Self-management of cloud applications and infrastructure for energy optimization

Frederico Alvares de Oliveira Jr.
ASCOLA Research Team
(Mines Nantes-INRIA, LINA)
Université de Nantes, France
frederico.alvares@univ-nantes.fr

Thomas Ledoux
ASCOLA Research Team
(Mines Nantes-INRIA, LINA)
Ecole des Mines de Nantes, France
thomas.ledoux@mines-nantes.fr

ABSTRACT

As a direct consequence of the increasing popularity of Cloud Computing solutions, data centers are amazingly growing and hence have to urgently face with the energy consumption issue. Available solutions rely on Cloud Computing models and virtualization techniques to scale up/down application based on their performance metrics. Although those proposals can reduce the energy footprint of applications and by transitivity of cloud infrastructures, they do not consider the internal characteristics of applications to finely define a trade-off between applications Quality of Service and energy footprint. In this paper, we propose a self-adaptation approach that considers both application internals and system to reduce the energy footprint in cloud infrastructure. Each application and the infrastructure are equipped with their own control loop, which allows them to autonomously optimize their executions. Simulations show that the approach may lead to appreciable energy savings without interfering on application provider revenues.

1. INTRODUCTION

With the popularization of Cloud Computing platforms, data centers have become bigger and bigger and the energy consumption due to IT infrastructure has grown dramatically [14]. Cloud Computing models [18], on the other hand, can be seen as a very powerful tool to face the problem of energy consumption. From the application provider point of view, it allows to precisely demand/release resources on the fly. It means that it is possible to dynamically adjust the right amount of resources needed to deliver the desired level of Quality of Service (QoS). From the infrastructure provider point of view, those models enable the mutualization by consolidating the workload of several applications in a few physical machines [8].

Management of both QoS applications and overall infrastructure energy consumption has been proposed through a unique system [12][23][26][19]. The objective is to maximize applications' QoS while minimizing the costs due to the infrastructure (e. g. energy consumption). Although it enables to scale up/down applications by querying/releasing resources according to their incoming charge, applications are considered as *black boxes*. This restrains the adaptation capability of applications in the sense that they are only able to add or remove resources based on performance attributes. From our point of view, it is not sufficient since applications may have specific requirements in terms of reactivity, fault-tolerance and others concerns. The QoS of one application is not only related to performance criteria such as response time but also to internal aspects that may differ from one application to another [4].

From our point of view, providing a unique framework generic enough to take into account every application particularity in addition to the infrastructure constraints seems to be too difficult. Instead, we advocate a per application autonomic system coupled to an infrastructure: each application is equipped with one autonomic loop in charge of determining the minimum amount of resources necessary to provide the best QoS possible while an additional loop manages the physical resources at the infrastructure level. More precisely, this article focuses on adding to the usual elasticity capability (scaling up/down) by considering all applications internals to be able to use several and different *application configurations*. Based on a constraint model, the autonomic loop may switch from one configuration to another according to the incoming charge, the QoS expectations and the infrastructure constraints. As a result, adaptation processes will be triggered in order to query/release Virtual Machines (VMs) to the infrastructure manager.

As a proof of concept, we implemented a prototype which relies on Constraint Programming [21] to model and solve the optimization problem. Results based on simulation show that we can reach appreciable levels of energy savings by considering the trade-off QoS (performance and other application-specific attributes) and energy cost.

The rest of this article is organized as follows: in Section 2, we describe the system architecture of our proposal. Section 3 presents the constraint model for self-management of applications in cloud infrastructures. We present some simulation-based results in section 4. Section 5 provides a discussion about the literature review. Finally, we conclude the paper in section 6.

This work is based on an earlier work: Self-management of applications QoS for energy optimization in datacenters, in Proceedings of the 2^{nd} International Workshop on Green Computing Middleware, © ACM, 2011.
http://doi.acm.org/10.1145/2088996.2088999

2. SYSTEM ARCHITECTURE

This section presents a System Architecture overview of the proposed framework. We first describe the two kinds of parts involved and then we describe how they interact together.

The application provider aims at increasing its applications' QoS while reducing the hosting costs. The infrastructure provider aims to sell as much resources (e. g. CPU and RAM) as possible, while reducing its power consumption costs. For this purpose, we rely on autonomic computing in order to conceive a system consisting of two kinds of control loop: the Application Manager (AM) and the Infrastructure Manager (IM). The choice for several control loops (one per application) instead of only one is justified by the fact that applications may have different characteristics and because these the autonomic phases (monitoring, analysis, planning and execution) may differ significantly from one application to another. For instance, the reaction time (i. e. the time interval between the end of one loop and the start of another) of an application that performs scientific calculation might be different than for a web application. Figure 1 shows the global architecture. For each application there is one AM in charge of maintaining the correct amount of resource according to the different configurations and the workload. At the low level, the IM manages the overall energy footprint by consolidating VMs on a minimum number of servers and arbitrates the allocation of resources according to the application demands [23].

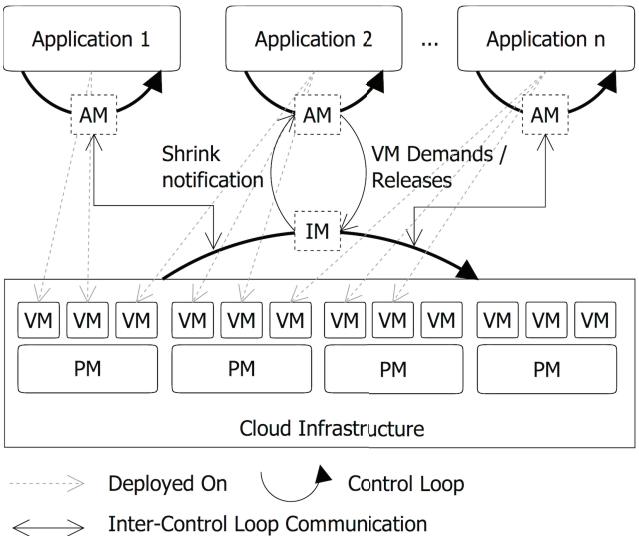

Figure 1: Interaction between Application Manager and Infrastructure Manager

To optimize its incomes, the application provider may decide to switch from one configuration to another more energy and QoS-compliant (e. g. by switching components off or replacing them). For example, let us suppose that an application makes use of a component that is in charge of logging the application's transactions. In this case, as the logging component can be disconnected from other components, the application has two possible configurations: with and without the logging component. Hence, the trade-off between QoS and energy footprint is determined by considering the impact in terms of both energy and QoS resulting of the execution of the logging component. Having the new application configuration, the AM identifies the application components that should be started/stopped, connected/disconnected, and determines in which order those actions should happen and performs the reconfiguration.

On the other hand, the IM gathers some information about the infrastructure such as the physical machines' (PM) utilization rates and the data center power availability. With this information the IM performs two tasks: (i) arbitrate about the distribution of resources among all applications in periods of low availability of resources; and (ii) handle the applications demands for allocation/release of resources. In the first task, the IM arbitrates on which applications should be more penalized based on a pre-defined arbitration policy. For instance, by defining different classes of clients (e. g. diamond, gold, silver, etc.). Then, the IM notifies those applications by informing them about the constraints in terms of infrastructure they must to meet. In the second task, based on the new resources requirements and the current state (mapping VM/PM), the IM consolidates these new VMs along with other applications' VMs so that the number of PMs is minimized [8]. The result of this procedure is a new and optimized VMs/PMs mapping.

It is noteworthy that due to the strong dependencies among the component instances and the hosting VMs, the executions at both levels must be synchronized. For instance, the VM that hosts one component (instance) cannot be stopped before the component has been disconnected from the rest of the application. These issues have already started being addressed in [5].

With respect to the optimization problems, we model the consolidation problem (at the IM) and the resource provisioning and application reconfiguration problems (at the AM) as Constraint Satisfaction Problems and solved them with Constraint Programming techniques.

3. CONSTRAINT MODEL

A Constraint Programming Model [21] is roughly composed of three elements: a set of problem variables, a set of domain functions (constraining the values each variable can assume) and a set of constraints. Given that, a solver engine is in charge of finding the possible solutions to the problem without any more details. For optimization problems, an objective function is defined and an optimal solution is then a solution that minimizes (or maximizes) that function. The managers presented in the previous section (AMs and IM) seek to optimize the trade-off revenues/costs at each layer in an autonomous fashion. In order to synchronize their actions, they must share some context data (e. g. current VM prices) that contributes to guide them to choose the proper configurations [5]. In this section, we present the constraint models we rely on to solve these optimization problems.

3.1 Application Constraint Model

3.1.1 Variables and Domains

One application is composed of a set of components expressed by the vector C. We call configuration the way

those components are linked one to another. The application might have one or several configurations $K = (k_1, k_2, \ldots, k_m)$ and $S_i \subseteq C$ is the set of components used in configuration k_i. In order to consume the application, end users pay an amount according to the quality delivered by the application's services. Thus, the QoS in general has an impact on the application provider's revenues. One configuration k_i has its QoS described in terms of performance $Q_i^{perf} \in [0, 1]$ and its internal configuration specific attribute $Q_i^{spec} \in [0, 1]$. The former corresponds to the number of client requests the applications is able to serve per time unit (e. g. per second), whereas the latter is the quality implied by the application configuration.

Each application component can be hosted by one or several VMs, meaning that a component may have several instances running on different places so that it is possible to apply techniques like load balancing to improve the performance dynamically. There might be several classes of VM $M = (m_1, m_2, \ldots, m_q)$, which are expressed in terms of CPU (m^{cpu}) , RAM (m^{ram}) and cost (m^{cost}), $\forall m \in M$. We define the allocation matrix $N_{n \text{ x } q}$ (Equation 1), in which each element $e_{ij} \in [0, \frac{cost^{max}}{m_j^{cost}}]$ corresponds to the number of VMs of class m_j allocated to component c_i, where $cost^{max}$ corresponds to the maximum cost (budget) for the application.

$$N = \begin{pmatrix} e_{11} & e_{12} & \ldots & e_{1m} \\ e_{21} & e_{22} & \ldots & e_{2m} \\ & \ldots & & \\ e_{n1} & e_{n2} & \ldots & e_{nm} \end{pmatrix} \quad (1)$$

We also define a configuration variable $W \in \{1, \ldots, m\}$ to express the chosen configuration. So, the constraint expressed in Equation 2 must hold. Basically, this constraint states that if a configuration k_l is chosen, there should not be any VM $e_{ij} = 0$ for all classes of VM m_j and for any component c_i, which is not part of configuration k_l ($c_i \notin S_l$).

$$W = l \Leftrightarrow \forall c_i \in C : c_i \notin S_l (\forall 1 \leq j \leq q (e_{ij} = 0)) \quad (2)$$

It is straightforward that the performance of one component service varies according to its demands the resource capacity allocated to it. Thus, for each component $c_j \in C$, we define a function $perf_j : (\lambda, \rho_j) \mapsto \mathbb{N}$ in terms of response time, where λ corresponds to the workload and ρ_j to the total amount of CPU allocated to component c_j. Finally, the QoS performance value Q^{perf} is determined by a utility function $u : \mathbb{N} \mapsto [0, 1]$ of the global response time. In other words, both Q^{spec} and Q^{perf} are normalized values that estimate the utility in terms of revenues of a given application configuration k running under a certain infrastructure configuration N. Indeed, the higher the response time the lower is the revenue. Similarly, the lower the specific QoS (QoS^{spec}), the lower is the revenue.

3.1.2 Constraints

We define a constraint stating that the application total cost cannot exceed $cost^{max}$ (Equation 3). The cost function $cost(N)$ is defined as percentage of $cost^{max}$ (Equation 4).

$$cost^{max} \geq \sum_{i=1}^{n} \sum_{j=1}^{q} e_{ij} * m_i^{cost} \quad (3)$$

$$cost(N) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{q} e_{ij} * m_i^{cost}}{cost^{max}} \quad (4)$$

3.1.3 Objective Function

The objective function is then defined as in Equation 5.

$$O_{N,W} = \max(\quad w_{perf} * Q_W^{perf}(\textstyle\sum_{\forall c_j \in S_W} perf_j(\lambda, \rho_j)) \\ + w_{spec} * Q_W^{spec} + w_{cost} * (1 - cost(N))) \quad (5)$$

Where $0 \leq w_{spec} + w_{perf} + w_{cost} \leq 1$, w_{spec}, w_{perf} and w_{cost} correspond to weights for Q^{spec}, Q^{perf} and $cost$, respectively.

3.2 Infrastructure Constraint Model

3.2.1 Variables and Domains

The infrastructure consists of a set of physical machines (PMs) expressed by vector $P = (pm_1, pm_2, \ldots, pm_p)$. pm_i^{cpu} and pm_i^{ram} defines respectively the CPU and RAM capacities of machine $pm_i \in P$. Let $V = (vm_1, vm_2, \ldots, vm_r)$ be the set of all VMs currently executing in the infrastructure. We define the placement matrix $H_{p \text{ x } r}$ (Equation 6), where $h_{ij} = 1 \iff vm_j$ is hosted by pm_i, $h_{ij} = 0$ otherwise, $1 \leq i \leq p, 1 \leq j \leq r$. Every time the IM receives a request for VMs from an application, it has to instantiate and place these VMs along with the existing ones. So, let $V' = (vm_1, vm_2, \ldots, vm_r, \ldots, vm_t)$ be the vector V concatenated with the VMs requested by an application. The IM handle this demand by producing another matrix $H_{p \text{ x } t}$ with the same characteristics of H, where $\forall 1 \leq j \leq r : h'_{ij} = h_{ij}$.

$$H = \begin{pmatrix} h_{11} & h_{12} & \ldots & h_{1r} \\ h_{21} & h_{22} & \ldots & h_{2r} \\ & \ldots & & \\ h_{p1} & h_{p2} & \ldots & h_{pr} \end{pmatrix} \quad (6)$$

3.2.2 Constraints

vm_j^{cpu} and vm_j^{ram} express the CPU and RAM capacities of VM $vm_j \in V$. The constraints 7 and 8 state that the CPU and RAM demands of VMs hosted in one PM should not exceed its CPU and RAM capacities.

$$pm_i^{cpu} \geq \sum_{j=1}^{t} h'_{ij} * vm_j^{cpu} : \forall 1 \leq i \leq p \quad (7)$$

$$pm_i^{ram} \geq \sum_{j=1}^{t} h'_{ij} * vm_j^{ram} : \forall 1 \leq i \leq p \quad (8)$$

3.2.3 Pricing

Let pm_i^{util} be the utilization rate of pm_i and $m_j \in M$ be the biggest VM class that fits in PM pm_i. If $pm_i^{util} < U_{min}$, for a period of time longer than τ, where U_{min} is an arbitrary minimum utilization value, one VM instance of class m_j is offered with a reduced price m_i^{disc}. We then redefine in Equation 9 the constraint presented in Equation 4.

$$cost(N) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{q}(e_{ij} - d_j) * m_i^{cost} + (d_j * m_i^{disc})}{cost^{max}} \quad (9)$$

Where d_j corresponds to the number of VM instances of class j with reduction of price. The pricing policy is useful to improve the synergy between AMs and IM. Indeed, prices with discount correspond to signals sent by IM willing to attract AMs to occupy a portion of the infrastructure that are not being fully utilized at a certain moment. For the IM, the objective is to improve the utilization of each PM and hence the energy efficiency of the data center.

3.2.4 Objective Function

Finally, the objective is to minimize the number of nodes necessary to host the VMs instances in V, as it can be seen in Equation 10.

$$O_H = min(\sum_{i=1}^{p}(u_i)), \text{ where } \begin{cases} 1, \exists vm_j \in H \mid h_{ij} = 1 \\ 0, \text{otherwise} \end{cases} \quad (10)$$

4. EXPERIMENTAL RESULTS

In this section, we present preliminary achievements obtained from simulation-based experiments. We have developed a Java-based simulator, which implements the Infrastructure and Application Managers. This simulator relies on Choco [22], a Constraint Programming Java library, to model and solve the optimization problems presented in Section 3. The next two subsections presents the simulation environment setup and the results obtained so far.

4.1 Simulation Setup

The experiments were performed on an Intel Core 2 Duo (2.53 GHz) computer with 4GB 1067 MHz DDR3 DRAM.

First, we evaluated a scenario with only one application. The application can be composed by a set of components, defined by vector $C = (comp_1, comp_2, comp_3)$. Vector $K = (k_1, k_2)$ states that there are two possible configurations, where $S_1 = (comp_1, comp_2)$ and $S_2 = (comp_1, comp_3)$. We define the specific QoS as $Q_1^{spec} = 1$ and $Q_2^{spec} = 0.85$ for configurations k_1 and k_2, respectively. It is noteworthy that those values are defined based on the application revenues running on each configuration. In this particular case, for the same amount of clients, we can state that configuration k_2 will have an income inferior than configuration k_1. Figure 2 presents the utility function (Figure 2 (a)), stating that $u(x) = 1$ iff $x \leq 600$ $u(x) = 0$ otherwise; the set of performance functions for each amount of resources allocated to components $comp_1$ and $comp_2$ (Figure 2 (b)) and $comp_3$ (Figure 2 (c)). The application has a budget (i. e. $cost^{max}$) of 22. Regarding the autonomic loop reaction times, we fixed the AM to trigger an analysis every $3s$, whereas for the IM it was fixed to $1s$. Finally, we weighted the QoS criteria and the cost as follows: $w_{perf} = 0.5$, $w_{spec} = 0.25$ and $w_{cost} = 0.25$.

The infrastructure is composed of 30 PMs $P = (pm_1, \ldots, pm_{30})$. $pm_i^{cpu} = 4GHz$ and $pm_i^{ram} = 4GB$, $\forall pm_i \in P$. Portions of those resource are made available in terms of VMs. There are two classes of VM $M = (m_1, m_2)$, where $m_1^{cpu} = 1GHz$, $m_1^{ram} = 1GB$, $m_1^{cost} = 0.18$, $m_1^{disc} = 0.1$, $m_2^{cpu} = 2GHz$, $m_2^{ram} = 1GB$, $m_2^{cost} = 0.27$ and $m_2^{disc} = 0.16$. Finally, the IM's reaction time was set to $20s$ and the parameters U_{min} and τ were set to 0.8 and $20s$, respectively.

We also evaluated our framework in a multi application scenario by launching four AMs, each one controlling a different instance of the same application as the one defined above. Although any kind of application with specific QoS and characteristics could have been used in this scenario, we decided, for the sake of clarity and simplicity, to reuse the same application definition. In this scenario we considered a four-times bigger data center in order to accommodate all four applications.

In order to determine the initial amount of VMs needed to host the application, it is necessary to given an arbitrary value for the workload variable (λ). In our case, we define the maximum capacity in terms of request/second the application can achieve with the given budget (i. e. 600 requests/sec).

4.2 Results

Figure 3 shows the number of PMs necessary to execute the application under a given workload (defined by the dotted curve) during a five minutes experiment execution. The other two curves represent: (i) the number of PMs when only one configuration (k_1) is considered; and (ii) when two configurations (k_1 and k_2) are considered. As it can be seen, there is a non-neglected energy savings.

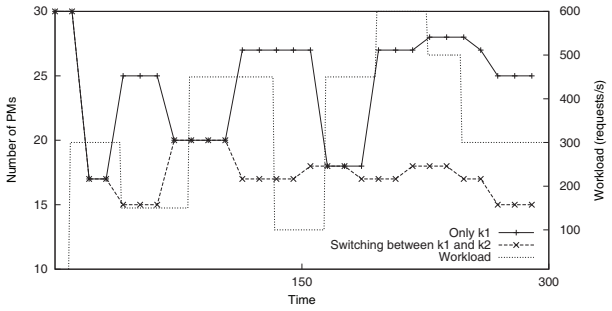

Figure 3: Resources consumed (in terms of PM) by the application under a certain workload.

The energy savings are important, but it is straightforward that when degrading the QoS less resource is needed and hence less energy is consumed. Figure 4, on the other hand

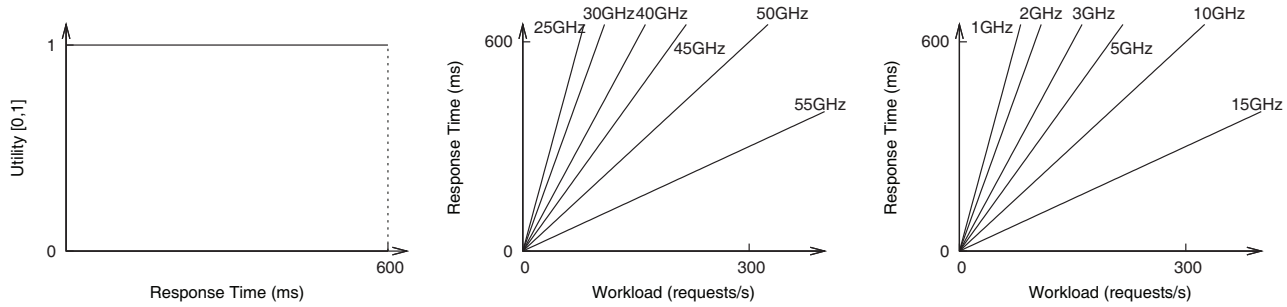

Figure 2: (a) The utility function; (b) The set of performance functions for components $comp_1$ and $comp_2$; and (c) The set of performance functions for components $comp_3$.

shows the ratio QoS per cost when considering only one configuration (k_1) and when considering the two configurations (k_1 and k_2). The ratio is used as a metric to compare the two approaches (considering only k_1 and considering k_1 and k_2) in terms of incomes (QoS) and expenses (cost). Figure 4 (d) shows when the application switches from one configuration to another during the execution. Figure 4 (a) shows the ratio Q^{perf} per cost. A significant difference is observed in favor of the approach that considers configuration switch. Indeed, when one application operates in a degraded mode (configuration with $Q^{spec} = 0.85$), it might require less resource for the same workload (i. e. to attend to the same amount of clients) in comparison to when it operates on a non-degraded mode (configuration with $Q^{spec} = 1$). Alternatively, the application operating on a degraded mode might be able to serve more clients with the same amount of resources than the application operating on a non-degraded mode. Concerning the Q^{spec} (Figure 4 (b)), we can also observe better results for the approach that allows to switch between configurations. Not surprisingly, this is also true when we compare the two approaches by considering the ratio of the average of both QoS criteria per cost (Figure 4 (c)).

With respect to the multi-application scenario, Figure 5 shows the IM's incomes per PM switched ON, when it applies (and when it does not) the discount policy presented in Section 3. The environment is composed of four applications running simultaneously, each one with a different workload (omitted due to space limitation). As it can be seen, with the discount policy, the ratio is up to 10% higher than without the discount policy, during the whole execution.

In conclusion, the synergy between AMs and IM can lead to significant energy savings. Furthermore, these savings may not impact negatively on the overall ratio revenues/cost.

4.2.1 Scalability

Finally, we evaluated the scalability of both constraint programming based optimization solutions. Figure 6 shows the execution time of the AM optimization algorithm based on the number of components and VMs allocated to each component. The algorithm takes approximately 2 seconds to find a solution for 10-component application, each component having up to 30 VMs. We believe that better results can be achieved by applying heuristics to select tree nodes in the search space during the execution. We plan to exploit this aspect as future works.

As previously mentioned, we rely on Entropy for the IM optimization problem. Although, it is perfectly is suitable for small and medium-sized data centers [8], we are aware that it does not scale for large-sized ones. To this end, we are currently investigating completely distributed approach for VM management whose objective is to be able to manage a large number of VMs and PMs in distributed manner. For further details, please see [20].

5. RELATED WORK

In this section, we discuss a selection of recent and relevant work about self-management on applications and cloud computing to address the problem of energy consumption in data centers. First, we briefly describe the state of the art on multi-level energy-aware approaches. Then, we provide some discussion on on the related works about the coordination of multiple autonomic loops.

5.1 Multi-level Energy-aware Approaches

Nguyen Van, Dang Tran and Menaud [23] proposed a Service Level Agreement (SLA)-based approach for cloud resources management running several applications at the same time. In their system, each application is profiled with its performance functions (based on the income workload and the amount of resources allocated to it) and a utility-based SLA. The objective is to determine the number of VMs necessary so that the utility is maximized and thereafter to pack those VMs into the minimum number of Physical Machines (PMs). This objective was split into two separate optimization problems that were modeled and solved by using Constraint Programming techniques. Our work extends this approach by allowing internal application adaptations.

In [2], an approach to improve the performance in multi-tier-based applications was proposed. It takes into account two levels of configuration (architectural and local) in order to improve the system performance. The architectural configuration means the cost in terms of physical machines and the local configuration corresponds to the maximum number of concurrent clients the servers can admit. The approach makes use of Mean Value Analysis queuing approach to model the performance (i. e. the latency and abandon rate) in each tier. Based on those parameters and on the

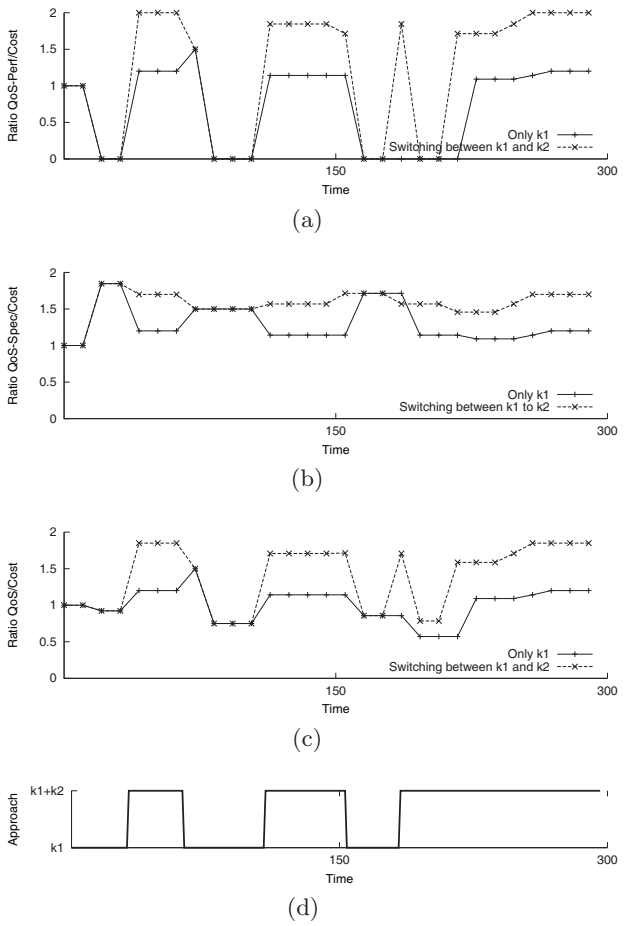

Figure 4: (a) Ratio Q^{perf} per cost; (b) Ratio Q^{spec} per cost; (c) Ratio Average QoS per cost; (d) Periods of time when the application operates on configurations k_1 or k_2

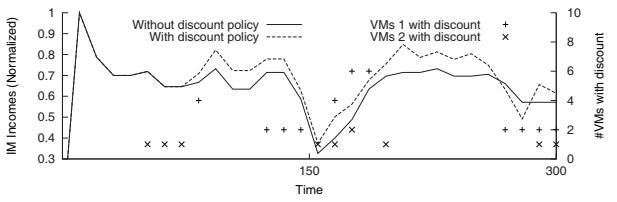

Figure 5: Ratio IM's incomes per PM switched ON.

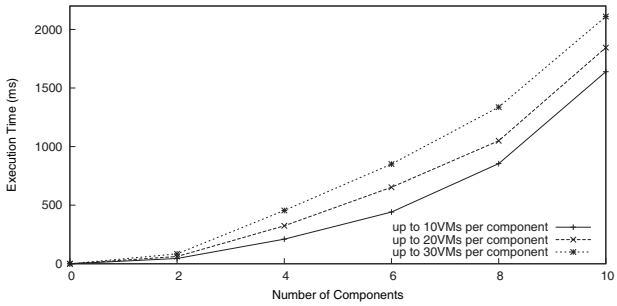

Figure 6: Application Manager Scalability Evaluation.

configuration cost, an objective function is provided. Two algorithms are provided: one to implement the model which is used to predict the latency, cost and abandon rate based on a given configuration and workload; and another one to find the optimal solution based on the objective function. The adaptation may occur in two levels: (i) degrade the application QoS by increasing the abandon rate (admission control); (ii) or by adding/suppressing PMs. Apart from the fact that we define our infrastructure in terms of VMs, we take into consideration that applications can internally be adapted. So, instead of controlling the QoS by admission control, we internally change the application to improve or decrease its QoS. In fact, that approach can be seen as a particular case of the proposed approach in this paper, where the QoS criterion considered is the availability.

In [19], the authors proposed an approach to optimize the energy consumption in a multi-application heterogeneous cluster environment. The objective is to determine the number of VMs needed for each application under a given workload in a way the overall energy consumption is minimized. Performance and power models are defined in terms of frequency and utilization rate of physical machines. The problem was modeled and solved by using Mixed-Integer Programming. Contrary to our work, the optimization problems (to determine the number of VMs and placing them) are performed in a single decision module. We believe that applications, having different kinds of workload may have different needs in terms of reconfigurations (different periods). Moreover, dealing with it as a single and more complex problem may lead to serious scalability problems.

SAFDIS [7] is a multi-level framework for self-adaptable distributed applications. The problem of adaptation is managed in a traversal way, from the application to the infrastructure, in the sense that applications and infrastructure are monitored and thereafter the adaptation takes place at the upper level (application) considering the current state of lower levels (infrastructure). For instance, a sensor may detect that an application is overloaded. The framework will then try to migrate the application from one resource to a more powerful one. Although the work proposes a cross-layered approach to adapt applications and infrastructure in function of their QoS, it is not the focus to deal with the energy issue.

BrownMap [24] is a methodology to enforce power budget in data centers. The objective is to cope with temporary reduction of power available in data-centers (brown-out). The authors proposed an approach based on VM live migration and VM resizing to adapt shared data-centers when brown-outs happen. Based on a power model and utility-based SLA specification, the methodology analyzes every

server (one-by-one) by performing VM resizing and/or VM replacement until a global power budget is met. In summary, the methodology aims at adapting the infrastructure to meet the power budget while minimizing utility drops. While the work focuses on the infrastructure adaptation to cope with power supply interruptions, our work relies on a strong interaction between layers. It allows the infrastructure to make applications aware about a possible resource shrink situation. As a consequence, applications are able to adapt themselves before the problem has happens.

5.2 Multi-control Loop Coordination Approaches

The issue of orchestrating autonomic managers has been addressed by IBM since 2005 [9] and the first interesting results have been proposed by J. Kephart and al. in order to achieve specific power-performance trade-offs [12]. The authors developed architectural and algorithmic choices allowing two managers to work together to act in accordance, resulting in power savings. This framework composed of two separate agents that manages performance and power. The former sends power-unaware information to the latter, which in turn manage the tradeoff between performance and power consumption. After receiving state information from the performance manager, which has its own power-unaware policies, the power manager tries to optimize the joint utility by applying its power policy to manage the trade-off between performance and power. The work makes use of Dynamic Voltage Frequency Scaling to implement its power management policies, which are related to a specific machine.

In [10], the authors propose an interface-based coordinated method for multiple applications and system layer. The idea is that only semantic-less numbers (i. e. integer numbers that can be increased or decreased) representing performance (QoS levels) and energy (e. g. processors P-States) are shared among applications and system. A coordination algorithm is provided to combine all applications needs in terms of QoS and energy requirements. Hence, the application has no information about the system power management modules, and similarly, the system has no details about the application performance levels. In order to trigger a reconfiguration process at the infrastructure level, all applications should agree and therefore that process may take too long to converge so the adaptation is no longer needed.

Ardagna et al. [1] proposed an autonomic approach to tackle with several energy-related sub-problems in two different control loops: short-term, for problems whose solutions have a low overhead; and long-term, for problem whose solutions have a high overhead. The two control loops share some decision variables, which ensure a better exploitation of the tradeoff energy costs and applications QoS. The work classifies the adaptations with regards to their complexities and overheads. For instance, the application adaptation and infrastructure adaptation (by DVFS mechanisms) are addressed in the same control loop, since they imply a low overhead. Also, the work combines the use of both DVFS and virtualization techniques. That may require a complex management of some VMs caused by changes on the processor frequency of the PM in which those VMs are hosted. Besides, our proposal takes into account the application internal reconfiguration in order to either reduce the energy footprint and/or meet the resources restrictions.

Coordinating multiple autonomic managers to achieve specific and common goal have been receiving a lot of attention in the last years. [16] identifies five different patterns of interacting control loops in self-adaptive systems where each pattern can be considered as a particular way to orchestrate the control loops. [6] goes further and proposes a collection of architectural design patterns addressing different classes of integration problems focusing on the possibly conflicting goals. [17] proposes a hierarchical model of control loops where a coordination manager orchestrates the other autonomic managers to satisfy properties of consistency. Finally, [25] extends MAPE-K loops with support for two types of coordination: intra-loop and inter-loop coordinations very close to ours. However, the implementation framework is dedicated to a self-healing use case and the contribution is weaved with the domain specific of traffic monitoring.

6. CONCLUSIONS AND FUTURE WORK

The popularization of new provisioning models like Cloud Computing has driven the recent increase of the energy consumption in IT infrastructures. Current approaches leverage autonomic computing to make cloud applications autoscalable with respect to the amount of resources necessary to maintain their Quality of Service (QoS). At the low level, virtualization techniques improve infrastructure utilization and consequently its energy efficiency.

In this article, we presented an approach that goes deeper in the adaptation process by taking into account application internals and their impact on the energy footprint, as well as the relationship between application and infrastructure providers. We argue for a software platform based on autonomic computing in which each application has its own control loop whose objective is to find the best trade-off between QoS and the amount of resources needed. Another control loop at the infrastructure layer takes care of applications requests while optimizing their placements. To this end, we provide a Constraint Programming-based model for these Constraint Satisfaction Problems. Finally, we showed the feasibility of our approach by performing simulation-based experiments.

We are currently working to set up a real infrastructure environment based on Grid'5000 [1] test-bed. The objective is to deploy real component services on a real infrastructure in order to validate our approach in a realistic environment. In parallel, we are also working on a synchronization mechanism to ensure that when a reconfiguration takes place at application level, all the underlying operations should be synchronized in order to avoid component disruption. A first result is presented in [5], where we proposed an approach to address the problem of synchronization and coordination of several autonomic loops in cloud environments. Currently, we are relying on transactional models for component-based applications previously proposed in [15] to incorporate transaction management.

We are also considering to take into account the overhead of reconfigurations. At the application layer, we believe that

[1] http://www.grid5000.fr

component operations (start, stop, bind and unbind) are neglected with respect to infrastructure costs (VM creation). So, the cost for creating a VM can be easily determined and incorporated in the AM constraint model. In addition, the cost referred to unavailability of application during reconfiguration procedures should also be taken into account so we can have a more realistic model. For actions triggered by the IM (e. g. PM switch-on/off), we rely on the reconfiguration cost model proposed in [8].

Finally, we are aware that the energy consumption within one machine can vary significantly and hence more fine-grained power models would help to achieve better results. Several works [13, 11, 3] propose more detailed power models that deal with power consumption in a per process / application / VM basis. Therefore, the power variations within a single PM can be more easily detected so the power models can be improved.

7. REFERENCES

[1] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-aware autonomic resource allocation in multi-tier virtualized environments. *IEEE Transactions on Services Computing*, 99(PrePrints), 2010.

[2] J. Arnaud and S. Bouchenak. Moka : Optimisation de services internet multi-étagés. In *NOTERE*, Montreal, Canada, July 2009.

[3] A. E. H. Bohra and V. Chaudhary. *VMeter: Power modelling for virtualized clouds*, pages 1–8. Ieee, 2010.

[4] M. Comuzzi and B. Pernici. A framework for qos-based web service contracting. *ACM Trans. Web*, 3(3):1–52, 2009.

[5] F. A. de Oliveira Jr., R. Sharrock, and T. Ledoux. Synchronization of multiple autonomic control loops: Application to cloud computing. In *Proceedings of the 14th International Conference on Coordination Models and Languages (Coordination) – To appear.*, June 2012.

[6] S. Frey, A. Diaconescu, and I. Demeure. Architectural integration patterns for autonomic management systems. In *Proc. of the 9th IEEE International Conference and Workshops on the Engineering of Autonomic and Autonomous Systems (EASe 2012)*. IEEE, April 2012.

[7] G. Gauvrit, E. Daubert, and F. André. SAFDIS: A Framework to Bring Self-Adaptability to Service-Based Distributed Applications. In *36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pages 211–218, Lille, France, Sept. 2010.

[8] 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*, pages 41–50, New York, NY, USA, 2009. ACM.

[9] IBM. An architectural blueprint for autonomic computing. Technical Report June, 2005.

[10] A. Kansal, J. Liu, A. Singh, R. Nathuji, and T. Abdelzaher. Semantic-less coordination of power management and application performance. *SIGOPS Oper. Syst. Rev.*, March 2010.

[11] A. Kansal, F. Zhao, and A. A. Bhattacharya. Virtual Machine Power Metering and Provisioning. In *ACM Symposium on Cloud Computing*, 2010.

[12] J. O. Kephart, H. Chan, R. Das, D. W. Levine, G. Tesauro, F. Rawson, and C. Lefurgy. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *Proceedings of the Fourth International Conference on Autonomic Computing*, Washington, DC, USA, 2007. IEEE Computer Society.

[13] R. Koller, A. Verma, and A. Neogi. Wattapp: an application aware power meter for shared data centers. In *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, pages 31–40, New York, NY, USA, 2010. ACM.

[14] J. G. Koomey. GROWTH IN DATA CENTER ELECTRICITY USE 2005 TO 2010. Technical report, Analytics Press, 2011.

[15] M. Lèger, T. Ledoux, and T. Coupaye. Reliable dynamic reconfigurations in a reflective component model. In L. Grunske, R. Reussner, and F. Plasil, editors, *Component-Based Software Engineering*, volume 6092 of *Lecture Notes in Computer Science*, pages 74–92. Springer Berlin / Heidelberg, 2010.

[16] R. D. Lemos and al. Software Engineering for Self-Adaptive Systems : A Second Research Roadmap (Draft Version of May 20, 2011). Technical Report October 2010, 2011.

[17] S. Mak-Karé Gueye, N. de Palma, and E. Rutten. Coordinating energy-aware administration loops using discrete control. In *Proc. of the 8th International Conference on Autonomic and Autonomous Systems (ICAS 2012)*, March 2012.

[18] S. P. Mirashe and N. V. Kalyankar. Cloud computing. *Communications of the ACM*, 51(7):9, 2010.

[19] V. Petrucci, E. V. Carrera, O. Loques, J. C. B. Leite, and D. Moss. Optimized Management of Power and Performance for Virtualized Heterogeneous Server Clusters, 2011.

[20] F. Quesnel and A. Lèbre. Cooperative Dynamic Scheduling of Virtual Machines in Distributed Systems. In *Euro-Par 2011 Workshops*, volume 7156 of *Lecture Notes in Computer Science*, pages 457–466, Bordeaux, France, Aug. 2011. Springer-Verlag.

[21] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

[22] C. Team. choco: an open source java constraint programming library. Research report 10-02-INFO, Ecole des Mines de Nantes, 2010.

[23] H. N. Van, F. D. Tran, and J.-M. Menaud. Performance and power management for cloud infrastructures. *IEEE International Conference on Cloud Computing*, 2010.

[24] A. Verma, P. De, V. Mann, T. K. Nayak, A. Purohit, G. Dasgupta, and R. Kothari. Brownmap: Enforcing power budget in shared data centers. In *Middleware*, pages 42–63, 2010.

[25] P. Vromant, D. Weyns, S. Malek, and J. Andersson. On Interacting Control Loops in Self-Adaptive Systems. In *Proc. of the 6th International Symposium on Software Engineering for Adaptive and*

Self-Managing Systems, pages 202–207. ACM, 2011.

[26] X. Wang and Y. Wang. Coordinating power control and performance management for virtualized server clusters. *IEEE Transactions on Parallel and Distributed Systems*, 22(2):245–259, 2010.