International Journal of Computer Trends and Technology
ISSN: 2231-2803 / https://doi.org/10.14445/22312803/IJCTT-V67I3P129

Volume 67 Issue 3,151-155, March 2019
© 2019 Seventh Sense Research Group®

Review Article

Analyzing Resource Allocation Strategies with Elasticity in Multi-Tenant Cloud Environment

Amit Kumar Chaturvedi[1], Praveen Sengar[2], Kalpana Sharma[3]

[1,3]Assistant Prof., MCA Deptt, Govt. Engineering College, Ajmer, India
[2]Ph.D. Scholar, CS Deptt., Bhagwant University, Ajmer

Abstract - *Sharing the resources for maximizing the uses and benefits is the basic idea behind cloud computing. Providers share their resources like networks, servers, storage, applications, and services to the clients in a ubiquitous, convenient, and on-demand way. Cloud is a multi-tenant environment that supports a customizable and easily configurable service model. SLA (Service Level Agreements) play a vital role in binding clients and providing negotiable and agreed rules and regulations. If anyone violates these rules and regulations, they will also be penalized. The concept of multi-tenancy increases the use of cloud resources to an extent, but it also increases the challenges of resource allocation strategies. Various researchers propose resource allocation strategies by taking different factors; one-factor elasticity is common in resource allocation. This paper will present a study of the resource allocation strategies with elasticity in a multi-tenant cloud environment. In this process, SLA is always at the center to do the whole process of providing the elasticity in resource allocation strategies.*

Keywords - *multi-tenant, elasticity, resource allocation, SLA, QoS Virtual Machine.*

I. INTRODUCTION

According to the National Institute of Standards and Technology (NIST), Cloud Computing is a model that enables providers to share their computing resources (e.g., networks, servers, storage, applications, and services) and users to access them in a ubiquitous, convenient and on-demand way with a minimal management effort [11]. Cloud computing offers a service model with IaaS, PaaS, and SaaS layers. All the service layers are accessible to multiple tenants through virtual machines. Using configurable process models, such a multi-tenant environment allows a cloud business process provider to deliver a customizable process that different tenants can configure according to their specific needs [13]. Cloud computing infrastructures allow creating a variable number of virtual machine instances depending on the application demands. The SaaS service layer provides flexibility in resource allocation, i.e., scalability to scale up or down application resources and pay for only what is used by the user. Hence, the resources are used cost-effectively with scalability in resource allocation.

Multi-tenancy means multiple tenants reside on the same server simultaneously and share the resources available at different service layers, i.e., IaaS, PaaS, or SaaS, as cloud computing provides a configurable process model that fulfills the isolated demand of multiple tenants and configures the resources accordingly with a minimum number of resources. From the provider's point of view at IaaS, it is essential to measure the total resource requirement (like processors, memory, OS configuration, applications, etc.). Providing secure and isolated access to the resources and maintaining the confidentiality of data is the prime focus of the multi-tenant environment. Elasticity is another key factor in cloud computing, but there is not a standard metric or procedure to quantify it, and it is rarely used. In this paper, we will propose an elasticity metric that will be general, flexible, simple, and easy to measure. This elasticity metric allows providers and users to analyze service elasticity enablers.

II. SERVICE LEVEL AGREEMENT

A service-level agreement (SLA) is a contract between a service provider and its customer on what services the provider will furnish. Maintaining a negotiated SLA (throughput) is to minimize the number of computational resources involved. The underlying resource management policy is based on a business model: each SLA violation leads to an associated penalization, whereas scaling up the computational resources involved has an associated cost. Therefore, the overall goal of the SLA is to maximize the utilization of the resources efficiently so that the client and/or service provider get the maximum profit with minimum incurred cost of these resources by having and maintaining them on their site. Hence, getting maximum revenue by investing less cost is also the goal of the SLA from the service provider's point of view.

III. QOS PROVISIONING COMPONENT

The Quality of Service (QoS) is a generic term collectively used to assess the usefulness of any system from the user's perspective. In computer networks, QoS involves adding mechanisms to control the network activity, such as transmission and error rates, to assure certain service parameters. The main goal of QoS provisioning is to achieve more deterministic network behavior so that information carried out by the network can be better delivered, and network resources are better utilized.

In cloud computing, QoS provisioning means making decisions about the allocation and redistribution of resources based on monitoring buffers and resource requirement requests by the tenants. The exceeding data in the buffer leads to different trigger actions based on occupancy thresholds: (1) dropping data from the buffer, (2) allocating additional resources to consume this additional data and (3) reallocation of resources from other streams. When the traffic is bursty, the requirement or demand of resources cannot be predicted, and it is decided at runtime.

IV. RELATED WORK ON MULTI-TENANCY

An important requirement for SaaS applications is the support of multiple tenants. A tenant is a customer that uses or provides a SaaS application. To exploit economies of scale, i.e., allow SaaS providers to offer the one SaaS application instance to multiple tenants, a SaaS application must be multi-tenant aware [3,12,13]. Multi-tenant aware means that each tenant can interact with the application as if it were the only user of the application. In particular, a tenant cannot access or view another tenant's data [12]. In a SaaS model, the multi-tenancy support can be applied to four different software layers [14]: the application, the middleware, the virtual machine (VM), and the operating system layers. In a multi-tenancy-enabled service environment, user requests from different tenants are served concurrently by one or more hosted application instances based on the shared hardware and software infrastructure. There are generally two kinds of multi-tenancy patterns [11,15]: multiple instances and native multi-tenancy; the former supports each tenant with its dedicated application instance over shared hardware, operating system, or a middleware server in a hosting environment, whereas the latter can support all tenants by a single shared application instance over various hosting resources. The two kinds of multi-tenancy patterns scale quite differently regarding the number of tenants they can support. Multi-instance is adopted to support a small number to hundreds of tenants. At the same time, native multi-tenancy is used to support a much larger number of tenants, usually in the hundreds or even thousands. It is interesting to note that the isolation level among tenants decreases as the scalability level increases [15].

In [16], the authors present a framework to deal with the issues of native multi-tenancy for SaaS applications. In [15], the challenges of SaaS applications for application vendors and providers are discussed, taking into account the need for customization of SaaS applications [17]. The traditional technique for implementing multi-tenancy is to add a tenant ID column to each table and share tables among tenants [13,18]. Another work is presented in [19], where the M-store system is proposed and developed, which provides storage and indexing services for a multi-tenant database system. These techniques create an isolated environment for tenants by separating one tenant's context from another. This tenant context isolation can be implemented from the data layer to execute a specific view.

The basic idea of using JVM instrumentation is to start an agent listener when JVM initiates and then intercept values or fields annotated as tenant-aware (called isolation points) and load them according to the tenant's configuration. Two crucial problems: efficient VM image management and intelligent resource mapping. VM image management includes image preparation and local image management of physical resources. iVIC is a platform for academic researchers to dynamically create customized virtual computing environments to launch scientific computing, simulations, and analysis by leveraging VM technology. In iVIC, common resources (e.g., a set of workstations, PC servers, and small clusters) are organized into a number of physical resource pools. Each physical machine is treated as a VM Container (VMC) responsible for providing VM environments.

Each VMC exposes controlling and querying interfaces to upper resource level managers via SOAP interfaces. VM container interacting with SOAP interfaces is the mechanism for monitoring and measuring virtualized resources. The selected cloud computing platform in this research, Eucalyptus, offers SOAP interfaces and enables on-demand deployment of VM instances.

In [1], a cloud computing mechanism is proposed as a raw computational on-demand resource for a grid middleware. The authors use Eucalyptus to manage resources for a grid middleware implementation called DIET-Solve in this work. In [2], the authors describe three key components, effectively covering "measurement, "modeling" and "management" (VM3) of shared resource implications on individual virtual machine performance. Authors also propose a decomposition model that estimates the potential performance loss when a virtual machine is consolidated with other machines. Such a decomposition model consists of three major components: (a) virtualization overheads, (b) core contention overheads, and (c) shared cache contention overheads. A relevant commercial tool is Amazon Auto Scaling [3]. It is a web service to automatically launch or terminate Amazon EC2 instances based on user-defined triggers. It allows applications to scale up instances seamlessly and automatically when demand spikes and automatically shed unneeded instances when demand subsides. It uses proprietary commands to create Auto Scaling Groups, representing an application running on multiple instances. However, this mechanism is a closed proprietary mechanism that depends totally on the Amazon EC2 platform. Second, it is based only on resource utilization, but it does not consider the nature of the applications. In this work, resource utilization is compared against web applications' performance (throughput) to determine whether or not a virtual machine is saturated.

The problem of allocating cloud resources can be seen as a bin packing problem. Quite often, bin packing approximation algorithms are used for cloud resource provisioning. The TDS (Tenant Defined Storage) system aims to automatically allocate and reallocate the storage resources required by the different tenants. When a tenant user accesses the multi-tenant application, he first connects to a load balancer (1 in the figure) to select one of the available server instances in a nearby data center (2). The selected application server needs to connect to the corresponding storage pool where the tenant's data is stored (3). This storage pool should be close to the application server, preferable within the same data center. Both the application servers and storage pools can be provisioned on the fly in an elastic cloud environment. The management and provisioning of these resources is the main task of the elasticity manager (4). This component monitors and evaluates the current load on the provisioned application server instances to achieve high scalability. As the load increases, additional instances will be provisioned to avoid overload.

Similarly, the component also monitors the usage of the provisioned storage pools. Suppose the usage of a single storage pool reaches a certain threshold. An additional storage pool is provisioned, and some of the existing tenant data will be reallocated before the storage pool runs out of space. On the other hand, if the load on the application servers or the usage of the storage pools decreases significantly, one or more application servers and/or storage pools should be de-provisioned, requiring the reallocation of some of the tenants, to minimize the operating costs. Whenever tenants are reallocated, the elasticity manager also notifies the load balancer (5) to guarantee the correct routing of incoming requests.

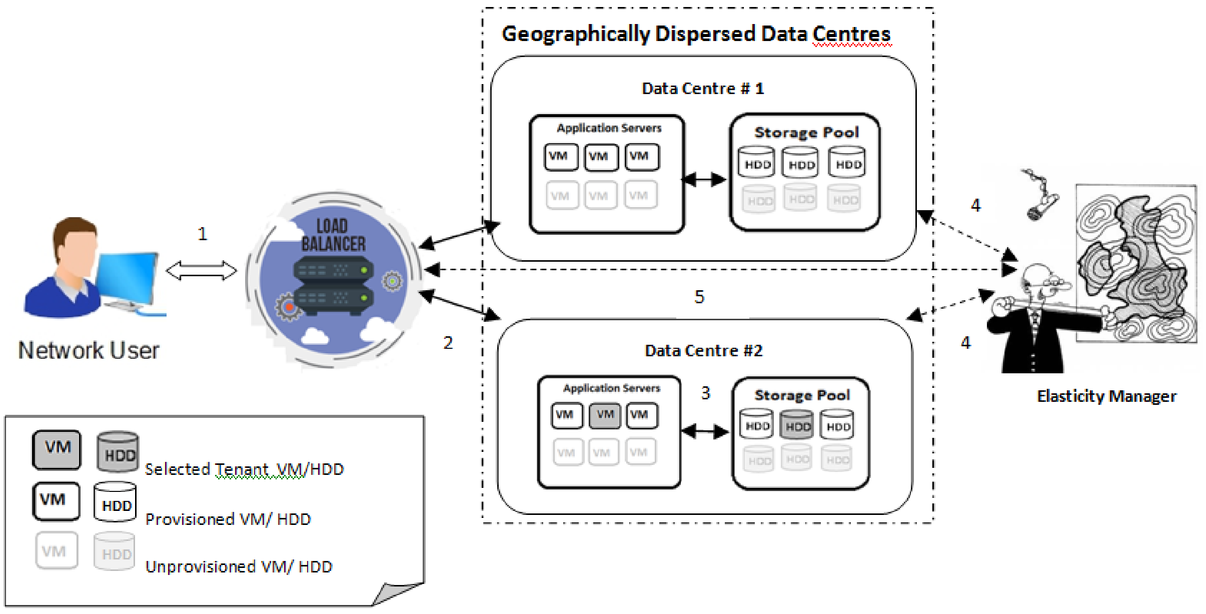

Fig. 1 General overview of the Tenant-Defined Storage system

Tenants are hierarchically organized using a tree structure, which we refer to as the tenant tree. There are several reasons to do so. First of all, multi-tenant applications are often used by a number of organizations the tenants. Large organizations, however, tend to consist of multiple independent divisions, introducing the need for subtenants or even sub-subtenants, and the tenant tree inherently supports this hierarchical structure. Secondly, when the application tenants are geographically distributed, it might be good practice to cluster them based on their location, and resources can be allocated from a resource pool close to the tenant. Tenants could also be clustered based on other characteristics, e.g., the selected SLA or other regulatory policies concerning the storage of sensitive data, and these characteristics could define the required type of (physical) hardware. In general, tenants can be clustered based on multiple characteristics, depending on the tenant's requirements, the possibilities of the application, and the infrastructure. The goal of the TDS system is to cluster related tenants together while minimizing migrations over time. In the resulting tenant tree, the most significant characteristics appear at the highest levels of the tree structure, as higher levels have a higher impact on the clustering of tenants.

Following are the important factors for the performance considerations of a clients job in a multi-tenant execution environment in cloud computing :

A. Execution isolation

In Cloud computing, multi-tenancy is a Cloud's design for sharing the computing resources that are in use among the different concurrent users. Isolation is the capability of perceiving one shared environment as dedicated and safe. Complete isolation among applications executed in PaaS environments can be achieved using multiple strategies19. Among them, the following approaches are identified:

B. Virtual Multi-tenancy

This approach relies on the isolation provided by resource virtualization (VMs) and hypervisors in the infrastructure management layer. Recently, these approaches have evolved to use Container technologies, although not yet widespread; Organic Multi-tenancy: This approach is based on isolation achieved at different PaaS component levels, such as application servers and DBMS…

C. Security at multiple levels

While Cloud computing offers a paradigm-shifting technological solution for computational resources and software, the concerns about privacy and confidentiality of data still are a major concern for adoption. It requires capabilities for underlying (data) security and resilience of resources delivered in the PaaS and IaaS multiple to enable users to uptake the Cloud-based delivery model.

D. Compliance

Public and Hybrid Cloud scenarios are characterized by a constant data flow that cannot be allocated to a particular place. This brings uncertainty regarding the various data protection legislation, which transcends national borders and complicates compliance with the data protection legislation worldwide. Enterprises or individuals using the PaaS to develop applications that handle confidential and private data need to safeguard their privacy. Therefore, from a legal point of view, providing mechanisms to enable data protection and privacy in Cloud environments should be basic functionality.

V. RELATED WORK ON ELASTICITY

Elasticity is the degree to which a system can adapt to workload changes by provisioning and de-provisioning resources autonomously. At each time, the available resources match the current demand as closely as possible. Again accuracy and time are considered. Being θ the average time to switch from a system configuration to another and μ is the average percentage of under-provisioned resources during the scaling process, the elasticity (el) is defined as:

$$\mathrm{El} = \frac{1}{\theta.\mu} \qquad (1)$$

elasticity is, in this case, a metric measured in time units−1 from 0 to 1 [14].

An elasticity metric is supposed to answer these two questions: how often does the system violate its requirements? And once these requirements are violated, how long does it take before the system recovers to a state in which requirements are met again? in this work, two metrics are defined to answer these questions, the number of slo (service level objectives) violations per time unit (from 0 to 1) and the meantime to quality repair or mttqr (in time units, from 0 to 1) [15].

There has been some work on elasticity measurement in cloud computing. In[14], elasticity is the degree to which workload changes are adapted by automatically provisioning and de-provisioning resources so that available resources match the current demand in time. In[16], elasticity for customers is the ability to quickly request, receive, and release as many resources as required. In[17], elasticity is measured by mapping a user's request to different resources. In[18], elasticity is defined dynamically to meet the varying workload of resources. [19], cost, quality, and available resources are treated as three elasticity dimensions for elastic cloud applications.
There are many approaches to predicting elasticity and deciding when and how resources are scaled in/out using heuristics and mathematical/analytical techniques. In[14], the elasticity metric captures key elasticity characteristics. In[11], execution platforms and configuration points are proposed to reflect the elasticity definition. In[1], elasticity benchmarking approaches are outlined for special workload design and implementation requirements. In[20], thread pools are used as a kind of elastic resource for JVM, and preliminary results of running a novel elasticity benchmark reveal the elastic behavior of thread pool resources.

VI. CONCLUSION

Multi-tenancy extended the reach and efficiency in resource utilization and offered cloud computing services to potential clients. Elasticity is the key factor in resource allocation under cloud computing. The main concern during the allocation and deallocation of resources is that there should be as few situations of overutilization and underutilization of resources to the client. This leads to improving the economies of scale for the clients and service providers. Saturation or overutilization occurs whenever resource utilization gets above the point of exhaustion. Resource underutilization occurs whenever virtual machines are not using some resources within a cloud computing infrastructure and an application is executed. Resource underutilization can be measured by the number of resources available by potential virtual machines and applications. There are multiple tools used to manage, measure, and monitor the dynamic resource allocation in cloud computing. These tools include the vsphere Web Client and the vsphere Client New Virtual Machine wizards and Virtual Machine Properties, editors.

The primary goal of the resource allocation system is to determine a feasible allocation of tenant data over the available resource pools. A feasible allocation should aim to minimize the number of instances (bins) to minimize the operational costs. Furthermore, the number of migrations overtime should also be minimized.

ACKNOWLEDGMENT

The authors are thankful to all directly or indirectly for preparing this research paper, especially Dr. K. K. Goyal, for his continuous support.

REFERENCES

[1] E. Caron, F. Desprez, D. Loureiro, Cloud computing resource management through a grid middleware: a case study with DIET and Eucalyptus, in IEEE International Conference on Cloud Computing, 2009, pp. 151–154.

[2] R. Iyer, R. Illikkal, O. Tickoo, L. Zhao, P. Apparao, D. Newell, VM3: measuring, modeling and managing VM shared resources, The International Journal of Computer and Telecommunications Networking (2009) 2873–2887.

[3] Amazon Web Services—auto-scaling. 2010. http://aws.amazon.com/autoscaling/.

[4] C. Devlin, SaaS capacity planning: transaction cost analysis revisited, MSDN Library, February 2008. http://msdn.microsoft.com/en-us/library/cc261632. aspx.

[5] C. Isci, J. Kephart, L. Zhang, E. Bouillet, D. Pendarakis, X. Meng, Efficient resource provisioning in compute clouds via VM multiplexing, in: Proceeding of the 7th International Conference on Autonomic Computing, June 2010.

[6] A.K. Mishra, J.L. Hellerstein, W. Cirne, C.R. Das, Towards characterizing cloud backend workloads: insights from Google compute clusters, SIGMETRICS Performance Evaluation Review 37 (4) (2010).

[7] Apache Software Foundation, Apache JMeter, 2010. http://jakarta.apache.org/ jmeter/usermanual/glossary.html.

[8] S. Wee, H. Liu, Client-side load balancer using the cloud, in Proceedings of the 2010 ACM Symposium on Applied Computing, March 2010.

[9] D. Dyachuk, R. Deters, A solution to resource underutilization for web services hosted in the cloud, in Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on the Move to Meaningful Internet Systems: Part I, November 2009.

[10] C. Matthews, Y. Coady, Virtualized recomposition: cloudy or clear? In: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, Washington, DC, USA, May 2009, pp. 38–43.

[11] Mell, P.M., et al.: The NIST definition of cloud computing 2011. (Technical report)

[12] Rosemann, M., van der Aalst, WMP: A configurable reference modeling language. Inf. Syst. (2007)

[13] Aalst, W.: Business Process Configuration in The Cloud: Support and Analyze Multi-Tenant Processes? In: ECOWS, IEEE (2011) 3{10)

[14] S. Herbst, N. Kounev, R. Reussner, Elasticity in cloud computing: What it is, and what it is not, in Proceedings of the 10th International Conference on Autonomic Computing, 2013, pp. 23–27.

[15] M. Becker, S. Lehrigy, S. Becker, Systematically deriving quality metrics for cloud computing systems, in Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering, 2015.

[16] L. Badger, T. Grance, R. Patt-Corner, and J. Voas, Draft Cloud Computing Synopsis and Recommendations, vol. 800, NIST Special Publication, 2011.

[17] R. Cohen, Defining Elastic Computing, 2009, http://www.elasticvapor.com/2009/09/defining-elastic-computing.html.

[18] R. Buyya, J. Broberg, and A. M. Goscinski, Cloud Computing: Principles and Paradigms, vol. 87, John Wiley & Sons, New York, NY, USA, 2010.

[19] S. Dustdar, Y. Guo, B. Satzger, and H.-L. Truong, “Principles of elastic processes,” IEEE Internet Computing, vol. 15, no. 5, pp. 66–71, 2011.

[20] M. Kuperberg, N. Herbst, J. von Kistowski, and R. Reussner, Defining and Quantifying Elasticity of Resources in Cloud Computing and Scalable Platforms, KIT, Fakultat fur Informatik, 2011.