ANTONIO CARLOS SALZVEDEL FURTADO JUNIOR

IMPROVED RESOURCE CONSOLIDATION IN CLOUD COMPUTING

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Departamento de Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientator: Prof. Dr. Eduardo Almeida

CURITIBA

2011

ANTONIO CARLOS SALZVEDEL FURTADO JUNIOR

IMPROVED RESOURCE CONSOLIDATION IN CLOUD COMPUTING

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Departamento de Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientator: Prof. Dr. Eduardo Almeida

CURITIBA

2011

CONTENTS

ABSTRACT

The concept of cloud computing has been gaining a lot of attention lately, both in the business world and in research papers. Its popularity resides at the low costs it propitiates, opposed to traditional ways of providing computational resources. The main advantage is its elasticity, in which they can be offered on-demand. On the other hand, new challenges arise because of this dynamism. A system needs to guarantee a proper scheduling of these resources, in order to fulfil an organization's demands. At the same time, it needs to meet its scheduling policies (minimize costs, heigh availability, and so on) and deal with peaks of demand.

The paper is dedicated to the study of an efficient algorithm.

CHAPTER 1

INTRODUCTION

Cloud computing has attracted a lot of attention lately. It was popularized as a business model by Amazon's Elastic Compute Cloud (EC2), which started selling virtual machines (VMs) in 2006. Over time, more cloud providers have appeared in the market, offering their computational resources (CPU, memory, and I/O bandwidth). This definition of cloud, in which the IT infrastructure is deployed through virtual machines, is referred as Infrastructure-as-a-service (IaaS). One of the most appealing benefits of this paradigm, when associated to cloud providers, is the ability to cut costs. Companies may base their IT strategies on cloud-based resources, spending very little or no money managing their own IT infrastructure. They pay for these resources on-demand, in contrast to the traditional resource provisioning model, in which they would need to deal with both under- and over- utilization of their own resources. Moreover, the cloud providers may offer lower prices because they are benefited from the economy of scale.

Of course, the gains cited above are only noticeable if we are talking about public clouds – clouds made available in a pay-as-you-go manner to the public by an external provider. Although the market has evolved around this type of cloud, organizations might build IaaS clouds using their own infrastructure, known as private clouds. Their aim is not to sell capacity over the internet, but to give local users an agile and flexible infrastructure to run service workloads in their administrative domains. These users are offered VMs, which are scheduled in a group of physical machines within their organization, which we call a cluster. This leads to a better utilization of resources, since services with little demand can be packed into the same machine, process known as server consolidation. Other benefits, such as the migration of VMs between hosts and the ability to dynamically change the amount of resources provided to it, enabled by the technology present in virtual machine monitors (VMMs), make it possible to deal with fluctuations in the workload.

These characteristics propitiate an elastic environment, which is good for a private cloud, and vital for the pay-on-demand model used in public clouds. It's also possible for an organization to mix these two types of clouds, creating a hybrid cloud, which is useful to supplement a private cloud's infrastructure with external resources from a public one. However, this paper will not get into the details of hybrid clouds.

A cloud is highly dependable on machine virtualization, essential to achieve its goals. Database management systems (DBMS), like other software systems, are also increasingly being run on virtualized environments for many reasons. Some of them are mentioned in [4], [5] and [6], which include the reduction on the cost of ownership, better provisioning and manageability of applications and the ability to migrate it among physical hosts. This paper focus on other motivation, which is the possibility to take a variety of databases that run on dedicated computing resources and move them to a shared resource pool, including the ability to reallocate these resources as needed. It was formalized as a problem in [5], defined as the virtualization design problem (a.k.a. resource consolidation problem) for relational database workloads. It can be defined as follows: *"Given N database workloads that will run on N database systems inside virtual machines, how should we allocate the available resources to the N virtual machines to get the best overall performance?"*. According to that paper, the virtualization design problem may find a better solution when applied to relational database systems due to three factors. First, relational database workloads consist of SQL queries with constrained and highly specialized resource usage patterns. Second , queries are highly variable in the way they use resources – one query might heavily need CPU, while another might need I/O bandwidth instead. Thus, they could benefit from the dynamism in resource allocation. Third, database systems already have a way of modelling their own performance, namely the query optimizer.

As mentioned, DBMSes have particularities involving their workloads. Therefore, the application running inside a VM shouldn't be treated as a black box. Instead, the database system cost model should be exploited. As VMMs have parameters to control the share of physical resources, database systems have tuning parameters to manage their own performance. These two sets need to be simultaneously analysed and tuned. In [6],

this is a principle of a proposed *virtualization design advisor*. It works by recommending configuration parameters for a group of VMs, each one containing a DBMS. These parameters determine how the shared resources will be allocated to each VM, and consequently to each DBMS. It uses information about anticipated workloads to specify the parameters offline. Furthermore, runtime information collected after the deployment of the recommended configuration can be used to refine this recommendation and to handle fluctuations in the workload. It has not been proposed to run this advisor in a cloud yet, rather it has been implemented and tested in a single physical machine, in which two DBMS instances were deployed.

This paper proposes an implementation of this virtualization design advisor in a cloud environment, in a distributed manner. The advisor should be able to configure all the VMs deployed in a cluster, considering only the resources of the physical host in which it was deployed. It is expected that this advisor provides a better utilization of resources in the cloud, even though the cloud is not perceived by the advisor. Its efficiency would need to be confirmed through tests that would need to be performed after implementing. The rest of this paper is structured as follows. In Section 2, the *virtualization design advisor* is described. Section 3 is used to show how a cloud infrastructure is managed. In section 4, it is proposed an integration of the advisor within the cloud management system. Finally, section 5 presents some final considerations and an idea about future work.

CHAPTER 2

VIRTUALIZATION DESIGN ADVISOR

In [6], the author considers a typical resource consolidation scenario, in which several DMBS instances, each one of them running in a separate VM, share a common pool of physical resources. The mentioned paper addresses the problem of optimizing the performance of these instances by controlling the configurations of the VMs in which they run. These configurations determine how these resources will be allocated to each DMBS instance. These physical resources belong to one server, in which all the VMs run. The scenario is illustrated in 2.1.

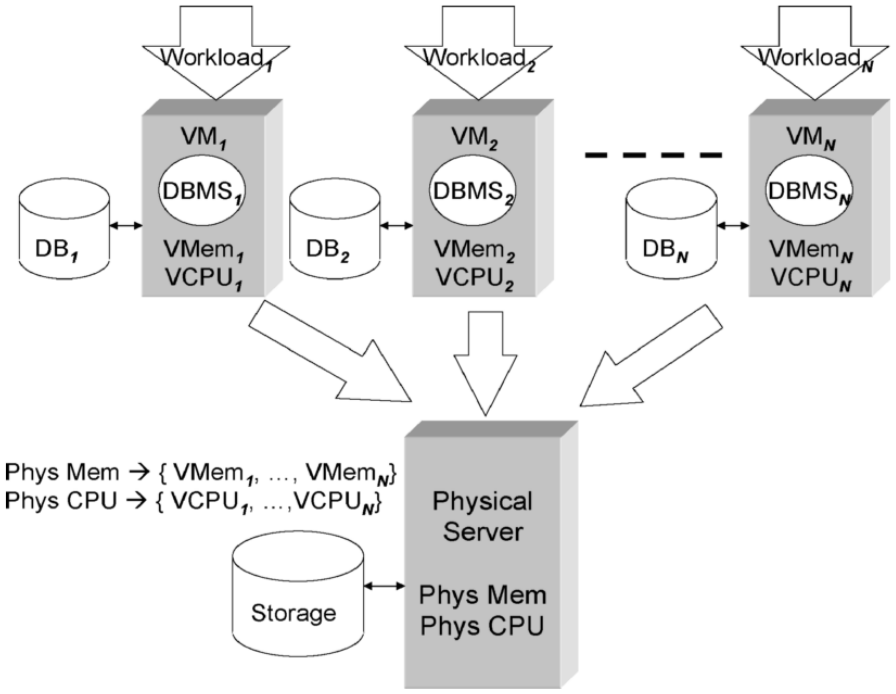

Figure 2.1: Resource Consolidation scenario

2.1 Problem definition

In order to model this problem, the author assumes that there are M types of resources , such as CPU capacity, memory, or I/O bandwidth. The notation used to represent the share of resources allocated to a VM running a workload W_i is $R_i = [r_{i1}, ...r_{iM}], 0 \leq r_{ij} \leq 1$. Each workload has the same monitoring interval for all the VMs (i.e. the workloads represent the statements processed by different DBMSes in the same amount of time).

Each workload is associated to a cost, which depends on the resource share allocated to the VM in which it runs. The notation $Cost(W_i, R_i)$ is used to represent the cost of running the workload W_i under resource allocation R_i. Considering that there are N workloads, the goal is to choose $r_{ij}, 1 \leq i \leq N, 1 \leq j \leq M$ such that

$$\sum_{i=1}^{N} Cost(W_i, R_i)$$

is minimized.

This problem was generalized to satisfy Quality of Service (QoS) requirements. One of these requirements is the possibility to specify the maximum increase in cost that is allowed for a workload under the recommended resource allocation. In order to model this requirement, it was defined a *cost degradation* as

$$Degradation(W_i, R_i) = \frac{Cost(W_i, R_i)}{Cost(W_i, [1, ..., 1])}$$

, where $[1, ..., 1]$ represents the resource allocation in which all of the available resources are allocated to W_i. It can be specified a *degradation limit* $L_i (L_i \geq 1)$, such that

$$Degradation(W_i, R_i) \leq L_i$$

for all i. This limit is set per workload, so it does not need information about other workloads that it will be sharing the physical server with.

Other QoS requirement introduced involves the ability to specify relative priorities among the different workloads. A *benefit gain factor* $G_i (G_i \geq 1)$ can be used to indicate

how important is to improve the performance of W_i. Each unit of improvement is considered to worth G_i cost units. When this parameter is applied to the problem, it may cause a workload to get more resources than its fair share. In order to incorporate it to our problem, the cost equation is modified to minimize the following

$$\sum_{i=1}^{N} G_i * Cost(W_i, R_i)$$

2.2 Architecture

The process of determining the allocation of resources to each VM is neither immediate, nor static. The proposed advisor follows a sequence of steps. Initially, it makes a resource allocation based on the workload descriptions and performance goals, which is performed offline (i.e. the VMs are not running yet). Then all of the subsequent steps are performed online. It adjusts its recommendations based on the difference between expected and actual workload costs to correct for any cost estimation errors made during the initial phase. At the same time, it uses continuing monitoring information to dynamically detect changes in the workloads. This last step is important because a workload cannot be considered static, its resource needs may change during execution. In both of these online steps, the resource allocation step may occasionally be performed again with updated cost models. This approach prevents the advisor from allocating resources to DBMS instances that will obtain little benefit from them. These resources need to be given to the workloads that need them the most.

Since the advisor does not consist in one single task, it makes sense to organize the flow of its tasks. An overview of this advisor in a modular way is given in 2.2. This paper intends to give a brief explanation of each task.

2.2.1 Cost estimation and initial allocation

Given a workload W_i, the cost estimator will estimate $Cost(W_i, R_i)$. The strategy used to implement this module is to leverage the cost models built into database systems for query

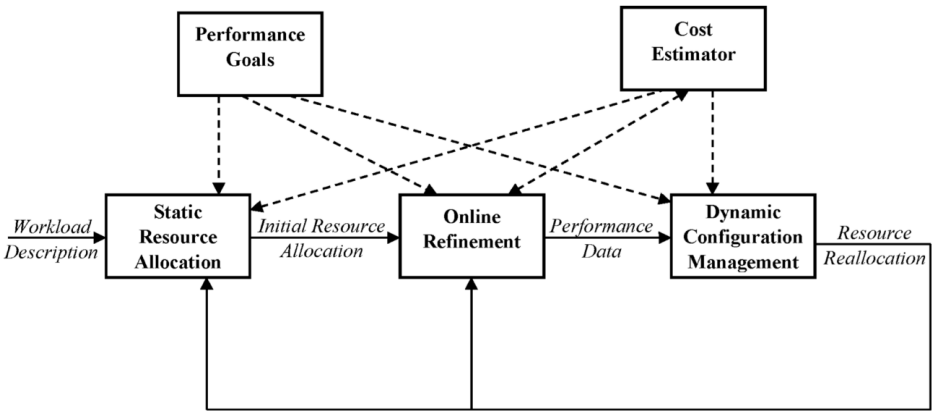

Figure 2.2: Advisor overview

optimization. The query optimizer cost model can be described as $Cost_{DB}(W_i, P_i, D_i)$, where W_i is a SQL workload, $P_i = [p_{i1}, .., p_{iL}]$ is a vector of parameters that are used to list both the available computing resources and DBMS configuration parameters relevant to the cost model, and D_i is the database instance.

The author identifies two problems in using only query optimizer cost models. The first problem is the difficulty of comparing cost estimates produced by different DMBSes. They may have different cost models. Even if they share the same notion of cost, the normalization process may differ. This problem is addressed partially by the advisor. It assumes that the DBMSes have the same notion of cost, and it proposes a renormalization step to make $Cost_{DB}(W_i, P_i, D_i)$ from different DBMSes comparable. This step is not considered for implementation, since the support of multiple DBMSes is out of the scope of this paper.

The second problem is that the query optimizer cost estimates depends on P_i, while the virtualization design advisor is given a candidate resource allocation R_i. The mapping of these two parameters is achieved through a calibration step. This step determines a set of DBMS cost model configuration parameters according to the different possible candidate resource allocations. It is supposed to be performed per DMBS on the physical machine before running the virtualization design advisor. Once the appropriate configuration parameters P_i are determined for every possible R_i, the DBMS cost model is used to generate $Cost_{DB}$.

The calibration step is performed on *descritive parameters*, which are used to characterize the execution environment. The approach to the *prescritive parameters*, which control the configuration of the DBMS itself, is to leave it for user definition. For instance, in 2.1 it is shown some descritive parameters used in PostgreSQL, while in 2.2 it is shown the prescritive ones.

Parameter	Description
random_page_cost	Cost of non-sequential page I/O
cpu_tuple_cost	CPU cost of processing one tuple
effective_page_size	size of file system's page size

Table 2.1: Descritive parameters

Parameter	Description
shared_buffers	shared bufferpool size
work_mem	amount of memory used by each sort and hashing operator.

Table 2.2: Prescritive parameters

The calibration step follows a basic methodology for each parameter $p_{ij} \in P_i$, described below:

- (1) Define a calibration query Q and a calibration database D, such that $Cost_{DB}(Q, P_i, D)$ is independent for all descritive parameters in P_i, except for p_{ij};

- (2) Choose a resource allocation R_i, instantiate D, and run Q under that resource allocation, and measure the execution time T_Q;

- (3) This step refers to the renormalization of the $Cost_{DB}$ provided by the DBMS. As mentioned earlier in this section, we are not going into the details of this step;

- (4) Repeat the two preceding steps for a variety of R_i allocations. $r_{ij} \in R_i$ only needs to be varied if p_{ij} describes that resource. For instance, query optimizer parameters that describe CPU, I/O and memory are independent of each other and can be calibrated independently. This shall avoid unnecessary calculations;

- (5) Perform regression analysis on the set of (R_i, p_{ij}) value pairs to determine a calibration function Cal_{ij} that maps resource allocations to p_{ij} values.

During the described methodology, calibration queries should be carefully chosen. They need to be dependent only on the parameter that is being calibrated. If it is not possible to isolate one parameter, a system of k equations is solved to determine the values for the k parameters.

After the calibration is performed, the advisor is ready to be run. First, it needs to provide an initial configuration. In [6], it was proposed a greedy algorithm to search for an initial configuration, as seen in 1.

Algorithm 1 Greedy search algorithm

for $i = 1 \rightarrow N$ **do**
  $R_i \leftarrow [\frac{1}{N}, .., \frac{1}{N}]$
end for
$done \leftarrow$ **false**
repeat
  $MaxDiff \leftarrow 0$
  for $j = 1 \rightarrow M$ **do**
    $MaxGain_j \leftarrow 0$
    $MaxLoss_j \leftarrow \infty$
    for $i = 1 \rightarrow N$ **do**
      $C' \leftarrow G_i{*}Cost(W_i, [r_{i1}, r_{ij}{+}\delta, r_{iM}])$ # Maximize gain
      if $C_i - C' > MaxGain_j$ **then**
        $MaxGain_j \leftarrow C_i - C'$
        $i_{gain} \leftarrow i$
      end if
      $C' \leftarrow G_i{*}Cost(W_i, [r_{i1}, r_{ij}{-}\delta, r_{iM}])$ # Minimize loss
      if $(C' - C_i < MaxLoss_j) \wedge (C' satisfies L_i)$ **then**
        $MaxLoss_j \leftarrow C' - C_i$
        $i_{loss} \leftarrow i$
      end if
    end for
    # Maximum benefit from adjusting this resource
    if $(i_{gain} \neq i_{lose}) \wedge (MaxGain_j - MinLoss_j > MaxDiff)$ **then**
      $MaxDiff \leftarrow MaxGain_j - MinLoss_j$
      $i_{maxgain} \leftarrow i_{gain}$
      $i_{maxloss} \leftarrow i_{loss}$
      $j_{max} \leftarrow j$
    end if
  end for
  if $MaxDiff > 0$ **then**
    $r_{i_{maxgain}j_{max}} \leftarrow r_{i_{gain}j_{max}} + \delta$
    $r_{i_{maxloss}j_{max}} \leftarrow r_{i_{loss}j_{max}} - \delta$
  else
    $done \leftarrow$ **true**
  end if
until done

The greedy search algorithm presented initially assigns $\frac{1}{N}$ share of each resource to each one of the N workloads. Then it iteratively considers shifting an amount δ of resources (e.g. $\delta = 5\%$) from one workload to another. It analyses which workload benefits more from gaining that resource and also which loses less. Based on this, it tries to obtain the maximum benefit from adjusting this resource. It covers all the resources. The *REPEAT* statement is performed until no more optimizations are possible. It's also important to

notice that this algorithm takes into consideration the QoS parameters discussed earlier in this paper, L_i and G_i.

From results of experiments, the creator of this algorithm observed that the greedy search is very often optimal and always within 5% of the optimal allocation

2.2.2 Online refinement

The initial allocation is based on the calibrated query optimizer cost model, as described earlier. This enables the advisor to make recommendations based on an informed cost model. However, this model may have inaccuracies that lead to sub-optimal recommendations. The *online refinement* is based on the observation the actual times of different workloads in the different virtual machines. It uses these observations to refine resource allocation recommendations. Then the advisor is rerun with the new cost models, so we can obtain an improved resource allocation for different workloads. These optimizations are performed until the allocations stabilize. It's important to notice that the goal of the *online refinement* is not deal with dynamic changes in the workload, which are dealt by another module, but to correct cost models. Thus, it's assumed that the workload is not going to change during this process.

In order to optimize the recommendations, it is necessary to identify the cost model behaviour. The author identifies two types of models. The first is the *linear model*, which can describe the allocation of CPU. For this resource workload completion times is linear in the inverse of the resource allocation level. Thus, the cost model in this case can be represented by

$$Cost(W_i, [r_i]) = \frac{\alpha_i}{r_i} + \beta_i.$$

The values α_i and β_i are obtained through a linear regression. This regression is performed on multiple points represented by estimated costs for different r_i values used during the initial allocation phase. This cost is adjusted by two parameters, Est_i and Act_i, they represent the estimated cost of workload and the runtime cost, respectively. When the cost is underestimated, these parameters are used to increase the slope of our

cost equation. From another standpoint, when this value is overestimated, we need to decrease the slope. This is achieved by

$$Cost(W_i, [r_i]) = \frac{Act_i}{Est_i} * \frac{\alpha_i}{r_i} + \frac{Act_i}{Est_i} * \beta_i.$$

However, not all resources are linear. The second type of cost model identified is the *piecewise-linear*, which describes the allocation of memory. Increasing this resource does not consistently result in performance gain. The magnitude and the rate of improvement change according to the query execution plan. The cost equation is similar to the linear cost model, it is given by

$$Cost(W_i, [r_i]) = \frac{Act_i}{Est_i} * \frac{\alpha_{ij}}{r_i} + \frac{Act_i}{Est_i} * \beta_{ij}, r_i \in A_{ij}.$$

The difference here is the parameter A_{ij}, which represents the interval of resource allocation levels corresponding to a particular query execution plan, that is represented by j. Its intervals are obtained during the initial phase, when the query optimizer is called with different resource allocations and returns different query execution plans with their respective costs. These query execution plans define the boundaries of the A_{ij} intervals. The end of this interval is the largest resource allocation level for which the query optimizer produced this plan. The initial values of α_{ij} and β_{ij} are obtained through linear regression. Together with A_{ij}, they are subsequently adjusted.

Both of the equations presented work within their cost model. Nevertheless, a physical machine has more than one type of resource. Therefore, the author extends the cost equations to multiple resources. He deals with the case in which M resources are recommended, such that $M - 1$ resources can be modelled using a linear function, while the resource M is modelled using a piecewise-linear function. The cost of workload W_i, given a resource allocation $R_i = [r_{i1}, ..., r_{iM}]$ can be given by

$$Cost(W_i, R_i) = \sum_{j=1}^{M} \frac{Act_i}{Est_i} * \frac{\alpha_{ijk}}{r_{ij}} + \frac{Act_i}{Est_i} * \beta_{ik}, r_{iM} \in A_{iMk},$$

in which k represents a certain query execution plan, which defines the boundaries of A_{iMk}.

During refinement, this equation is supposed to be performed iteratively, in order to optimize the parameters α_{ijk} and β_{ijk}. This iteration is shown below.

$$
\begin{aligned}
Cost(W_i, R_i) &= \sum_{j=1}^{M} \frac{Act_i}{Est_i} * \frac{\alpha_{ijk}}{r_{ij}} + \frac{Act_i}{Est_i} * \beta_{ik} = \sum_{j=1}^{M} \frac{\alpha'_{ijk}}{r_{ij}} + \beta'_{ik}, r_{iM} \in A_{iMk} \\
Cost(W_i, R_i) &= \sum_{j=1}^{M} \frac{Act_i}{Est_i} * \frac{\alpha'_{ijk}}{r_{ij}} + \frac{Act_i}{Est_i} * \beta'_{ik} = \sum_{j=1}^{M} \frac{\alpha''_{ijk}}{r_{ij}} + \beta''_{ik}, r_{iM} \in A_{iMk} \\
&\vdots
\end{aligned}
$$

The author in [6] proposes an heuristic that changes the equation in the first iteration. Instead of considering the interval A_{iMk}, the first iteration scales all the intervals of the cost equation (i.e., for all k). This is done because the estimation errors may reside in a bias in the query optimizer's view of resource allocation levels. In the second iteration and beyond, this cost will be refined according to the interval A_{iMk}, where the resource r_{iM} will be scaled.

These refinement iterations stops when the refinement continues beyond M iterations. In this case the refinement process continues, but through a linear regression model to the observed points. The query estimates wouldn't be necessary anymore. This process only finishes when the number of iterations reach an upper bound, placed manually. It is used to guarantee termination.

After refining the cost equations, the advisor is rerun. If the newly obtained resource allocation is the same as the original, the refinement process is stopped. Otherwise, it goes on.

2.2.3 Dynamic configuration management

Even if we have an optimal resource allocation, the workload may change its behaviour during execution. They may occur due to the intensity of the workload (e.g. increased number of clients), or the nature of the workload (e.g. new set of queries with different

resource needs). These changes are not dealt by our online refinement, that's why the dynamic configuration management is needed.

The proposed approach consists in monitoring the relative changes in the average cost per queries between periods. If the change in the estimated cost per query is above a threshold, θ, we classify this a major change. When this is identified, it's decided to make the virtualization design advisor to restart its cost modelling from initial state, before online refinement. The cost model needs to be discarded, since it no longer reflects information about the workload.

In order to deal with minor changes, it's introduced a new metric E_{ip}. It represents the relative error between the estimated and the observed cost of running workload W_i in monitoring period p. We analyse two consecutive periods. If both $E_{i(p-1)}$ and E_{ip} are below some threshold (e.g. 5%), or if $E_{ip} - E_{i(p-1)} > 0$, then we continue with online refinement. In this case, the errors are either small, or are decreasing. Both of these situations can be efficiently dealt by some iterations of online refinement. However, if this condition is not satisfied, we discard the cost model again.

CHAPTER 3

CLOUD INFRASTRUCTURE MANAGEMENT

This section will be used to described OpenNebula, that is a virtual infrastructure (VI) manager. Organizations can use it to manage and deploy VMs, individually or in groups that must be co-scheduled on local or external resources, which means that it supports hybrid clouds. Some of its key features are:

- It provides a homogeneous view of resources, regardless of the underlying hypervisor (e.g. KVM, Xen). This makes the virtualization much less restrictive. The physical machine in which the VM is being run does not need to be tied to a specific virtualization technology, causing incompatibility issues;
- Manage a VM's full life cycle, like managing storage requirements and setting up the network;
- Supports configurable resource allocation policy.

The OpenNebula architecture is illustrated by 3.1. It basically can be split into three layers. The core (middle layer) has three management areas. One of them is dealing with the creation of virtual networks, which is done through the Virtual Network (VN) manager. This component keeps track of leases (a set form by one IP and one MAC address valid on a particular network), and their associations with VMs and physical bridges they are using. Second, it needs to manage and monitor the physical, which is responsibility of the host manager. And finally, there is the VM manager, responsible for taking care of a VM's life cycle. It prepare the disk images for them, by controlling the image and storage technologies. It also needs to control the hypervisors, for creating and controlling them. And combined with the VN manager, it provides the VM with the network environment. These managers count with a SQL Pool, which saves the OpenNebula state, and in case of a failure, it is possible to restore it.

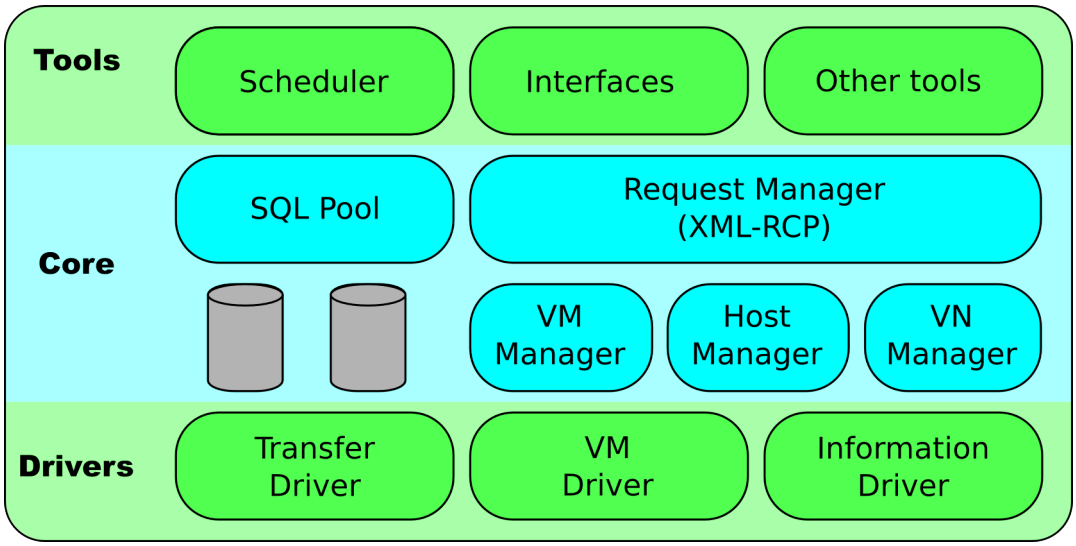

Figure 3.1: OpenNebula architecture

One important feature of the core module is the support of service deployment. The Request manager exposes a XML-RPC interface that decouples most of the functionality present in the core. As OpenNebula is in continuing development, it is expected that this API will support even more functionality. Currently, it is known that dynamic resource allocation is not supported yet. We'll discuss this issue on the next section, as this functionality is important for the advisor.

All these described activities are performed through pluggable drivers, present in the bottom layer. This modularity makes the system easier to extend and avoids tying it to any specific technology. The drivers shown in 3.1 address some particular areas, such as virtualization (by controlling the hypervisor), storage operations, gather monitoring information and authenticate user requests.

In the top layer tool there are both user interfaces to access OpenNebula, but also APIs extended from the XML-RPC, which can work between the core and other tools. Even more relevant for this paper is the OpenNebula's scheduler, also in this layer. Its job is to make VMs placement decisions, i.e. it assigns physical machines for VMs. It has access to all requests OpenNebula receives. Based on these requests, it keeps track on allocations and sends the appropriate deployment commands to OpenNebula core. As other modules, it can be replaced by third party solutions, such as Haizea[1], which

[1] http://haizea.cs.uchicago.edu/

offers more sophisticated placement policies. However, in this paper we will stick to the OpenNebula's default scheduler.

The scheduler already present in OpenNebula works only with immediate provisioning. Its concept is pretty straightforward, the resources are only provisioned at the moment they are required, if that's not possible, then the requirement is ignored. The requirements are done in a manual and static way, in which amount of resources, among other configurations, are defined in a file. The assignment is performed through a classification policy. The administrator needs to set a *RANK* variable, which defines which host is more suitable to host a VM. Each VM has its own RANK, and the scheduler assigns will assign a host with the highest value for this variable to a VM. It's possible to define a VM template in OpenNebula, so you can share the same scheduling policy to a group of VMs. For instance, here are some possible *RANK* definitions, each one serving a specific policy:

- Load-aware Policy
  - **Heuristic:** Use nodes with less load;
  - **Implementation:** Use nodes with more *FREECPU* first.
  - *RANK* = *FREECPU*
- Striping policy
  - **Heuristic:** Spread the VMs in the cluster node;
  - **Implementation:** Use the nodes with less VMs running first.
  - *RANK* = "- *RUNNING_VMS*"

CHAPTER 4

PROPOSED IMPLEMENTATION

In this paper, it's proposed an implementation of the virtualization design advisor over OpenNebula. The aim is to optimize the distribution of resources inside the private part of a cloud for VMs running database workloads. Since it's not possible, nor makes sense to optimize an external provider's infrastructure, the public part of the cloud is just ignored. We also stick to the PostgreSQL[1] as the DBMS used in our solution, although the support for other DBMSs could be extended in future work.

As already described, the virtualization designer advisor has only been modelled and tested against one server. The major issue of this paper is to port it to a cloud, which can contain several hosts. Other issue is that a cloud is an heterogeneous environment. In spite of the homogeneous view of resources provided by the VMM, the hosts may be different themselves. To address these problems, the proposed approach consists in having N instances of our advisor, being N the number of nodes in the private cloud. Therefore, the advisor will not see the cloud as a whole, but only the host it was assigned to work with, the same as it was working wit only one server.

The first step of this implementation would be the creation of a module responsible for the calibration process. As discussed earlier, this calibration will be used to map query optimizer's cost model, which depends on its parameters P_i, to the advisor's cost model, based on resources R_i. This process is supposed to be executed before any VM deployment, since the initial allocation depends on the latter cost model. Thus, it will be executed whenever a new physical host is added to a cluster. As this paper is limited to one DBMS, the renormalization step will not be performed after calibration.

After calibration, we expect the OpenNebula's default scheduler to define which host will be assigned to the VM that is to be deployed, as usual. At this step, the only provision

[1] http://www.postgresql.org/

that needs to be taken is to set the *RANK* variable properly. In this paper, we intend to deal with only two types of resources: memory and CPU. Therefore, we propose the following heuristic for the *RANK*:

- $RANK = \alpha * FREECPU + (1 - \alpha) * FREEMEMORY$, $0 \leq \alpha \leq 1$.

In this equation, we expect α to be used to prioritize one type of resource over another. Since the scheduler assigns the host before being possible to run the advisor, this heuristic will be used for all kinds of workloads. So it doesn't matter whether they are more or less CPU intensive or memory intensive. We consider this to be a limitation. However, we still expect that this heuristic will be able to distribute well the VMs among the hosts.

Once the VMs are placed on a machine, the initial configuration step from the virtualization design advisor can be started. In order to deal with their initial configuration, it's proposed the use of the greedy search algorithm, described earlier in this paper. A problem that has already been identified, which affects the greedy algorithm, is the lack of current support for dynamic resource reallocation in OpenNebula. One possible approach would be the use of libvirt[2], which defines itself as "A toolkit to interact with the virtualization capabilities of recent versions of Linux (and other OSes)". It offers an API that works with all the hypervisors supported by OpenNebula, offering many features, including resource reallocation. Currently, OpenNebula has an libvirt API, which enables this tool to manage the VMs over the core layer. It should be possible to implement a solution by using libvirt under the core layer, to extend the drivers' capabilities.

The online refinement and the dynamic configuration management steps should have direct implementations, following their descriptions. The initial configuration is defined by the greedy algorithm (the static resource allocation module), it stops when it find it's the best allocation, which may be the optimal or close to it. Once this task is performed, the online refinement is started. It updates the cost model by observing estimated and actual times of workloads. It returns an optimized cost model to the first step, where the greedy algorithm is restarted and searches for a new recommendation with the optimized cost model. The optimizer only stops when the newly obtained recommendations don't

[2]http://libvirt.org/

differ from the original ones (i.e. it stabilizes). The dynamic configuration management module is also started after the initial configuration was performed. It also uses optimized cost models to monitor changes in the workload. It may restart the workload when major changes are detected.

The performance of this advisor is expected to improve as optimized cost models are obtained, and so less calls to the DBMS's query optimizer are needed. This is due to the fact that these calls have a high computational cost.

CHAPTER 5

FINAL CONSIDERATIONS

Even though OpenNebula's capabilities of scheduling enables a way of distribution of resources, this is not enough for consolidating resources. Its goal is not to analyse the application workload inside a VM. That's why the resources need to be required in a static way, in which it's not possible to reallocate resources.

The virtualization design advisor is a way of distributing these resources by considering the database workload. Although it generates overhead, it provides a way of dealing with both over- and under- utilization of resources.

For a future work, the cloud infrastructure should be more exploited. Through live migration of VMs, a technology already supported by most hypervisors and OpenNebula, it's possible to transfer VMs from one host to another. This could be useful in cases which avoiding over-utilization of resources is not possible (e.g. Too many VMs with high CPU needs). This paper was limited to deal with the problem of resource consolidation inside each host. By considering other hosts into the model, it could be possible to broaden to deal with server consolidation.

REFERENCES

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, 2009.

[2] Carlo Curino, Evan P.C. Jones, Samuel Madden, and Hari Balakrishnan. Workload-aware database monitoring and consolidation. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 313–324, New York, NY, USA, 2011. ACM.

[3] Donald Kossmann and Tim Kraska. Data management in the cloud: Promises, state-of-the-art, and open questions. *Datenbank-Spektrum*, 10:121–129, 2010. 10.1007/s13222-010-0033-3.

[4] U.F. Minhas, J. Yadav, A. Aboulnaga, and K. Salem. Database systems on virtual machines: How much do you lose? In *Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on*, pages 35 –41, april 2008.

[5] A.A. Soror, A. Aboulnaga, and K. Salem. Database virtualization: A new frontier for database tuning and physical design. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 388 –394, april 2007.

[6] Ahmed A. Soror, Umar Farooq Minhas, Ashraf Aboulnaga, Kenneth Salem, Peter Kokosielis, and Sunil Kamath. Automatic virtual machine configuration for database workloads. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 953–966, New York, NY, USA, 2008. ACM.

[7] B. Sotomayor, R.S. Montero, I.M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *Internet Computing, IEEE*, 13(5):14 –22, sept.-oct. 2009.

ANTONIO CARLOS SALZVEDEL FURTADO JUNIOR

IMPROVED RESOURCE CONSOLIDATION IN CLOUD COMPUTING

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Departamento de Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientator: Prof. Dr. Eduardo Almeida

CURITIBA

2011