of our model. This is due to a number of factors. To begin, CloudSim is a free and publicly accessible Java-based simulator. Additionally, it has various submodules that mimic the primary elements and levels of cloud environments. This allows the simulation to be easily tailored to specific designs by adding or removing components as needed. In this study, we used CloudSim 3.0.3, which was the most recent version as of the middle of 2017. For the purpose of modelling and assessing cloud computing infrastructures and services, Calheiros et al. (2011) developed CloudSim, a generic and extensible simulation model. It allows for event queuing and processing, the introduction of new cloud system elements including hosts,

data centers, brokers, and virtual machines, as well as inter-component communication and timekeeping management (Calheiros et al., 2011).

What follows is a list of CloudSim's fundamental classes:

The data center represents the primary hardware architecture of the cloud and is under the providers' control.

The term "broker" stands in for a broker module that handles interactions between clients and service providers.

"Host" means it simulates a server in a data center.

An implementation of a virtual machine (VM) that operates on the host and carries out the duties.

SaaS layer modeler Cloudlet: models' cloud-based apps and services (in this model it is denoted by the term task).

VM Allocation: a policy provided in the data center characteristics that defines the process of assigning VMs to hosts.

VmScheduler is a method through which the host's CPU resources are divided up among the several virtual machines. It is a programme that is installed on every physical host in the data centre and is responsible for allocating processing power to virtual machines.

Cloudlet Scheduler is the virtual machine's (VM) unique strategy for allocating CPU time to cloudlets.

Several changes were made to the CloudSim simulator classes in order to implement the model and make it applicable to the given problem. To this end, updates were made to preexisting classes and new ones were created to aid in areas such as load computation, host clustering, and negotiation.

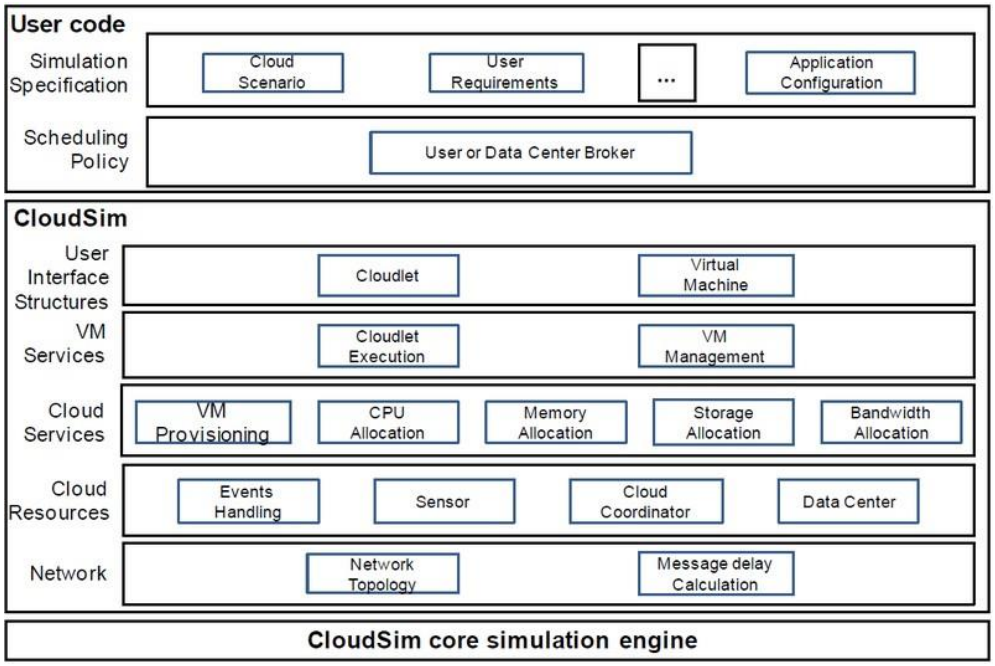

Figure 1: CloudSim Architecture

After the simulation was done, particular parameters were analyzed in order to evaluate the effectiveness of the suggested model. The average waiting time, average execution time, throughput, resource usage, cost, profit, and power consumption were the primary factors utilized as indications of the model's aims (see Section 1.3). Here are the specifics of each variable:

Commonly abbreviated as "AWT," this metric measures how long it takes for one job to complete on average, based on the number of other tasks that have to finish in the meanwhile. By subtracting the time, a task is submitted to the system from the time execution of all tasks begins, as stated in Equation 3.1, we can get the average waiting time. Negotiation time, task mapping time, data transfer time, and VM migration time are all included in the waiting time.

$$\mathbf{A}\text{WT} = \sum^{m} \ (ST(\boldsymbol{ii}) - SubT(\boldsymbol{ii}))/\mathbf{m} \quad (3.1)$$

where: t=**00**

AWT is the average waiting time of all tasks in seconds m denotes the number of tasks running in the system per unit of time ST (i) denotes the start time of execution of the task i in seconds SubT (i) denotes the time for submission of the task i in seconds.

A task's average completion time is the sum of the time it took to carry out all its individual steps. Time is tracked by subtracting the moment a job was handed off from the moment it was completed.

$$\mathbf{AA}\text{CT} = \sum^{m} \ (Ext(\boldsymbol{ii}) - Subt(\boldsymbol{ii}))/\mathbf{m} \quad (3.2)$$

$$t=\mathbf{00}$$

where:

Standardized Average Completion Time (ACT) is the sum of all task times in seconds.

When we say "m," we're referring to the number of tasks in use at any one moment. The last time that task I was executed is denoted by the expression Ext I where I is a number of seconds. Time in seconds at which task I was submitted is denoted by subt I

The throughput (TH) is a metric for gauging how well a system performs as a whole. Throughput measures how many jobs our model can do in a given amount of time.

Where TH = (C/T) FF0000 (3.3)

The system's throughput is denoted by the symbol TH. C is the sum of all actions taken. T is the number of seconds used in the simulation.

In terms of AVU, or average virtual machine usage: All jobs' use of the VM's CPU, memory, storage, and bandwidth is reflected here. Each virtual machine's CPU consumption is calculated by dividing the sum of its CPU, memory, storage, and bandwidth utilizations by 4. (Which is the number of factors including in computing VM utilization). As a percentage of the virtual machine's available CPUs, CPU utilization measures how much of the VM's CPUs are really being used. As illustrated in Equations 3.4, 3.5, 3.6, and 3.7, memory usage, storage utilization, and bandwidth utilization are calculated in the same way as CPU utilization.

CVM (i) = $\sum n \quad (UC(\boldsymbol{ii}) / AC(\boldsymbol{ii})) * \mathbf{FF0000}$ (3.4)

$\boldsymbol{ii}$=**00**

MVM (i) = $\sum n \quad (UM(\boldsymbol{ii}) / AM(\boldsymbol{ii})) * \mathbf{FF0000}$ (3.5)

$\boldsymbol{ii}$=**00**

SVM (i) = $\sum n \quad (US(\boldsymbol{ii}) / AS(\boldsymbol{ii})) * \mathbf{FF0000}$ (3.6)

$\boldsymbol{ii}$=**00**

BVM (i) = $\sum n \quad (UB(\boldsymbol{ii}) / AB(\boldsymbol{ii})) * \mathbf{FF0000}$ (3.7)

$\boldsymbol{ii}$=**00**

UVM (i) = (CVM (i) + MVM (i) + SVM (i) +BVM (i))/4 (3.8)

where:

UVM (i) is the average utilization of VM i CVM (i) is the CPU utilization of VM i

UC (i) is the used CPU for all tasks executed in VM i AC (i) is the total CPU of VM i

MVM is the memory utilization of VM i

UM (i) is the used memory for all tasks executed in VM i AM (i) is the total memory of VM i

SVM (i) is the storage utilization of VM i

US (i) is the used storage for all tasks executed in VM i AS (i) is the total storage of VM i

BVM (i) is the bandwidth utilization of VM i

UB (i) is the used bandwidth for all tasks executed in VM i AB (i) is the total bandwidth of VM i

n denotes the number of VMs

Average Resource Utilization (ARU): Host utilization in our model represents the total utilization of all VMs running on the host. The average resource utilization is computed by summing the host utilization of all available hosts in the data center (Equations 3.9 and 3.10).

$$HU(j) = (\sum_{ii=n} UVM(ii))/n \quad (3.9)$$

where:

HU is the average host utilization for host j

UVM (i) denotes the utilization of all VMs in the host j n denotes the number of VMs in host j

$$ARU = \sum_{jj=m} HU(jj)\ /\mathrm{m} \quad (3.10)$$

where:

ARU denotes the resource utilization

HU is the average host utilization for host j as shown in Equation 3.9 m denotes the number of hosts in the data center.

Conclusion

This thesis proposes a paradigm for resource allocation in cloud computing, with the goals of bettering service level agreement (SLA) negotiation, job scheduling, and virtual machine (VM) allocation. These modules were optimized using many forms of particle swarm optimization tailored to the specifics of each situation. When negotiating service level agreements (SLAs) between users and several distant data centers with varying capacities, parallel PSO is used to enhance the process. Through the application of the parallel PSO algorithm, negotiations may be automated to save time and effort while still producing a solution of acceptable quality. When compared to PSO and SPPSO, the suggested algorithm for SLA negotiation cuts waiting time by around 30% and 20%, respectively. If you compare it to the PSO algorithm, you'll see a throughput boost of around 20%. When compared to PSO, the percentage of SLA violations is reduced by around a quarter.

References

1. Abdel-Kader, R. F. (2010, February). Genetically improved PSO algorithm for efficient data clustering. In Machine Learning and Computing (ICMLC), 2010 Second International Conference on (pp. 71-75). IEEE.

2. Abdi, S., Motamedi, S. A., & Sharifian, S. (2014, January). Task scheduling using Modified PSO Algorithm in cloud computing environment. In International conference on machine learning, electrical and mechanical engineering (pp. 8-9).
3. Rajkumar, V., and V. Maniraj. "HYBRID TRAFFIC ALLOCATION USING APPLICATION-AWARE ALLOCATION OF RESOURCES IN CELLULAR NETWORKS." Shodhsamhita (ISSN: 2277-7067) 12.8 (2021).
4. Abdullah, R., & Talib, A. M. (2012, October). Towards integrating information of service level agreement and resources as a services (RaaS) for cloud computing environment. In Open Systems (ICOS), 2012 IEEE Conference on (pp. 1-5). IEEE.
5. Adamuthe, A. C., Pandharpatte, R. M., & Thampi, G. T. (2013, November). Multiobjective virtual machine placement in cloud environment. In Cloud & Ubiquitous Computing & Emerging Technologies (CUBE), 2013 International Conference on (pp. 8-13). IEEE.
6. Rajkumar, V., and V. Maniraj. "RL-ROUTING: A DEEP REINFORCEMENT LEARNING SDN ROUTING ALGORITHM." JOURNAL OF EDUCATION: RABINDRABHARATI UNIVERSITY (ISSN: 0972-7175) 24.12 (2021).
7. Adrian, B., & Heryawan, L. (2015, November). Analysis of K-means algorithm for VMallocation in cloud computing. In Data and Software Engineering (ICoDSE), 2015International Conference on (pp. 48-53). IEEE.
8. Ahmad, R. W., Gani, A., Hamid, S. H. A., Shiraz, M., Yousafzai, A., & Xia, F. (2015).A survey on virtual machine migration and server consolidation frameworks for cloud data centers. Journal of Network and Computer Applications, 52, 11-25.
9. Rajkumar, V., and V. Maniraj. "PRIVACY-PRESERVING COMPUTATION WITH AN EXTENDED FRAMEWORK AND FLEXIBLE ACCESS CONTROL." 湖南大学学报 (自然科学版) 48.10 (2021).
10. Aissi,H., Bazgan, C., & Vanderpooten, D. (2005). Complexity of the min–max and min–max regret assignment problems. Operations research letters, 33(6), 634-640.
11. Ahmadyfard, A., & Modares, H. (2008, August). Combining PSO and k-means to enhance data clustering. In Telecommunications, 2008. IST 2008. International Symposium on (pp. 688-691). IEEE.
12. Rajkumar, V., and V. Maniraj. "Software-Defined Networking's Study with Impact on Network Security." Design Engineering (ISSN: 0011-9342) 8 (2021).
13. Abulkhair, M. F., Alkayal, E. S., & Jennings, N. R. (2017, September). Automated Negotiation using Parallel Particle Swarm Optimization for Cloud Computing Applications. In Computer and Applications (ICCA), 2017 International Conference on (pp. 26-35). IEEE.

14. Al-Ayyoub, M., Jararweh, Y., Daraghmeh, M., & Althebyan, Q. (2015). Multi-agent based dynamic resource provisioning and monitoring for cloud computing systems infrastructure. Cluster Computing, 18(2), 919-932.
15. Alkayal, E. S., Jennings, N. R., & Abulkhair, M. F. (2016, November). Efficient Task Scheduling Multi-Objective Particle Swarm Optimization in Cloud Computing. In Local Computer Networks Workshops (LCN Workshops), 2016 IEEE 41st Conference on (pp. 17-24). IEEE.
16. Rajkumar, V., and V. Maniraj. "HCCLBA: Hop-By-Hop Consumption Conscious Load Balancing Architecture Using Programmable Data Planes." Webology (ISSN: 1735-188X) 18.2 (2021).
17. Alkhashai, H. M., & Omara, F. A. (2016). An Enhanced Task Scheduling Algorithm on Cloud Computing Environment. International Journal of Grid and Distributed Computing, 9(7), 91-100.