ANTONIO CARLOS SALZVEDEL FURTADO JUNIOR

AUTOMATIC ELASTICITY 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

AUTOMATIC ELASTICITY 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 a big potential to change the Information Technology (IT) world. It was popularized as a business model by Amazon's Elastic Compute Cloud, 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 type 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. Also, the cloud providers may offer lower prices because they are benefited with the economy of scale.

The benefits mentioned earlier are only noticeable when using public clouds – clouds made available in a pay-as-you-go manner to the public by a cloud 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 sell capacity over the internet, but give local users an agile and flexible infrastructure to run service workloads in their administrative domains. These users are offered VMs, which are scheduled among a group of physical machines from their organization, which we call a cluster. This leads to a better utilization of resources, since services with little demands 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, which are enabled by the technology present in virtul machine monitors (VMMs), make it possible to deal with fluctuations in the workload. This

provides 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 create a hybrid cloud, which are useful to supplement a private cloud's infrastructure with external resources from a public one.

The machine virtualization proposed by the cloud concept is essential to achieve its benefits. Database management systems (DBMS), like other software systems, are also increasingly being run on virtualized environments with the same goal. In [1], it is discussed the virtualization design problem for relational database workloads, which 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 it, the problem of virtualization design 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 patters. Second , queries are highly variable in the way they use resources – one query might heavily need CPU, while another needs I/O bandwidth instead. Thus, they could benefit from elasticity. Third, database systems already have a way of modeling their own performance, namely the query optimizer.

As mentioned, DBMSs have particularities involving their workloads. Therefore, the application running inside a VM shouldn't be treated as a black box. Instead, it should exploited the database system cost model. 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 analyzed and tuned. In [2], it is proposed a *virtualization design advisor*. It works by setting the configuration parameters of a VM containing a DBMS. These parameters determine how the shared resources will be allocated to each DBMS instance. It uses information about their 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 was not proposed to run this advisor in a cloud, rather it was implemented in a single physical machine, in which two DBMS

instances were deployed.

Our study proposes to implement this virtualization design advisor in a cloud environment. The advisor should be able to configure all the VMs deployed in a cluster, considering the resources of all physical machines. It is expected that this advisor provides automatic elasticity for the cloud. 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, we propose an integration of the advisor within the cloud management system.

CHAPTER 2

VIRTUALIZATION DESIGN ADVISOR

In [2], it is considered a typical scenario of resource consolidation, 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. It's also considered that the physical resources belong to one server, in which all the VMs run. The scenario is illustrated in 2.1.

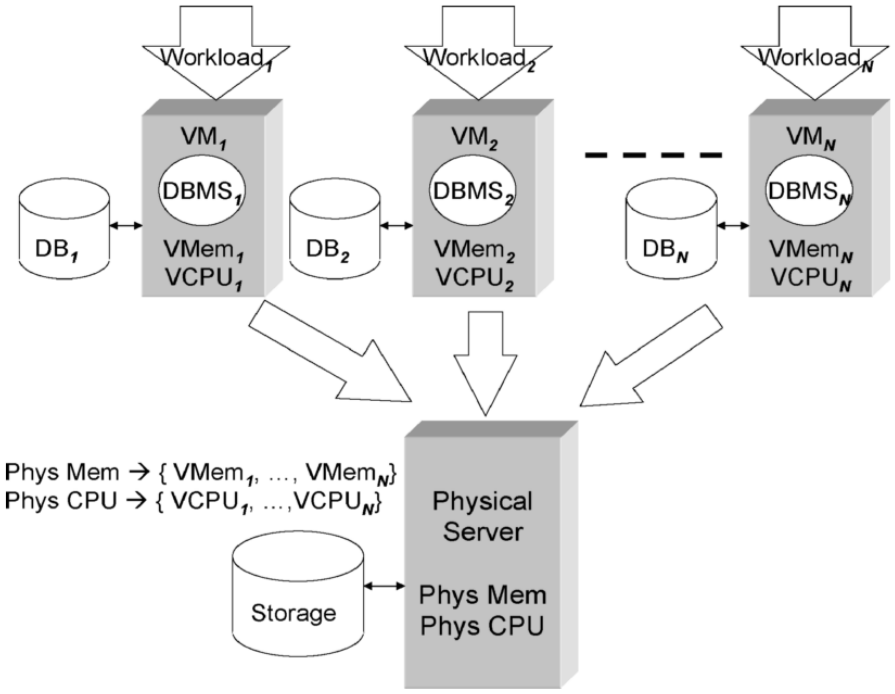

Figure 2.1: Resource Consolidation scenario

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 $R_i = [r_{i1}, ...r_{iM}], 0 \leq r_{ij} \leq 1$. This workload represents the statements processed by the different DMBS instances at the same amount of time.

Each workload has a cost, which depends on the resource share allocated to the VM in which it runs. It is used $Cost(W_i, R_i)$ 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 is generalized to satisfy Quality of Service (QoS) requirements.On of these requirements is to specify the maximum increase in cost that is allowed for a workload under the recommended resource allocation. 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 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.

The other QoS requirement introduced is 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 not immediate, nor static. The proposed advisor follows a sequence of steps. Initially, it makes resource allocations based on the workload descriptions and performance goals, which is performed offline, i.e., the VMs are not running yet. Then two following steps are performed online. First, it adjusts its recommendations based on workload costs to correct for any cost estimation erros made during the initial phase. Second, 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. This approach prevents the advisor from allocating resources to DBMS instances that will obtain little benefit from them.

Since the advisor has a considerable number of sub-processes, it makes sense to organize it in modules. An overview of this advisor in a modular way is given in 2.2. This paper intends to give a brief explanation of each module.

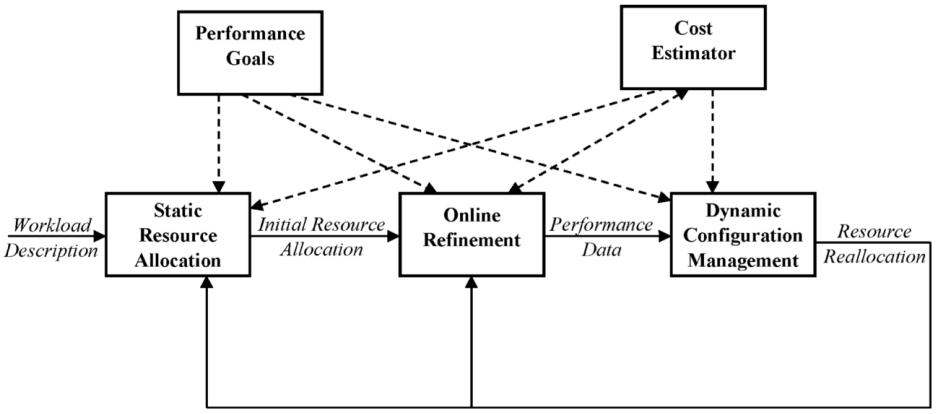

Figure 2.2: Advisor overview

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 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, they normalize costs differently. This problem is addressed partially by the advisor. Although it only considers that the DBMSes have the same notion of cost, it proposes a renormalization step, in order to make $Cost_{DB}(W_i, P_i, D_i)$ from different DBMSes comparable. For our purpose this step is not considered for implementation, since the support of multiple DBMSes is out of 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 done 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, the 2.1 show some descritive parameters used in PostgreSQL, while 2.2 show the prescritive ones.

The calibration follows the basic methodology for each parameter $p_{ij} \in P_i$:

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

- (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, associating with each a value of p_{ij}, which describes a certain resource. For instance, query optimizer parameters that describe CPU, I/O and memory are independent of each other and can be calibrated independently;

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

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 [2] 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 will stop under two situations. The first is when the newly resource allocation recommendation is the same as the second. The second situation is 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.

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 modeling 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 analyze 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 erros are either small, or are decreasing. Both of these situations can be effitiently 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. Each component is specialized in one aspect of VI management. The core takes care of the VM's life cycle. It has three management areas. The first is the preparation of disk images for the VMs. It needs to control the image and storage technologies. The second area is the setup of the network environment for the VMs . And finally, it needs to control the hypervisors for creating and controlling them. All of these activities are performed through pluggable drivers. This modularity makes the system easier to extend and avoids tying it to any specific technology. Other benefit of the core is the ability to support services

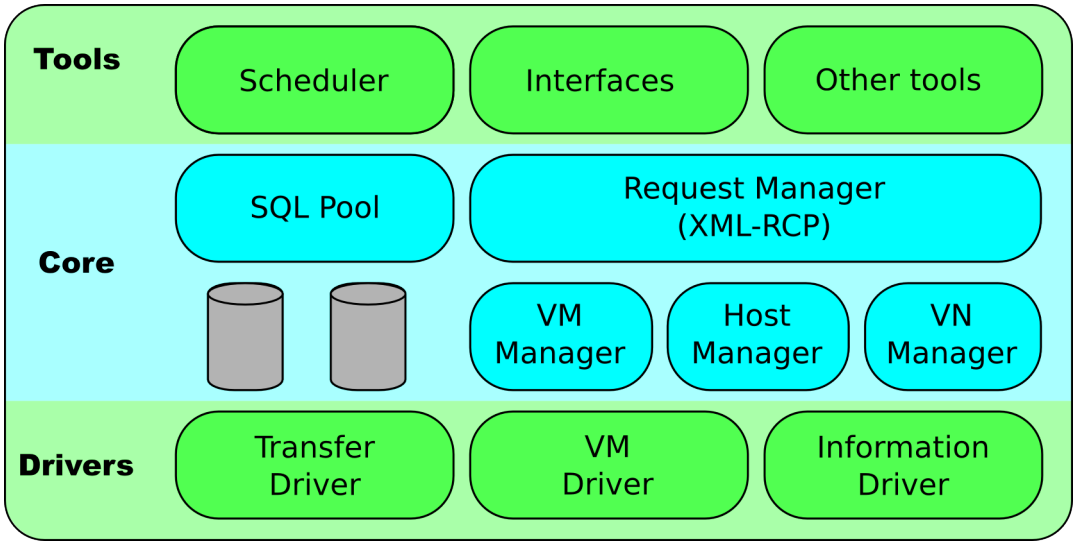

Figure 3.1: OpenNebula architecture

BIBLIOGRAFIA

[1] 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.

[2] 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.

ANTONIO CARLOS SALZVEDEL FURTADO JUNIOR

AUTOMATIC ELASTICITY 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