ISSN: 0975-766X
CODEN: IJPTFI
Available Online through ***Research Article***
www.ijptonline.com

A NOVEL APPROACH FOR SELF ADAPTIVE MULTI-AGENTS RESOURCE ALLOCATION IN CLOUD ENVIRONMENTS

Mohammed Nisar Faruk [a], Dhandapani Sivakumar[b,*]
[a]Department of Computer Science and Engineering, Bharath University, Chennai-600073.
[b]Department of Information Technology, Easwari Engineering College, Chennai-600089.
Email: dgsivakumar@sify.com

Received on: 15.10.2016 *Accepted on: 22.11.2016*

Abstract

The creation of virtualization and Cloud Computing technologies assures a number of features such as improved flexibility, stabilized energy efficiency and lower operating costs for IT domain. However, highly erratic workloads make it demanding to offer quality-of-service assurance in the mean while promising competent resource utilization. To evade breach of service-level agreements (SLAs) or unproductive resource usage, resource allocations need to be tailored endlessly during operation to reflect dynamic application workloads. In this paper, we present a novel approach to self-adaptive multi-agent resource allocation in cloud environments based on online architecture-level performance models. We present a comprehensive setup of a delegate enterprise application, the new Virtenterprise_Cloudapp benchmark, adopted in a virtualized cluster environment.

Keywords: Resource Allocation; Virtualization, Cloud computing, Agents framework.

1. Introduction

In Recent development virtualization and Cloud Computing intend at decoupling applications and services from the primary hardware infrastructures. This delivers rapid flexibility because resources (e.g., CPU, memory, bandwidth) can be allocated on demand basis and personalized in response to dynamic system workloads. Henceforth the Cloud Computing resource allocations would be enlarged up and down in an elastic mode, dazzling the load intensity and resource anxiety of running applications. however, virtualization permits reducing the number of substantial servers in data centers by running multiple autonomous virtual machines (VMs) on the equivalent physical hardware. By humanizing energy efficiency, this creates considerable cost savings for both infrastructure providers and service

providers. moreover, the above mentioned benefits arrive at the cost of bigger system difficulty and dynamics, making it demanding to deliver Quality-of-Service (QoS) guarantees with highly changeable application workloads. Service providers intended to face the following issues during establishments: number of resources allocated to a new service mounted in the virtualized infrastructure on-the-fly, in order to assure Service-Level Agreements (SLA). Number of resource allocations of running applications and dynamic assignment of services based on their workloads. Number of additional resources are necessary to prolong at increasing load conditions due to floating workloads. How much resources can be deployed, without compromising SLAs. Responding such questions necessitate the capability to predict at run-time performance of running applications would affected if workload is dynamic, as well as the capability to predict the outcome of changing resource allocations to acclimatize the system. This paper refers as online performance prediction. The latter permits to proactively settle in the system to the fresh workload conditions evading SLA breach or inefficient resource utilization. Over the past decade, a numerous performance prediction methods depend on architecture-level performance model have been originated by the performance engineering community [15]. However, these techniques are embattled for offline use at system design and consumption time, and are generally engaged for evaluating substitute system designs and for sizing and competence planning before engaging the system into production. The benefit of such techniques, balanced to techniques, e.g., [10, 11, 13], depend on traditional performance models.

They could potentially permit to explicitly confine the performance influences of software architecture, the application custom profiles as well as the completing environment. In the meanwhile the software architecture characteristically does not change throughout the operation, the application handling profiles, the resource allocations at the different levels of the execution domain may change regularly. However, though the software architecture would not change frequently, its performance-lever behavior has considered. In this context, the input parameters conceded to a service may have direct contact on the group of software components concerned in executing the service and their internal performance and resource demands. Henceforth, the detailed performance model is confining the performance-relevant features of both the software level architecture and the multi-layered domain. In this paper, we deliver a novel approach to self-adaptive multi-agent resource allocation in cloud environments. We investigate the use of such models to treat online performance forecast allowing to predict the dynamic changes in cloud user workloads, and to predict the effects of relevant

reconfiguration actions, commencement to avoid SLA violations or inefficient resource utilization. We detailed experimented setup with a representative application, the new Virtenterprise_Cloudapp benchmark1, adopted in a virtualized cloud environment. The setup defined as a proof-of-concept presenting the feasibility of using architecture level performance models at run-time. The contributions of this article are: i) a autonomous self-adaptive control loop and a relevant resource allocation algorithm for virtualized cloud environments depend on online performance models, ii) an implementation of our approach in the framework with a enterprise application, the new Virtenterprise_Cloudapp benchmark, of a sensible size and complexity, iii) an investigational evaluation of the urbanized framework representing its effectiveness and convenient applicability. The rest of this paper is prepared as follows: Section 2 offers some background on performance models; Section 3 illustrates our self-adaptive multi-agent resource allocation approach. In Section 4, we depict the architecture of the Virtenterprise_Cloudapp benchmark, the ensuing performance model, and the results of the appraisal of our approach. Finally, we review connected work. in Section 5.

We differentiate between evocative architecture-level performance samples and extrapolative performance samples. The former illustrates performance-relevant features of software architectures and execution environments (e.g., UML models embedded with performance observations). The latter confine the chronological system behavior and that can be used for performance forecasting by means of logical or simulation techniques. Over the past decade, a number of architecture-level concert meta-models for describing performance-related aspects of software architectures and execution domains have been implemented by the performance engineering community, the most outstanding examples are meta-models include CSM, PCM and KLAPER and UML SPT and MARTE profiles [9,14]. The general goal of these works is to make it possible to forecast the system performance by transforming architecture-level performance samples into predictive performance samples in a mechanical or semi-automatic manner. They are usually tend to evaluate alternative system designs or for sizing and capacity forecasting before deploying the system into production. We investigate the use of architecture-level performance samples as a means for online performance prediction through system operation [5]. These samples allows for modeling the architectural layers and their configurations and the nature of provided services explicitly. Therefore, the comparative performance of different execution domains or software heaps can be captured. Particularly when modeling virtualized cloud environments, where virtualization layers may tend to change during operation, an explicit modeling of the performance manipulation is valuable. In modern trend, with the

increasing deployment of component based software engineering, the software performance evaluation community has focused on embedding and extending predictable performance engineering methods to support component- based machines which typically used as base for building modern enterprise level applications. The most prominent component-based performance modeling languages, in the form of parameterization with supporting tool is the Palladio Component Model (PCM) [8,9]. In this paper, we define the PCM as architecture-level performance model by allowing to explicitly sampling dissimilar usage profiles and resource allocations. To obtain predictions from the PCM models, we depict the Simucom framework that inherits a queueing-network based simulation [1]. In order to detain the time behavior and resource utilization of a component, PCM has four factors [1]. Perceptibly, the component's execution affects its performance. in addition, the component may fall on external services whose performance need to be considered. Moreover, in both way the component is used (the usage profile together with service input parameters, and the execution domain in which the component is running). PCM permits model processing resources like, CPUs and HDD. The allocation model explains the mapping of component occurrences from the system model to resources detailed in the resource environment model. The usage model describes the user activities. It confines the services that are referred at run-time, the request frequency in which services are referred and the input parameters approved to them.

2. Self-Adaptive Resource Management

This segment describes our self-adaptive multi-agent resource management algorithm based on a control loop model.

2.1 Adaptation Control Loop

The deployment of control loop detailed in Figure 1. The Forecasting Agent, we consider that changes of the workload are either declared by the customers (sales promotion) or by methods similar to workload forecasting [3]. In the utility agent, we use the detailed software performance samples to predict the effect of changes and to conclude which actions need to take. The Forecasting phase further implements the reconfiguration options taken into consideration. The Modern cloud virtualization and middleware technologies offer several possibilities for dynamic resource allocation with system reconfiguration. The virtualization permits to add/remove virtual servers cores to virtual machines or to modify the hypervisor scheduling constraints, for example, increasing/decreasing the capability (CAP) parameter. Application servers naturally provide means to generate application server clusters and animatedly add/remove cluster nodes. Henceforth, virtualization allows to switch virtual machines from one physical server to another physical server.

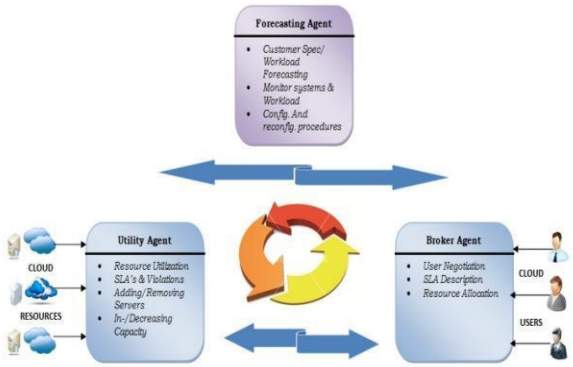


Fig. 1: The Model configuration process.

The described dynamic reconfiguration alternates have their precise advantages and reasonable drawbacks. Most of them used at run-time (on-the-fly) but require a extraordinary system setup or introduce elevated reconfiguration overhead (Virtual Machines live migration). Hence, for our self-adaptive multi-agent resource allocation algorithm, we spotlight on adding/removing virtual CPUs and adding/removing application servers to an server cluster.

2.2. Resource Allocation Algorithm

In this segment, we deliver a prescribed definition of multi-agent resource allocation algorithm. This algorithm involves of two phases: GET phase and SET phase. The GET phase allocates required resources until all client Service Level Agreement are fulfilled. The SET phase optimizes the resource effectiveness by deallocating resources in which the resources are not utilized efficiently. We deliver the algorithm in generic terms, through which it can be applied on different types of resources and resource allocation platforms.

Formally, the Virtual Cloud environment can be symbolized as a 3-tuple

O = (R, S,N) where,

$R = \{r_1, r_2, ..., r_o\}$ is the set of resource types (Different types

of Virtual Machines executed in the cloud environment),

$S = \{s_1, s_2, ..., s_n\}$ is the set of services accessible in the cloud environment,

$N = \{n_1, n_2, ..., c_l\}$ is the set of client workloads and corresponding

SLAs. Each $n_q \in N$ is represented as triple (s, φ, ρ) where

$s \in S$ is the service is being used, φ is the workload intensity of different requests

(expected arrival rate), and ρ is the average response time (SLA) requested by the client.

We include the following functions:

$E \in [S \rightarrow 2^R]$ type of resource are required by service $s \in S$,$F \in [S \times R \rightarrow Q^{s,t}]$ modulated to as resource allocation function .Assignment of each service $s \in S$ a set of instances $Q^{s,t}$ of resource type $r \in R$ (Virtual machines instances). For Each resource type instance is considered to be allocated a number of equal processing resources (HDDs, CPUs). officially, the resource type instances $q \in Q^{s,t}$ is represented in the form of triple (□ , □ , □), where □ defines the processing rate of its processing virtual resources, □ is the no. of processing virtual resources currently Allocated to the clients (allocated virtual CPUs), and □ is the maximum No. of processing resources can be allocated (no. of CPUs mounted on a physical machine). $D \in [S \rightarrow T$ □] specifies the demand factor of service $s \in S$ in the equivalent unit as the processing rate π of the virtual resource type. We conclude the subsequent performance metrics: $N_{tot}(n)$ is the total number of requests described in client workload $n \in N$ completion time per unit (request throughput), $T_{avg}(n)$ is the avg. response time of each service request in the specified client workload $n \in N$, $R_{util}(r)$ is the avg. utilization of virtual resource type $r \in R$ over all instances of the virtual resource, $R_{Util}(r)$ is the maximum permitted avg. utilization for resource type $r \in R$. Finally, we classify the subsequent predicates:$P(N_{tot}(n))$ for $c \in C$ is defined as $(N_{tot}(n) = c[\varphi])$, $P(T_{avg}(n))$ for $c \in C$ is defined as $(T_{avg}(n) \leq c[\rho])$, $P(R_{util}(r))$ for $r \in R$ is defined as $(R_{util}(r) \leq R_{Util}(r))$. For a configuration characterized by a resource allocation function *F* to be satisfactory by the following condition must have $(\forall n \in N : P(N_{tot}(n)) \wedge P(R_{util}(r)) \wedge (\forall r \in R : P(R_{util}(r))$. This condition is verified in terms of our online performance prediction method.

The client workloads changes dynamically every time $N \rightarrow N$ (for example a new client workload $n = (s, \varphi, \rho)$ is programmed for execution or a transform in the workload intensity φ of an previous workload forecast), we assume that the online performance prediction method to predict the effect of this dynamic change in the overall client system workload. If an SLA violation is identified, the GET phase of our algorithm is mounted in which allocates additional resources until all client Service level agreement are satisfied. After the GET phase completed, the SET phase is started to optimize the resource efficiency. If there is no SLAs violation, the SET phase starts immediately. In the following section we elaborate GET and SET phases in more detail.

2.3 Get Phase: The representation of this algorithm described in mathematical style pseudo code presents more fundamental heuristic for allocating resources to different kinds of services through which the client SLAs are satisfied.

while $n \quad N: \quad P(R_{util}(r))$ do

for all $r \in E(n[s]) \dot{:}= P(R_{util}(r))$ do while $cap(n, r) \leq cap(n, r)$ do

if $\exists q \in F(n,[s], r) : q(\mu) < q(\mu)$ then $q[\mu] \leftarrow q[\mu] + 1$

Else

$F(n,[s], r) \leftarrow F(n,[s], r) \cup \{q\hat{}\}$

end if

end while

end for

end while

essentially, while existence of a client response time and the SLA is violated, the algorithm dramatically increases the quantity of allocated resources for all virtual resource types incurred by the service that currently go beyond their maximum permitted utilization $R_{util}(r)$ .This is based on the hypothesis that violations are rooted by at least one virtual resource type and the violated SLA has becomes serious bottleneck. By Increasing the number of allocated resources mechanism as follows, If there are instance of the over utilized virtual resource type r (VM's) which has a few processing resources availability (virtual CPUs) that are not allocated yet, the additional resources are need to be allocated. Or else, a new instance of the resource type $q\hat{}$ is included (a new VM is initiated). In our algorithm describes, an additional resource instance increases sequentially to the total capacity.

2.4 Set Phase

The SET phase targeted to optimize the resource efficiency by demanding to release virtual resources which is not utilized at the maximum by the respective client workloads.

for all $n \in N$ do

while $\exists r \in E(n[s]) : R_{Util}(r) - R_{util}(r) \geq \varepsilon$ do if $\exists q \in F(n,[s], r) : q(\mu) > 0$ then

$q[\mu] \leftarrow q[\mu] - 1$

if $\neg P(R_{util}(r))$ then

$q[\mu] \leftarrow q[\mu] + 1$

end if

if $q[\mu] = 0$ then

$F(n,[s], r) \leftarrow F(n,[s], r) \setminus \{q\}$

end if end if

end while end for

The optimization algorithm is elaborated to overall client workloads n ∈ N. While there is a virtual resource type t assigned to service s of the currently assumed workload n whose delta between the max. utilization R_{util} (r) and current utilization

R_{util} (r) is higher than a predefined constant □ , the amount of resources allocated to this current service will be decreased, Hence for a resource type instance q of r which presently has some resources allocated (virtual CPUs), the amount of allocated resources is reduced. If the client SLAs are predicted to be violated after this drastic change, the change is partially reversed. In case after the change, the instance has no outstanding allocated resources, the instance q can be detached from the set of virtual resource type instances (VM may be shut down). Note that the group of a virtual resource type instances can also turn out to be empty,, if there is no service left in the queue using the respective virtual resource type r.

3. Experimental Setup

In the experimental setup of hardware environment, we mount six blade servers from a cloud cluster virtual environment. Each server is prepared with two or three Intel Xeon E5430 4-core CPUs working with 2.66 GHz and 32 GB of main memory. The machines are linked by a 1 GBit LAN. The above figure shows the experiment environment. On each machine has Citrix XenServer 5.5 as the cloud virtualization layer in every the XenServer's VMs, we sprint the benchmark components (Supplier emulator, application servers, load balancer, driver agents).

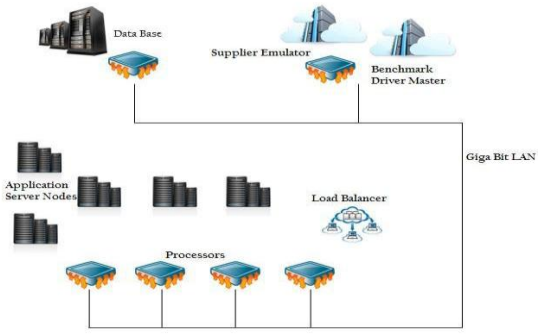


Fig 2: The Experiment setup.

Each component works with its own Virtual Machines, at initial stage this equipped with two virtual CPUs. The operating system point of view these VMs perform CentOS 5.3. As Java Enterprise Edition application server, we assume the Oracle Weblogic Server (WLS) 10.3.3. The load balancer processed with haproxy 1.4.8 using round-robin mechanism as a load balancing strategy.

The Benchmark driver agents are incorporated by the Faban framework that is migrated with this benchmark. The database is an Oracle 11g database server instance mounted on each Virtual Machines(VM) with eight VCPUs on a individual node on Windows Server 2008. The Virtenterprise_Cloudapp benchmark application is implemented in a cluster of WLS nodes.

For the evaluation perspective, we mentioned reconfiguration options relating to the WLS cluster nodes and the VCPUs the VMs are prepared with; WLS nodes may added or removed with WLS cluster and also the VCPUs are added or removed with VM. These reconfigurations are applicable at run-time(on-the-fly), Therefore this can be applied while the Virtenterprise_Cloudapp application is running. In our simulations, the VMs map to virtual resource type instances in reallocation algorithm and their VCPUs are map to the capacity parameter(cap).

4. Evaluation

We evaluate our approach in different scenarios presented in the following.

4.1 Adding a New Service

In the first scenario is proposed to evaluate the results of allocation when a new service is deployed in the virtual cloud environment at run time. Consider that there are four services proposed in our virtual environment running on one node with two VCPUs by default configuration mentioned as C_0 .The SLAs of currently running services are described as: (Create-VehicleEJB, 17, 84ms), (Purchase, 11.5, 90ms), (Manage, 10.5, 90ms), (Browse, 35, 96ms). This specification defines the same data as the more common client workload specifications. Figure 3.

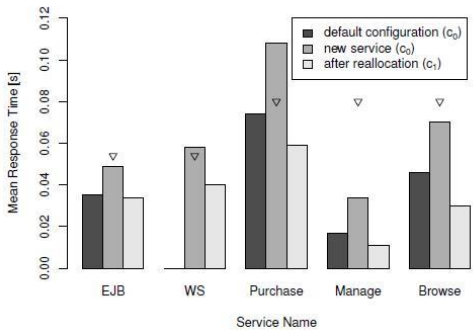
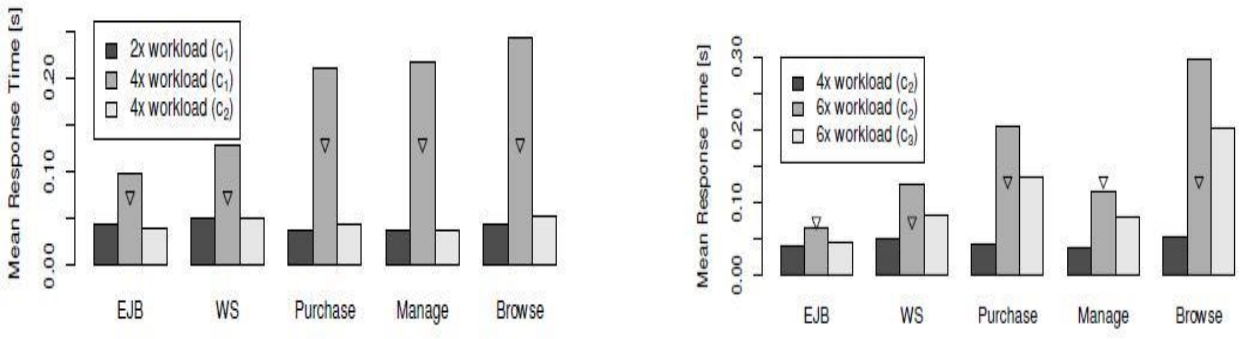


Fig. 3: The response times each services and their respective SLAs (represented by ∇) before and after reallocation when mounting a new service.

Now a new service mounted with the new SLA (Create-VehicleWS, 17, 84ms) is added. To guarantee that all SLAs are still preserved after deployment of each new service, our self-adaptive multi-agent resource allocation mechanism is

triggered. The results are depicted in *Figure 4(a)*. After mounting the new service to the model, the mechanism predicts SLA violations for the Create-VehicleWS and Purchase services. Hence, the GET Phase of the reconfiguration algorithm initiates and refers a capacity increase by one, henceforth adds an extra VCPU to the existing node which is mentioned as C_1. After this variation, the simulation indicates fulfilled SLAs, therefore the algorithm go into the SET-Phase and tries to decrease the overall quantity of used resources with all workload classes, but this fails because after the SLAs of Create-VehicleWS and Purchase service are again violated. Therefore, the resultant configuration of our algorithms involves of one node with three VCPUs. The above behavior was established in our experiments depicted in *Figure 4 (b)* . The measurements depicts that with the default resource allocation the SLA for Create-VehicleWS and Purchase service cannot be continued. However, subsequent resource allocation proposed by our algorithm, all SLAs are satisfied.



4.a) increasing workload from 2x to 4x **4.b) increasing workload from 4x to 6x**

Fig. 4 The response times with dynamic workload from 2x to 4x and 4x to 6x, respectively (SLAs denoted by ∇).

Our initial point is that five services are processing on one cluster node with three VCPUs (C1) with doubled the standard workload and the subsequent SLAs (Create-Vehicle EJB, 50, 94ms), (Create-Vehicle WS, 40, 84ms), (Purchase, 45, 140ms), (Manage, 35, 110ms), (Browse, 50, 130ms) which are partially satisfied at initial stage. Now, we boost the workload to 4x with the standard load. For this newly mounted workload, the reallocation algorithm identifies a violation of the SLAs and prefer the reallocating the system resources which using two nodes. Applying this configuration to our benchmark application, the SLAs are satisfied. For the measurement results *see Figure 4 a).* In the next step, we increase the cluster workload to 6x with the standard load, without changing the SLAs. Again, this leads to a breach of the SLAs in simulation results. Henceforth, we deply our algorithm, finding a new appropriate configuration with three nodes. The experiment results are described Figure 4 b). However, the results show that after reallocation the

SLA of the Browse service is still somewhat violated. This is not due to mistakenness of our method, but rather due to elasticity problems of the database machine which is present, again which is not commanding enough to handle the newly mounted workload while satisfying the original SLAs. Hence, we are certain to give a more powerful database, the SLAs should be satisfied.

4.2 Workload Decrease

This feature is proposal is to calculate our approach in situations where the workload reduced. The intention is to release resources which are not utilized proficiently and therefore increase the system efficiency. Consider that the situation all services is processed with 6x the common workload on three nodes - a total number of 11 VCPUs and all SLAs are satisfied (C3). Now, we reduce the workload into 4x the standard load. For this change, our approach predicts that two nodes with a total number of 7 VCPUs (C4) are adequate to handle the reduced workload. The dimension results are depicted in Figure 5, representation that the recommendation is correct. One can see that the configuration C4 the average response time increases, but the SLAs are still pleased.4

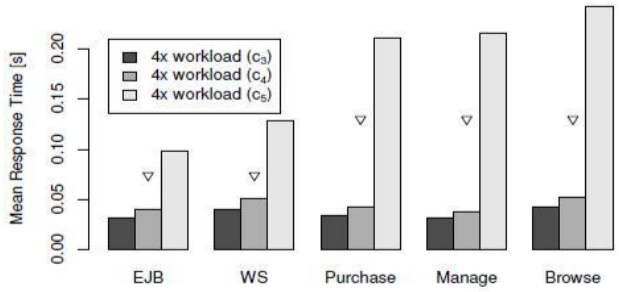


Fig. 5: The response times for processing 4x workload before and after reconfiguration and with fewer resources.

To confirm that resource allocations may not be further condensed while satisfying the SLAs, we further compact the allocated resources manually to 1 node with 4 VCPUs (c5). The consequences for this configuration show that it would infringe the SLAs, henceforth the previously found configuration would be valid.

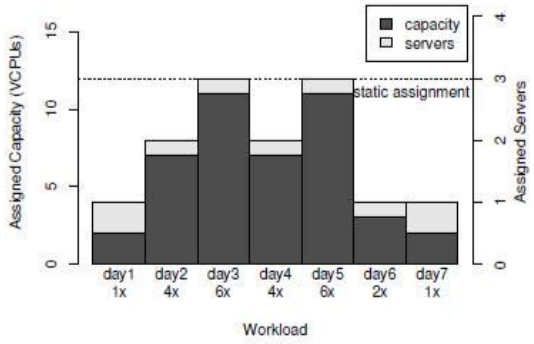


Fig. 6: Allotted capacity and servers dynamic workload distribution more than 7 days

4.3 Resource Usage and Efficiency

After evaluating the functionality of our approach, this section describes its benefits. Consider that the workload distribution over ten days like depicted in *Figure 6*. In a constant scenario, one would assign three or four dedicated servers to guarantee the SLAs for the peak load. Moreover with our approach the system can dynamically assign the system resources during run time. In the static scenario, the system would use 7x3 = 21 servers; hence our approach requires only 1+2+3+2+3+1+1 = 13 servers. Hence, in such a situation, only 62% of the resources of the constant assignment are needed and almost 40% of the resources will be saved.

4. Related Work

Many related work which is configured resource allocation algorithms is done offline mode that can be identified in the areas of capacity planning and resource management. In recent trends the virtualization and Cloud Computing also other variations on automatic resource management will appear. In this segment, we give a detailed summary of the latter aspects and then deliver some works in detail. Many researches worked with resource allocation problem using various approaches similar to bin packing, multiple knapsack issues, etc. in context the dynamic resource allocation was previously research pointed this issues using linear optimization techniques [11]. Non-linear optimization techniques are purely based on simulated annealing [7] and fuzzy logic [8]. However, the resource allocation issues in virtualized environments are more difficult because of the depletion of virtual resources. The growth of cloud computing with multiple platforms, there have been quite a few approaches on QoS and resource management techniques at runtime [2, 8, 11, 15]. Moreover, these advancements are frequent on a very top level of resource management issues and contract with very coarse-grained resource allocation issues (e.g. are limited to only adding/removing Virtual Machines[2]) or target on different optimization techniques.

5. Conclusions

In this paper, we invented a novel approach to self-adaptive multi-agent resource allocation on-the-fly. We described performance models to predict the effect of fluctuations in the cloud service workloads and the corresponding system reconfiguration actions. By using cloud virtualization techniques, we implement these dynamic allocation to the Virtenterprise_Cloudapp benchmark to assess the samples for online performance prediction. The resultatent values depicts that our methodology can be applied to dynamic environment to discover efficient resource allocations and

satisfying specified SLAs. In an example, we defined that this method can save and utilize up to 40% of the resources. In the future, we plan to expand our resource allocation with enhanced heuristics for discovering resource allocations. Moreover, we sketch to evaluate our approach with different types of resources. In addition, we propose to consider the consequence of shared resources in virtualized cloud environments by enlarging the performance methods with virtualization and variant physical resources network and storage.

References

1. GENI System Overview. http://www.geni.net/.
2. J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian, "Resource Management in the Autonomic Service-Oriented Architecture," in ICAC '06: IEEE International Conference on Autonomic Computing, 2006, pp. 84–92.
3. D. Chess, A. Segal, I. Whalley, and S. White, "Unity: Experiences with a Prototype Autonomic Computing System," in 2004. Proceedings. International Conference on Autonomic Computing, 2004, pp. 140–147.
4. Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.
5. X. Wang, D. Lan, G. Wang, X. Fang, M. Ye, Y. Chen, and Q. Wang,"Appliance-Based Autonomic Provisioning Framework for Virtualized Outsourcing Data Center," in Autonomic Computing, 2007. ICAC '07. Fourth International Conference on, 2007, pp. 29–29.
6. F. Hermenier, X. Lorca, J. M. Menaud, G. Muller, and J. Lawall, "Entropy: A Consolidation Manager for Clusters," in VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 2009, pp. 41–50.
7. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. Intl. Journal of Supercomputer Applications, 11(2):115– 128, 1997.
8. J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor- G: A Computation Management Agent for Multi-Institutional Grids. Cluster Computing, 5(3):237– 246, 2002.
9. M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, pages 59–72, New York, NY, USA, 2007. ACM.

10. Chenhong Zhao, Shanshan Zhang, Qingfeng Liu, "Independent Tasks Scheduling Based on GeneticAlgorithm in Cloud Computing", IEEE Computer society,978-1-4244-3693-4,2009.
11. Hai Zhong, Kun Tao, Xuejie Zhang, "An Approach to Optimized Resource Scheduling Algorithm for Open-source Cloud Systems" , IEEE Computer Society, 978-0-7695-4106-8, 2010.
12. B. Sotomayor, K. Keahey, I. Foster," Combining batch execution and leasing using virtual machines", ACM 17th International Symposium on High Performance Distributed Computing, pp. 87-96, 2008.
13. M. Mazzucco, D. Dyachuk, and R. Deters, "Maximizing Cloud Providers' Revenues via Energy Aware Allocation Policies," in 2010 IEEE 3rd International Conference on Cloud Computing. IEEE, 2010, pp. 131– 138.
14. A.-C. Orgerie, L. Lefevre, and J.-P. Gelas, "Demystifying energy consumption in grids and clouds," in Green Computing Conference, 2010 International, 2010, pp. 335–342.
15. B. Rajkumar, B. Anton, and A. Jemal, "Energy efficient management of data center resources for computing: Vision, architectural elements and open challenges", in International Conference on Parallel and Distributed Processing Techniques and Aplplications, Jul. 2010.