Meryn: Open, SLA-driven, Cloud Bursting PaaS

Djawida Dib[1] Nikos Parlavantzas[2] Christine Morin[1]
[1]INRIA - [2]INSA, Campus de Beaulieu, 35042 Rennes cedex, France
{djawida.dib, nikos.parlavantzas, christine.morin}@inria.fr

ABSTRACT

PaaS (Platform as a service) systems are revolutionizing the way modern applications are developed and hosted. Current PaaS offerings provide limited support for managing SLAs (Service Level Agreements) that constrain application quality properties, such as response times. To overcome this limitation, PaaS systems should be capable of dynamically adjusting resource allocations to meet provider objectives. At the same time, PaaS systems should retain their extensibility to host diverse application types. In this paper we propose Meryn, an open, SLA-driven, PaaS system that aims at maximizing the provider profit and providing SLA guarantees to diverse application types. Meryn provides support for cloud bursting and applies a decentralized protocol for selecting resources to run applications. This protocol tries to minimize the cost of running applications without affecting their agreed quality properties. We implemented a prototype of the Meryn system and performed a preliminary evaluation. The results show that the effectiveness of Meryn in optimizing the provider profit is very promising.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Cloud computing

Keywords

Cloud Computing, Platform as a Service (PaaS), Service Level Agreement (SLA), Virtual Cluster, Cloud Bursting.

1. INTRODUCTION

The PaaS (Platform as a Service) cloud model is revolutionizing the way applications are developed and managed. PaaS customers deploy their applications on a complete hosting environment, which is delivered as a service by PaaS providers. This environment typically relies on virtualized resources owned by the provider or leased from public clouds on demand. Customers are shielded from managing the underlying resources, thus drastically simplifying application development and administration.

Interactions between the PaaS customers and providers are governed by Service Level Agreements (SLAs), specifying the obligations of each party as well as associated payments and penalties. A major limitation of current PaaS offering is that they provide no QoS (Quality of Service) guarantees to their customers. Indeed, current PaaS SLAs constrain exclusively application resource usage (e.g., number of instances, memory size, bandwidth), rather than application QoS properties, such as response time or throughput. PaaS customers are thus burdened with ensuring QoS properties for their applications, which limits the value of PaaS systems.

Integrating QoS support in PaaS requires an automated resource management solution that dynamically distributes the private and leased resources among customers, taking into account the SLAs and adapting to changing workloads, resource capabilities, and resource prices. An important challenge in developing such a solution comes from the need to support extensibility with regard to programming frameworks. A great variety of programming frameworks have proven to be widely useful and are currently part of PaaS offerings. Examples include web application frameworks, MapReduce, batch frameworks, and task farming frameworks. These frameworks employ sophisticated scheduling and resource management policies, optimized to support the quality objectives of framework-based applications. The challenge therefore lies in integrating independently-developed frameworks with heterogeneous objectives in order to meet overall provider objectives, such as profit maximization.

Consider, for example, a resource management solution in which most decisions are delegated to a central PaaS component. Whereas such a solution could easily satisfy provider objectives, it would require extensive modifications to existing and new frameworks, hurting extensibility. Moreover, the solution could potentially involve prohibitive communication and computation costs. Conversely, a solution that leaves the programming frameworks almost unmodified could easily support extensibility but at the cost of making it difficult to satisfy PaaS-wide objectives.

In this paper, we present Meryn, an open SLA-driven PaaS system that supports optimizing the provider profit while being extensible with respect to programming frameworks. Meryn relies on a decentralized scheme to control the dynamic distribution of private resources among programming frameworks and to manage bursting to public clouds, when necessary. Resource distribution is guided by the SLAs

ORMACloud'13, June 17, 2013, New York, NY, USA.

of hosted applications and resource prices. We have implemented a prototype of Meryn and demonstrated its effectiveness in meeting the profit objective.

The remaining of this paper is organized as follows. In Section 2 we discuss the design principles of Meryn. In Section 3 we present the architecture of Meryn. In Section 4 we introduce the cost optimization policy used in Meryn. In Section 5 we present our prototype and discuss preliminary evaluation results. In Section 6 we discuss related work. Finally we conclude in Section 7.

2. APPROACH

Three key design decisions were made in designing the Meryn platform. The first decision concerns how the PaaS-owned physical resources are shared. Meryn partitions the physical machines into VMs with the well-known advantages of virtualization in terms of flexible resource control and isolation. Each VM is then assigned to a single programming framework (e.g., Hadoop, Torque) managing applications of a particular type (e.g., MapReduce, batch). Assigning a VM to a single framework, rather than multiple frameworks, simplifies application deployment and allows integrating frameworks that assume exclusive control of a machine.

The second decision concerns the division of resource management responsibilities between the frameworks and the PaaS system. Meryn adopts a decentralized architecture in which the frameworks collaborate though exchanging resources with the aim of increasing the provider profit. Compared to a centralized architecture, the advantage of this approach is that it imposes minimal changes on the frameworks, thus facilitating the extensibility of the PaaS system. Indeed, the frameworks continue to take most of the resource management decisions regarding their hosted applications, taking advantage of their application type-specific knowledge to better satisfy SLA objectives.

The final decision concerns the granularity of the resources exchanged among frameworks. Meryn adopts a coarse-grained approach in which the frameworks exchange VMs of fixed capacities. Although a fine-grained approach would improve infrastructure utilization, the benefit of the current approach is that it corresponds to the one supported by current IaaS offerings. As a consequence, private VMs and VMs dynamically leased from public clouds can be managed in a similar way, which greatly simplifies the decentralised protocol for resource exchange.

3. SYSTEM ARCHITECTURE

3.1 Overview of Meryn System

The overall architecture is illustrated in Figure 1, where private resources consist of a fixed number of VMs shared between multiple elastic Virtual Clusters (VCs). We define a VC as a set of VMs running on private resources plus possibly some VMs provisioned from public clouds. Each VC is associated with a specific application type (e.g., batch, MapReduce) and is managed by a specific programming framework (e.g., Torque, Hadoop). The initial division of resources among VCs could be fair or based on past traces. Then, according to specific policies VCs may exchange resources between each other. The overall resource management is fully decentralized where each VC autonomously manages its own resources and decides when to be extended with public cloud resources.

Users do not see what is under the hood and they do not have to care about it. They submit their applications through a common and uniform interface, whatever the type of their applications.

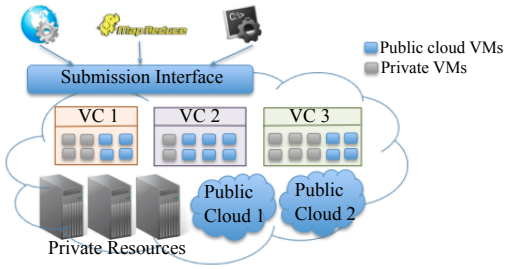


Figure 1: Architecture Overview

3.2 Meryn Components

Figure 2 shows the main components of the Meryn system, which consists of a *Client Manager*, a *Cluster Manager* for each VC, a set of *Application Controllers* one for each submitted application and a *Resource Manager*.

Most programming frameworks consist of one master daemon and a set of slave daemons. In Meryn, we run the master daemon separately on a private VM that we call *Master VM*, and the slave daemons either on private VMs or public cloud VMs and we call them *Slave VMs*. On the master VM, we deploy the corresponding Cluster Manager and instantiate an Application Manager for each application running on the VC, while we use the slave VMs only to run the applications. In the following we describe each Meryn component.

- **Client Manager:** it is the entry point of the system, and it provides users with a uniform submission interface. The Client Manager is responsible for receiving submission requests and transferring them to the corresponding Cluster Manager. It also enables users to get the results of their applications. Meryn may have several Client Managers in order to avoid a potential bottleneck, which could happen in peak periods.

- **Cluster Manager:** it consists of two parts, a generic part and a framework-specific part. The generic part is the same for all Cluster Managers and consists in managing resources and deciding when to horizontally scale up and down. More precisely, the Cluster Manager has to decide when to release resources to other VCs, when to acquire resources from other VCs and when to rent resources from public clouds. The framework-specific part differs from one Cluster Manager to another and depends on the hosted type of applications. This part consists in proposing SLAs and negotiating them with users through the Client Manager. It consists also in translating users submission requests to submission requests compatible with the corresponding programming framework.

- **Application Controller:** it is responsible for monitoring the execution progress of its associated application and the satisfaction of its agreed SLA.

- **Resource Manager:** it is responsible for the initial system deployment and for transferring VMs from one VC to another. The Resource Manager interacts with a VM management system, such as Nimbus [2], OpenNebula [13] or Snooze [6].

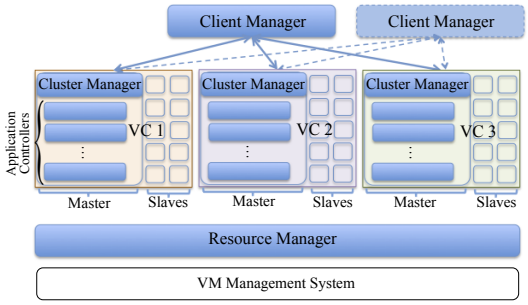


Figure 2: Meryn Components

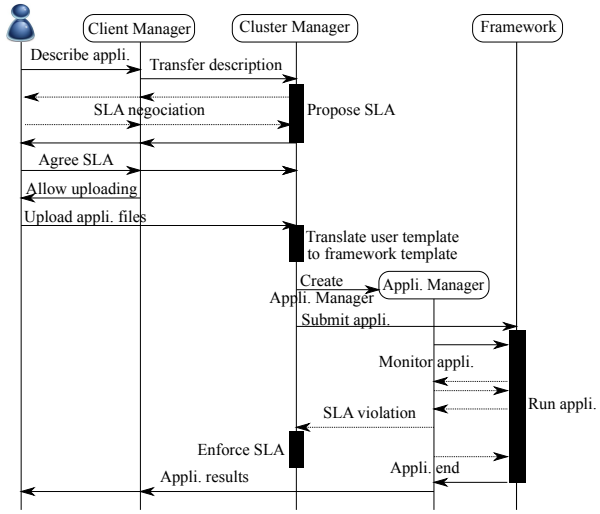


Figure 3: Application Submission Process

3.3 Application Submission

The application submission process operates as shown in Figure 3. First, the user contacts the Client Manager and uses its submission interface to describe her application. The application description consists of its characteristics in terms of resource consumption and its requirements in terms of software dependencies. The submission interface provides a standardized template to specify the application type and to describe the application whatever is its type. Based on the application type the Client Manager determines its corresponding VC and transfers the application description template to the corresponding Cluster Manager. Then, the Cluster Manager proposes SLA terms and negotiates them with the user. Once both the Cluster Manager and the user agree on one SLA contract, the Client Manager lets the user upload the executable file and the possible input data of her application toward the corresponding VC.

Afterwards, the Cluster Manager translates the application description template to another template compatible with its programming framework. Then the Cluster Manager launches a new Application Controller instance and submits the application to the framework. The Application Controller monitors the progress of its application and checks the satisfaction of its SLA contract until the end of its execution. If during the application execution, the Application Controller detects a violation of the SLA contract, it informs the Cluster Manager. The Cluster Manager proceeds to address the SLA violation according to specific policies that are not treated in this paper. When the application execution ends, the Client Manager provides a way for the users to retrieve the results from the Cluster Manager.

3.4 Exchanging VMs between VCs

The VM exchange between two VCs, a *source VC* and a *destination VC*, occurs when the destination VC requires additional VMs and the source VC can provide them. The VMs exchange procedure operates as follows. First, the Cluster Manager of the source VC selects the list of VMs to remove and removes them from its programming framework. Then, it requests the Resource Manager to shutdown the VMs. Once this operation ends, the Cluster Manager of the source VC informs the Cluster Manager of the destination VC that the VMs are available. The Cluster Manager of the destination VC requests the Resource Manager to start new VMs with the disk image of its corresponding framework. Once the Resource Manager starts the VMs, the Cluster Manager of the destination VC configures them and adds them to the framework resources.

3.5 Cloud Bursting

We enable the cloud bursting of a VC when the other VCs cannot provide it with the VMs it requires. Before adding cloud VMs to VCs, we first save the VM disk images in the different clouds that may be used. Note that, for each framework there is a customized VM disk image that contains all the necessary software and libraries. When a VC requires a cloud VM, its Cluster Manager sends a request to a public cloud. Once the cloud VM is created, the Cluster Manager gets its IP address and performs some configuration on it such as starting the framework daemons. Depending on the framework, the Cluster Manager may perform additional configuration on the other VMs of the cluster for updating them. Finally, when the VC finishes using the cloud VM, its Cluster Manager first removes the VM from the framework, then asks the public cloud to stop the VM.

4. COST OPTIMIZATION

We assume that a user pays for running an application in Meryn. If this payment is the same whether the application runs on private resources or public cloud resources, then to maximize the platform profit we should minimize the cost of running the application; namely, we should favor the use of private resources. However, when a new application arrives and all the private resources are used by other applications, it is difficult to know if it is better to overload the private resources or to rent public cloud resources. To address this issue, we have defined a *resource selection protocol* that estimates the cost of the different options a framework has to run a new application. This protocol is applied in a decentralized way where each Cluster Manager has its own way to compute the cost for providing resources for the new application. The cost computation method depends on the

applications performance model and SLA. We call this cost a *bid*. In the next subsections we first present our resource selection protocol. Then, we illustrate how the cluster bid may be computed in the case of a batch framework.

4.1 Resource Selection Protocol

The objective of the resource selection protocol is to find the VMs with the cheapest cost to run a new application. This protocol is inspired by decentralized, auction-based algorithms such as [10]. The protocol operates as shown in Algorithm 1. When a new application is submitted, according to its defined SLA its corresponding Cluster Manager, called here *local cluster manager*, knows how many VMs the application requires. The local cluster manager needs to decide among executing the application on its own available private VMs (*local-vms*), obtaining VMs from other VCs (*vc-vms*), or renting the VMs from public clouds (*cloud-vms*). To take this decision, the local cluster manager has five options.

In the first option, the private available VMs of its VC are sufficient for running the new application. Thus, the new application runs using the local-vms. Otherwise, the local cluster manager invites all the other Cluster Managers to propose a bid for the required VMs. The Cluster Managers propose a bid equal to zero if they have unused VMs to give to the local cluster manager. Otherwise, they calculate their bid as the estimated change in their running applications revenues arising from reducing their used VMs.

In the second option, at least one of the proposed bids is equal to zero. Thus, the local cluster manager gets the VMs from the corresponding VC and runs the new application using the vc-vms. Otherwise, the local cluster manager (1) also calculates its local bid in the same way as the other Cluster Managers, (2) gets the smaller bid proposed by the other Cluster Manager, and (3) requests a set of public clouds their current market VM prices and gets the cheapest cloud VM price. The local cluster manager compares all the bids and according to the lowest bid determines its action.

In the third option, the local bid is the lowest bid. Thus, the number of the used VMs by the locally running applications is reduced and the new application runs using the local-vms. In the fourth option, the smaller VC bid is the lowest bid. Thus, the number of the used VMs by the running applications in the corresponding VC is reduced and the new application runs using the vc-vms. Finally, in the fifth option the cheapest cloud VM price is the lowest bid. Thus, the new application runs using leased cloud-vms from the corresponding public cloud.

The main assumption of this protocol is that Cluster Managers can calculate revenue changes of their running applications when the number of their used resources is reduced. This calculation is based on the price functions defined in the SLA contract and can exploit application type-specific performance models.

Algorithm 1 Resource Selection Protocol

Input: New application (*app.vms*)
Output: VMs to run the application
  if *local_vms* $\geqslant$ *app.vms* **then**
    Run *application* on *local_vms*
  else
    Request all Cluster Managers to propose an *bid*
    vcs_list_bids = proposed *bids*
    if ($\exists$ *bid* $\in$ *vcs_list_bids*, *bid* = 0) **then**
      Get VMs from the corresponding VC
      Run *application* on *vc_vms*
    else
      Propose *local_bid*
      vc_bid = the smaller *bid* in *vcs_list_bids*
      Request a set of public clouds their current VM prices
      cloud_bid = the cheapest *cloud_vms* price;
      min_bid = *get_min*(*cloud_bid*, *vc_bid*, *local_bid*)
      if *min_bid* = *local_bid* **then**
        Reduce the used VMs by locally running apps
        Run *application* using *local_vms*
      else if *min_bid* = *vc_bid* **then**
        Reduce the used VMs by the running apps in the corresponding VC
        Get *vc_vm*
        Run *application* using *vc_vms*
      else
        Rent *cloud_vms*
        Run *application* using *cloud_vms*
      end if
    end if
  end if

4.2 Batch Framework Example

To illustrate the calculation of revenue changes, we study the case of a batch framework. In the next subsections, we first define an SLA function for batch applications. Then we propose a method for calculating the batch cluster bid.

4.2.1 Service Level Agreement (SLA)

To define an SLA contract for a batch application, its corresponding user should describe its characteristics and requirements. The application characteristics may be information about its consumption of CPU, memory, storage and network resources. For parallel applications it may also be the number of its processes. The application requirements may be specific resources or software dependencies. Based on the application description the batch Cluster Manager proposes to the user two SLA metrics: *the deadline* and *the price*. The deadline is the overall time to run an application and give results to the user. The price is the amount of money paid by the user in order to run her application.

For the SLA negotiation, the Cluster Manager provides the user with a set of pairs (deadline, price) and lets her choose one of them. If the user does not agree with any proposed pairs she may impose one of the SLA metrics. If the user has budget constrains, she may impose a price and the Cluster Manager gives the corresponding deadline. Otherwise, if the user's application is urgent, she may impose a deadline and the Cluster Manager gives the corresponding price. If the user does not agree with the metric given by the Cluster Manager, she may change the value of the metric that she proposed and launch a new negotiation round, and so on until she agrees with the two metrics. If the Cluster Manager exceeds the agreed deadline, the price will be reduced proportionally to the delay.

On the Cluster Manager side the computation of the deadline and the price is based on some background assumptions. First, we configure the batch framework scheduler so that it attributes a number of VMs to each single application. Then, we assume that the batch Cluster Manager may deduce the application execution time based on its dedicated

number of VMs and vice versa. Based on the application execution time and the used VMs, the batch Cluster Manager computes the deadline and the price of the application. The deadline is the sum of the execution time and the required time for processing the application submission. The price is the product of the execution time, the number of the used VMs and the VM price fixed by the platform provider. We assume that the VMs are charged according to the execution time instead of a per hour charging as it is currently done in public clouds like Amazon EC2 [1]. Initially we define the deadline and the price as follows:

$$deadline = execution\_time + processing\_time \quad (1)$$

$$price = execution\_time * nb\_vms * vm\_price \quad (2)$$

We differentiate here between the VM price for users and the VM cost for the platform provider. The VM cost depends on the VM being executed on top of private resources or on top of a public cloud. To limit the platform losses the VM price should be greater or equal to the cost of the public cloud VM.

If the system exceeds the agreed deadline, a *delay penalty* is computed proportionally to the delay and deduced from the initial price. The delay penalty may be bounded to a maximum value in order to limit the platform losses. We define the delay penalty as follows:

$$delay\_penalty = (delay * nb\_vms * vm\_price) \div N, \ N > 0 \quad (3)$$

Note that the N value may be fixed by the provider or determined during the SLA negotiation. This value determines how fast the delay penalty will be high. As an example, we assume that the delay is equal to the execution time. With N=1 the delay penalty will equal the price, which means that the provider revenue will be zero and the user will pay nothing for the run of her application. While with N=2 the delay penalty will be the half of the price, which means that the provider revenue will be halved and the user will pay the half of the initial price. To generalize, we can say that a high N value is more advantageous for the provider while a low N value is more advantageous for the user.

4.2.2 Bid Computation

Algorithm 2 shows the method we propose to compute the bid of a batch VC. The bid computation request requires two parameters: a number of VMs and a duration. The duration represents the period during which the VMs are used and possibly given back. When the batch VC receives a new bid computation request from another VC (*requester VC*), its Cluster Manager first checks its available VMs. If the available VMs are sufficient for the request, the Cluster Manager replies with a bid equal to zero because it may provide the VMs without any particular cost. Otherwise, the Cluster Manager considers suspending separately each application running on its VC during the requested duration, and estimates for each application the potential loss of revenues that may results. Then, the Cluster Manager proposes a bid based on the estimated potential loss of revenues. The loss of revenues represents the cost for the platform provider to suspend an application during the requested duration. This supposes that if the batch VC provides VMs after suspending an application, it expects the requester VC to give back the VMs before the end of the requested duration.

To compute the potential loss of revenues, the Cluster Manager selects only the running applications that holds a number of VMs greater or equal to the requested VMs, and computes the possible cost of their suspension. This cost consists of a minimal suspension cost and a possible delay penalty. The minimal suspension cost may be the cost of keeping the data used by the application in storage during the lending duration, or the possible extension of the application execution time caused by the application suspension. The delay penalty is computed based on the possible delay caused by the application suspension.

To compute the possible application's delay, we compute the application's spent time, progress time, finish time and free time (see Fig. 4). The application's spent time is the duration that the application spent in the system, from the submission time until the current time. The application's progress time is the current execution duration of the application, more specifically it is the difference between the current time and the time the application started actually running. The application's finish time is the remaining time to the end of the execution of the application, more specifically it is the difference between the predicted application execution time and the application progress time. The application's free time is the margin between the deadline and the predicted end of the application's execution.

If the application's free time is longer than the requested duration, the cost of suspending the application is only the minimal suspension cost. Otherwise, the cost of suspending the application is the sum of the minimal suspension cost and the delay penalty. The application's delay is estimated as the difference between the requested duration and the application's free time. Then, based on the estimated delay we compute the delay penalty according to equation 3. Once the suspension cost of all the selected applications is calculated, we consider the smallest estimated cost as the VC bid. Finally, the Cluster Manager sends the estimated bid to the requester VC.

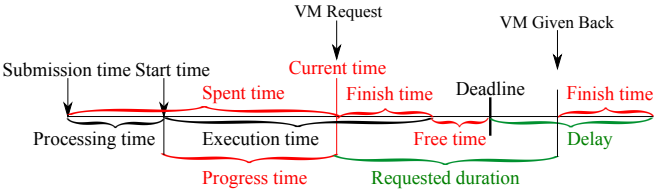


Figure 4: Application Times

5. EVALUATION

5.1 Implementation

To evaluate the presented architecture, we use two clusters on two different sites. One cluster provides the private resources and the second cluster represents the public cloud. Both clusters run Snooze VM manager software, one instance on each cluster. On top of the private Snooze we implemented the Meryn prototype in about 3,000 lines of shell script. The prototype supports batch and MapReduce applications using respectively Oracle Grid Engine OGE 6.2u7 and Hadoop 0.20.2 frameworks. We also implemented the bid computation algorithm for batch applications in about 300 lines of Java code. To evaluate the resource selection

Algorithm 2 Bid Computation for Batch VC

Input: *nb_vms* , *duration*.
Output: *bid*.

```
if available_vms ⩾ nb_vms then
  bid = 0
else
  for app ∈ list_running_apps do
    if app.vms ⩾ nb_vms then
      app.spent_t = current_t − app.submit_t
      app.progress_t = current_t − app.start_t
      app.finish_t = app.exec_t − app.progress_t
      app.free_t = app.deadline − (app.spent_t + app.finish_t)
      if app.free_t ⩾ duration then
        suspension_cost = app.min_suspension_cost
      else
        app.delay = duration − app.free_t
        Compute delay_penalty as in equation 3
        suspension_cost = delay_penalty + app.min_suspension_cost
      end if
      Add suspension_cost to list_costs
    end if
  end for
  bid = minimal suspension_cost in list_costs
end if
Send bid to the requester VC
```

protocol, we deployed two batch frameworks on the private resources. We compared Meryn with a *static approach* that statically partitions the virtual clusters and enables them to be extended only with public cloud resources

5.2 Environment

We carried out our experiments using the Grid'5000 experimental testbed [3]. We separately deployed the Meryn prototype and the static approach, each one on 9 nodes of the parapluie cluster of the Rennes site that consists of 40 HP Proliant DL165 G7 nodes supplied with 2 AMD Opteron(tm) 6164 HE processors (each with 6 cores at 1.7 GHz), 48 GB of memory, and Gigabit Ethernet network interfaces. We defined a public cloud on 12 nodes of the edel cluster of the Grenoble site that consists of 72 Bullx Blade B500 nodes supplied with 2 Intel Xeon E5520 processors (each with 4 cores at 2.27 GHz), 24 GB of memory, and Gigabit Ethernet network interfaces.

We used a VM instance model similar to the Amazon EC2 medium instance[1] that consists of 2 CPUs and 3.75 GB of memory, in private resources and in the public cloud. The VM hosting capacity in private resources was fixed to 50 VMs. We shared fairly the VMs between the two batch VCs, 25 VMs for each VC, in both Meryn and the static approach. We assume that the VM hosting capacity in the public cloud is infinite.

5.3 Workload

For a preliminary evaluation, we ran a synthetic workload consisting of 65 applications submitted at a fixed inter-arrival time of 5s, 50 applications submitted to the first batch VC (VC1) and 15 applications submitted to the second batch VC (VC2). For simplicity reasons, we ran each application on only one VM. The batch application we have used is the Pascal example, provided with the OGE framework, with a depth equal to 600. The measured execution time of the batch applications is about 1550s on a private VM and about 1670s on a cloud VM. We believe that this difference is because the processor speeds are not the same in the two clusters.

[1]http://aws.amazon.com/ec2/instance-types/

Table 1: Processing Time Measurement

Case	Processing time [s]
local-vm	7~15
vc-vm	40~58
cloud-vm	60~84
local-vm after suspension	10~17
vc-vm after suspension	60~68

To compute the application deadline we use as execution time the measured cloud execution time, and for the processing time we measured the required time to submit an application in all the possible cases (see Table 1). The maximum value (84s) is used as the processing time for all the submitted applications. In order to compute the cost of running the applications, we assumed that the cost of a public cloud VM is twice the cost of a private VM. More specifically we set the private VM cost to 2 units and the cloud VM cost to 4 units.

5.4 Results

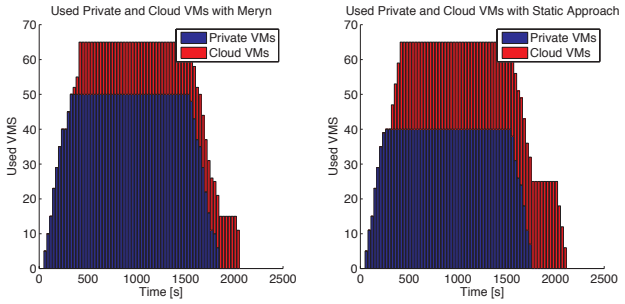
Figure 5(a) and Figure 5(b) show the used private and public cloud VMs with respectively Meryn and the static approach, during the execution of the synthetic workload. The comparison of these two figures clearly shows that Meryn uses less cloud VMs than the static approach, while the workload completion time is almost the same, 2021s with Meryn and 2091s with the static approach. In this experiment, the number of the used cloud VMs was up to 25 VMs in the static approach while it was only 15 VMs in Meryn. We noticed that in the static approach VC1 have used 25 private VMs and 25 cloud VMs to run its 50 applications, while VC2 have used 15 private VMs to run its 15 application and its remaining 10 private VMs were left unused. In Meryn the VC2 instead of keeping its 10 private VMs unused, it transferred them to the VC1. Thus, VC1 have used 25 private VMs, 10 VC2 VMs and 15 cloud VMs to run its 50 applications. In this experiment scenario, no application suspension was performed because the cost of suspending an application was higher than running the last applications on the cloud VMs.

Figure 6(a) compares the overall workload completion time and the average execution time of applications (application submitted on VC1, applications submitted on VC2 and all applications together) between Meryn and the static approach. The average execution time of the VC2 applications is almost the same, 1518s with Meryn and 1514s with the static approach because with both the applications were executed on private VMs. The average execution time of the VC1 applications is 3,33% better with Meryn compared to the static approach because with Meryn less applications were executed on cloud VMs. Therefore, the average execution time of all the applications and the overall workload

completion time are respectively 2,57% and 3,34% better with Meryn compared to the static approach.

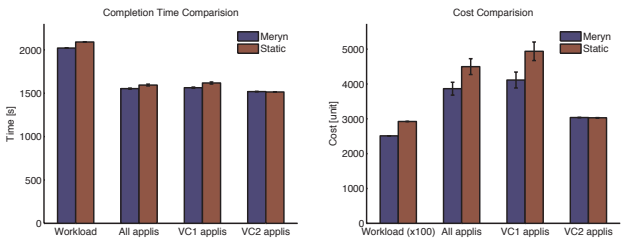
Figure 6(b) compares the overall workload cost and the average cost of applications (application submitted on VC1, applications submitted on VC2 and all applications together) between Meryn and the static approach. The average cost of the VC2 applications is almost the same, 3037 units with Meryn and 3029 units with the static approach because the execution time and execution location of these applications were similar. The average cost of the VC1 applications is 16,72% better with Meryn compared to the static approach because Meryn used less cloud VMs than the static approach and the cloud VM cost was set two times more than the private VM cost. Therefore, the average cost of all the application and the overall workload cost are both 14,07% better with Meryn compared to the static approach.

In this experiment the deadline of each application was satisfied with both Meryn and the static approach.



(a) (b)

Figure 5: The proportion of the used private and cloud VMs in (a) Meryn and (b) the Static Approach.



(a) (b)

Figure 6: Comparison of (a) the overall workload completion time and the average execution time of applications, and (b) the overall workload cost and the average cost of applications.

5.5 Discussion

The results have shown that Meryn helps decreasing the use of the cloud VMs without impacting the completion time of the applications. Consequently, the cost of running all the workload applications with Meryn was reduced by 41158 units compared to the static approach. As the deadlines of all the applications were satisfied the revenues with Meryn and the static approach were equal. Thus, the platform provider profit was higher with Meryn compared to the static approach.

We believe that the observed results are very promising, and we will perform more experiments in the future with workloads representative of real data centers workloads.

6. RELATED WORK

We survey the related work in three fields: SLA, PaaS and virtual clustering.

6.1 SLA (Service Level Agreement)

Using the QoS parameters or more recently the SLA contracts as constraints for optimizing resource allocation is a challenging problem. This problem attracted a lot of attention from the research community. In the following we provide a review of some relevant prior work.

In [4] the authors proposed a QoS-based workflow scheduling algorithm in utility grids. The objective of the algorithm is to create a schedule that minimizes the total execution cost of a workflow, while meeting a user-defined deadline for the total execution time. The algorithm tries to schedule a critical path of the workflow such that it completes before the user's deadline and the execution cost is minimized. Then it finds a partial critical path to each scheduled task on the critical path and executes the same procedure in a recursive manner. In [7] the authors considered the SLA-based resource allocation problem for multi-tier applications in cloud computing. The objective of this work is to optimize the total profit from the SLA contracts and lost from operational cost. A solution is proposed based on generating an initial solution inspired from the profit upper-bound and resource consolidation technique based on the force-directed search. In [16] a resource allocation algorithm for SaaS (Software as a Service) providers is proposed to minimize infrastructure cost and SLA violation. To achieve this goal, mapping and scheduling mechanisms are proposed to deal with the customer side dynamic demands and resource level heterogeneity. The mechanisms minimize the cost by optimizing the resource allocation within a VM, and thus allowing a cost effective usage of resources.

The resource selection protocol proposed in this paper differs from the above works in one important aspect. With our protocol no modification of the framework's resource allocation policy is required. Thus, the protocol may be applied on various frameworks.

6.2 PaaS (Platform as a Service)

Recently much research work has been done on the level of Platform as a Service (PaaS). In the following we provide a review of some of relevant works.

Themis [5] is a private PaaS system that shares cloud resources between competing applications using a fine-grained, market-based allocation mechanism. In Themis, applications autonomously adapt their resource demand to price variations with the aim to meet application SLAs. Unlike Meryn, Themis has no support for optimizing provider objectives and manages the resources of a single cloud.

Qu4DS [9] is a PaaS-level framework that assists providers in honoring SLAs while maximizing their profit. The framework includes mechanisms for SLA negotiation, translation, and enforcement and supports both performance and reliability QoS properties. However, the framework is restricted

to applications of a single type (master/worker) running on resources from a single IaaS cloud.

ConPaaS [15] is an open-source PaaS for hosting scientific and web applications in the cloud. ConPaaS aims at being able to run over a wide variety of public and private IaaS clouds. However, ConPaaS does not provide a policy to select the IaaS cloud provider and does not provide a support for optimizing the PaaS provider profit.

6.3 Virtual Clustering

In the field of virtual clustering, related work can be classified in two categories. The first category focuses on sharing private physical resources between several virtual clusters or frameworks, and includes Mesos [8] and VOC [14]. In Mesos [8] the sharing policy is based on negotiation between virtual clusters and a master. In VOC [14] the job queue is monitored and the sharing policy is based on job requirements. The second category focuses on extending private resources with public cloud resources, and includes Elastic Site [11] and Elastic Cluster [12]. In Elastic Site [11] a site is extended by integrating remote cloud resources on demand based on the number of queued jobs. In Elastic Cluster [12] a cluster is self-resizable to adapts to the workload and to meet the timing requirements of applications. At the best of our knowledge, Meryn is the only system belonging to both categories. Meryn enables at the same time the sharing of the private physical resources between multiple VCs and the cloud bursting of each VC. Moreover, none of these systems provide an SLA contract to the applications.

7. CONCLUSION AND FUTURE WORK

We presented Meryn, an open, SLA-driven, cloud bursting PaaS that provides an execution environment and SLA guarantees to diverse types of applications. Meryn makes use of diverse elastic virtual clusters and dynamically resizes them, either through exchanging VMs between the virtual clusters or using public cloud VMs. We have also presented a resource selection protocol that aims at optimizing the platform provider profit. This optimization is performed by optimizing the use of private resources before renting any public cloud resources. We have implemented a prototype of Meryn that supports batch and MapReduce applications. We presented a preliminary evaluation of the Meryn prototype using a synthetic workload. Results showed that the resource selection protocol can effectively improve the provider profit by reducing the cost of running applications.

In future work we plan to further investigate other application models, and in particular to propose a bid computation model and an SLA function for MapReduce applications. We plan also to perform more experiments with workloads representative of real data centers workloads.

Acknowledgment

We cordially thank Dr. Eugen Feller for providing documentation support concerning Snooze. D.Dib's PhD Grant is is co-funded by the Brittany Council. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

8. REFERENCES

[1] Amazon ec2. `http://aws.amazon.com/ec2/`.

[2] Nimbus. `http://www.nimbusproject.org/`.

[3] Grid5000. `https://www.grid5000.fr/`.

[4] S. Abrishami, M. Naghibzadeh, and D. Epema. Cost-driven scheduling of grid workflows using partial critical paths. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, 2010.

[5] S. V. Costache, N. Parlavantzas, C. Morin, and S. Kortas. Themis: Economy-Based Automatic Resource Scaling for Cloud Systems. In *14th IEEE International Conference on High Performance Computing and Communications (HPCC 2012)*, 2012.

[6] E. Feller, L. Rilling, and C. Morin. Snooze: A scalable and autonomic virtual machine management framework for private clouds. In *CCGRID*, 2012.

[7] H. Goudarzi and M. Pedram. Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, CLOUD '11. IEEE Computer Society, 2011.

[8] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: a platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, 2011.

[9] A. Lage Freitas, N. Parlavantzas, and J.-L. Pazat. An Integrated Approach for Specifying and Enforcing SLAs for Cloud Services. In *The IEEE 5th International Conference on Cloud Computing (CLOUD 2012)*, Honolulu, États-Unis, 2012.

[10] S. Malek, M. Mikic-Rakic, and N. Medvidovic. A decentralized redeployment algorithm for improving the availability of distributed systems. In *Proceedings of the Third international working conference on Component Deployment*, 2005.

[11] P. Marshall, K. Keahey, and T. Freeman. Elastic site: Using clouds to elastically extend site resources. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, 2010.

[12] G. Mateescu, W. Gentzsch, and C. J. Ribbens. Hybrid computing-where hpc meets grid and cloud computing. *Future Gener. Comput. Syst.*, 2011.

[13] D. Milojicic, I. M. Llorente, and R. S. Montero. Opennebula: A cloud management tool. *Internet Computing, IEEE*, 2011.

[14] M. A. Murphy, M. Fenn, and S. Goasguen. Virtual organization clusters. In *Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Washington, DC, USA, 2009. IEEE Computer Society.

[15] G. Pierre and C. Stratan. ConPaaS: a platform for hosting elastic cloud applications. *IEEE Internet Computing*, 2012.

[16] L. Wu, S. Garg, and R. Buyya. Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, 2011.