An Economic Approach for Application QoS Management in Clouds

Stefania Costache[1,2,*], Nikos Parlavantzas[2,3], Christine Morin[2], and Samuel Kortas[1]

[1] EDF R&D, France
[2] INRIA Centre Rennes - Bretagne Atlantique, France
[3] INSA Rennes, France
{Stefania.Costache,Nikos.Parlavantzas,Christine.Morin}@inria.fr, Samuel.Kortas@edf.fr

Abstract. Virtualization provides increased control and flexibility in how resources are allocated to applications. However, common resource provisioning mechanisms do not fully use these advantages; either they provide limited support for applications demanding quality of service, or the resource allocation complexity is high. To address this problem we propose a novel resource management architecture for virtualized infrastructures based on a virtual economy. By limiting the coupling between the applications and the resource management, this architecture can support diverse types of applications and performance goals while ensuring an efficient resource usage. We validate its use through simple policies that scale the resource allocations of the applications vertically and horizontally to meet application performance goals.

1 Introduction

Managing resources of private clouds while providing application QoS guarantees is a key challenge. A cloud computing platform needs to host on its limited capacity a variety of applications (e.g., web applications, scientific workloads) that possibly require different QoS guarantees (e.g., throughput, response time). Thus, the resource management system is required to be flexible enough to meet all user demands while ensuring an efficient resource utilization. The flexibility of the resource management can be achieved by decoupling the application performance management from the infrastructure resource management and passing information about applications to the infrastructure in a generic way. An efficient resource management is possible by using virtualization technologies to dynamically provision the resources in a fine-grained manner and to transparently balance the load between physical machines. However, common resource management systems either fail to address these requirements or they achieve them through algorithms that have a high computational complexity and would not scale well with the size of the infrastructure [6].

[*] This work is supported by ANRT through the CIFRE sponsorship No. 0332/2010.

In this paper we present a resource management architecture for cloud platforms that addresses the flexibility and efficiency issues through a market-based approach. Each application is managed by a local agent that determines the resource demand that meets the application's performance goal, while a global controller performs the infrastructure resource management based on the agent's communicated application preferences. The agent communicates its application preferences by submitting bids expressing their willingness to pay for resources. The global controller uses a proportional-share rule [5] to allocate resources to applications according to their bid. The resource price variation provides service differentiation between applications while the proportional share ensures a maximum utilization of infrastructure resources. While this model does not necessarily lead to a global optimal resource allocation, it allows applications to closely meet their performance goals while keeping a simple resource management. We illustrate how this model supports application performance goals through agents that scale the allocation of their applications using feedback-based control policies. We simulated our architecture and validated the policies in contention scenarios using the CloudSim toolkit [2].

This paper is organized as follows. In Section 2 we give an overview of our solution and describe the main architecture elements and in Section 3 we describe how the architecture can be used to execute different application types. Section 4 describes the related work. Finally, we conclude and present future steps in Section 5.

2 Architecture

In this Section we describe the architecture of our solution. We detail the main components and the interaction between them. We then describe the current implementation of the proportional-share allocation algorithm and the assumptions that we make regarding the infrastructure's virtual currency management.

Overview. Figure 1 shows the main architecture components. Our architecture consists of distributed *application managers* that receive a *budget of credits* from a *budget manager* and execute applications submitted by users. To request resources for their applications, the managers communicate with a *resource controller* that provisions them virtual clusters (i.e., groups of virtual machines) from a *virtual infrastructure manager* and charges them for their used resources. This *virtual infrastructure manager* (e.g. OpenNebula [9]) supports operations related to creation, destruction, and dynamic placement of virtual machines. We also consider that it is capable of providing monitoring information about the physical hosts and virtual machines to the resource controller.

The application managers are started when the applications are submitted to the infrastructure and manage the application's life-cycle. A manager requests resources for its application by submitting bids of the form $b(n, r_{min}, s)$ to a resource controller. This bid specifies the size of the virtual cluster, n, a minimum resource allocation, r_{min}, that a resource controller should ensure for any

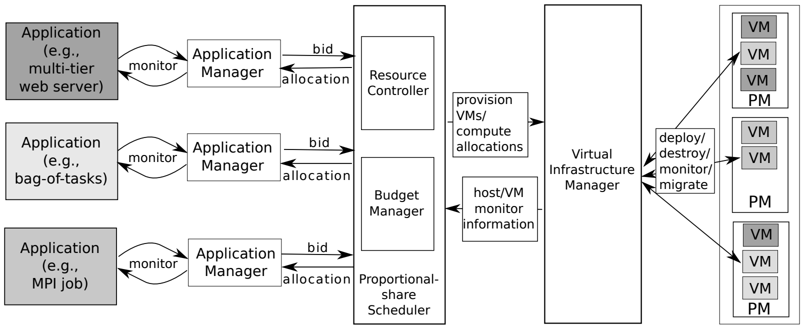
Application (e.g., multi-tier web server)
monitor
Application Manager
bid
allocation
Application (e.g., bag-of-tasks)
monitor
Application Manager
bid
allocation
Application (e.g., MPI job)
monitor
Application Manager
bid
allocation
Resource Controller
Budget Manager
Proportional-share Scheduler
provision VMs/ compute allocations
host/VM monitor information
Virtual Infrastructure Manager
deploy/ destroy/ monitor/ migrate
VM
VM
VM
PM
VM
VM
PM
VM
VM
VM
PM

Fig. 1. Architecture overview

instance of the virtual cluster and the manager's willingness to pay for the allocated resources, s (spending rate). After its virtual cluster is allocated, the manager starts its application. During the application execution the manager monitors the application and uses application performance metrics (e.g., number of processed tasks/time unit), or system information (e.g., resource utilization metrics) to adapt its resource request to its application performance goal. This can be done in two different ways: (i) by changing the virtual cluster size; (ii) by changing the spending rate for the virtual cluster.

The resource controller allocates a resource fraction (e.g., 10% CPU or 1MB memory) on a physical node for each virtual machine instance of a virtual cluster. This allocation is enforced by a Virtual Machine Monitor (e.g., Xen [1]) and is proportional with the manager's spending rate and inversely proportional with the current resource price. If the allocation becomes lower than the minimum resource allocation requested by the manager then the virtual cluster is preempted.

Resource Allocation. The resource controller recomputes the allocations for all running virtual machines periodically. At the beginning of each time period, the resource controller aggregates all newly received and existing requests and distributes the total infrastructure capacity between them through a proportional-share allocation rule. This rule is applied as follows.

We consider the infrastructure has a total capacity C that needs to be shared between M virtual machine instances. Each virtual machine receives a resource amount defined as $a = \frac{b_j}{P} \cdot C$, where s_i is the spending rate per virtual machine and $P = \Sigma_{i=1}^{M} s_i$ is the total resource price. However, because the capacity of the infrastructure is partitioned between different physical nodes, after computing the allocations we may reach a situation in which we cannot accommodate all the virtual machines on the physical nodes. Thus, instead of computing the allocation from the total infrastructure capacity, we compute the allocation considering the node capacity and we try to minimize the resulting error. For simplicity we assume that the physical infrastructure is homogeneous and we treat only the CPU allocation case.

The algorithm applied by the resource controller has the following steps. To ensure that the allocation of the virtual machine instances belonging to the same group is uniform, the spending rate of the group is distributed between the virtual machine instances in an equal way. Then, the instances are sorted in descending order by their spending rates s. Afterwards, each virtual machine instance from each virtual cluster is assigned to the node with the smallest price $p = \Sigma_{k=1}^{m} s_k$, given that there are m instances already assigned to it. This ensures that the virtual machine gets the highest allocation for the current iteration, fully utilizing the resources and minimizing the allocation error. The resource allocations for the current period are computed by iterating through all nodes and applying the proportional-share rule locally.

Finally, the application managers are charged with the cost of using resources for the previous period, $c = \frac{s}{M} \Sigma_{i=1}^{M} u_i$; u_i represents the total amount of used resource by a virtual machine instance i belonging to the virtual cluster of size M.

Budget Management. The logic of distributing amounts of credits to application managers is abstracted by the budget manager component of our architecture. For now we consider that this entity applies a credit distribution policy that follows the principle "use it or loose it". That is, each manager receives an amount of credits at a large time interval. To prevent hoarding of credits, the manager is not allowed to save any credits from one time interval (i.e., *renew period*) to another. We also consider that this amount of credits can come from an user's account, at a rate established by the user itself; we don't deal with the management of the user's credits in the rest of this paper.

3 Use Cases

We illustrate how the agents can adapt either their spending rates or their virtual cluster size to take advantage of the resource availability and to meet specific application goals. We consider two examples: (i) a rigid application (e.g., MPI job) that needs to execute before a deadline; (ii) an elastic application (i.e., bag-of-tasks application) composed of a large number of tasks that can be executed as soon as resources become available on the infrastructure; we assume that a master component keeps the tasks in a queue and submits them to worker components to be processed. For the first case the manager requires a virtual cluster of fixed size to the resource controller and then it controls the virtual cluster's allocation by scaling its spending rate. For the second case the manager requires a virtual cluster with an initial size which is then scaled according to infrastructure's utilization level. Both application models are well known in the scientific community and are representative for scientific clouds.

We analyzed the behavior of our designed managers by implementing and evaluating our architecture in CloudSim [2]. We don't consider the overheads of virtual machine operations as we only want to show the managers behavior and not the architecture's performance. As we focus on the proportional-share of CPU resources, we consider that the memory capacity of the node is enough to accommodate all submitted applications. We describe next the design and behavior of each manager.

3.1 Adapting the Agent's Spending Rate

In this case we design a manager that uses application progress information to finish the application before a given deadline while being cost-effective. We describe the manager logic and we analyze its behavior under varying load.

Application Management Logic. To provision resources for its application, the manager uses a policy that adapts its spending rate based on a *reference progress*. This reference progress represents how much of the application needs to be processed per scheduling period to meet its deadline:

$$p_{reference} = \begin{cases} min(\frac{total\_length}{execution\_time}, \frac{length}{deadline - now}), \text{ if } now < deadline \\ \frac{total\_length}{execution\_time}, \text{ otherwise} \end{cases} \quad (1)$$

The length is a parameter specific to the application: it can be number of files that the application needs to process to finish its execution, number of iterations or instructions. The execution time represents the time in which the application finishes if it runs alone on the infrastructure. If the current time is smaller than the application deadline, the reference progress is computed as the remaining application length distributed over the remaining execution time. Otherwise, the application is already delayed, so it is desirable to make a maximum amount of progress in its computation.

The manager monitors its application and receives information about the progress made in the last scheduling period. To save its budget for future use, if the application made enough progress then the manager decreases its bid. When the application cannot meet its reference progress the manager uses all its saved credits. To adapt the bid, the manager uses a subtractive decrease/multiplicative increase rule:

$$b = \begin{cases} max(p_r, b - \alpha * p_r), \text{ if } p_{current} \geq p_{reference} \\ min(b_{max}, \beta \cdot b), \text{ otherwise} \end{cases}, \quad (2)$$

where α and β are configurable parameters that establish the scaling rate of the bid and p_r is the minimum price of using resources. To avoid depleting its budget before the application completion, the manager limits its maximum submitted bid to an amount b_{max}. For a more efficient use of the budget, we choose the smallest time period between the remaining time to the budget renew and the estimated remaining execution time of the application and we distribute the current budget over it. The remaining execution time is estimated as the remaining time to completion if the application continues to make $p_{current}$ progress each scheduling period. Given a budget B, the manager computes b_{max} as follows:

$$b_{max} = B_{current}/(min(renew - now, remaining\_execution\_time) \cdot C_{node}) \quad (3)$$

Evaluation. To illustrate the advantage of using a feedback-based control manager, we simulate the execution of a deadline-driven application under varying workload. We consider that the infrastructure is used to run best-effort and

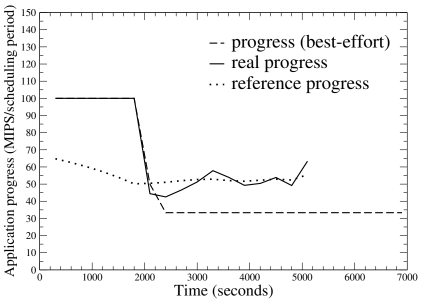

Fig. 2. Application progress variation in time

deadline-driven applications. For the best-effort application we define a manager that distributes its budget equally over the renew period.

For our experiment we consider the following settings. The managers are given an amount of *450000 credits* that is renewed at *3600 seconds*. The reserve price is set to $1credit/second$. 3 applications, each of them with a single task of length of 360000 MIPS are submitted to a single physical node with 100 MIPS. The first application is submitted with a deadline of 5400 seconds while the other two are best-effort. These best-effort applications are submitted after 1800 seconds at a distance of 5 minutes each. The scheduling period is set to 5 minutes. To scale its bid, the manager uses the feedback control rule parameters: $\alpha = 0.5$ and $\beta = 2$.

Figure 2 shows the results of adapting the bid to follow the application's reference progress. During the first 1800 seconds the application executes alone on the node so it makes a maximum amount of progress. Thus, its reference progress also drops. After the first 1800 seconds the other two applications start executing one by one so the manager needs to adapt its bid to follow the reference progress. The fluctuations in the real progress represent the result of this adaptation. We compare this case with the best-effort manager. In our case, the application completes before its deadline. However, in the case of the best-effort manager, the application completes much later (1600 seconds past the deadline), because the manager is not aware of the competition for resources.

3.2 Adapting the Virtual Cluster Size

We design a manager that uses its past virtual cluster resource allocation as a feedback and scales its application to minimize its completion time. The manager is willing to spend all its budget at a constant rate. We describe its logic and behavior next.

Application Manager Logic. To scale its application, the manager applies an additive increase/multiplicative decrease rule and uses its virtual cluster past average CPU allocation as a congestion signal. To compute the past average CPU allocation, the manager uses an EWMA filter. As long as the application master has tasks in its queue, the manager expands the virtual cluster. To ensure that the application's tasks already submitted to virtual machines are processed as fast as possible, the manager shrinks the virtual cluster when the existing virtual machines don't have enough CPU. The virtual cluster size (i.e., the number of virtual machines), n, is updated as follows:

$$n = \begin{cases} n + \alpha, \text{ if } a_{avg} \geq T_a \text{ and } remaining\_tasks\_to\_process > 0 \\ \lceil \frac{n}{2^{\beta}} \rceil, \text{ otherwise} \end{cases} \tag{4}$$

where α and β are configurable parameters that establish the scaling rate of the virtual cluster size and T_a is a threshold on the virtual cluster allocation.

Evaluation. To illustrate the benefits of the elastic scaling on the application execution time, we analyze the behavior of the elastic application manager under varying load. For our experiment we consider the following settings. The elastic manager is given a budget of *1.800.000 credits* and the other managers *120.000 credits*; their budgets are renewed at *3600 seconds*. The infrastructure has 10 nodes each with 100 MIPS and the scheduling period is set to 5 minutes. An application with 200 tasks, with an average execution time of 10 minutes each, starts executing. After 200 seconds 15 applications with a length of 360000 MIPS are submitted with an exponential inter-arrival time distribution, with an average inter-arrival time of 160 seconds. The virtual cluster average allocation threshold is set to 85% of C_{node}. The manager is conservative in scaling the virtual cluster and uses the feedback control rule parameters: $\alpha = 1$ and $\beta = 0.5$.

Figure 3 shows the resource allocation variation in terms of CPU (a) and number of virtual machines (b). The manager starts its application with an initial number of 5 virtual machines at full capacity. When the demand is low, the manager gets more resources for its existing virtual machines and expands its virtual cluster. This is noticed after the application is submitted and after the other applications finish their execution. When all the submitted applications are running, the allocation for the existing virtual machines drops and the manager shrinks its virtual cluster to 4 virtual machines. Because the average allocation is greater than the given threshold, when the infrastructure is free the manager actually creates more virtual machines than the infrastructure's capacity. Setting a higher threshold would avoid this behavior.

We compare our proportional-share mechanism to a static allocation mechanism. With the static allocation mechanism the manager doesn't receive any feedback from the infrastructure and is not able to scale its application. When the application is executed with our proportional-share mechanism it finishes in *300 minutes* while in the static allocation case it finishes in *417 minutes*. The elastic behavior of the manager leads to a better resource usage, as seen in Figure 3 (c), and to a smaller execution time of the application.

4 Related Work

Many recent research efforts focused on designing algorithms for dynamic resource provisioning in shared platforms. However few of them decouple the

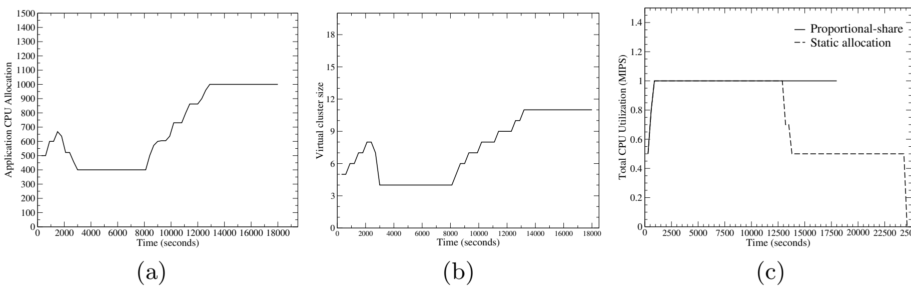

Fig. 3. Application allocation in terms of CPU (a), number of virtual machines (b) and datacenter utilization (c) in time

application performance management from the resource management. This decoupling can be achieved with two mechanisms: i) using utility functions with which applications express their valuation for resources to the resource manager; ii) using an economic model with which both applications and the resource manager act selfishly to maximize their own benefit.

Utility functions were used to dynamically control the resource allocation for applications in a virtualized [6] and non-virtualized [11] datacenter. The users specify their valuation for certain levels of performance, which is then expressed as a function of the application's resource allocation (i.e., resource-level utility). By knowing the resource-level utilities of all applications, a resource manager computes the resource configuration according to a global objective, i.e., maximize the sum of all resource-level utilities [11], ensure a (max-min) fair allocation [3]. As the resource controller needs to determine the most efficient allocation by considering any fraction of resource the application would get, the computational complexity is high. Scaling with the size of the infrastructure and the number of hosted applications clearly demands resource management algorithms with a low run-time complexity.

Opposed to this approach, we use an economic model to dynamically provision resources to applications. Through an economic model [12] the resource control becomes decentralized. Each entity from the system acts selfishly: each application tries to meet its own performance goal while the resource provider tries to maximize its own revenue. Applying this model to dynamically allocate resources between competitive applications is not new. Both Stratford et. al. [10] and Norris et. al. [7] proposed to use the dynamic pricing of resources as a mechanism to regulate the resource allocation between competitive applications. In both cases resources were traded using a commodity market model. However, this model would have a high communication overhead and it would be difficult to use in a large scale system.

A popular approach to regulate access to resources in distributed systems is to use an auction-based market. In auctions the price of the resource is given by the bids of the participants. However, when considering divisible resources, most auction models suffer from the same computational complexity as the utility functions, as the resource manager must compute an efficient allocation. From this perspective, the simplest auction mechanism for resource allocation is the proportional-share introduced by Lai et. al. [5]. This mechanism has a low complexity as it applies a simple computational rule to distribute the resource between competitive users and thus can scale with the size of the infrastructure and the number of applications. We propose in our work to use this mechanism for virtual machine provisioning to allow applications to adapt their resource allocations according to their performance goals.

Several market-based systems [5, 4, 8] propose a proportional-share approach but they do not specifically target cloud infrastructures. From this perspective, the most similar to our work is Tycoon [5]. In Tycoon, resources are allocated through a proportional-share rule on each physical node while agents select the nodes according to user's preferences and budget. In our architecture, the proportional-share rule is applied for the entire infrastructure capacity instead of one physical machine, decoupling the resource provisioning from the physical placement. Our agents are concerned with meeting application goals through intelligently managing their budgets and adapting to the fluctuating resource availability.

5 Conclusions

In this paper we presented a new architecture for managing applications and resources in a cloud infrastructure. To allocate resources between multiple competitive applications, this architecture uses a proportional-share economic model. The main advantage of this model is the decentralization of the resource control. Each application is managed by an independent agent that requests resources by submitting bids to a resource controller. The manager's bid is limited by its given budget. To meet its application performance goals the manager can apply different strategies to vary its bid in time. Through this approach, our architecture supports different types of applications and allows them to meet their performance goals while having a simple resource management mechanism.

We validated our architecture by designing and simulating application managers for rigid and elastic applications. We showed how managers can use simple feedback-based policies to scale the allocation of their applications according to a given goal. This opens the path towards designing more efficient managers that optimize their budget management to meet several application performance goals. For example, in the elastic application case, the manager would take decisions to manage its budget and scale its virtual cluster based on an estimated finish time of the tasks and a possible deadline. A further step would be then to consider applications with time-varying resource demands. Optimizing the resource allocation mechanism and adding support for multiple resource types will also be our next focus. To improve the support of many application types, we plan to add the possibility for applications to express placement preferences. Finally, we plan to implement and validate our architecture in a real system.

References

[1] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP 2003), pp. 164–177. ACM Press, New York (2003)

[2] Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience 41(1), 23–50 (2011)

[3] Carrera, D., Steinder, M., Whalley, I., Torres, J., Ayguade, E.: Utility-based placement of dynamic web applications with fairness goals. In: IEEE Network Operations and Management Symposium, pp. 9–16 (2008)

[4] Chun, B.N., Culler, D.E.: REXEC: A Decentralized, Secure Remote Execution Environment for Clusters. In: Falsafi, B., Lauria, M. (eds.) CANPC 2000. LNCS, vol. 1797, pp. 1–14. Springer, Heidelberg (2000)

[5] Lai, K., Rasmusson, L., Adar, E., Zhang, L., Huberman, B.: Tycoon: An implementation of a distributed, market-based resource allocation system. Multiagent and Grid Systems 1(3), 169–182 (2005)

[6] Nguyen Van, H., Dang Tran, F., Menaud, J.-M.: SLA-aware virtual resource management for cloud infrastructures. In: 9th IEEE International Conference on Computer and Information Technology (CIT 2009), pp. 1–8 (2009)

[7] Norris, J., Coleman, K., Fox, A., Candea, G.: Oncall: Defeating spikes with a free-market application cluster. In: Proceedings of the First International Conference on Autonomic Computing (2004)

[8] Sandholm, T., Lai, K.: Dynamic Proportional Share Scheduling in Hadoop. In: Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2010. LNCS, vol. 6253, pp. 110–131. Springer, Heidelberg (2010)

[9] Sotomayor, B., Montero, R., Llorente, I., Foster, I.: An Open Source Solution for Virtual Infrastructure Management in Private and Hybrid Clouds. IEEE Internet Computing 13(5), 14–22 (2009)

[10] Stratford, N., Mortier, R.: An economic approach to adaptive resource management. In: Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems, HOTOS 1999. IEEE Computer Society (1999)

[11] Tesauro, G., Kephart, J.O., Das, R.: Utility functions in autonomic systems. In: ICAC 2004: Proceedings of the First International Conference on Autonomic Computing, pp. 70–77. IEEE Computer Society (2004)

[12] Yeo, C.S., Buyya, R.: A taxonomy of market-based resource management systems for utility-driven cluster computing. Softw. Pract. Exper. 36, 1381–1419 (2006)