PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

APRIMORANDO A ELASTICIDADE DE APLICAÇÕES DE BANCO DE DADOS UTILIZANDO VIRTUALIZAÇÃO EM NÍVEL DE SISTEMA OPERACIONAL

ISRAEL CAMPOS DE OLIVEIRA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. César Augusto F. De Rose

Porto Alegre
2015

Dados Internacionais de Catalogação na Publicação (CIP)

O48a Oliveira, Israel Campos de
Aprimorando a elasticidade de aplicações de banco de dados utilizando virtualização em nível de sistema operacional / Israel Campos de Oliveira. – Porto Alegre, 2015.
95 p.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. César Augusto F. De Rose.

1. Informática. 2. Computação em Nuvem. 3. Máquinas Virtuais. I. De Rose, César Augusto F. II. Título.

CDD 004.36

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Aprimorando a Elasticidade de Aplicações de Banco de dados Utilizando Virtualização em Nível de Sistema Operacional" apresentada por Israel Campos de Oliveira como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 13/03/2015 pela Comissão Examinadora:

Prof. Dr. César Augusto Fonticielha De Rose – Orientador — PPGCC/PUCRS

Prof. Dr. Tiago Coelho Ferreto – PPGCC/PUCRS

Profa. Dra. Andrea Schwertner Charão – UFSM

Homologada em 05/11/2015, conforme Ata No. 020 pela Comissão Coordenadora.

Prof. Dr. Luiz Gustavo Leão Fernandes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

DEDICATÓRIA

Dedico este trabalho a minha esposa e ao meu filho.

“The art of simplicity is a puzzle of complexity.”
(Douglas Horton)

AGRADECIMENTOS

Agradeço a DELL Computadores por acreditar na pesquisa e financiar este trabalho durante dois anos. Ao orientador Prof. Dr. Cesar A. F. De Rose que buscou subsídios, recursos computacionais e norteou o trabalho durante dois anos. Seus conselhos e direções foram fundamentais para o sucesso desta pesquisa. Suas exigências contribuíram fortemente para o meu crescimento pessoal, bem como profissional. Meus sinceros agradecimentos.

Meus agradecimentos ao professor Dr. Tiago Ferreto que acompanhou todas as etapas da pesquisa, desde sua concepção até sua finalização, enriquecendo-a com suas sugestões.

Ao Laboratório de Alto Desempenho da PUCRS (LAD) por ter oportunizado este trabalho. Aos colegas do LAD, em especial ao Fábio Diniz Rossi e ao Miguel Gomes Xavier que sempre me apoiaram e estiveram presentes nos momentos de angústia e de felicidade. Foram inúmeras as vezes que discutimos assuntos sobre temas atuais e direções futuras para esta pesquisa.

Agradeço a minha fiel companheira Liliane, por sua paciência e a apoio incondicionais. Por fim, agradeço aos meus pais, pela preocupação e pelos sinceros conselhos dados nestes dois anos

APRIMORANDO A ELASTICIDADE DE APLICAÇÕES DE BANCO DE DADOS UTILIZANDO VIRTUALIZAÇÃO EM NÍVEL DE SISTEMA OPERACIONAL

RESUMO

Visando manter um serviço em execução dentro de níveis de QoS aceitáveis, os administradores de sistemas devem provisionar recursos suficientes para lidar com as flutuações de carga de trabalho. Recursos provisionados de maneira inadequada podem reduzir os lucros das empresas ou degradar o desempenho do serviço. Vários trabalhos apresentam soluções elásticas para alocação de recursos em ambientes de virtualização. Porém, em ambientes de virtualização tradicionais, a ação da elasticidade fica limitada pela fatia de recursos estabelecida durante a inicialização das máquinas virtuais, não permitindo a adição de recursos em tempo de execução. No entanto, a virtualização em nível de sistema operacional consiste em uma nova abordagem que permite a manipulação em tempo de execução da totalidade dos recursos disponíveis e oferecidas pelo nodo de forma compartilhada. Portanto, neste trabalho avaliamos o impacto da estratégia de alocação dinâmica de recursos proposta por Dawoud et al., primeiramente desenvolvida para ambientes tradicionais de virtualização, posteriormente adaptada à virtualização em nível de sistema operacional. A avaliação de nossa abordagem utilizou aplicação de banco de dados, pois consiste em uma plataforma bastante utilizada em ambientes virtualizados, além de seus *workloads* utilizarem tanto processador, quanto memória e rede. Os resultados mostraram que nossa abordagem é eficaz quanto a economia dos recursos disponíveis, impactando nas métricas de desempenho.

Palavras-Chave: containers, elasticidade, nuvem, virtualização.

IMPROVING ELASTICITY IN VIRTUALIZED ENVIRONMENTS FOR DATABASE APPLICATIONS THROUGH OS-LEVEL VIRTUALIZATION

ABSTRACT

In order to maintain a service running within the acceptable Qos levels, cloud providers should adapt resource provision to handle workload fluctuations. In case of under provisioned resources service performance tend to be affected while over provisioned resources reduce providers profit. To cope with this scenario several studies present elastic solutions for resource allocation in virtualization environments. However, in traditional virtualization environments, the elasticity is limitedby the limits established during initialization of virtual machines, not allowing the addition of resources in runtime. Instead, OS-level virtualization is a new approach that allows the manipulation of all available resources shared by a host at runtime. Therefore, this study evaluates the impact of applying the dynamic resource allocation strategy proposed by Dawoud et al., firstly developed for traditional virtualization environments, to OS-level virtualization. To evaluate our approach, we use database applications because they are widely used in virtualized environments, and their workloads use processor, memory, and network. The results show that our approach has a more efficient use of available resources, resulting in a positive impact on performance metrics.

Keywords: cloud, containers, elasticity, virtualization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

BD – Banco de Dados

CPU – Central Processor Unit

MV – Máquina Virtual

OLTP – On-Line Transaction Processing

QOS – Quality of Service

SGBD – Sistema Gerenciador de Banco de Dados

SLA – Service Level Agreement

SLO – Service Level Objective

SO – Sistema Operacional

TPS – Transações por Segundo

VMM – Virtual Machine Monitor

SUMÁRIO

1. INTRODUÇÃO

A computação em nuvem não é um conceito novo mas sim o ressurgimento do paradigma de *utility computing* cujo objetivo é fornecer serviços de fácil acesso e de baixo custo aos usuários garantindo características tais como disponibilidade, elasticidade e escalabilidade. Em linhas gerais, isto deve-se aos avanços das técnicas de virtualização e do modelo de negócio *pay-per-use* inerente à computação em nuvem. O NIST *(National Institute of Standards and Technology)* [Mel11] conceitualiza a computação em nuvem da seguinte forma: infraestrutura com capacidades de provisionamento de recursos de forma rápida e elástica, em alguns casos automática, aumentando ou diminuindo a quantidade de recursos. Para o usuário, tais capacidades muitas vezes parecem ser ilimitadas, podendo ser provisionadas em qualquer quantidade e a qualquer momento.

A definição do NIST enfatiza que a elasticidade, capacidade de alocação e/ou desalocação de recursos sob demanda de acordo com o comportamento do serviço, é uma das características mais marcantes da computação em nuvem. A elasticidade vai ao encontro da computação verde, possibilitando um melhor aproveitamento dos recursos providos pela infraestrutura. Assim como é possível aumentar o número de recursos, eles também podem ser reduzidos à medida que não são mais utilizados. Esta medida economiza custos, evitando o desperdício de energia e impactando em economia financeira e sustentabilidade.

Essa dinamicidade na quantidade de recursos conforme o comportamento da aplicação é regrada através de métricas, sejam elas de desempenho, disponibilidade ou segurança, que são regidas pelos acordos de níveis de serviço (SLA). O SLA consiste em um contrato entre o prestador de serviços e seus clientes e pode ser dividido em um ou mais objetivos de nível de serviço (SLO). Podemos exemplificar um SLO com a seguinte assertiva: *A página inicial deve ser completamente carregada em não mais de 2 segundos*. Como pode ser visto, SLO consiste em três partes: métrica de qualidade de serviço (QoS) (o tempo de resposta), o limite máximo de tempo no atendimento da requisição (2 segundos), e um operador relacional (não mais do que). A violação destes objetivos normalmente está associados com penalidades para o provedor do serviço.

No entanto, o mapeamento dinâmico de métricas de QoS em recursos de infraestrutura, como CPU e memória, consiste em um grande desafio. Assim sendo, a abordagem mais comum consiste em estabelecer limites (*thresholds*) superiores e inferiores (fixos ou dinâmicos) que são comumente usados para determinar ações de elasticidade [CMKS09]. Embora amplamente utilizados, limites que apresentam valores fixos permitem a quebra de SLAs quando o comportamento da aplicação se mostra irregular. Já limites dinâmicos podem onerar métricas de desempenho devido às seguidas mudanças na quantidade de recursos.

A fim de superar o desafio em diminuir *trade-offs* ocasionados pela utilização de métricas fixas, dinâmicas ou híbridas, diversos trabalhos têm proposto diferentes abordagens. Zhu et al. [ZWS06] desenvolveu um modelo matemático com o objetivo de assegurar que uma determinada máquina virtual receba recursos suficientes para manter um serviço dentro dos níveis de eficiência desejados. Este modelo foi utilizado por Dawoud et al. [DTM12] na construção de sua arquitetura elástica, onde também implementou um atuador em nível de aplicação que realiza alterações em tempo real na configuração em um servidor Apache, tornando a solução por ele proposta dependente desta plataforma.

Todavia a solução de Dawoud et al. [DTM12] foi desenvolvida com base em um sistema de virtualização tradicional onde somente é possível gerenciar os recursos atribuídos na inicialização das instâncias virtuais. Esta limitação é um fator crítico, pois caso o serviço do usuário necessite de mais recursos do que alocado inicialmente para manter a qualidade deste serviço, a arquitetura elástica não conseguirá alocar os recursos necessários. A utilização da virtualização em nível de Sistema Operacional irá resolver essa limitação, pois neste tipo de virtualização, é possível alocar recursos independentemente do que foi definido na inicialização da instância virtual.

Além disso, trabalhos como [XNR+13] e [LCXDR13] mostram que a virtualização em nível de Sistema Operacional apresenta melhor desempenho frente ao modelo tradicional de virtualização, o que irá contribuir em um melhor aproveitamento dos recursos, trazendo benefícios tanto para usuário, que poderá obter melhor desempenho para suas aplicações utilizando menos recursos, quanto para o provedor de serviços que irá economizar estes recursos.

Desta forma o objetivo deste trabalho é avaliar o impacto do uso da virtualização em nível de Sistema Operacional sobre a estratégia de alocação dinâmica de recursos proposta por Dawoud et al. [DTM12], com foco no uso de aplicações de banco de dados e com a finalidade de manter um determinado SLO.

Sistemas de gerenciamento de banco de dados (SGBDs) são candidatos potenciais para a implantação em nuvem, isso porque, geralmente as instalações destes sistemas são complexas e envolvem uma grande quantidade de dados, ocasionando um custo elevado, tanto em *hardware* quanto em *software* [SMMM10] [XDODPDR14]. Para muitas empresas o pagamento baseado no uso do modelo de computação em nuvem, juntamente com o suporte para manutenção do *hardware*, é muito atraente [Lom].

As principais contribuições deste trabalho são as seguintes: (i) desenvolvimento de controladores de CPU e memória baseados em [ZWS06]; (ii) avaliação de desempenho da virtualização em nível de Sistema Operacional quanto à eficiência na alocação e desalocação de recursos frente à um modelo tradicional de virtualização; (iii) verificação desta estratégia de alocação de recursos utilizando como carga de trabalho aplicações de banco de dados.

1.1 Organização do Trabalho

O presente capítulo apresentou a introdução deste trabalho. Os próximos capítulos estão organizados como segue: os Capítulo 2 e 3 apresentam o embasamento teórico deste trabalho, concentrando-se nas áreas de Virtualização e Computação em Nuvem respectivamente. O Capítulo 4 apresenta o Estado da Arte, mostrando como a elasticidade está inserida nos principais serviços de computação em nuvem, como também os trabalhos da academia que oferecem soluções para os principais problemas relacionados à elasticidade.

No Capítulo 5 , será feita uma análise em profundidade da arquitetura elástica proposta por Dawoud et al. [DTM12] na qual esse trabalho se baseia, bem como as modificações necessárias que realizamos para a implementação desta arquitetura utilizando a técnica de virtualização em nível de sistema de operacional. No Capítulo 6 serão demonstrados os resultados das avaliações que realizamos. As considerações finais e os trabalhos futuros são apresentados no Capítulo 7.

2. VIRTUALIZAÇÃO

O conceito de virtualização surgiu na década de 60, quando a IBM introduziu no mercado o *mainframe* S/370 no qual era possível a criação de máquinas virtuais logicamente separadas, dando ao usuário a ilusão de que estaria acessando diretamente uma máquina física [BK12]. Esta solução permitiu que os usuários deste tipo de equipamento, que era de alto custo, utilizassem a totalidade dos recursos da máquina, trazendo um melhor retorno de investimento para o usuário [VMW].

Estas partições virtuais, também chamadas de máquinas virtuais (MVs), permitiam a execução várias tarefas, ou seja, vários aplicativos e processos ao mesmo tempo e de forma concorrente sobre os mesmos recursos. Desta forma uma máquina virtual consiste em um ambiente operacional auto-suficiente que abstrai as características de *hardware* e *software* da máquina física na qual ela está sendo executada [Car08]. Isto torna-se possível devido ao Monitor de Máquina Virtual (*Virtual Machine Monitor - VMM*, também conhecido por *hypervisor* [VR13].

O VMM é um componente de *software* responsável pela criação, isolamento e preservação do estado da máquina virtual. É responsável pela orquestração de acesso aos recursos do sistema que hospeda as máquinas virtuais, mantendo controle sobre os recursos compartilhados por estas, tais como, processadores, dispositivos de entrada e saída, memória e armazenamento. Também é função do VMM escalonar qual máquina virtual vai executar a cada momento, semelhante ao escalonador de processos do sistema operacional. A Figura 2.1 mostra a relação entre o VMM e máquinas virtuais.

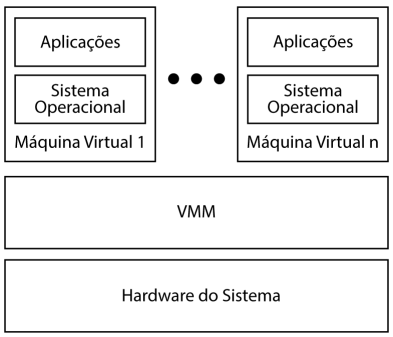

Figura 2.1 – Virtual Machine Monitor (VMM)

Goldberg [Gol73] classificou a virtualização de duas formas distintas: virtualização de Tipo I e Tipo II. Na virtualização de Tipo I o VMM é executado diretamente no *hardware* físico, já na virtualização de Tipo II o VMM é executado sobre um sistema operacional.

Recentemente surgiu um novo tipo de virtualização denominada híbrida [SdO07], por absorver características dos Tipo I e II. Na virtualização de abordagem híbrida o VMM e a máquina virtual operam principalmente no *hardware* físico, mas usam o sistema operacional hospedeiro para realizar operações de E/S. Essa abordagem é utilizada pelas soluções

de virtualização da Microsoft, como Hyper-V [VV09] e Virtual-PC [Hon03]. A Figura 2.2 ilustra os três tipos de implementação do VMM.

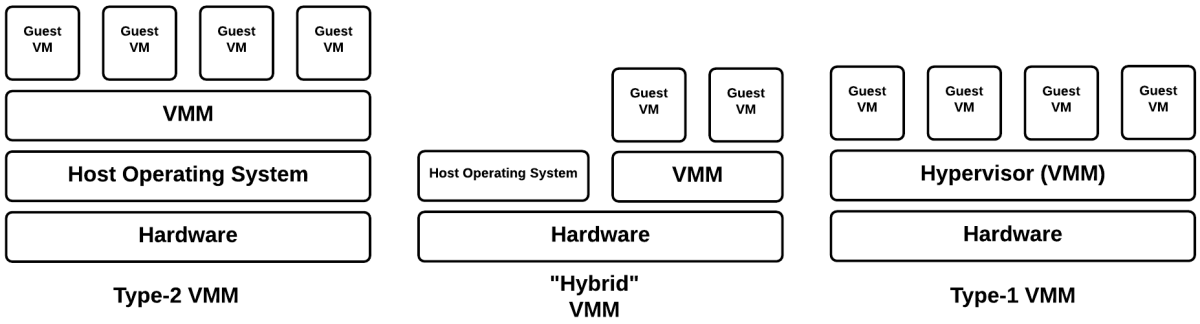

Figura 2.2 – VMM

Apesar de todas as vantagens a virtualização foi abandonada durante as décadas de 80 e 90 quando os aplicativos cliente-servidor passaram a dominar o mercado e os *desktops* e servidores x86 tornaram-se equipamentos mais acessíveis. Todavia o crescimento das implantações deste tipo de equipamento levou ao surgimento de novos desafios operacionais e de infraestrutura de TI. Segundo [VMW] esses desafios incluem:

A pouca utilização da infraestrutura :
As implantações típicas de servidor x86 alcançam uma média de utilização de apenas 10 a 15% de sua capacidade total [VMW]. As organizações normalmente executam um aplicativo por servidor para evitar o risco de as vulnerabilidades de um aplicativo afetarem a disponibilidade de outro no mesmo servidor.

Aumento dos custos da infraestrutura física :
Os custos operacionais para oferecer suporte à crescente infraestrutura física tiveram um aumento constante. A maior parte da infraestrutura computacional deve permanecer em operação o tempo todo resultando em custos - como os de consumo de energia, refrigeração e instalações - invariáveis com os níveis de utilização.

Aumento dos custos de gerenciamento de TI :
A medida que os ambientes de computação ficam mais complexos aumenta o nível de conhecimento especializado e a experiência exigidos da equipe de gerenciamento da infraestrutura, bem como também os custos associados a ela. As organizações gastam tempo e recursos desproporcionais em tarefas manuais associadas à manutenção de servidores e, portanto, exigem mais pessoas para concluir essas tarefas.

Motivado por esses desafios a virtualização ressurgiu com força total nos anos 2000, habilitando os sistemas x86 à lidar com muitas destas questões transformando-os

em uma infraestrutura de *hardware* compartilhada de uso geral que oferece isolamento, mobilidade e liberdade de escolha do sistema operacional para os ambientes de aplicativos.

Desta forma, pode-se definir virtualização como uma técnica que permite a execução de várias máquinas virtuais isoladas que rodam sistema operacional, aplicações e serviços de redes de forma independente uma das outras, compartilhando recursos de uma única máquina física [Car08].

Atualmente podem-se utilizar diferentes arquiteturas de virtualização dependendo da necessidade de desempenho, do *hardware* ou ainda do tipo de *software* que irá ser utilizado. A seguir apresentaremos com mais detalhes as arquiteturas existentes.

2.1 Arquiteturas de Virtualização

Existem similaridades entre as arquiteturas de virtualização, em que a diferença entre elas está no nível de abstração e nos métodos utilizados para a implementação da virtualização [SMI05]. Neste trabalho abordaremos quatro das principais arquiteturas de virtualização: virtualização total, paravirtualização, virtualização assistida por *hardware* e virtualização no nível do Sistema Operacional.

2.1.1 Virtualização Total

A Virtualização Total algumas vezes é interpretada por emulação de *hardware* por prover uma máquina virtual com emulação completa do *hardware* subjacente. Neste caso, um sistema operacional não modificado é executado utilizando um VMM para receber e traduzir as instruções em tempo real para as máquinas virtuais. Esta técnica requer que cada característica do *hardware* seja devidamente refletida na máquina virtual, incluindo todo o conjunto de instruções de E/S, acesso à memória e todos os demais elementos utilizados por um *software* que fosse executado em uma máquina real [LP10]. Esse processo pode levar a perdas significativas de desempenho. No entanto, novas estratégias estão sendo utilizadas para agregar múltiplas instruções e traduzi-las em conjunto [CCW+08].

Embora a Virtualização Total ainda possua penalidades de desempenho, essa arquitetura permite a execução de sistemas operacionais não modificados, o que é ideal principalmente quando os códigos fonte do sistema não estão disponíveis como os da família de sistemas operacionais da Microsoft.

2.1.2 Paravirtualização

A possibilidade de modificar a *interface* entre o VMM e os sistemas convidados, oferecendo a estes um *hardware* virtual que é similar mas não idêntico ao *hardware* real, denomina-se paravirtualização [LM08]. Na paravirtualização o sistema operacional da máquina virtual tem conhecimento de que está operando em um ambiente virtualizado, tendo o seu código modificado, removendo todas as instruções que poderiam fazer acesso direto aos recursos físicos e que venham a comprometer a integridade das máquinas virtuais, substituindo-as por chamadas explícitas ao VMM [LP10]. O VMM, por sua vez, provê *interfaces* de comunicação para operações críticas de *kernel*, como gerenciamento de memória, controle de interrupções e *time keeping*.

A vantagem dessa arquitetura é que suas máquinas virtuais geralmente superam em nível de desempenho as máquinas virtuais implementadas utilizando a arquitetura de Virtualização Total. A desvantagem está na necessidade de modificar o sistema operacional. Isso acarreta implicações para os sistemas operacionais sem código fonte disponível, tais como Windows 2000/XP/Vista/8, tendo assim, desvantagens de compatibilidade e portabilidade [CCW+08]. Esta arquitetura de virtualização é bastante utilizada devido ao seu desempenho razoável e por suas facilidades de gerenciamento.

2.1.3 Virtualização Assistida por Hardware

Virtualização Assistida por *Hardware* ou Virtualização Nativa utiliza suporte de *hardware* para virtualização, com intuito de auxiliar na redução do impacto da camada de virtualização utilizando novas *features*, diretamente no processador. Estes aprimoramentos, como por exemplo as tecnologias de virtualização da Intel (VT-x) e AMD (AMD-v) [Fis06] privilegiam a execução de instruções de CPU, através de uma tecnologia que permite que o VMM seja executado em um modo que privilegia as chamadas que são automaticamente capturadas pelo VMM, removendo a necessidade de tradução binária ou paravirtualização.

Nessa técnica, o estado da máquina virtual é armazenado no *Virtual Machine Control Structures* (VT-x) para família de processadores Intel ou *Virtual Machine Control Blocks* (AMD-V) para processadores AMD [Fis06].

Esta arquitetura também permite que múltiplos sistemas operacionais não modificados sejam executados em paralelo. Assim, todos os sistemas operacionais são capazes de rodar sobre o mesmo processador, realizando acessos diretamente ao *hardware*, ou seja, sem a emulação de um processador. Isso é diferente da técnica de Virtualização Total, na qual é possível rodar um sistema operacional em um processador emulado, embora, geralmente com degradação do desempenho.

2.1.4 Virtualização em nível do Sistema Operacional

Virtualização em Nível de Sistema Operacional é a forma menos intrusiva de virtualização, ou seja, a camada de virtualização afeta minimamente as aplicações. Ao contrário da Virtualização Total e da Paravirtualização essa forma não depende de um VMM. Em vez disso o sistema operacional é modificado de forma a isolar múltiplas áreas de usuários (*User-Spaces*), em mais alto nível, intituladas *containers*.

Essa arquitetura viabiliza esse cenário por meio de duas entidades: *namespaces* e Gerência de Recursos. Um *namespace* é um ambiente abstrato criado para conter um agrupamento lógico de identificadores únicos ou de símbolos. Um identificador definido em um *namespace* está associado ao próprio *namespace*. O mesmo identificador pode ser definido independentemente em diversos *namespaces*. Um exemplo disso são os diretórios em um sistema de arquivos, que possuem diferentes arquivos em uma mesma unidade de disco.

As tecnologias mais recentes baseadas em *containers* utilizam *namespaces* para introduzir maior isolamento, geralmente providos pelo próprio sistema operacional. Deste modo, assegura-se que diferentes usuários utilizem diferentes espaços no mesmo sistema operacional, ou seja, os processos de um usuário tornam-se incapazes de se comunicar com os processos de outros usuários, garantindo o isolamento do sistema como um todo. Uma fundamentação sobre os principais *namespaces* providos pelo *kernel* do Linux é apresentado a seguir [LWN] :

- ***Namespace* de Sistema de Arquivos:** tem a função de limitar o escopo do sistema de arquivos para um processo ou um grupo de processos. Desta forma, o processo fica restrito à uma subárvore limitada de diretórios e arquivos. No Linux, este *namespace* é implementado pela ferramenta *chroot()*. Na virtualização baseada em *container*, é implementado de forma a restringir os *containers* a determinados diretórios na máquina hospedeira via *cgroups*. Esta abordagem apresenta diversas vantagens como: (i) não necessitar de um dispositivo de bloco separado ou uma partição de disco para cada *container*; (ii) o administrador da máquina hospedeira possui acesso a todos os arquivos em todos os *containers*. Assim, tanto manutenção quanto *backup/restore* tornam-se tarefas triviais; (iii) permite o provisionamento de *containers* em larga escala.

- ***Namespace* PID:** garante que todos os processos em um *container* tenham seus identificadores únicos e o primeiro processo dentro de um *container* possua o PID 1. Desta forma os *containers* só podem visualizar seus próprios processos, não podendo visualizar ou acessar processos em outros *containers*.

- ***Namespace* IPC:** assegura que cada *container* possua o seu próprio System V IPC (Inter- Process Communication), ou seja, seus próprios segmentos de memória compartilhada, semáforos e mensagens.

- ***Namespace* de Rede:** proporciona isolamento de rede garantindo que cada *container* possua seus próprios dispositivos de rede, endereços IP, roteamento, regras de *firewall*, *caches* de rede, números de porta, entre outros.

- ***Namespace* /proc e /sys**: assegura que cada *container* possua sua própria representação do /proc e do /sys, sistemas de arquivos especiais usados para exportar algumas informações do kernel para aplicações. Em poucas palavras, são os subconjuntos de um sistema hospedeiro Linux.

- ***Namespace* de Usuário:** consiste de tabelas de identificadores de usuários (uid), de forma que, o mesmo uid em diferentes *containers* possa se referir a diferentes usuários.

- ***Namespace* UTS:** permite que cada *container* tenha seu próprio *hostname*, além de um nome de domínio NIS (*Network Information Service*). Isto pode ser útil para a inicialização e configuração de *scripts* que adaptam suas ações com base nesses nomes. O termo "UTS"deriva do nome da estrutura passada para a chamada de sistema *uname(): struct utsname*. O nome dessa estrutura, por sua vez deriva de *"UNIX Time-sharing System"*

Desta forma podemos definir um *container* como uma entidade que utiliza todos os *namespaces* necessários em conjunto. Assim sendo, exite apenas um único *kernel* do sistema operacional em execução, que é compartilhado de forma isolado pelos *containers*.

Devido a um modelo único de *kernel*, existe uma única entidade que controla todos os recursos do sistema operacional: o próprio *kernel*. Todos os *containers* compartilham o mesmo conjunto de recursos: CPU, memória, disco e rede. Os recursos para cada *container* não são pré-alocados, eles são apenas limitados. Isso significa que: todos os recursos podem ser alterados dinamicamente, em tempo de execução; se um recurso não é usado, ele fica disponível para outros *containers*.

Essa abordagem possui uma baixa sobrecarga de desempenho em analogia às demais arquiteturas de virtualização. No entanto, incorre em algumas consequências: (i) o mesmo sistema operacional é executado em todos os *containers* e na máquina hospedeira. Apesar desta limitação, diferentes distribuições Linux podem coexistir em diferentes *containers*; (ii) dispensa a necessidade de um *kernel* em execução para sustentar um *container*, resultando em um menor volume de dados comparando com as máquinas virtuais tradicionais; (iii) a pilha de *software* que encontra-se entre o *hardware* e as aplicações do usuários é similar a pilha de *software* de uma arquitetura sem virtualização. Em outras palavras, os

containers possuem um desempenho similar à uma máquina nativa, com um custo mínimo da virtualização (em comparação com as MVs).

A desvantagem desse modelo consiste na falta de isolamento, pois os recursos são compartilhados entre os *containers*, ao contrário dos modelos tradicionais de virtualização em que os recursos são alocados única e exclusivamente a cada nova máquina virtual. Outra desvantagem dessa arquitetura é que todos os *containers* compartilham o mesmo *kernel*. Assim, se o *kernel* do sistema operacional falhar ou ser comprometido, todos os *containers* serão comprometidos. Todavia, ainda pode-se encontrar vantagem nesse ponto pois o fato de se possuir uma instância única do *kernel* do sistema operacional acarreta em menor consumo de recursos.

Vários ambientes que provêm serviços computacionais já utilizam a virtualização em nível de Sistema Operacional. O Mesos [Hin11], um gerenciador de *clusters* que simplifica a complexidade da execução de aplicativos em um *pool* compartilhado de servidores, utiliza *containers* em sua arquitetura. Recentemente a Amazon [AWS] lançou o *AWS Elastic Beanstalk*, um serviço de implementação e gerenciamento de aplicativos na nuvem, com suporte a *containers* através da utilização do Docker [Doc]. O Docker tem se tornado um sistema padrão para a criação e gerenciamento de *containers* na nuvem.

A seguir serão apresentadas as tecnologias de virtualização que se relacionam com o nosso trabalho, a saber o Xen e o LXC *Linux Containers*

2.2 Tecnologias de Virtualização

Atualmente exitem diversas tecnologias que visam suportar as arquiteturas de virtualização vistas na Seção 2.1. Essas tecnologias são compostas por ferramenta de gerenciamento que procuram facilitar a administração dos ambientes virtualizados.

Esta seção apresenta as tecnologias de virtualização que são importantes para o desenvolvimento deste trabalho como veremos a seguir.

Xen

O VMM Xen foi criado por Keir Fraser e Ian Pratt, como parte do projeto de pesquisa *Xenoserver* na Universidade de *Cambridge* no final de 1990. O Xen permite compartilhar uma única máquina para vários clientes rodando sistemas operacionais distintos [BDF+03].

O Xen utiliza o conceito de paravirtualização em que o sistema operacional rodando sobre uma máquina virtual é modificado, tendo a ilusão de estar sendo executado diretamente sobre o *hardware*. O Xen se encarrega de organizar as requisições feitas pelas máquinas virtuais e repassá-las ao sistema principal. Ele se limita a repassar as instru-

ções, sem interpretá-las como faria um emulador, o que minimiza a perda de desempenho oferecida pela camada de virtualização.

Também permite aos usuários aumentar o nível de utilização de servidores por meio da consolidação de grupos de servidores em uma única máquina, reduzindo a complexidade e reduzindo o custo final [BDF+03]. Uma vez que o Xen é uma aplicação de código aberto, ele é amplamente utilizado em ambientes de Nuvem Computacional, como na Amazon AWS [AWS].

LXC - Linux Container

O LXC é um dos mais populares representantes da virtualização em nível de Sistema Operacional. Utiliza *namespaces* do *kernel* do Linux para fornecer isolamento de recursos entre todos os *containers*. Durante a inicialização de um *container*, por padrão, PIDs, IPCs e pontos de montagem são virtualizados e isolados por meio de namespace de PID, de IPC e de sistema de arquivos, respectivamente [XNR+13]

O gerenciamento de CPU e de memória são realizados unicamente por meio de grupos de controle (CGroups) [RED]. O CGroups tem a função de restringir o número de CPUs e limitar a quantidade de memória por *container*, bem como, isolar os processos de diferentes *containers*. A utilização do CGroups, permite aos administradores de sistema terem controle refinado sobre a alocação, prioridades, negação, gerenciamento e monitoramento do sistema, permitindo que os recursos de *hardware* sejam divididos inteligentemente entre os *containers* aumentando a eficiência geral.

Os *containers* no LXC são gerenciados pela ferramenta *lxc-tools*, fornecida pelos desenvolvedores do próprio LXC [Hel09]. Com essa ferramenta é possível criar, congelar ou destruir os *containers* em execução.

Ambas as tecnologias que foram apresentas tem sido amplamente utilizadas em ambientes computacionais na nuvem, conteúdo que será abordado no próximo Capítulo.

3. COMPUTAÇÃO EM NUVEM

Observamos na Seção anterior que o aumento do poder computacional e o uso da virtualização tornaram possível a construção de um ambiente computacional extremamente poderoso e flexível, que hoje conhecemos por computação em nuvem. A seguir serão abordados o conceito de computação em nuvem, assim como suas principais características e modelos de implantação.

O termo computação em nuvem, (*cloud computing*) tem se difundindo amplamente na indústria da tecnologia da informação, tornando-se uma metáfora para a internet ou infraestrutura de comunicação entre os componentes arquiteturais, baseada em uma abstração que oculta a complexidade de infraestrutura [SMM09].

O NIST *(National Institute of Standards and Technology)* [Mel11] afirma que a computação em nuvem é um paradigma em evolução. Suas definições, casos de uso, tecnologias, problemas, riscos e benefícios serão redefinidos em debates entre os setores público e privado e essas definições, atributos e características evoluirão com o tempo.

Ainda não se tem uma definição amplamente aceita. O NIST apresenta a seguinte definição para computação em nuvem: “Computação em nuvem é um modelo que possibilita acesso, de modo conveniente e sob demanda, a um conjunto de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente adquiridos e liberados com mínimo esforço gerencial ou interação com o provedor de serviços”.

Ainda Armbrust et al.[Arm09] propõe a seguinte definição: “A computação em nuvem é um conjunto de serviços de rede ativados que proporcionam escalabilidade, qualidade de serviço e infraestrutura barata de computação sob demanda, que pode ser acessada de uma forma simples e pervasiva”.

A computação em nuvem pode ser implementada com a finalidade de oferecer diferentes níveis de serviço. Essa implementação se dá através dos Modelos de Serviço que apresentaremos a seguir.

3.1 Modelos de Serviço

Os modelos de serviço são importantes na medida que definem um padrão arquitetural para soluções de computação em nuvem. Pode-se ter três modelos de serviços: SaaS, PaaS e IaaS que serão descritos em mais detalhes a seguir.

3.1.1 *Software* como Serviço (SaaS)

O modelo SaaS disponibiliza *softwares* com propósitos específicos, que estão acessíveis aos usuários a partir de diversos tipos de dispositivos através da Internet. No SaaS, com exceção de algumas configurações específicas, o usuário não administra ou controla a infraestrutura subjacente, incluindo rede, servidores, sistemas operacionais, armazenamento ou mesmo as características individuais da aplicação. Esta abordagem permite que os desenvolvedores se concentram em inovação e não na infraestrutura, possibilitando desta forma um desenvolvimento mais ágil destes *softwares*.

Como o *software* está disponível pela Internet, pode ser acessado pelos usuários de qualquer lugar e a qualquer momento, permitindo maior integração entre unidades de uma mesma empresa ou outros serviços de *software*. Desta forma, novos recursos podem ser incorporados automaticamente à estes sistemas sem que os usuários percebam estas ações, tornando transparente sua atualização. Como exemplos de SaaS podemos destacar os serviços da Microsoft com o Office 365, o Google Docs e o Dropbox.

3.1.2 Plataforma como Serviço (PaaS)

O PaaS oferece uma infraestrutura de alto nível de integração para implementar e testar aplicações na nuvem. O usuário não administra ou controla a infraestrutura subjacente incluindo rede, servidores, sistemas operacionais ou armazenamento, mas tem controle sobre as aplicações implantadas, inclusive podendo realizar configurações das aplicações hospedadas nesta infraestrutura. A PaaS fornece um sistema operacional, linguagens de programação e ambientes de desenvolvimento para as aplicações, auxiliando na implementação de *softwares*, já que contém ferramentas de desenvolvimento e colaboração entre desenvolvedores.

Os desenvolvedores tem a sua disposição ambientes escaláveis e flexíveis. Todavia existem restrições sobre o tipo de *software* que se pode desenvolver devido a limitações que o ambiente impõe na concepção das aplicações até a restrições quanto a utilização de determinados tipos de SGBDs. Do ponto de vista do negócio o PaaS permite a melhora do gerenciamento do trabalho e das responsabilidades das equipes de TI das empresas. Como exemplo de PaaS podemos destacar o Google App Engine [Goo].

3.1.3 Infraestrutura como Serviço (IaaS)

O principal objetivo do IaaS é tornar mais fácil e acessível o fornecimento de recursos tais como servidores, rede, armazenamento e outros recursos de computação fundamentais para construir um ambiente sob demanda que podem incluir sistemas operacionais e aplicativos.

A IaaS possui algumas características, tais como uma *interface* única para administração da infraestrutura como *hosts*, *switches*, balanceadores de carga, roteadores e o suporte para a adição de novos equipamentos de forma simples e transparente. Em geral, o usuário não administra ou controla a infraestrutura da nuvem, mas tem controle sobre os sistemas operacionais, armazenamento e aplicativos implantados, e, eventualmente, seleciona componentes de rede, tais como *firewalls*.

O termo IaaS se refere a uma infraestrutura computacional baseada em técnicas de virtualização de recursos de computação. Esta infraestrutura pode escalar dinamicamente, aumentando ou diminuindo os recursos de acordo com as necessidades das aplicações. Do ponto de vista de um melhor aproveitamento da infraestrutura, ao invés de comprar novos servidores e equipamentos de rede para a ampliação de serviços, pode-se aproveitar os recursos disponíveis e adicionar novos servidores virtuais à infraestrutura existente de forma dinâmica. O Amazon Elastic Cloud Computing (EC2) [AWS] é um exemplo de IaaS.

A seguir analisaremos as principais características que em conjunto, definem exclusivamente a computação em nuvem e fazem distinção com outros paradigmas.

3.2 Características

Podemos definir as características do ambiente de nuvem como vantagens que este ambiente oferece. A seguir analisaremos as cinco principais características apontadas como pelo NIST [Mel11] como essenciais para se compor um ambiente computacional na nuvem.

- ***Self-service* sob demanda:** Permite que o usuário adquira unilateralmente recursos computacionais na medida em que necessite e sem precisar da interação com os provedores serviço. O *hardware* e o *software* dentro de uma nuvem podem ser automaticamente reconfigurados sendo que estas modificações são apresentadas de forma transparente para os usuários, que possuem perfis diferentes e assim podem personalizar os seus ambientes computacionais.
- **Amplo acesso:** Os recursos são disponibilizados através da rede e acessados através de mecanismos padronizados que possibilitam o uso por plataformas, tais como

celulares, *laptops* e *tablets*. Os *softwares* clientes instalados localmente para o acesso à nuvem geralmente são leves e fáceis de utilizar.

- ***Pooling* de recursos:** Os recursos computacionais do provedor são organizados em um *pool* para servir múltiplos usuários usando um modelo *multi-tenant*, com diferentes recursos físicos e virtuais, dinamicamente atribuídos e ajustados de acordo com a demanda dos usuários. Estes usuários não precisam ter conhecimento da localização física dos recursos computacionais, podendo somente especificar a localização em um nível mais alto de abstração, tais como o país, estado ou centro de dados.
- **Elasticidade rápida:** Recursos podem ser adquiridos de forma rápida e elástica, caso haja a necessidade de escalar devido o aumento da demanda, e liberados, na diminuição dessa demanda. Para os usuários, os recursos disponíveis para uso parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer momento.
- **Serviço medido:** Sistemas em nuvem automaticamente controlam e otimizam o uso de recursos por meio de uma capacidade de medir estes serviços. A automação é realizada em algum nível de abstração apropriado para cada tipo de serviço. O uso de recursos pode ser monitorado e controlado, possibilitando transparência para o provedor e o usuário do serviço utilizado.

Este conjunto de características define um ambiente altamente escalável, seguro e disponível, que é possibilitado fortemente pela camada de virtualização e suas características, tendo como ponto chave para este trabalho a capacidade de prover um ambiente elástico, onde é possível aumentar ou diminuir a quantidade de recursos de uma máquina virtual, de acordo com as flutuações da carga de trabalho em execução. Na próxima Seção esta característica será abordada com mais detalhes.

3.3 Elasticidade

Um dos principais atrativos para a utilização da Computação em Nuvem consiste em sua capacidade de adquirir recursos de maneira dinâmica. Do ponto de vista do consumidor, a nuvem parece possuir recursos infinitos, pois ele pode obter mais ou menos recursos computacionais conforme a necessidade de suas aplicações [CSGS13]. Por sua vez, o provedor de serviços de nuvem é responsável pela aquisição, gestão e manutenção da infraestrutura de nuvem. Além disso, também deve fornecer os mecanismos para permitir o acesso aos serviços de elasticidade. Algumas plataformas de nuvem oferecem *interfaces* e *APIs* que permitem aos usuários gerenciar seus recursos manualmente [GdB12].

Comumente, o termo elasticidade é usado como sinônimo de escalabilidade, no entanto, eles são conceitos diferentes e nunca devem ser usados como sinônimos. A es-

calabilidade é a capacidade do sistema crescer a um tamanho que era esperado para acomodar um aumento futuro ou a sua capacidade para melhorar o rendimento, quando são adicionados novos recursos [SSRA+11], [AEADE11]. Em uma nuvem escalável é possível adicionar recursos sempre que a demanda aumenta, com a finalidade de manter as aplicações dentro de um nível de qualidade estabelecido. Quanto à aplicação, essa é dita escalável quando sua eficácia é mantida a medida que a quantidade de recursos e o tamanho do problema são aumentados proporcionalmente. A escalabilidade de uma aplicação reflete sua capacidade em fazer uso dos recursos disponíveis com eficácia [KHvKR11].

Todavia, segundo o NIST, elasticidade é a habilidade de rápido provisionamento e desprovisionamento, com capacidade de recursos virtuais praticamente infinita e quantidade adquirível sem restrição a qualquer momento [Mel11].

Taurion [TAU], define a elasticidade como a capacidade do ambiente computacional da nuvem aumentar ou diminuir de forma automática os recursos computacionais demandados e provisionados para cada usuário.

Ainda, Han et al. [HGG+14] define a elasticidade como a habilidade de escalar recursos de maneira adaptável para atender à variação da demanda das aplicações.

Desta forma, podemos concluir que a elasticidade diz respeito à capacidade proativa ou reativa para aumentar ou diminuir recursos de um serviço em tempo de execução. Podemos observar que a elasticidade proposta por ambientes de nuvem permite uma escalabilidade adaptativa de recursos, para mais ou menos, a qualquer momento, sem nenhum compromisso de longo prazo. Ao longo desta Seção vamos analisar a elasticidade quanto a suas modalidades e políticas de alocação de recursos A seguir apresentaremos as modalidades existentes de elasticidade

3.3.1 Modalidade

Ambientes de nuvem permitem elasticidade em dois sentidos:

- *scale-up:* ou elasticidade vertical, geralmente consiste em adicionar mais processadores ou memória, sobre um mesmo nodo aumentando sua capacidade. As vantagens deste modelo consistem em um menor consumo de energia do que aplicações executando em uma quantidade maior de nodos, custos com resfriamento são reduzidos devido à menor quantidade de nodos necessários, menor complexidade para sua implementação, menor custo de licenças, e geralmente utiliza uma menor taxa dos dispositivos de rede. Por outro lado, existem pontos únicos de falha.
- *scale-out:* ou elasticidade horizontal, geralmente se refere à adição de mais nodos ao ambiente. A vantagem deste tipo de proposta consiste em uma escalabilidade praticamente infinita, embora existam limitações de *software* ou outros atributos da

infraestrutura. Como vantagem, *scale-out* fornece balanceamento de carga e disponibilidade. Por outro lado, existe uma grande emissão de gases pelo *data center*, aumentam os custos de eletricidade e resfriamento, é necessária uma maior quantidade de equipamentos de rede, além de que as aplicações que se utilizam destes ambientes são mais complexa devido à adaptabilidade necessária.

Assim, quando o número de clientes que acessam um determinado serviço aumenta, a aplicação pode utilizar a totalidade dos recursos do nodo. Quando existe a necessidade de recursos acima da capacidade do nodo, novas instâncias sobre novos nodos são lançadas com a finalidade de manter a qualidade do serviço. Isso significa que os novos recursos físicos são alocados e se tornam parte do *pool* de recursos disponíveis. Quando o número de clientes diminui, esses recursos são liberados. Assim, este tipo de aplicação deve ser projetada para escalar a partir de um único servidor, até um conjunto de milhares de máquinas.

3.3.2 Política de Alocação de Recursos

Quanto a política de alocação de recursos, Galante e Bona [GdB12] apresentam esse item em duas abordagens: manual ou automática. A política manual exige a intervenção do usuário, enquanto a automática ainda divide-se em outras duas: reativa e proativa. A abordagem reativa é marcada pelo emprego do mecanismo regra-condição-ação, como pode ser visto na Figura 3.1.

Uma regra é composta por um conjunto de condições que quando satisfeitas, disparam ações de elasticidade na nuvem. Cada condição considera um evento ou uma métrica do sistema que é comparado com um limiar. Um limite superior é usado para a alocação, enquanto o inferior é útil para a consolidação de recursos [BMVO12] [RBA11]. As informações sobre os valores de métricas e eventos geralmente são fornecidas pelo o sistema de monitoramento da infraestrutura da nuvem ou pela própria aplicação do usuário. Já a abordagem proativa usa técnicas de predição para antecipar o comportamento de carga da aplicação e, assim, decidir pelas ações de elasticidade.

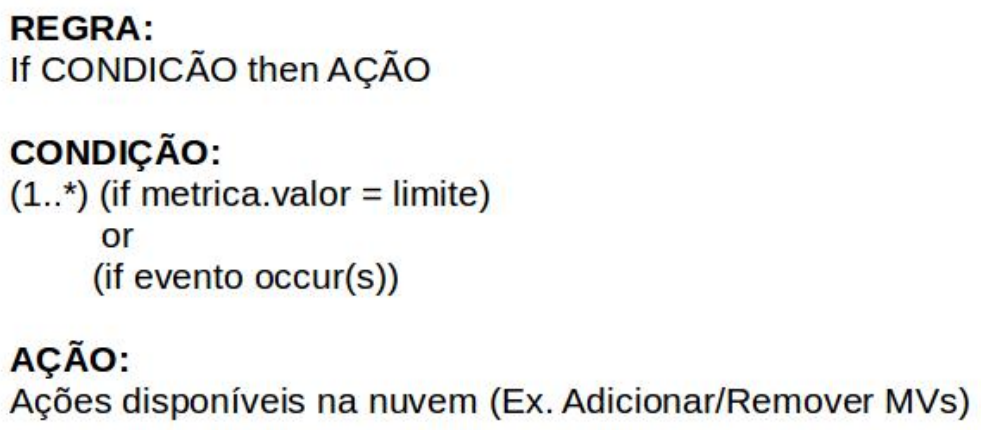

Figura 3.1 – Regra-Condição-Ação

3.3.3 Estratégias

Ainda conforme Galante e Bona [GdB12], podem-se classificar as estratégias de elasticidade na nuvem em dois grupos: replicação e redimensionamento.

A replicação consiste em adicionar ou remover instâncias virtuais sobre o ambiente de nuvem. Atualmente é o método mais utilizado para proporcionar elasticidade, sendo utilizado na maioria dos provedores de nuvem [DTM12], e discutido em diversos trabalhos acadêmicos [LBCP09] [MLS10] [CVKB12] [FPG10] [SSGW11]. Para apoiar a elasticidade de alguns tipos de aplicações, plataformas de nuvem como a Amazon [AWS] e Azure [MSA] oferecem recursos adicionais como balanceamento de carga, com a finalidade de dividir a carga entre as várias réplicas.

No que se refere ao redimensionamento, esta estratégia engloba alterações de configuração de recursos, em que é possível alterar o nível de utilização de CPU, largura de banda, disco ou memória de uma ou mais máquinas virtuais [GdB12]. Essa abordagem é diferente da replicação, na qual adaptações do serviço não são realizadas.

No próximo Capítulo apresentaremos as principais soluções para a computação em nuvem e como elas apresentam a elasticidade para seus usuários.

4. ESTADO DA ARTE

Como podemos ver anteriormente, a computação em nuvem proporciona um ambiente dinâmico que provê acesso aos recursos sob demanda. Essa capacidade de provisionamento tem chamado a atenção tanto da academia quanto da indústria. Na Seção a seguir analisaremos como as principais plataformas de nuvem tratam a elasticidade.

4.1 As Plataformas de Nuvem e a Elasticidade

Os provedores de nuvem como *Amazon AWS* e *Windows Azure*, oferecem a elasticidade como um serviço denominando *Auto Scaling* que permite iniciar ou encerrar automaticamente instâncias virtuais com base em políticas definidas pelo usuário. Os recursos podem ser dimensionados de forma dinâmica ou estática: dinamicamente com base nas condições especificadas pelo usuário; ou estaticamente de acordo com um cronograma definido pelo usuário. [AWS]

A *Amazon AWS*, uma dos mais tradicionais provedores de nuvem, disponibiliza uma ferramenta gráfica na qual o usuário escolhe uma métrica para ser monitorada e define limites para tal métrica. Quando esta métrica atingir o limite pré-definido, uma ação de elasticidade será executada. O monitoramento destas métricas é realizado pelo *CloudWatch* [CLO] e incluem monitoramento de uso de CPU, tráfego de rede e disco. O *CloudWatch* também inclui o monitoramento dos balanceadores de carga que são utilizados para distribuir a carga de trabalho entre as instâncias virtuais ativas. A elasticidade de fato é gerida de forma reativa por outra ferramenta denominada *AutoScaling* [AUT].

No Windows Azure [RG11] é oferecida uma biblioteca *AutoScaling Application Block* a qual o usuário deve adicionar ao *core* de sua aplicação proporcionando suporte à elasticidade. Além dessa opção em nível de programação, o Windows Azure também oferece uma ferramenta que por meio de uma interface gráfica permite a alteração do número de instâncias associadas a um determinado serviço. Ainda o Azure conta com o AzureWatch [Azu] uma ferramenta que permite controlar o número de instâncias. Ele é um arcabouço adaptado para o Windows Azure que trabalha com informações de uso de CPU, dados das máquinas virtuais, histórico, tempo de requisições, e também é dirigido por regras explícitas de atuação.

A RackSpace, provedor de nuvem que utiliza um ambiente suportado pelo *Openstack* tem as métricas monitoradas pelo *Ceilometer* [OPE] e disponibiliza uma ferramenta gráfica na qual o usuário define as regras que irão disparar as ações de elasticidade.

O Nimbus [NIM] utiliza o Phantom [PHA] para monitorar o comportamento dos recursos e disparar de forma reativa ações de elasticidade. O usuário deve configurar o

Phantom completando parâmetros como: (i) número máximo e mínimo de MVs; (ii) número de MVs que serão lançadas quando um limite superior for atingindo e; (iii) número de MVs que serão consolidadas quando um limite inferior for atingido.

GoGrid [GOG] implementa mecanismos de replicação mas ao contrário da Amazon e o do Windows Azure, não possui serviços de elasticidade automáticos nativos. O GoGrid oferece uma API para controlar a quantidade de máquinas virtuais instanciadas, deixando por conta do usuário a implementação de mecanismos automatizados mais elaborados. Para superar a falta de mecanismos automatizados, ferramentas como RightScale [Rig] e Scalr [Sca] têm sido desenvolvidas.

RightScale [Rig] é uma plataforma de gerenciamento que fornece controle e capacidades de elasticidade para diferentes provedores de nuvem pública (Amazon, Rackspace, GoGrid, entre outros) e também para soluções de nuvem privada (CloudStack, Eucalyptus e OpenStack). A solução fornece mecanismos automáticos reativos com base em um *Daemon* cuja função é monitorar as filas de entrada para instanciar MVs, visando processar os trabalhos na fila, dependendo de diferentes métricas pré-estabelecidas.

Scalr [Sca] é um projeto de código aberto cujo objetivo é oferecer soluções de elasticidade para aplicações *web*. O Scalr é compatível com várias soluções para nuvens, tais como, Amazon, Rackspace e Eucalyptus. Atualmente suporta Apache e ngnix, banco de dados MySQL, PostgreSQL, Redis e MongoDB. Da mesma forma, RightScale utiliza métricas tanto de *hardware* como de *software* para disparar ações de elasticidade.

A soluções para nuvem vistas até aqui oferecem subsistemas parametrizáveis para monitoramento de serviços e gerenciamento da elasticidade. Os usuários devem criar as regras e definir os limites superior e inferior de uma métrica a ser monitorada, bem como as condições e as ações de elasticidade que devem ser executadas [Rig13].

Todavia, a criação destas regras não é uma tarefa trivial uma vez que o usuário, na maioria das vezes, não sabe ao certo qual o exato comportamento da aplicação e quais os recursos mais impactam sobre ela. Com a finalidade de amenizar esse problema, várias iniciativas da academia têm pesquisado maneiras de automatizar esse processo. Veremos alguns destes trabalhos na Seção a seguir.

4.2 Iniciativas da Academia

Ao analisar os trabalhos da academia relacionados com nosso trabalho, podemos agrupá-los segundo ao método de operação. Algumas soluções são preditivas, propondo diversas estratégias para estimar com antecedência o comportamento das cargas de trabalho, aumentando ou diminuindo a quantidade de recursos de uma máquina virtual sempre que necessário. Outras reativas, apenas reagem as mudanças da carga de trabalho. Ainda,

apresentamos soluções híbridas, reativas e preditivas, realizando predições de carga de trabalho e reagindo, o mais rápido possível, à demanda por recursos.

Como soluções preditivas relacionadas a nossa proposta, alguns trabalhos atuais são encontrados na literatura como o de Zhenhuan et al. [GGW10], Kalyvianaki et al. [KCH09], e Ali-Edin et al. [AETE12].

O PRESS, trabalho desenvolvido por Zhenhuan et al. [GGW10], emprega uma abordagem para captar padrões de curto prazo sobre a demanda de recursos, usando cadeias de Markov para prever futuras demandas. Esses modelos de previsão de recursos são atualizados à medida que os padrões de utilização de recursos mudam. O trabalho foi implementado sobre o Xen, utilizando o *benchmarks* Rubis e realiza redimensionamento no uso de CPU como ação de elasticidade. Os experimentos mostram uma boa precisão na previsão de recursos futuramente necessários, com menos de 5% de erro de sobre-estimação e quase zero de erro de subestimação.

O trabalho desenvolvido por Kalyvianaki et al. [KCH09] integra o filtro de Kalman em controladores para alocar dinamicamente os recursos de CPU em máquinas virtuais. A utilização de CPU é monitorada e a quantidade deste recurso é provisionada e atualizada dinamicamente. A solução foi desenvolvida utilizando a plataforma Xen e como carga de trabalho o *trace* do servidor *web* que hospedou o sistema da copa do mundo de 1998. Como resultado, foram utilizados 15% a menos de recursos.

Ali-Edin et al. [AETE12] propôs um controlador de elasticidade horizontal adaptável, em que o serviço de nuvem é modelado utilizando teoria das filas, e a carga de trabalho é estimada permitindo controles preditivos. Os experimentos foram realizando em ambiente simulado utilizando o mesmo *workload* de [KCH09], com uma economia de recursos em torno de 15% a menos de recursos.

Como soluções reativas, foram encontrados alguns trabalhos relacionados com essa proposta, como os de Dawoud et al. [DTM12] e Marshall et al. [MKF10].

Dawoud et al.[DTM12] apresenta uma arquitetura elástica para o gerenciamento dinâmico de recursos e otimização de aplicações em ambiente virtualizado utilizando o Xen. O trabalho implementou três atuadores: de CPU, Memória e Aplicação. Esses atuadores executados em paralelo garantem a eficiente alocação de recursos e otimização do desempenho da aplicação, reorganizado as máquinas virtuais dinamicamente sobre os recursos disponíveis. Os experimentos tinham por objetivo minimizar a quebra de SLA (tempo de resposta) de um servidor Apache e validar a eficiência do controlador de aplicação desenvolvido neste trabalho. Os resultados comprovam a eficiência da arquitetura elástica, onde quando comparado a um sistema tradicional de alocação, praticamente não houveram quebras de SLO.

O trabalho desenvolvido por Marshall et al. [MKF10] propôs um modelo elástico reativo capaz de responder às mudanças na demanda de recursos, buscando o melhor

aproveitamento dos mesmo sobre um ambiente de nuvem. A solução foi desenvolvida para a plataforma Nimbus [NIM] utilizando o Xen como virtualizador. Foram criadas três políticas de alocação de recursos, sendo que a elasticidade se deu por meio da replicação de máquinas virtuais.

Também encontrou-se na literatura trabalhos que apresentam abordagens híbridas como os de Zhiming et al. [SSGW11], Urgaonkar et al. [USC+08], e Brasileiro et al. [MBL+13].

Zhiming et al. [SSGW11] apresentou o *CloudScale*, um sistema de escalonamento de recursos elástico que visa satisfazer exigência de SLA dos aplicativos em execução na nuvem, utilizando o mínimo de recursos (com foco em economia de energia). Para tal fim, foi desenvolvido um algoritmo para a predição de demanda. Este algoritmo utiliza uma abordagem híbrida, utilizando uma *Fast Fourier Transform* (FTT) para identificar padrões de comportamento da carga de trabalho, denominados assinaturas. Se uma assinatura for descoberta, um modelo de predição é utilizado para estimar futuras demandas por recursos. Caso contrário, o modelo de predição utiliza uma cadeia de Markov para prever a necessidade de recursos em um futuro próximo. Além disso, foi desenvolvido um modulo de correção da predição, que adiciona um valor extra à predição de recursos a fim de evitar subestimação. Quanto às técnicas de elasticidade, o *CloudSale* utiliza principalmente a replicação , além da modificar a frequência das CPUs usando DVFS [XNR+13]. O *CloudScale* foi implementado sobre a plataforma Xen, e como carga de trabalho foram utilizados o *benchmark* Rubis [RUB] e uma aplicação *MapReduce*. Os resultados experimentais mostraram que *CloudScale* minimizou em 83% as quebras de SLO quando comparado a outras soluções. Quanto à economia de energia. o *CloudScale* trouxe economia entre 8 a 10% no consumo total de energia.

Urgaonkar et al. [USC+08] mostrou uma nova técnica provisionamento dinâmico para aplicações *web* com múltiplas camadas, empregando um modelo de filas flexível para determinar a quantidade de recursos a ser alocados para cada camada, e uma combinação de métodos preditivos e reativos para determinar quando da provisão destes recursos. Os experimentos foram realizados em um *pool* de quarenta máquinas físicas utilizando Xen, com a execução do *benchmark* Rubis. Este trabalho concentra-se na elasticidade horizontal, provendo recursos através da alocação/desalocação de instâncias virtuais. Os resultados dos experimentos mostraram resultados satisfatórios quanto à eficiência do provisionamento, mantendo metas de SLA pré-definidas, como por exemplo, tempo de resposta.

Brasileiro et al. [MBL+13] apresentou um *framework* para o provisionamento de recursos flexível e não intrusivo, que emprega abordagens tanto reativas, quanto preditivas. Baseou-se no uso de um conjunto configurável de preditores de demanda dos serviços, além de um mecanismo de seleção que decide periodicamente o melhor preditor a ser empregado. O trabalho também propôs correção sobre predições subestimadas, otimizando as decisões de alocação de recursos e reduzindo o número de quebras de SLA. Os ex-

perimentos foram realizado através de simulações com *traces* de utilização de clientes da HP. Os resultados mostraram que é possível obter uma economia de até 37% de recursos, enquanto a probabilidade de quebra de SLA é mantida na média de 0.008%.

Tabela 4.1 – Relação dos Trabalhos Acadêmicos.

Método	Autor	VMM	Estratégia	Workload	Solução
Preditivo	Zhenhuan et al.	XEN	Redimensionamento	e-Commerce	Predição com base em processamento de sinais e aprendizagem estatística
	Kalyvianaki et al.	XEN	Redimensionamento	App Web	Filtros de Kalman
	Ali-Eldin et al.	XEN	Replicação	App Web	Teoria das filas
Reativo	Dawoud et al.	XEN	Redimensionamento	App Web	Arquitetura para sistema elástico - Tunning da aplicação
	Marshall et al.	XEN	Replicação	HPC	Gerenciador de recursos Nimbus.
Híbrido	Zhiming et al.	XEN	Redimensionamento	e-Commerce	Correção de predição,-,DVFS
	Urgaonkar et al.	XEN	Replicação	App Web	Mecanismo de provisão de recursos
	Morais et al.	XEN	Replicação	App Web	Arcabouço para provisionamento automático de recursos

Ao analisar os trabalhos vistos na tabela 4.1 podemos verificar que existem uma série de desafios para pesquisa no que se refere à alocação dinâmica de recursos. Podemos citar entre outros, a questão do *overhead* das soluções: alocação de recurso de maneira preditiva geralmente é bastante intrusiva, causando alta sobrecarga sobre o ambiente, ao passo que soluções reativas podem acarretar em aumento no custo operacional devido às seguidas mudanças na configuração do ambiente.

Quanto ao controle de elasticidade, a configuração manual de limites físicos para controle de métricas de QoS permitem uma alta quantidade na quebra de SLAs [Rig13]. No entanto, limites dinâmicos podem apresentar situações de *thrashing* em picos de processamento, em que seguidas mudanças no ambiente, ao invés de proporcionar a redução dos custos, implicam em justamente o contrário. Em poucas palavras, soluções que reconfiguram o ambiente de forma lenta podem afetar diretamente o desempenho das aplicações, enquanto soluções rápidas de realocação de recursos causam sobrecarga ao ambiente.

Alguns dos trabalhos analizados utilizam como estratégia de elasticidade, a replicação. Entendemos que esta abordagem oferece algumas limitações: (i) concentradas em aplicações *web*, não se aplicam à aplicações com outros comportamentos; (ii) dependem de um balanceador de carga, que pode ser um ponto único de falha; (iii) admitem uma máquina virtual como unidade de elasticidade, e não a quantidade de recursos disponíveis no nodo físico.

Ainda as pesquisas atuais, em sua maioria, utilizam como carga de trabalho aplicações *web*, como sistemas de comércio eletrônico sobre servidores *web*. Desta forma, este trabalho se diferencia dos demais, no que tange às características da aplicação, pois utilizaremos aplicações de banco de dados.

Embora a implantação de aplicações de banco de dados em ambientes de nuvem venham reduzir o custo de armazenamento e melhorar o acesso aos dados, existe a necessidade de que esses sistemas possam escalar quando é necessário, com o objetivo de garantir operações consistentes e confiáveis considerando cargas de trabalho máximas [SMMM10]. Conforme Aboulnaga et al. [ASS+09] e Agrawal et al. [AAB+08], técnicas adaptativas e dinâmicas deverão ser desenvolvidas para tornar os SGBDs em nuvem viáveis, além de ferramentas que automatizem tarefas tais como a alocação de recursos, vindo essa necessidades ao encontro deste trabalho.

Outrossim, todas as soluções apresentadas pela academia utilizam o modelo tradicional de virtualização, o que impõe a estas soluções limitações quanto ao desempenho e a alocação de recursos. Com intuito de superar estas limitações, este trabalho propôs a utilização de técnicas de virtualização em nível de Sistema Operacional.

Dentre os trabalhos analisados, nos chamou a atenção a arquitetura elástica proposta por Dawoud et al., por se tratar de uma solução que alcançou resultados satisfatórios, além de apresentar grande potencial de adaptabilidade à outros tipos de aplicações. Desta forma, acreditamos que a aplicação da técnica de virtualização em nível de Sistema Operacional sobre esta arquitetura irá minimizar as limitações quanto a alocação de recursos, limitações estas que estão relacionadas com a solução de virtualização utilizada por Dawoud et al., a saber o Xen.

Portanto, este trabalho consiste em uma avaliação do impacto do uso da virtualização em nível de Sistema Operacional sobre a estratégia de alocação dinâmica de recursos proposta por Dawoud et al. [DTM12], com foco no uso de aplicações de banco de dados, capaz de ajustar os recursos conforme as flutuações da carga de trabalho, com a finalidade de manter a qualidade do serviço dentro de níveis desejados. No Capítulo a seguir abordaremos com mais detalhes a arquitetura de Dawoud et al. e as modificações que foram necessárias para adapta-la ao uso de *containers*

5. APRIMORANDO A ELASTICIDADE COM VIRTUALIZAÇÃO EM NÍVEL DE S.O

Este Capítulo explora a arquitetura desenvolvida por Dawoud et al. [DTM12] bem como a sua adaptação para a utilização da virtualização em nível de sistema operacional. Primeiramente, na Seção 5.1 são abordados os fatores que nos motivaram a realizar esse trabalho; na Seção 5.2 é apresentada a arquitetura desenvolvida por Dawoud et al.; finalizando, na Seção 5.3 é apresentada a nova proposta modificada para a utilização de *containers*.

5.1 Motivação

A computação em nuvem oferece a seus usuários fácil acesso a infraestrutura computacional a um custo relativamente baixo. A elasticidade, umas das principais características da nuvem, por sua vez permite que recursos computacionais sejam adquiridos a medida que são necessários, podendo ser desalocados quando não são mais necessários.

Todos os serviços de nuvem analisados no Capítulo 4.1 possuem serviços elásticos, porém a sua utilização depende da criação de regras, o que não consiste em uma tarefa trivial [Rig13]. Saber a real necessidade de recursos computacionais para aplicações de comportamento arbitrário, como são muitas das aplicações atuais, com o objetivo de criar regras eficientes é uma tarefa praticamente impossível por parte do usuário, pois vários fatores podem impactar nessa decisão, como compartilhamento de recursos pelo provedor de nuvem, picos de processamento momentâneo versus uma alta carga de requisições durante um grande período, abstração do *hardware* real, entre outros [EBMD10]. Desta forma inúmeros trabalhos da academia, como os que vimos no Capítulo 4.2, têm procurado soluções para este problema propondo serviços de alocação dinâmica de recursos.

A alocação dinâmica de recursos diz respeito a alocação ou desalocação dos recursos para as máquinas virtuais sem a interferência do usuário do serviço. Por exemplo, pode existir uma maior demanda por processamento do que foi alocado inicialmente pelo usuário. Por outro lado, o usuário pode ter realizado uma alocação superestimada de recursos levando ao desperdício dos mesmos. Basicamente, uma arquitetura elástica pretende transformar a situação exemplo mostrada na Figura 5.1 para a situação mostrada na Figura 5.2. Na Figura 5.1, as máquinas virtuais 1 até 5 possuem disponíveis os mesmos limites máximos de recursos, mostrados pela linha vermelha.

Como pode ser visto na Figura 5.1, as máquinas virtuais 3 e 5 necessitam de mais recursos do que possuem, enquanto as máquinas virtuais 2 e 4 não estão utilizando todos

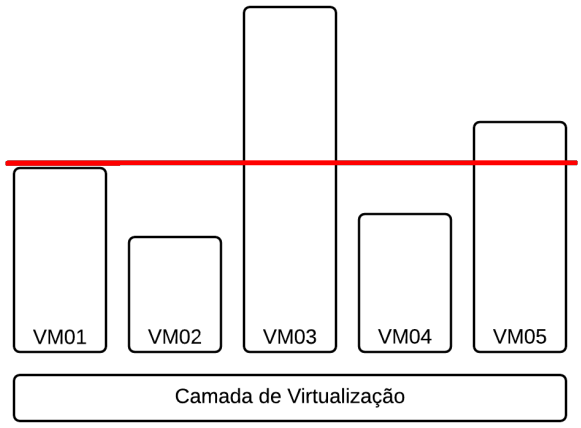

Figura 5.1 – Sistema de Alocação Fixo

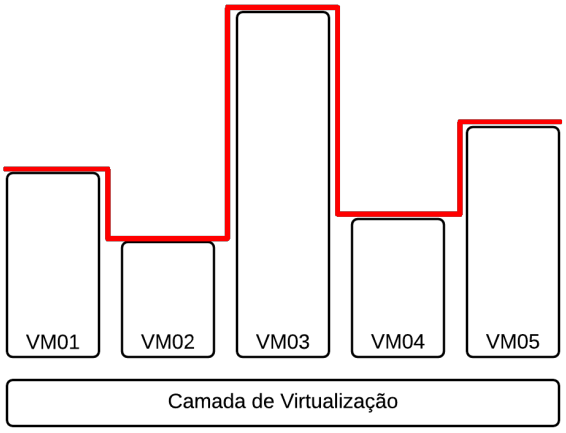

Figura 5.2 – Sistema de Alocação Dinâmico

os recursos que lhe foram disponibilizados. A arquitetura elástica irá redistribuir a alocação de recursos conforme mostrado na Figura 5.2.

Portanto, o objetivo da alocação dinâmica consiste em que todas as máquinas tenham os recursos que necessitem para atender determinados requisitos de qualidade, e que não existam recursos superutilizados ou subutilizados. Dentre as propostas da academia analisadas no Capítulo 4.2, nos chamou atenção a proposta desenvolvida por Dawoud et al. [DTM12].

A arquitetura de Dawoud et al. [DTM12] foi escolhida como base para este trabalho, por trazer uma série de benefícios para o usuário tais como: (i) facilidade de uso, pois não são necessárias análises prévias do comportamento da aplicação e tão pouco conhecimento avançado do ambiente para a configuração das ações de elasticidade; (ii) redução de custos, através de um melhor aproveitamento dos recursos alocados, pois esta arquitetura permite o uso mais otimizado dos recursos, evitando que ações de elasticidade venham

acontecer sem a real necessidade; (iii) manter o desempenho do serviço em execução em níveis aceitáveis.

Apesar dos benefícios apresentados, a solução desenvolvida por Dawoud et al. apresenta limitações quanto a alocação de recursos. Essas limitações estão relacionadas com a solução de virtualização utilizada por ele em sua arquitetura, a saber o Xen.

Como podemos ver na Figura 5.3 (a), em que uma vez definida a fatia de recursos para uma determinada máquina virtual, a amplitude de redimensionamento fica limitada a essa fatia fixa de recursos. Esta limitação é um fator crítico, pois caso a aplicação do usuário necessite de uma quantidade maior de recursos do que o destinado a ela na inicialização, visando manter um determinado SLA, a arquitetura elástica de Dawoud et al. não conseguirá alocar os recursos necessários.

A utilização da virtualização em nível de Sistema Operacional, irá resolver esse problema , pois em ambientes de *container* como mostra a Figura 5.3 (b), essa limitação não existe, sendo possível expandir um determinado *container* até a totalidade de recursos do nodo físico.

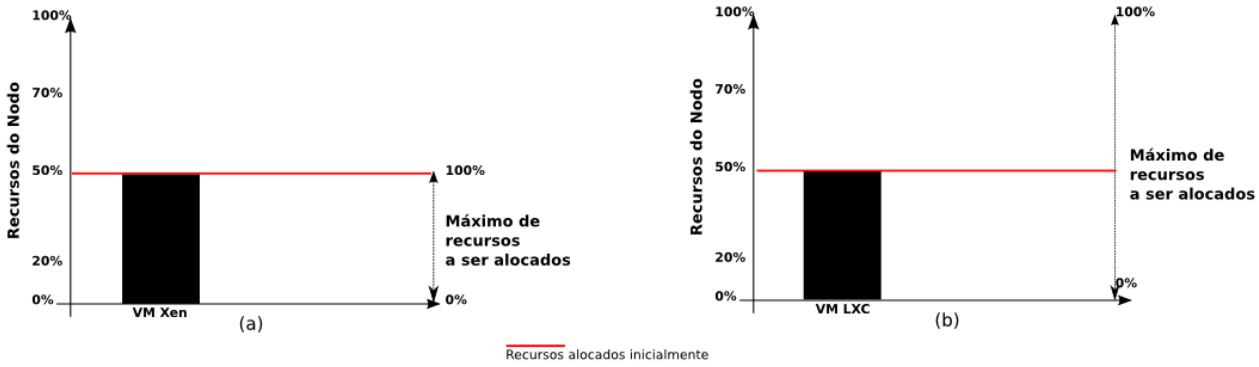

Figura 5.3 – Alocação Dinâmica de Recursos Xen X LXC

Ainda, ao contrário de outras técnicas de virtualização, como a para-virtualização ou a virtualização total, a virtualização em nível de Sistema Operacional, não depende de um monitor de máquinas virtuais, o que permite que os *containers* tenham desempenho de CPU, memória, disco e rede, similares ao de um ambiente nativo [XNR+13]. Essa menor sobrecarga pode acrescentar alguns benefícios à arquitetura desenvolvida por Dawoud et al. tais como: (i) menor intrusividade; (ii) redução dos problemas referentes à E/S (determinados SLAs, como por exemplo transações por segundo, sofrem influência direta deste tipo de recurso) e ; (iii) otimização do uso dos recursos.

Também, os ambientes de virtualização baseado em *containers* trazem benefícios quanto ao desempenho de aplicações de banco de dados quando comparado a outros ambientes virtualizados [LCXDR13] que também suportam bancos de dados.

Desta forma, nossa pesquisa poderá agregar benefícios ao trabalho no qual ela se baseia, suprindo as limitações geradas pelo uso de um método de virtualização tradicio-

nal. Nas próximas Seções, apresentaremos a arquitetura implementada por Dawoud et al. [DTM12]. Seus módulos serão explicados em detalhe. Após, discutiremos adaptações que se fizeram necessárias para sua implementação em ambiente de virtualização baseado em *containers*.

5.2 Arquitetura Elástica de Dawoud

A arquitetura elástica para alocação dinâmica de recursos, desenvolvida por Dawoud el al. [DTM12] tem o objetivo de assegurar que uma determinada máquina virtual receba recursos suficientes para manter seus serviços em execução dentro de níveis de qualidade determinados pelo usuário como aceitáveis.

Com esta finalidade, foi implementando o Controlador de QoS, que por sua vez é composto por três atuadores distintos: de CPU e Memória, que são responsáveis pelo cálculo da alocação ideal destes recursos, e de Aplicação que através de uma heurística define as configurações ideais de um servidor Apache, para que um determinado SLO seja atendido.

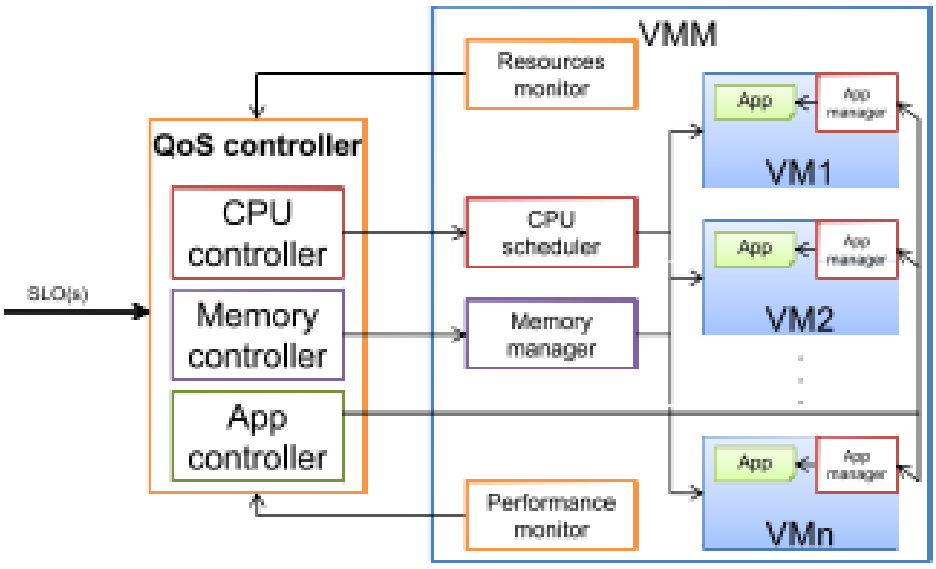

Figura 5.4 – Arquitetura de *Dawoud et al.* [DTM12]

Na Figura 5.4, que representa a arquitetura de Dawoud et al., podemos observar o "Controlador de QoS", componente principal desta arquitetura, que se comunica com os outros módulos implementados na camada do *hypervisor* e no nível de máquinas virtuais como veremos a seguir.

5.2.1 Monitoramento de Recursos

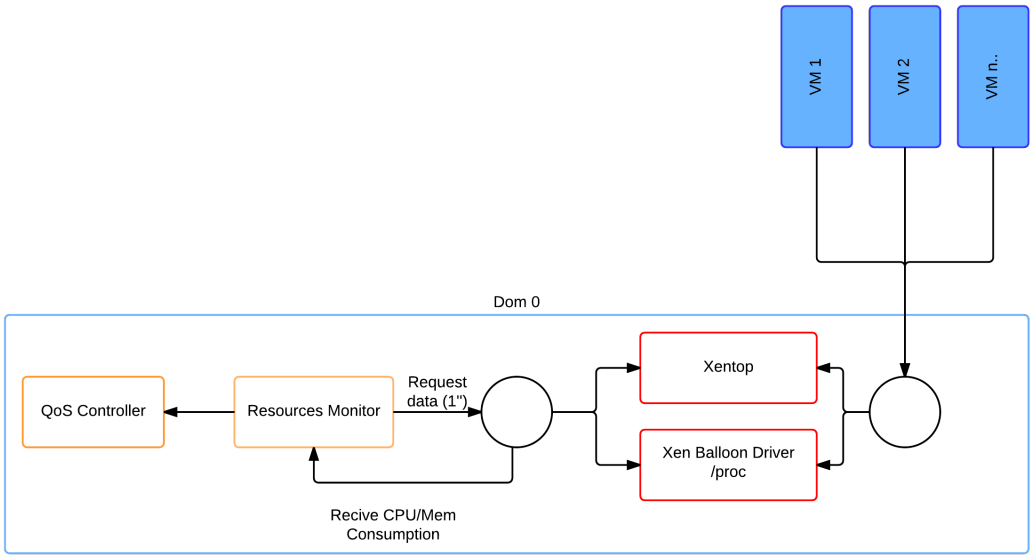

Figura 5.5 – Xen - Monitoramento de Recursos

O módulo de monitoramento de recursos (*Resources Monitor*) tem o objetivo de medir dinamicamente o consumo de CPU e de memória, atualizando o controlador de QoS com estas métricas.

O Consumo de CPU é obtido através da utilização comando *xentop* [XEN] do Xen. O *xentop* exibe informações sobre as máquinas virtuais de forma contínua e em tempo real, podendo seu resultado ser filtrado visando manipular essas informações. Já o uso de memória é obtido a partir do *Xen Balloon Driver*, cujas informações estão presentes em arquivos do diretório */proc* do sistema.

A Figura 5.5 ilustra o funcionamento do módulo, onde podemos ver que o Monitor de Recursos a cada segundo busca as informações sobre o consumo dos recursos, atualizando em seguida o Controlador de QoS.

5.2.2 Escalonador de CPU

O escalonador de CPU (*CPU Scheduler*) é implementado para alterar dinamicamente a alocação de CPU das máquinas virtuais de acordo com valores calculados pelo controlador de QoS. Este módulo é executando no Dom 0 do Xen e depende do *Xen Credit Scheduler* para definir as configurações de CPU para as máquinas virtuais.

O *Xen Credit Scheduler* [GND+07] é um escalonador desenvolvido com o objetivo de manter uma divisão de trabalho justa entre todos os processadores. A cada máquina virtual são atribuídos dois valores: um peso e um limite. Uma máquina virtual com peso 1024 terá quatro vezes mais prioridade à execução do que uma máquina virtual com peso 256. Os valores de peso variam de 1 a 65535, sendo o padrão 256. O limite geralmente fixa o máximo de CPUs que uma máquina virtual vai poder utilizar. O limite de uma CPU é expresso em porcentagem do conjunto de CPUs físicas disponíveis, de maneira que o valor 100 consiste em uma CPU física, 50 é meia CPU, 400 são 4 CPUs, e assim por diante.

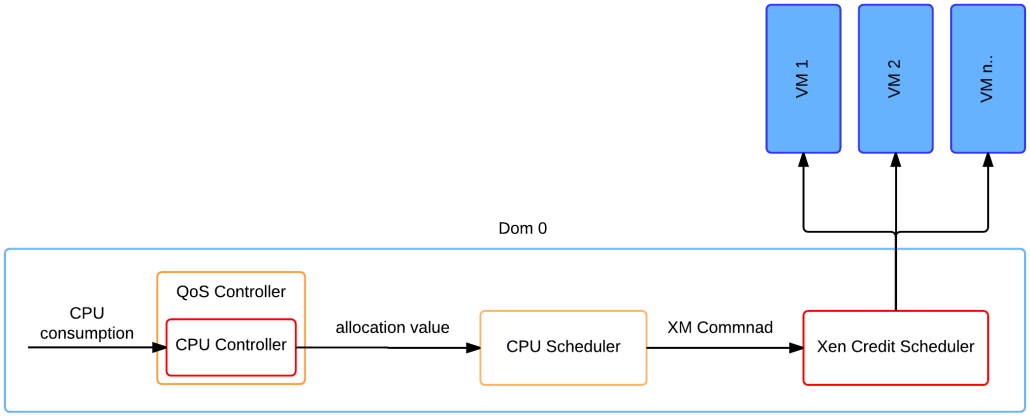

Figura 5.6 – Xen - Escalonador de CPU

A Figura 5.6 mostra em detalhes o funcionamento desse módulo, assim como a interação com o *Xen Credit Scheduler*. Assim que o escalonador de CPU recebe o valor de alocação do Controlador de QoS, é realizada uma chamada ao comando do *XM* para que o *Xen Credit Scheduler* efetue o ajuste. Assim que o comando é executado, a máquina virtual alvo da configuração tem o limite de CPU alterado. Na próxima subseção analisaremos em maiores detalhes o Gerenciador de Memória.

5.2.3 Gerenciador de Memória

O gerenciador de memória (*Memory Manager*) foi implementado com suporte do *Xen Balloon Driver* visando alterar dinamicamente a alocação de memória para as máquinas virtuais.

Cada máquina virtual é configurada inicialmente com um valor máximo de memória *(maxmem)*. O valor de *maxmem* é a soma do valor do balão e do valor disponível para alocação. Se uma máquina virtual não utiliza toda a sua memória, esta fatia não utilizada pode ser transferida para outra máquina virtual que possa estar necessitando. Isto é feito através da ampliação do balão (reduzindo a valor disponível para alocação) da primeira

máquina virtual e a redução do balão (aumentando o valor disponível para alocação) na segunda máquina virtual. Esse mecanismo permite que múltiplas máquinas virtuais possam ser configuradas com uma quantidade maior de memória total (*maxmem*) do que o tamanho da memória física disponível.

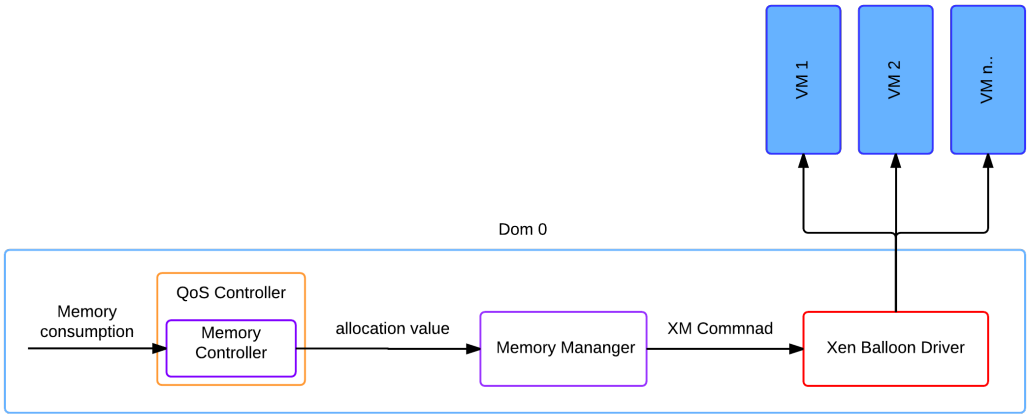

Figura 5.7 – Xen - Gerenciador de Memória

A Figura 5.7 retrata o funcionamento do módulo de gerenciamento de memória, onde podemos observar que após receber o valor de alocação determinado pelo Controlador de QoS, o gerenciador de memória realiza o ajuste de memória, através de uma chamada ao comando *XM*. A seguir analisaremos o funcionamento do Monitor de Desempenho.

5.2.4 Monitor de Desempenho

O monitor de desempenho (*Performance Monitor*) mantém o *Controlador de QoS* atualizado com as métricas de desempenho, ou seja, o tempo médio de resposta e a taxa de transferência de arquivos. As métricas de desempenho são informações base para o funcionamento do Controlador de QoS, que veremos em mais detalhes na Seção 5.2.6

Este módulo é implementado no dispositivo de rede do VMM, para que ele possa monitorar tanto o tráfego de entrada quanto o de saída.

5.2.5 Gerenciador de Aplicativos

O gerenciador de aplicativos (*App Mananger*) é implementado em nível de máquina virtual e sua função é atualizar o arquivo de configuração do *Apache*, alterando o

valor da variável *MaxClients*. A variável *MaxClients* determina a quantidade máxima de clientes que o servidor *Apache* poderá atender. Conforme estudo feito por Dawoud et al., a manipulação correta desta variável pode trazer ganhos consideráveis de desempenho.

O atuador de aplicativos informa ao gerenciador de aplicativos o valor que deverá ser atribuído à variável *MaxClients*. Após alterar esse valor, o gerenciador de aplicativos reinicia o serviço do servidor *Apache*, para que a alteração passe a ter efeito.

5.2.6 Controlador de QoS

O controlador de QoS *(QoS Controller)* é o componente principal da arquitetura proposta por Dawoud et al. [DTM12]. Composto por três atuadores: de CPU, memória e de aplicação que funcionam independentes uns dos outros, mas com o mesmo objetivo, de manter o sistema dentro de níveis de qualidade aceitáveis.

No lado esquerdo da Figura 5.4 está representado o controlador de QoS e os atuadores que o compõem. A seguir estes atuadores serão apresentados com maiores detalhes:

Atuador de CPU:

O atuador de CPU foi projetado por *Zhu et al.* [ZWS06] e é representado pela seguinte equação:

$$a_{cpu}(k+1) = a_{cpu}(k) - K_1(k)(u_{ref}^{cpu} - u_{cpu}(k)) \tag{5.1}$$

onde

$$k_1 = \alpha.c_{cpu}(k)/a_{cpu}(k) \tag{5.2}$$

Este atuador tem a função de calcular a próxima alocação de CPU $a_{cpu}(k+1)$, baseando-se na última alocação de CPU $a_{cpu}(k)$, além do consumo $c_{cpu}(k)$. A variável u_{cpu} consiste na última utilização de CPU e é representado por $u_{cpu}(k) = c_{cpu}(k)/a_{cpu}(k)$. O parâmetro α em K_1 é o ganho constante, que determina a agressividade do atuador.

A desvantagem deste atuador é a necessidade da definição manual do valor de referência sobre a taxa de utilização da CPU u_{ref}^{cpu}, necessária para manter um determinado SLA. No entanto, a definição deste valor não é uma tarefa trivial, uma vez que geralmente a manutenção de SLAs não depende apenas da utilização de CPU, mas também de outros fatores tais como consumo de memória, vazão de rede e E/S, que podem ser arbitrários. Assim, é mais realista que u_{ref} seja atualizado pelo controlador de QoS.

Com essa preocupação, outro atuador também foi projetado por [ZWS06] visando ajustar o valor de utilização de CPU u_{ref} dinamicamente, para assegurar que uma métrica

de QoS (no caso do trabalho de *Dawoud et al.* tempo de resposta) seja atendido. Esse controlador é expresso pela seguinte equação:

$$u_{ref}^{cpu}(i+1) = u_{ref}^{cpu}(i) + \beta(RT_{ref}^{cpu} - RT_{cpu}(i))/RT_{ref}^{cpu} \tag{5.3}$$

Onde $u_{ref}^{cpu}(i+1)$ é a utilização de CPU desejada, $RT_{cpu}(i)$ é o tempo de resposta medido, e RT_{ref}^{cpu} é o tempo de resposta desejado, determinada pelo SLA. Este atuador garante que o valor de utilização CPU esteja sempre dentro de um intervalo aceitável de utilização da CPU. A interação entre os dois atuadores pode ser vista na Figura 5.8

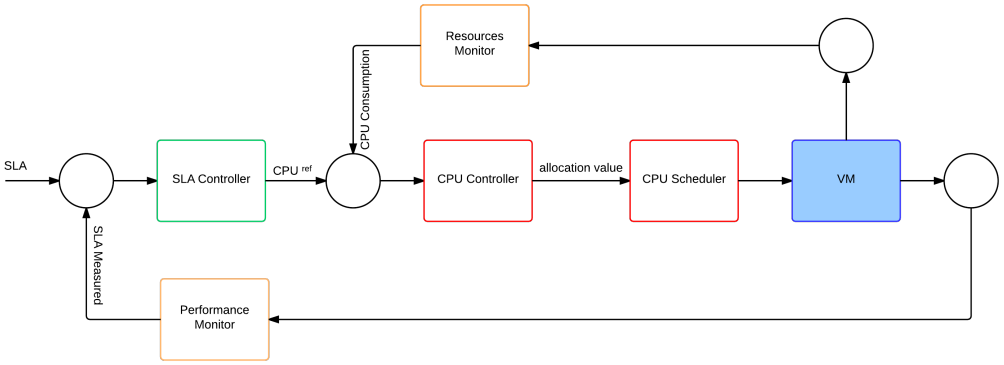

Figura 5.8 – Atuador de CPU

Atuador de Memória:

Projetado por *Zhu et al.* [ZWS06], é representado pela seguinte equação:

$$a_{mem}(i+1) = a_{mem}(i) + k_2(i)(u_{ref}^{mem} - u_{mem}(i)) \tag{5.4}$$

onde

$$K_2(i) = \lambda.u_{mem}(i)/u_{ref}^{mem} \tag{5.5}$$

Com o objetivo de manter o sistema longe de gargalos de memória, o atuador aloca agressivamente mais memória, quando a memória alocada anteriormente está próximo da saturação (por exemplo, 90%), e diminui lentamente a alocação de memória quando a alocação está em uma região longe de gargalos. A Figura 5.9 apresenta o diagrama que ilustra o funcionamento deste atuador.

Atuador de Aplicação:

Consiste na implementação de uma heurística com objetivo de encontrar o melhor valor do parâmetro de configuração *MaxClients* do servidor *Web* Apache. Este atuador

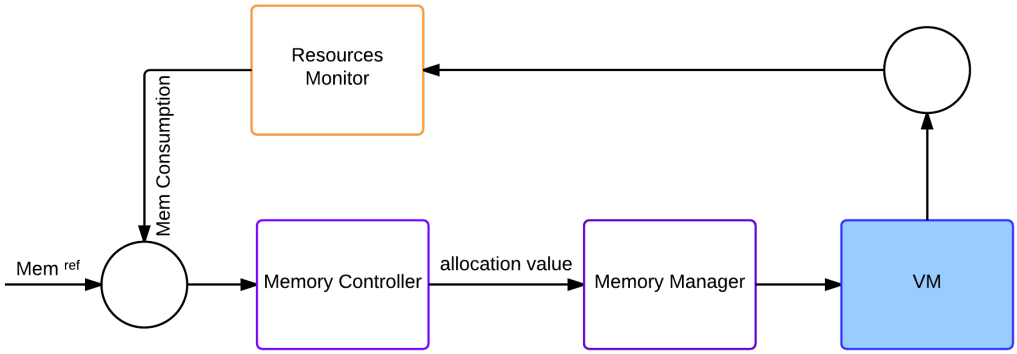

Figura 5.9 – Atuador de Memória

monitora o tempo de resposta, *throughput*, utilização de CPU e o número de processos do Apache que estão em execução. O controlador salva o melhor registro desses valores, que é definido pela procura do registro que satisfaça o SLA com a menor utilização da CPU. A cada nova medição dos valores monitorados, o atuador compara o registro atual com os novos valores, mantendo o melhor.

Na próxima Seção, serão abordadas as modificações que foram realizadas com a finalidade de adaptar a arquitetura proposta por Dawoud et al. [DTM12] para ambiente de *containers*.

5.3 Adaptação para o uso de *containers*

Nesta Seção, abordaremos as modificações realizadas na arquitetura de Dawoud et al. com a finalidade de adapta-la à utilização do modelo de virtualização em nível de Sistema Operacional. A utilização desta técnica de virtualização tem a finalidade de resolver as limitações apresentadas na Seção 5.1, ocasionadas pelo uso de modelo tradicional de virtualização.

Ao analisar a arquitetura vista na Seção 5.2, podemos verificar sua dependência à recursos específicos do Xen, como o *Xen Credit Scheduler*, o *Xentop* e o *Xen Balloon Driver*, o que ocasiona mudanças pontuais na estrutura interna desta arquitetura ao adaptá-la para a utilização de *containers*. A Figura 5.14 contempla estas alterações realizadas na arquitetura desenvolvida por Dawoud et al. [DTM12].

Para realizar essa adaptação utilizamos a linguagem Python, na versão 3.0. O *Python* foi a linguagem escolhida por fornecer a biblioteca *python-lxc* que permite a fácil manipulação dos *containers*.

5.3.1 Monitoramento de Recursos

Manteve-se a proposta inicial deste módulo, porém foi necessário a construção de um módulo totalmente novo devido as diferenças entre o Xen e o LXC. O LXC ao contrário do Xen não disponibiliza uma ferramenta que forneça as informações que são necessários para o Controlador de QoS.

Para a obtenção dos dados referentes ao consumo de CPU utilizamos as informações obtidas através do *cgroups*, mas especificamente do subsistema de *accounting* de CPU chamado *CPUACCT*. Esse subsistema gera relatórios em tempo real sobre recursos de CPU utilizados pelas tarefas em cada *container*. Dentre os relatórios disponíveis nós utilizamos o *cpuacct.usage* que reporta o total de tempo de CPU consumido por todas as tarefas para cada *container*, incluindo tarefas de baixa hierarquia.

Ainda, foi necessária a manipulação destes dados para obtermos o valor de entrada exigido pelo controlador de QoS, que é o percentual de utilização das CPUs alocadas para a máquina virtual.

Já para a obtenção dos dados referentes ao consumo de memória, utilizamos o subsistema *memory.usage_in_bytes* , que reporta em tempo real o consumo de memória total de um determinado *container*. A Figura 5.10 mostra em detalhes o funcionamento deste módulo

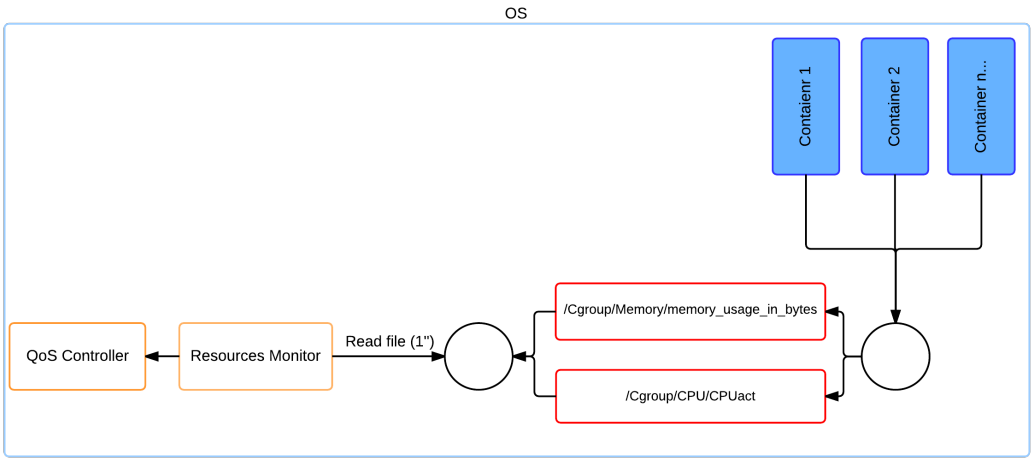

Figura 5.10 – LXC - Monitoramento de Recursos

5.3.2 Alocador de CPU

O alocador de CPU (*CPU Scheduler*) tem o objetivo de realizar a alocação ou a desalocação de CPU, conforme valores fornecidos pelo controlador de QoS. Este módulo

foi construído utilizando a biblioteca *python-lxc*, que permite a configuração em tempo real dos valores de alocação de CPU.

A definição da alocação de CPU consiste na configuração do subsistema *cpu.cpuset* do cgroups. Este subsistema permite que se especifique a quais CPUs as tarefas de um determinado *container* possuem acesso.

Todavia, o cálculo realizado pelo controlador de QoS, pode resultar em valores decimais, como por exemplo, sugerir a alocação de 1.15 CPUs. Com a finalidade de buscar um melhor aproveitamento dos recursos, criamos uma regra de arredondamento que compara se o valor decimal for maior ou igual a .5, então o valor de alocação é arrendondado para cima. Caso o valor decimal seja menor do que .5, o valor de alocação é arrendondado para baixo.

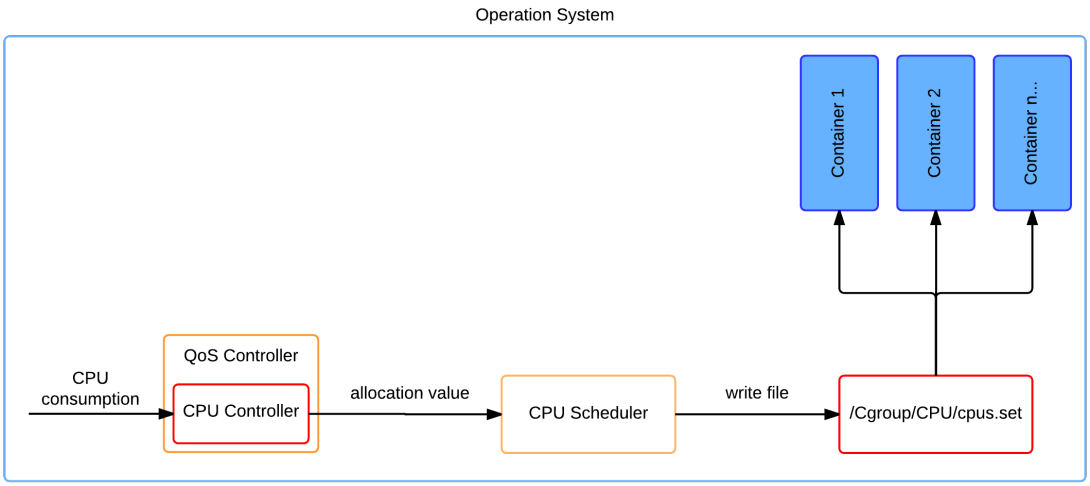

Figura 5.11 – LXC - Escalonador de CPU

Ainda, definimos que o valor mínimo de alocação de CPU seria de 1 núcleo físico. Isso é necessário porque, caso o *container* fique em *idle*, o controlador de QoS irá sugerir a alocação de valores menores que 1 núcleo podendo comprometer os serviços em execução nesse *container*.

A Figura 5.11 ilustra o funcionamento do alocador de CPU, bem como a sua interação com os demais módulos da arquitetura.

5.3.3 Gerenciador de Memória

O gerenciador de memória (*Memory Manager*) é responsável pela alocação ou desalocação de memória para os *containers*. Este módulo foi desenvolvido utilizando a biblioteca *python-lxc*, que permite a configuração em tempo real dos valores de alocação

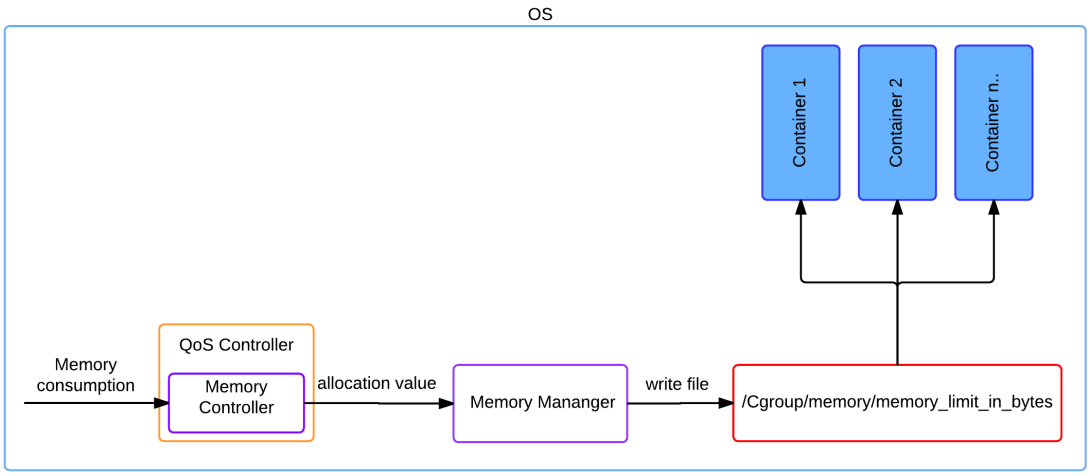

Figura 5.12 – LXC - Gerenciador de Memória

de memória. Essa configuração consiste na definição dos valores oriundos do Controlador de QoS, no subsistema *memory.limit_in_bytes* do cgroups, que permite que seja definido a quantia máxima de memória de um *container*.

Esse módulo também foi desenvolvido utilizando a biblioteca *python-lxc*, pois além de permitir a manipulação das informações sobre CPU, Rede e E/S também permite a manipulação dos valores de alocação de memória em tempo real.

Com a finalidade de garantir que haja memória suficiente para o funcionamento do *container*, definimos que o valor mínimo de alocação de memória seria de 512 MB. Todavia, ao contrário do Xen, o LXC permite a alocação de memória independente do valor alocado na inicialização do *container*. A Figura 5.12 apresenta o diagrama que ilustra o funcionamento do deste módulo, bem como a sua interação com os demais módulos da arquitetura.

5.3.4 Monitor de Desempenho

O monitor de desempenho (*Performance Monitor*) tem a função de informar ao controlador de QoS quanto às métricas de desempenho. Como o foco deste trabalho está voltado para sistemas de banco de dados, utilizamos como métrica o número de transações que são executadas por segundo (TPS) em um determinado banco de dados.

Para atingir esse objetivo desenvolvemos um módulo que a cada segundo busca no SGBD o número de transações que ocorreram, sendo que o total de transações por segundo se dá pela diferença entre o número de transações que aconteceram no segundo atual, pelo número de transações que aconteceram no segundo anterior. A Figura 5.13

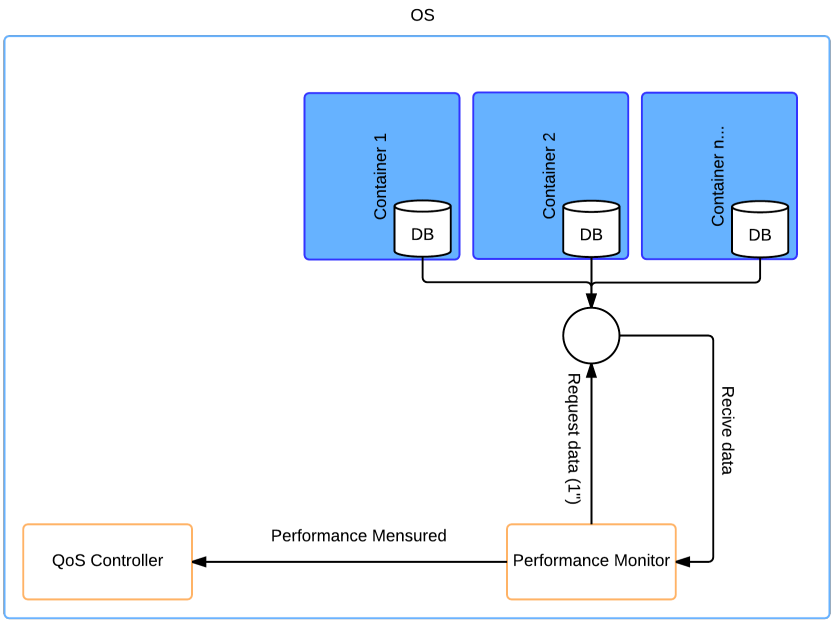

Figura 5.13 – LXC - Monitor de Desempenho

ilustra o funcionamento desse módulo em detalhes, bem como sua interação com os demais módulos da arquitetura.

5.3.5 Controlador de QoS

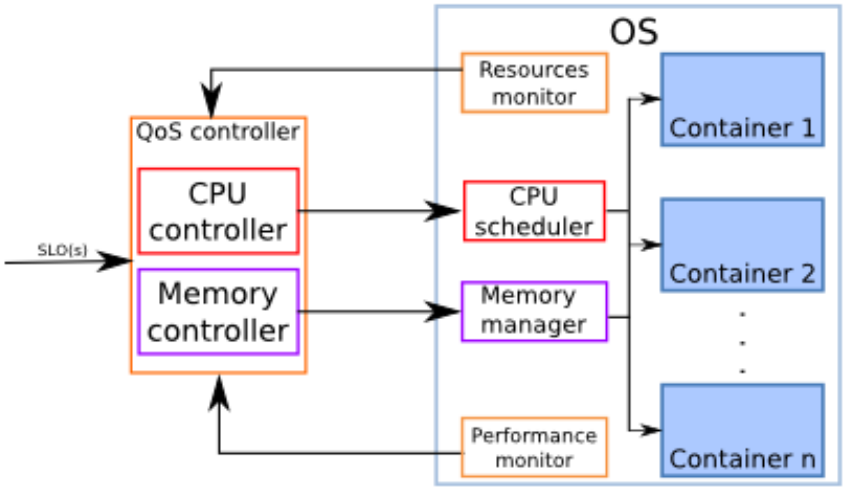

Figura 5.14 – Arquitetura de Dawoud et al. adaptada ao uso de *containers*

Os atuadores de CPU e memória não sofreram alterações. Já uma das diferenças mais marcantes está na ausência do atuador de aplicação em nossa implementação. A

construção deste controlador não faria sentido em nossa abordagem já que utilizaremos como carga de trabalho aplicações de banco de dados, ao contrário de Dawoud et al. que utilizou cargas de trabalho voltadas a servidores web. Além disso, a implementação deste tipo de atuador torna esta proposta dependente de uma plataforma específica (neste caso específico, o Apache), tornado a arquitetura menos flexível. Ainda o fato de ser necessário a reinicialização dos serviços de banco de dados após novas configurações, o que interromperia o serviço, inviabiliza a construção deste módulo. A Figura 5.14 retrata a arquitetura após as modificações realizadas para a utilização de *containers*.

A seguir no próximo Capítulo, serão demonstradas às avaliações que foram realizadas afim de verificar a eficiência de nossa implementação em manter uma aplicação dentro de um determinado SLA.

6. AVALIAÇÕES DA ARQUITETURA ADAPTADA AO USO DE *CONTAINERS*

Este capítulo apresenta a avaliação de nossa implementação da arquitetura elástica de Dawoud et al. [DTM12] adaptada para o uso de *containers*.

Na Seção 6.1 abordamos os experimentos que foram realizados com a finalidade de avaliarmos individualmente a eficiência dos atuadores de CPU e Memória. A seguir na Seção 6.2 descreveremos o ambiente, bem como as ferramentas que utilizamos para a avaliação de nossa arquitetura.

Por fim na Seção 6.3 serão apresentados os resultados obtidos no processo de avaliação, bem como um comparativo da arquitetura original de Dawoud et al. [DTM12] com a arquitetura adaptada ao uso de *containers*

6.1 Avaliação Individual dos Atuadores

Esta primeira avaliação tem por objetivo verificar individualmente a eficiência dos atuadores de CPU e Memória que foram implementados neste trabalho.

Nosso ambiente para estes experimentos foi composto por um servidor Dell R610, com dois processadores *Intel Xeon E5520 Six Core 2.27GHz* e com 24 GB de memória RAM. O sistema operacional utilizado foi o *Ubuntu Server 14.04 64-bit*, rodando a versão 1.0 do LXC.

Como ferramenta para realização destes testes utilizamos o *DaCapo Benchmark (V. 9.12)*. Desenvolvido pela IBM, o DaCapo [dac] consiste em um conjunto de aplicações *open source* com cargas de trabalho reais e não triviais. Em nossos experimentos, selecionamos quatro destas aplicações: *tradebeans*, *tradesoap*, *h2* e *lusearch*. A escolha destas aplicações foi motivada por apresentarem comportamento semelhante a aplicações de banco de dados, foco desse trabalho.

Em ambos os experimentos foram utilizados dois *containers*. O primeiro com os recursos de CPU e memória limitados apresentado nos gráficos como "Estático", e o segundo com os recursos gerenciados pelos atuadores de memória e CPU apresentado nos gráficos como "Dinâmico".

A seguir apresentaremos os resultados referentes a avaliação do atuador de CPU.

6.1.1 Avaliação do Atuador de CPU

O primeiro experimento realizado tinha o objetivo de verificar a eficiência do atuador de CPU em oferecer melhor desempenho quando comparado a um modelo de alocação fixa. Na primeira fase do experimento limitamos os recursos do *container* em 1 núcleo de CPU, sem limitação de memória, visando garantir que este último recurso não gerasse interferência nos testes. Já na segunda fase do experimento, iniciamos o *container* com a mesma configuração da primeira, porém ativamos o atuador de CPU com a finalidade de que este, realizasse a alocação dinâmica deste recurso.

A Figura 6.1 apresenta os resultados do experimento, onde fica claro o melhor desempenho do sistema quando a alocação de CPU é gerenciada pelo atuador. Já na Tabela 6.1, podemos comparar os tempos de execução em ambos os casos, estático e dinâmico, onde podemos observar que o atuador de CPU é capaz de aumentar o desempenho em mais de 40% em todos dos casos.

Para melhor compreendermos os resultados, acompanhamos o processo de alocação de CPU durante a execução do *benchmark h2*. O *h2*, que simula uma série de transações em uma aplicação bancária, foi selecionado para essa análise detalhada por ter características semelhantes às de aplicações que fazem uso intensivo de banco de dados [dac]

A Figura 6.2 comprova a eficiência do atuador em oferecer uma melhor alocação de CPU, pois aloca mais núcleos de processamento a medida que estes se fazem necessários, desalocando-os a medida que a demanda por processamento diminui.

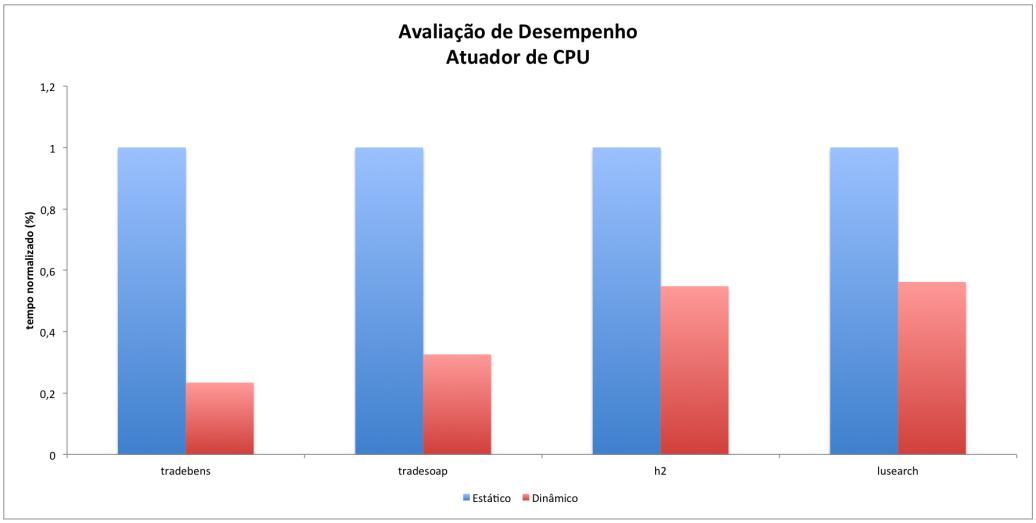

Figura 6.1 – Atuador de CPU - Avaliação de Performance

Desta forma podemos concluir que o atuador de CPU é capaz de manter a eficiência do sistema em picos de processamento, oferecendo economia de recursos ao melhorar o desempenho das aplicações. A seguir avaliaremos o atuador de memória.

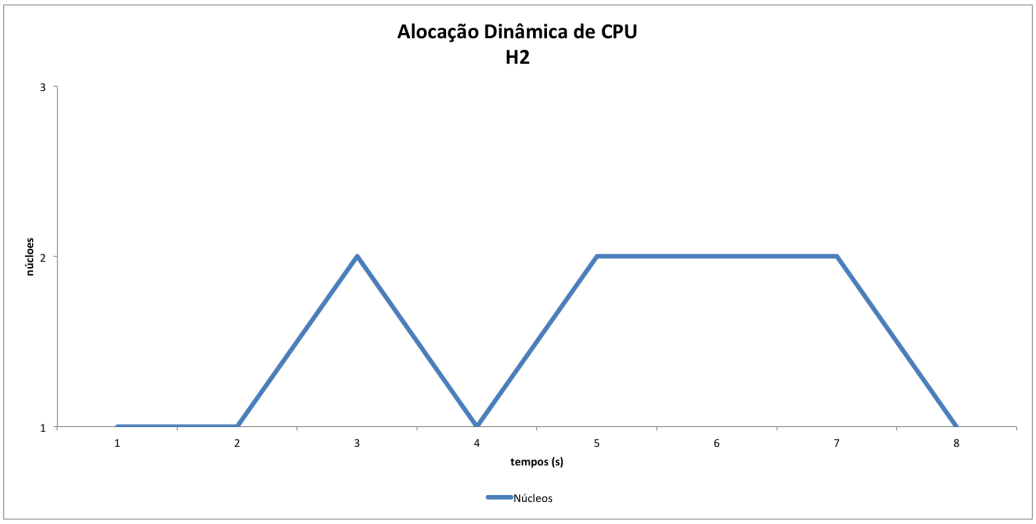

Figura 6.2 – Alocação de CPU durante a execução do Benchmark h2

Tabela 6.1 – Avaliação de Desempenho - Atuador de CPU

Bechmark	Estático (s)	Dinâmico (s)	Desempenho (%)
tradebeans	58.6	13.7	76.62
tradesoap	158.4	51.6	67.42
h2	11.5	6.3	45.22
luseach	12.1	6.7	43.80

6.1.2 Avaliação Atuador Memória

O segundo experimento foi realizado com a finalidade de verificar se o uso do atuador de memória poderia oferecer melhor desempenho frente a utilização da alocação fixa deste recurso. Limitamos a utilização de memória RAM em 1024MB para a realização dos experimentos com alocação fixa de recursos. Na segunda etapa dos experimentos, o *container* teve a alocação de memória inicialmente limitada a 512MB, porém o atuador de memória estava ativado. A Figura 6.3 mostra que o uso de alocação dinâmica de memória oferece um ganho considerável de desempenho frente a alocação fixa.

Na Tabela 6.2 podemos comparar os tempos de execução, onde podemos verificar que o uso do atuador de memória trouxe um ganho de desempenho considerável em praticamente todos as aplicações. Chama a atenção o menor ganho de desempenho na aplicação *tradebeans*, porém trata-se de uma aplicação em que a memória não é fator determinante para um melhor desempenho, sendo esta aplicação afetada diretamente pela capacidade de processamento.

Como em 6.1.1, com o objetivo de entender melhor esses resultados, realizamos um terceiro experimento verificando o consumo de memória durante a execução do *h2*.

A Figura 6.4 mostra como se deu a utilização de memória durante o teste. Podemos observar que no caso do *container* que sofreu a limitação, o valor definido (1024MB)

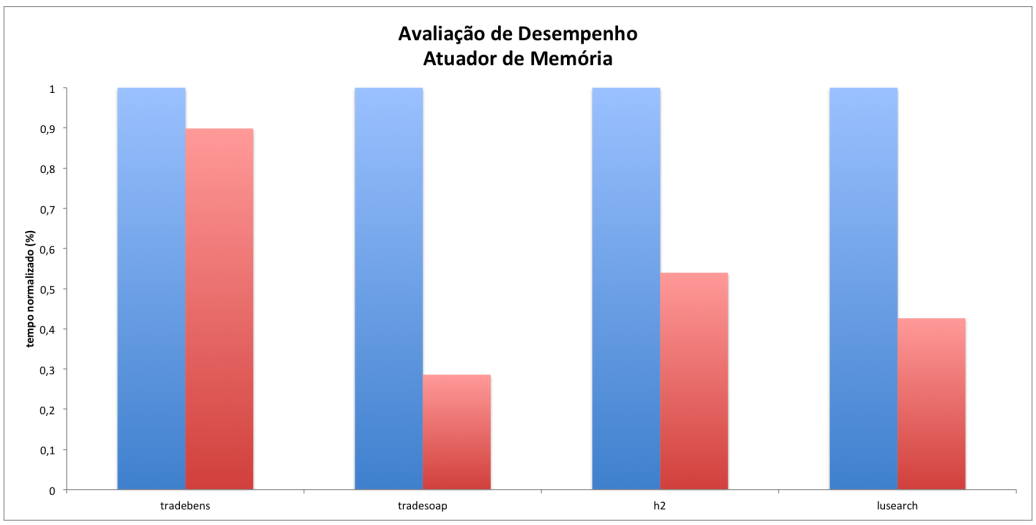

Figura 6.3 – Atuador de Memória - Avaliação de Performance

Tabela 6.2 – Avaliação de Desempenho - Atuador de Memória

Bechmark	Estático (s)	Dinâmico (s)	Desempenho (%)
tradebeans	12.8	11.5	10.16
tradesoap	172.4	49.3	71.40
h2	13.1	6.1	53.44
luseach	25.1	10.7	57.37

foi utilizado em sua totalidade, inclusive gerando uma grande utilização de *swap*, como a Figura 6.5 demonstra. Isso mostra que a utilização de *swap* pode gerar sobrecarga ao sistema, degradando em muito o desempenho da aplicação em execução.

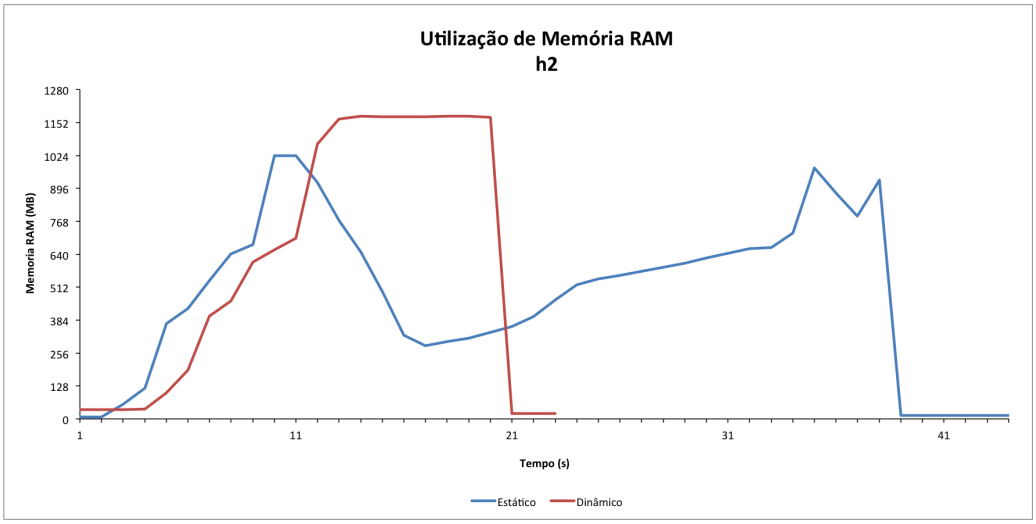

Figura 6.4 – Atuador de Memória - Utilização de Memória RAM

Por outro lado, ao executar os experimentos utilizando o atuador de memória, podemos verificar na Figura 6.4 que ao perceber a necessidade de mais memória, esta foi alocada aumentado a eficiência do sistema, sendo que esta memória foi liberada após a utilização. Ao analisar a Figura 6.5, podemos concluir que o atuador de memória foi capaz

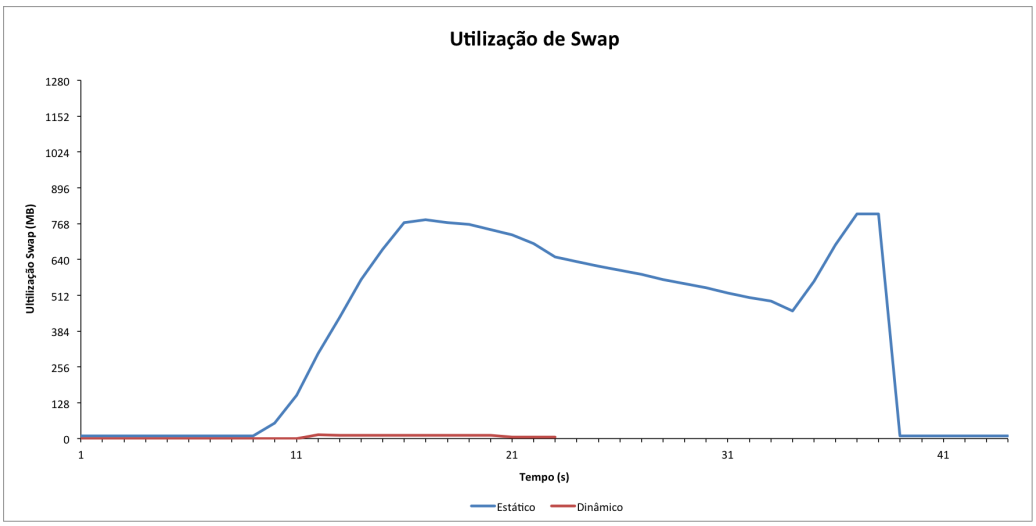

Figura 6.5 – Atuador de Memória - Utilização de Swap

de manter o sistema livre de gargalos, evitando a utilização de *swap*, o que contribuiu para um melhor desempenho.

Por fim, esses experimentos comprovaram a eficiência dos atuadores de CPU e memória, o que sustenta a nossa hipótese, de que a utilização destes atuadores poderá contribuir com o objetivo deste trabalho. Na próxima Seção 6.2 apresentaremos os experimentos que avaliam a nossa arquitetura.

6.2 Avaliação da Arquitetura

Nesta Seção, apresentaremos os experimentos realizados com a finalidade avaliar a capacidade da arquitetura proposta em manter a eficiência de um sistema de banco de dados. Na Subseção a seguir, apresentaremos em detalhes o ambiente que foi utilizado para realizar estes experimentos.

6.2.1 Ambiente de avaliação

O ambiente de execução é formando por dois servidores Dell PowerEdge R810 com dois processadores Intel®Xeon®6500 com 64GB de memória RAM cada um. Ainda utilizamos um servidor Dell PowerEdge R610, com dois processadores Intel®Xeon®X5690 e 24GB de memória RAM, de onde o *benchmark* foi executado. Os servidores estavam interligados em rede através de uma conexão GigaBit Ethernet.

No primeiro servidor foi instalado o *Citrix Xen Server 6.2*. Optou-se por essa distribuição do Xen, por ser uma distribuição comercial *opensource* amplamente utilizada pela

indústria e academia. Neste servidor foram executados experimentos utilizando a arquitetura proposta por Dawoud et al. [DTM12] como podemos ver em maiores detalhes na Seção 6.2.2.

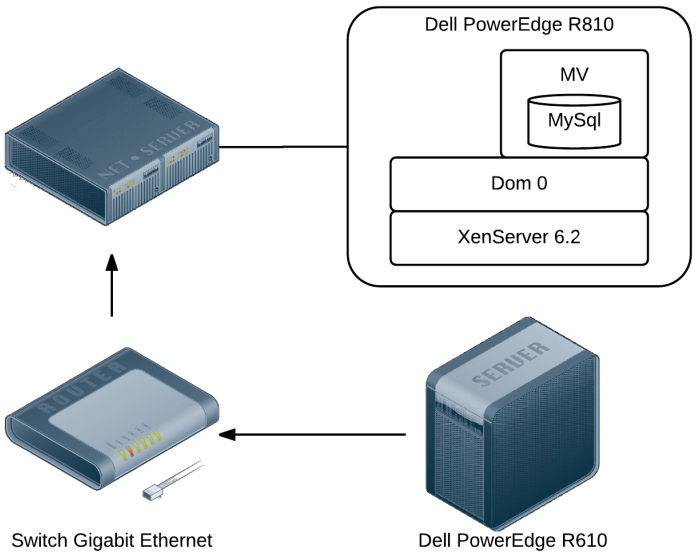

Figura 6.6 – Ambiente de avaliacao - XEN

Já o segundo servidor foi utilizado para realizar os experimentos da arquitetura adaptada para a utilização de *containers*. Neste servidor optamos por utilizar como sistema operacional o *Ubuntu Server 14.04*. Como solução para virtualização em nível de sistema operacional, utilizamos o LXC versão 1.0, que conforme Xavier et al. [XNR+13], apesar de apresentar problemas quanto a isolamento de recursos tem essa limitação compensada pelas facilidades de uso e gerenciamento. Ainda, o fato do LXC estar contido de forma *mainline* nos fontes do *kernel* do Linux, assegura que a tecnologia evolua linearmente com o desenvolvimento do *kernel*, evitando sua descontinuidade.

Como SGBD optamos por utilizar o MySql Server 6.2. O MySql é um sistema gerenciador de banco de dados relacional, amplamente utilizado em sistemas para internet, o que nos motivou a utilizá-lo. Configuramos o MySql para utilizar o InnoDB como *Storage Engine*, por apresentar bom desempenho, dar suporte à transações e ser totalmente adequada às propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), propriedades estas indispensáveis a qualquer SGBD. Optamos por seguir a configuração padrão do banco de dados, para evitar que possíveis otimizações no MySql viessem a interferir nos resultados.

As Figuras 6.6 e 6.7 apresentam em detalhes os ambientes de testes que foram utilizados. Na próxima Subseção apresentaremos os testes que foram executados bem como a ferramenta para realização dos mesmos.

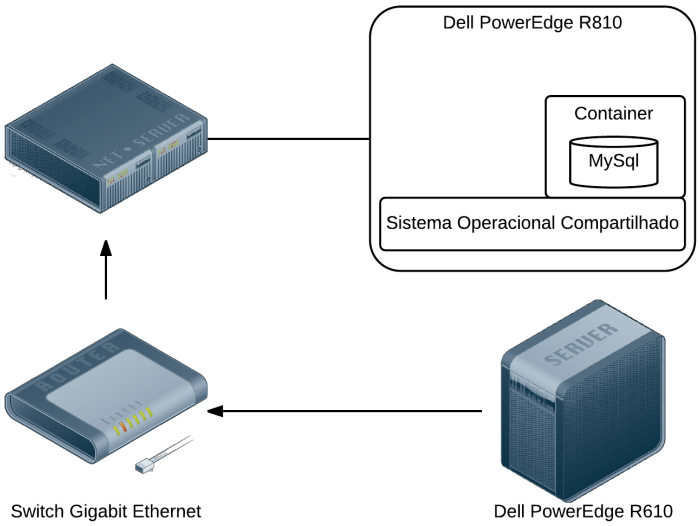

Figura 6.7 – Ambiente de avaliacao - LXC

6.2.2 *Benckmark* e descrição dos testes

Para avaliar aplicações de banco de dados em ambientes virtualizados, utilizando operações típicas de aplicações que demandam recursos computacionais, foram utilizadas operações do tipo *On-Line Transaction Processing (OLTP)* [TPC].

Sistemas OLTP são caracterizados por suportar vários usuários simultâneos, com execução de múltiplas transações (seleção, atualização e exclusão de dados) simultâneas. Estes tipos de transações permitem avaliar as operações típicas de ambientes que exigem uma elevada disponibilidade de acesso simultâneo aos dados com alto desempenho.

Para analisar o desempenho de sistemas OLTP, foi utilizado o *benchmark Sysbench*. O Sysbench é uma ferramenta de *benchmark multithreaded* para avaliação dos parâmetros que são importantes para um sistema de gerência de banco de dados sob carga de trabalho intensiva [Kop12].

Entre os muitos modos de teste, o teste de OLTP é concebido para medir o desempenho do servidor de banco de dados. O teste OLTP executado através desta ferramenta não é uma aproximação de um teste de OLTP, mas sim uma verdadeira referência lastreada em banco de dados que realizam consultas transacionais para uma instância do MySQL em um ambiente Linux [Kop12].

O Sysbench tem três modos de teste: transacional complexo, transacional simples e não-transacional. O modo transacional complexo, inclui: *points query, range query, range sum query, range order by query, update query, insert query* e *delete query* O modo transacional simples e o modo não transacional contém subconjuntos de testes do modo transacional complexo [CCS10]. Ainda pode-se optar pela execução que faz operações

de leitura e escrita, onde é observada cerca de 75% de operações de leitura e 25% de operações de escrita ou pela execução somente leitura.

Como resultado, o *Sysbench* fornece diversas informações de desempenho, entre elas a quantidade de transações por segundo (TPS) que foram executadas.

A Figura 6.8 apresenta a arquitetura do *software Sysbench-OLTP* que é composto por dois módulos: clientes e controlador. Os clientes são responsáveis pela criação de transações especificadas pelo utilizador, enquanto o controlador gerencia o início e término do *benchmark*. Todos os clientes, que são implementados usando *threads* que se comunicam com MySQL usando *sockets*. O *benchmark* é inteiramente escrito em linguagem C [CCS10].

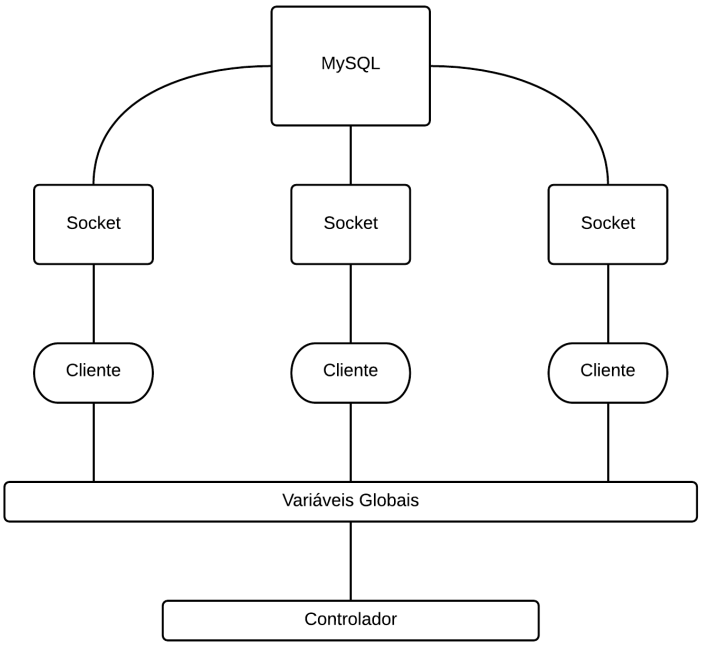

Figura 6.8 – Arquitetura Sysbench

O ambiente de testes foi configurado de maneira que apenas o SGBD estivesse em execução em cada caso de teste, e tanto o Sistema Operacional quanto o SGBD sempre estivessem no mesmo estado inicial para cada teste. Quanto ao *benchmark*, para nossos experimentos, nós selecionamos o modo de teste transacional complexo e optamos pela execução tanto utilizando operações de leitura e escrita, quanto operações de somente leitura. Todos os testes foram executados durante 300 segundos. Foram realizadas 15 execuções do *benchmark* para cada caso de teste, com intervalo de confiança chegando a 95%.

Ainda quanto as configurações do controlador de QoS, mantivemos as configurações originais, onde Dawoud et al. definiu a agressividade do atuador de CPU (α) e de memória (λ) em 1.5. Ainda quanto o atuador de memória, também mantivemos o valor de

referência em .90, ou seja, sempre a utilização de memória atingir 90%, mais memória deve ser alocada para que não haja utilização de *swap*.

Com objetivo de avaliar o desempenho da arquitetura desenvolvida por Dawoud et al. [DTM12] em um ambiente de banco de dados, executamos experimentos utilizando a arquitetura como ela foi projetada originalmente, utilizando Xen como VMM, porém sem o atuador de aplicação, pois a utilização deste módulo não faria sentido em nossos testes. Ainda executamos o mesmo experimento utilizando a arquitetura adaptada para o uso de *containers* com a finalidade de realizar um comparativo entre as soluções. Os resultados obtidos nessa avaliação podem ser vistos na Seção 6.3.1.

Com a finalidade de verificar a eficiência da arquitetura adaptada para o uso de *containers* em manter diferentes SLAs executamos uma série de experimentos, cujos resultados podem ser vistos na Seção 6.3.3

6.3 Resultados Obtidos

Esta Seção apresenta os resultados dos experimentos que realizamos com a finalidade de avaliar a nossa implementação da arquitetura elástica baseada em *containers* bem como um comparativo com a arquitetura elástica utilizando Xen como VMM.

6.3.1 Avaliação da arquitetura elástica utilizando Xen

Esse experimento foi realizado em um ambiente com máquina virtual utilizando Xen e em um *container* utilizando o LXC, ambos com 4 núcleos de processamento e 4096 MB de memória RAM. Para que não houvesse uma interferência menor de operações de E/S nos resultados, o teste em questão foi executado em modo somente leitura.

Para a definição do SLA, executamos o *benchmark* em ambas as instâncias virtuais para verificar o máximo de transações por segundo que seria alcançado. Observamos que no Xen obtivemos o máximo de 600 TPS enquanto no LXC obtivemos o máximo de 1800 TPS. Com base nestes dados, definimos que o SLA seria de 50% deste valor, ou seja, 300 TPS no caso do Xen e 900 TPS no caso do LXC.

A Figura 6.9, ilustra o comportamento do teste que foi realizando no ambiente Xen. Podemos observar um comportamento irregular tanto nas TPS quanto na alocação de CPU. Na Tabela 6.3 podemos verificar que neste experimento em 33.66% do tempo de execução houve quebra de SLA.

Já na execução no ambiente de *Containers*, ilustrado na Figura 6.17, observamos um comportamento mais regular quanto as TPS, onde houve quebra de SLA em 6.33%

do tempo de execução, porém ainda verificamos um comportamento irregular quanto a alocação e desalocação de núcleos de processamento.

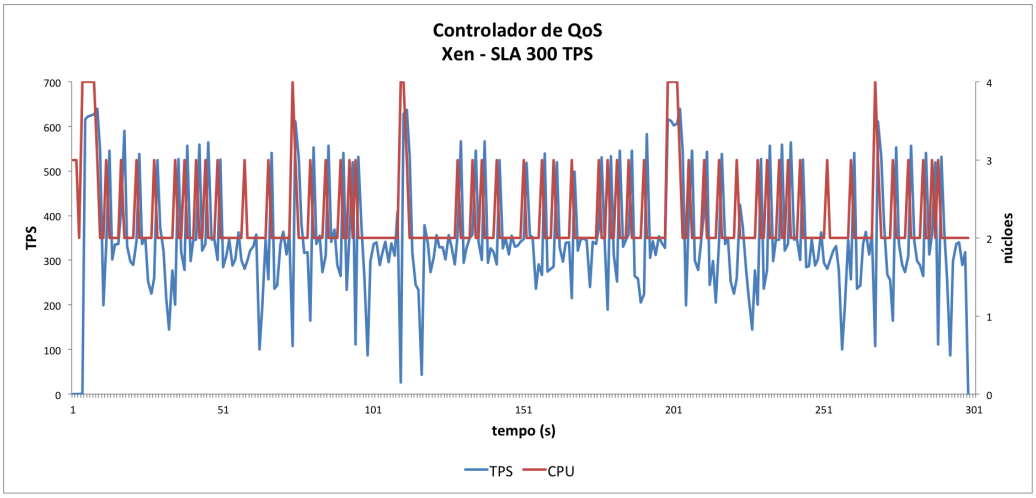

Figura 6.9 – Avaliação da Arquitetura com Xen. SLA 300TPS

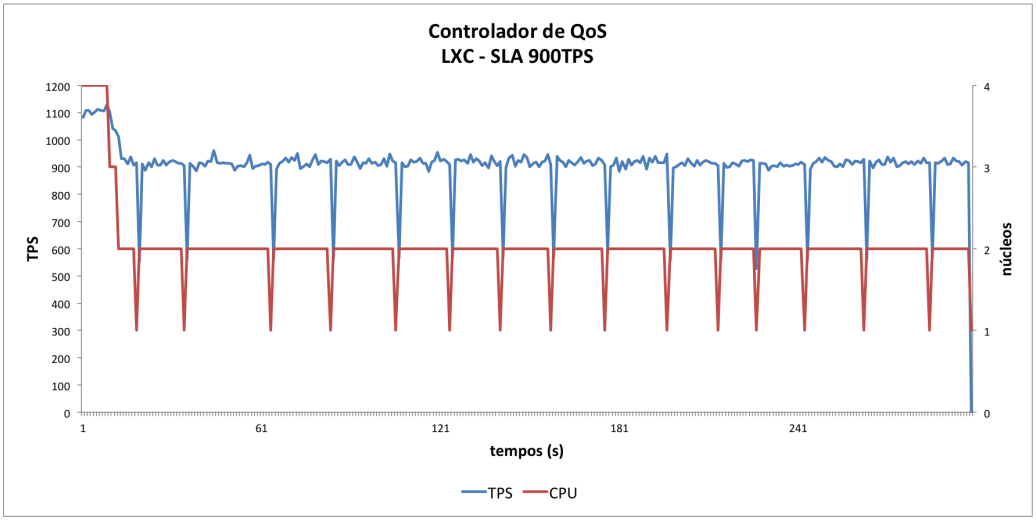

Figura 6.10 – Avaliação da Arquitetura com LXC. SLA 900TPS

Tabela 6.3 – Controlador de QoS - Comparativo Xen x LXC

Ambiente	Total de Transações	Média de TPS	SLA Atendido	Quebra SLA
XEN	103451	344,83	67.33%	33.66%
LXC	270020	900,06	93.66%	6.33%

Essa irregularidade na alocação e desalocação de CPUs ocorre devido a proposta original de Dawoud et al. [DTM12] não ter sido projetada para máquinas virtuais com mais de um núcleo de processamento, nem para ambientes virtualizados como *linux containers* que permitem manipulação da totalidade dos recursos do nodo. Assim, quando o controlador de QoS verifica a possibilidade de diminuir a alocação de CPU, por exemplo de 2 núcleos para 1 núcleo, existe uma queda considerável de desempenho fazendo com que

o controlador retorne a alocar 2 núcleos, criando uma situação de *thrashing* devido ao *overhead* de seguidas alocações/desalocações de recursos.

Todavia, ainda foi possível perceber que o controlador de QoS, proporcionou economia de recursos principalmente no caso do LXC, pois em 93.66%, dois núcleos de CPU foram suficientes para manter a qualidade do serviço gerando uma economia de 50% do recursos de CPU em relação a alocação inicial. Quanto ao consumo de memória, a economia foi de cerca 85% no caso do Xen e de 90% no caso do LXC.

Outro fato que chama a atenção é o ganho de desempenho do LXC frente ao Xen. Esse comportamento é esperado e pode ser explicado, como visto em [LCXDR13], devido à inclusão da uma camada do VMM necessária para a execução da máquina virtual no Xen. A seguir demostraremos os resultados de uma avaliação mais detalhada da arquitetura adaptada para o uso de *containers*

6.3.2 Avaliação preliminar da arquitetura elástica adaptada ao uso de *containers*

Para realizar esta avaliação, instanciamos um *container* sem limitação tanto de CPU quanto de memória, deixando a alocação de recursos totalmente a cargo do controlador de QoS. O objetivo deste experimento era fazer uma análise do problema relacionado a alocação de CPU, como vimos na Seção 6.3.1. Para esse experimento determinamos um SLA de 1000 TPS.

Na Figura 6.11 podemos observar novamente a irregularidade quanto a alocação e desalocação de CPUs. Esse comportamento leva ao mal funcionamento de Controlador de QoS, levando a quebra de SLA a 51% do tempo de execução do *benchmark*. Ainda analisando melhor o gráfico, podemos observar que o valor ideal de alocação seria um valor entre 1.0 e 2.0 núcleos.

Com a finalidade de resolver essa limitação realizamos algumas alterações no Controlador de QoS. Ao iniciar a execução, o Controlador de QoS passa por um período de "aprendizagem", para identificar os valores de alocação mínimos para atender o SLA informado. Após essa fase, o Controlador de QoS passa a não permitir a alocação de CPU com valores inferiores ao mínimo identificado.

A Figura 6.12 mostra os resultados após as modificações no controlador de QoS, em que existe uma maior regularidade no processo de alocação de CPU. Também percebe-se uma melhora significativa no desempenho do Controlador de QoS em manter a qualidade do serviço, pois nesse experimento o SLA foi atendido em 99.33% do tempo de execução.

Na próxima Seção apresentamos a avaliação de nossa arquitetura modificada para o uso de *Containers*, utilizando o Controlador de QoS modificado.

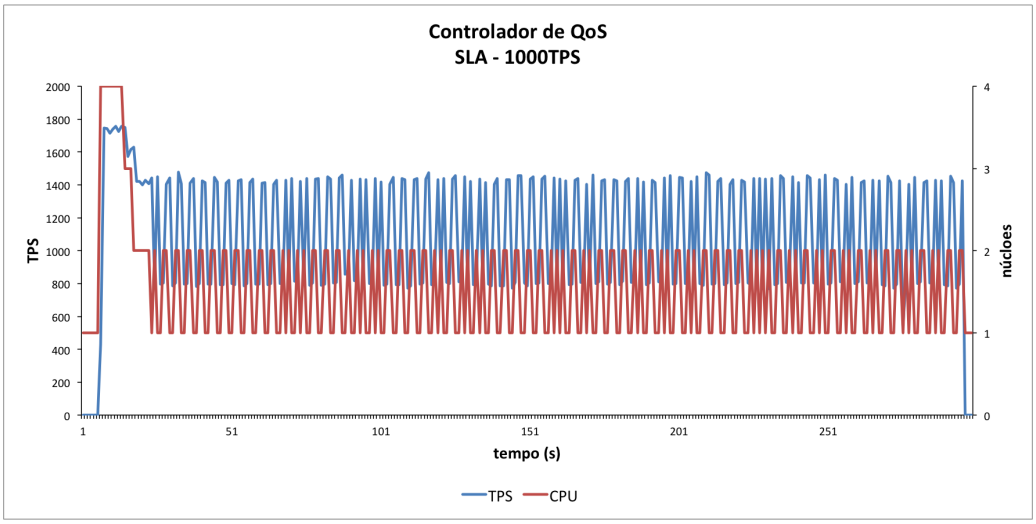

Figura 6.11 – LXC - SLA 1000 TPS - Arquitetura Original

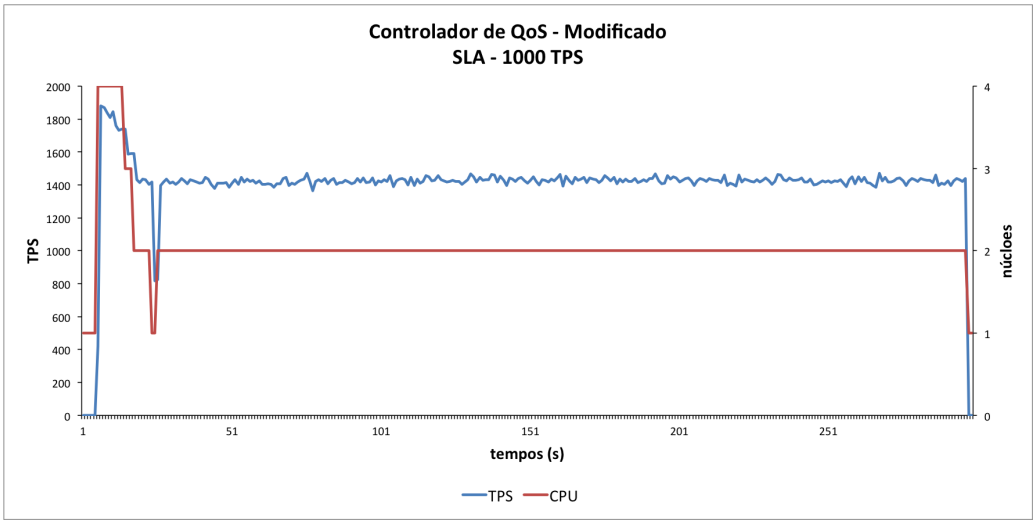

Figura 6.12 – LXC - SLA 100 TPS - Arquitetura Modificada

6.3.3 Avaliação da arquitetura elástica adaptada ao uso de *containers*

Com a finalidade de verificar a eficiência da arquitetura em manter os SLAs definidos, realizamos uma série de testes utilizando dois modos destintos de execução oferecidos pelo Sysbench: somente leitura e leitura/escrita. Para esses experimentos instanciamos um *container* sem limitação tanto de CPU quanto de memória, deixando a alocação de recursos totalmente a cargo do controlador de QoS. A seguir podemos observar os resultados na execução somente leitura.

Avaliação de desempenho quanto a operações de leitura

Neste experimento selecionamos três SLAs distintos: 800, 1300 e 1800 TPS respectivamente. Esses valores foram selecionados com base nos valores da execução do *benchmark* no ambiente nativo, com a finalidade de abranger um teto mínimo (800 TPS) um teto máximo (1800 TPS) e um valor médio (1300 TPS).

Na Figura 6.13, podemos observar o resultado obtido na execução do teste com SLA definido em 800 TPS. Nesse experimento houve manutenção do SLA em 99.4% do tempo de execução. Quanto a utilização de CPU, observamos que logo no início da execução o controlador de QoS alocou 3 núcleos de processamento, desalocando-os uma vez que identificou que 1 núcleo de CPU era suficiente para atender o SLA definido. A alocação de memória RAM não excedeu em nenhum momento 512MB.

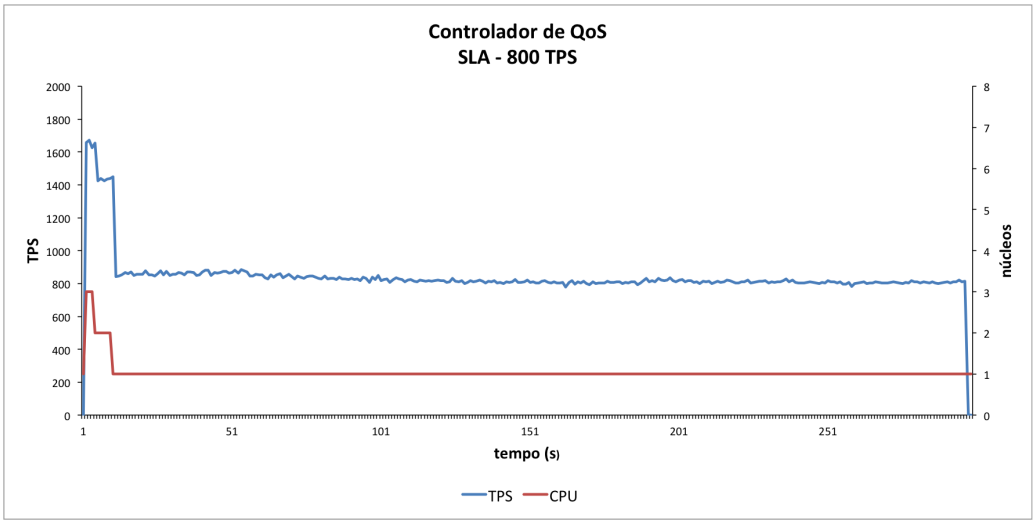

Figura 6.13 – LXC - SLA 800 TPS

No teste com SLA definido em 1300 TPS, também observamos a manutenção da qualidade de serviço em 99.4% do tempo de execução. Quanto a alocação de CPU, o controlador definiu 2 núcleos de processamento como valor ideal. A Figura 6.14 apresenta os resultados deste experimento. Como no caso anterior a alocação de memória não excedeu os 512MB

O último teste realizado, apresentado na Figura 6.15, tinha o objetivo de garantir no mínimo 1800 TPS, o que aconteceu em 96.4% do tempo de realização do teste. Esse percentual menor em relação aos outros experimentos deve-se ao maior tempo que Controlador de QoS necessitou para estabilizar a alocação de CPU, que neste caso levou 41 segundos, contra 12 segundos no teste com SLA em 800 TPS e 15 segundos na execução com SLA em 1300 TPS. Para essa execução, o Controlador de QoS definiu que seriam necessários 5 núcleos de CPU e 512MB de memória.

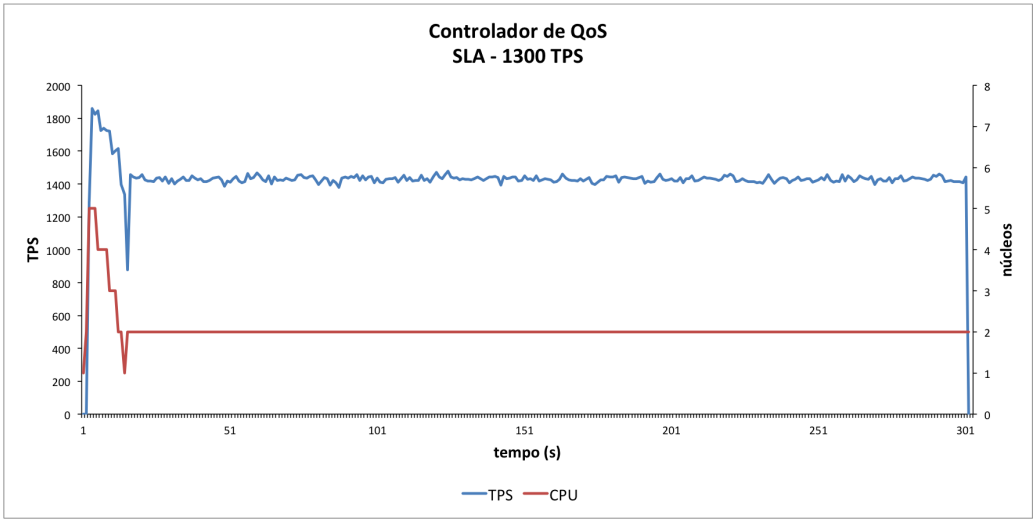

Figura 6.14 – LXC - SLA 1300 TPS

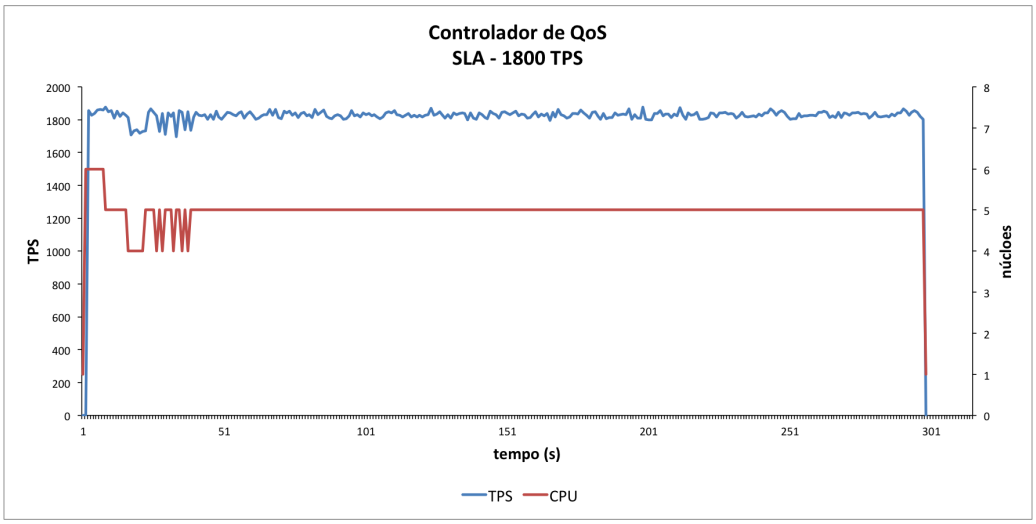

Figura 6.15 – LXC - SLA 1800 TPS

A Tabela 6.4 apresenta um resumo deste experimento. A seguir avaliaremos a arquitetura quanto a sua capacidade de manter a qualidade do serviço em transações que envolvem tanto leitura quanto escrita de dados

Tabela 6.4 – Avaliação Arquitetura - Somente Leitura

SLA (TPS)	Transações	Média TPS	SLA	Quebra SLA
800	250535	835.11	99.4%	0.6%
1300	429464	1431.54	99.4%	0.6%
1800	540708	1802.36	96.4%	3.6%

Avaliação utilizando operações de leitura/escrita

Esta avaliação consistiu na execução do *benchmark* no modo leitura/escrita. Para esse experimento selecionamos três SLAs distintos, 600, 900 e 1200 TPS, obedecendo o

mesmo critério do teste anterior que foi de selecionar valores que representassem o mínimo, a média e o máximo de TPS, observados em uma execução com o controlador de QoS desativado.

Na Figura 6.16, observamos o resultado da execução do *benchmark* com o SLA configurado para 600 TPS, onde o SLA foi atendido em 98% do tempo de execução do teste. Podemos observar que o Controlador de QoS na maior parte do tempo alocou 2 núcleos de CPU.

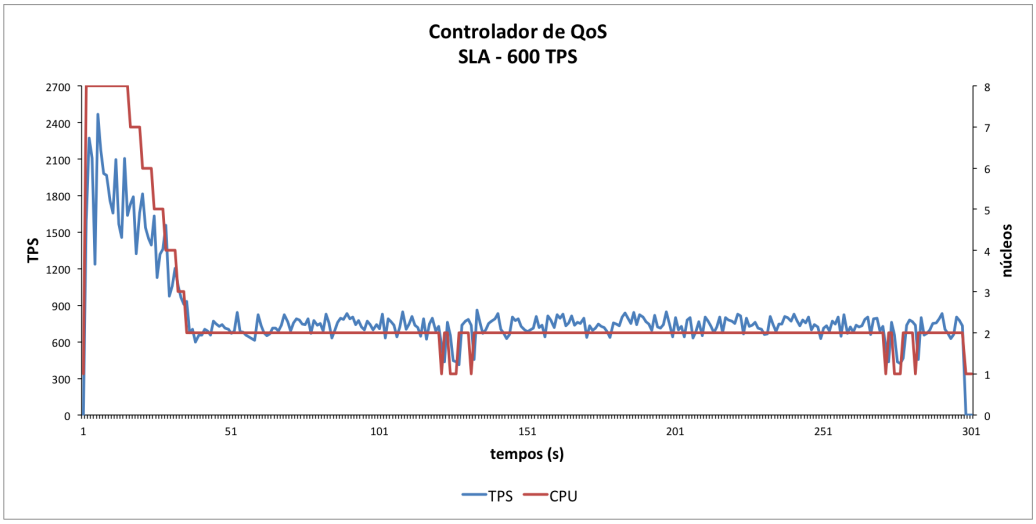

Figura 6.16 – LXC - SLA 600 TPS

Já na execução com SLA em 900 TPS como mostra a Figura 6.17, o objetivo foi atendido em 86.66%, tendo como média 1097.2 TPS. Podemos observar que quando as operações de E/S se intensificaram, aliviando a pressão sobre o processamento, o Controlador de QoS teve dificuldades em manter o SLA.

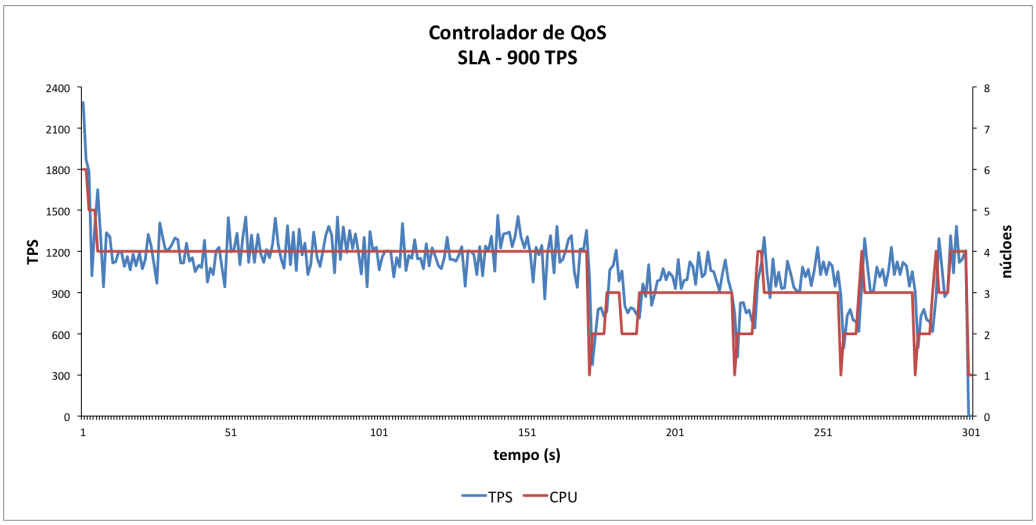

Figura 6.17 – LXC - SLA 900 TPS

Apesar de um comportamento diferente do experimento anterior (Figura 6.18), podemos observar um resultado semelhante, quando definimos o SLA em 1200 TPS, onde o objetivo foi alcançado em 87% do tempo de execução, com uma média de 1291.23 TPS.

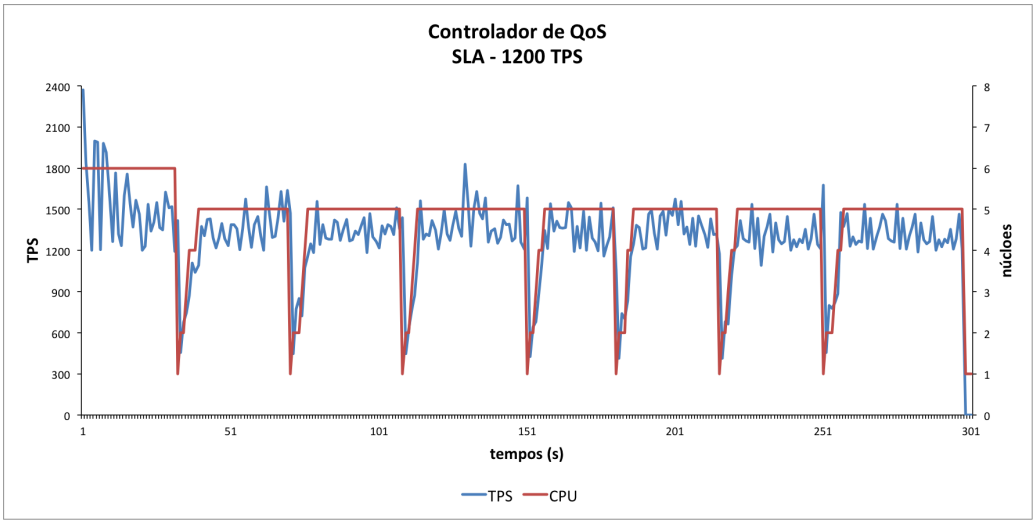

Figura 6.18 – LXC - 1200 TPS

Quanto ao consumo de memória, durante os três experimentos alocação de memória não ultrapassou os 1024MB. A Tabela 6.5 apresenta um resumo dos resultados obtidos nesta etapa de experimentos. A seguir, avaliamos o comportamento com múltiplos *containers* ao mesmo tempo.

Tabela 6.5 – Avaliação modo leitura/escrita

SLA (TPS)	Total de Transações	Média TPS	SLA Atendido	Quebra SLA
600	244401	814,67	98%	2%
900	329283	1097.61	86.66%	13.34%
1200	387369	1291.23	87%	13%

Avaliação da arquitetura com múltiplos *containers*

Esta avaliação consistiu na execução do teste utilizando dois *containers* sem limitação inicial de recursos, deixando este processo a cargo do Controlador de QoS. O objetivo desse experimento era verificar o comportamento do controlador ao manipular mais de uma instância virtual.

Para esse experimento executamos o *benchmark* no modo somente leitura, definindo para o *Container 1* um SLA de 900 TPS e para o *Container 2* um SLA de 1800 TPS.

Na Figura 6.19 podemos observar o resultado do teste, onde em ambos os *containers* o SLA foi atendido em 99.64% do tempo de execução. A Tabela 6.6 apresenta os resultados com maiores detalhes.

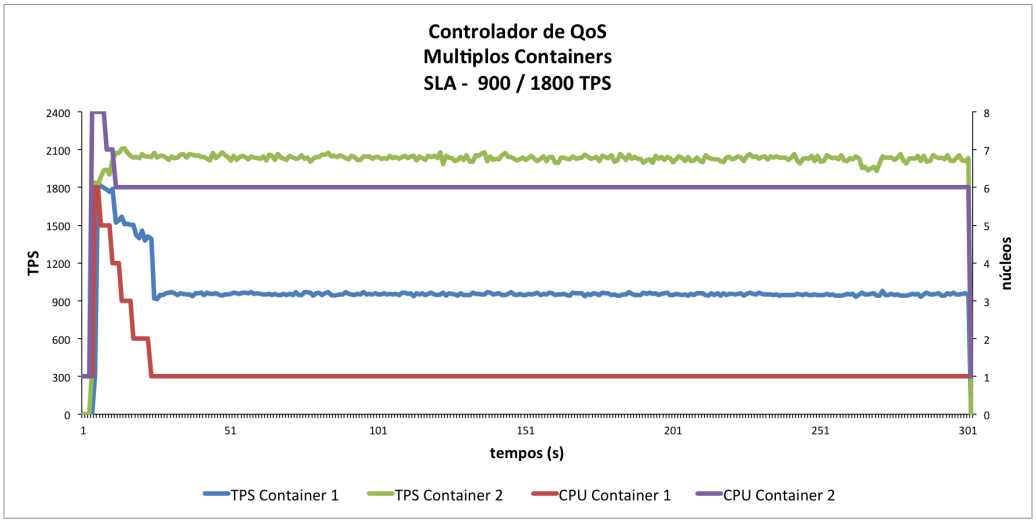

Figura 6.19 – Avaliação Múltiplos *Containers*

Com este experimento podemos concluir que o Controlador de QoS obteve resultado satisfatório ao se utilizar dois *containers*, todavia para a execução em um ambiente de produção, utilizando múltiplos *containers*, é necessário implementar componentes que venham amenizar algumas limitações do LXC, como por exemplo a falta de isolamento de recursos. [XNR+13]

Tabela 6.6 – Avaliação Múltiplos *Containers*

SLA	Total Transações	Media TPS	SLA Atendido	Quebra SLA
900	294050	980,16	99.34%	0.66%
1800	603432	2011,44	99.34%	0.66%

6.4 Considerações Finais

Nesta seção, os experimentos com o objetivo de avaliar a arquitetura elástica adaptada ao uso de *containers* foram apresentados, executados e avaliados.

O primeiro experimento executado, como visto na Seção 6.1, demonstra a capacidade individual dos atuadores em lidar com a flutuação da carga de trabalho, oferecendo ganhos consideráveis de desempenho quando comparado a um modelo tradicional de alocação de recursos.

Os experimentos, apresentados nas Seções 6.3.1 e 6.3.2, mostram que a utilização da virtualização em nível de Sistema Operacional trouxe flexibilidade e melhoria de desempenho considerável à arquitetura original. Ainda através destes experimentos identificamos a possibilidade de realizar melhorias na arquitetura, como a otimização para o uso de instâncias virtuais *multicore* e a implementação de uma fase de aprendizagem, possibilitando que resultados ainda melhores fossem atingidos.

Por fim, os experimentos vistos na Seção 6.3.3 demonstram a capacidade de nossa arquitetura em ajustar os recursos da instância virtual conforme as flutuações da carga de trabalho, mantendo a qualidade do serviço dentro de níveis desejados, utilizando como carga de trabalho aplicações de banco de dados.

No Capítulo a seguir apresentamos a conclusão deste trabalho bem como os trabalhos futuros a serem desenvolvidos.

7. CONCLUSÃO E TRABALHOS FUTUROS

A computação em nuvem trouxe inúmeros benefícios para os usuários que tem a sua disposição recursos computacionais com alta disponibilidade e escalabilidade. Junto com as diversas vantagens que a computação em nuvem oferece, vieram também inúmeros desafios, e dentre estes, um dos mais importantes consiste em alocar recursos de forma inteligente para determinadas aplicações.

Porém, a decisão sobre qual a real necessidade de recursos, como processador, memória e rede, que a aplicação necessita não é uma tarefa trivial. Devido ao comportamento arbitrário de muitas aplicações atuais, a mensuração e o gerenciamento manual por parte do usuário pode ser algo impossível. Outro fator importante consiste em que, quando o usuário aloca uma máquina virtual com uma certa quantidade de recursos, os recursos reais do nodo em que essa instância estará em execução são abstraídos do usuário. Isso significa que suas aplicações que executam sobre ambientes de nuvem podem estar compartilhando recursos, o que levaria a uma estimação errada sobre a necessidade da aplicação, devido a interferência causada pelos fatores externos ao conhecimento do usuário, como outras máquinas virtuais instanciadas no mesmo nodo e competindo pelos mesmos recursos.

Com foco nessas questões, alguns trabalhos tentam realizar alocações dinâmicas dos recursos conforme o comportamento da aplicação e com uma visão holística sobre a utilização dos recursos, com o mínimo ou sem a necessidade de intervenção do usuário. A alocação dinâmica de recursos, além de garantir que o serviço execute dentro dos níveis de qualidade exigidos pelos usuário, com o mínimo de recursos possível, ainda traz uma série de benefícios para os provedores de nuvem como a melhor utilização de recursos, o que acarreta em um menor consumo de energia e resfriamento, impactando diretamente nas métricas de sustentabilidade.

O melhor aproveitamento dos recursos se deve às características intrínsecas da própria virtualização, que permite que máquinas virtuais possam aumentar ou diminuir a quantidade de recursos necessárias para prover qualidade de serviço à aplicação, dentro de níveis estabelecidos entre o usuário e o provedor. Porém, a camada de virtualização tradicional também traz consigo algumas limitações, como por exemplo o *overhead* de desempenho causado pelo *hypervisor* e limitações quanto a alocação de recursos, que fica restrita a valores definidos pelo usuário no momento da criação de uma máquina virtual.

Tentando resolver essas questões, novas propostas de virtualização tem surgido, e este trabalho discute a utilização de uma destas, chamada virtualização em nível de sistema operacional. Essa nova proposta de virtualização apresenta taxas de desempenho muito próximas a um ambiente não-virtualizado, pois ao invés de existir uma camada de virtualização que suporte e interprete chamadas advindas de máquinas virtuais, o ambiente

computacional é dividido em *namespaces* que agregam processos de um mesmo usuários, os mantendo dentro de *containers*. Como o controle desses *containers* é realizado diretamente pelo *kernel*, praticamente não existe *overhead*, como também não existe restrições quanto a alocação de recursos, ficando esta limitada a totalidade de recursos existentes na máquina física. A virtualização em nível de sistema operacional prima pelo desempenho e simplicidade de gerenciamento, portanto algumas questões como isolamento ainda devem ser melhor implementadas [XOR+15] [XNR+13].

Com base nessa nova proposta de virtualização, este trabalho teve como objetivo avaliar impacto do uso da virtualização em nível de Sistema Operacional sobre a estratégia de alocação dinâmica de recursos proposta por Dawoud et al. [DTM12]. Em nossas avaliações, utilizamos aplicações de banco de dados devido a sua grande utilização em ambientes virtualizados, bem como aproveitamento da experiência do grupo de pesquisa com esse tipo de aplicação. A alocação dinâmica de recursos desse trabalho buscou manter SLAs voltados ao desempenho das aplicações de forma que refletisse um menor impacto sobre questões, como por exemplo a qualidade de experiência dos usuários.

Para tanto, descrevemos os conceitos teóricos envolvidos na pesquisa, bem como a proposta, implementação e avaliação da arquitetura elástica desenvolvida por Dawoud et al. [DTM12] modificada para utilização de *containers*. O componente principal da arquitetura consiste no Controlador de QoS, que através do atuador de CPU e do Atuador de Memória, calculam valores referentes à quantidade de recursos necessários para atender o SLA definido.

Após, realizamos testes utilizando diferentes aplicações com comportamentos variados com o objetivo de mostrar a validade de nossa proposta frente à heterogeneidade na utilização de recursos, característica encontrada nos ambientes de nuvens. Conforme apresentamos no Capítulo 6, os resultados obtidos pela arquitetura utilizando *containers* são promissores, pois já e possível verificar que consegue-se um bom nível de atendimento aos SLAs com a utilização da estratégia proposta. Nossos resultados mostraram uma quebra de SLAs em cerca de 13% no pior caso, explicado devido à algumas limitações dessa proposta quanto a inexistente alocação de E/S. No melhor caso, a manutenção de SLAs ocorre em mais de 99% do tempo de execução total das aplicações.

Porém, existem ainda melhorias que podem ser feitas com a finalidade de aprimorar ainda mais esta estratégia, como veremos a Seção a seguir.

7.1 Trabalhos Futuros

Considerando os benefícios proporcionados neste trabalho, os seguintes tópicos podem ser candidatos de um estudo em maior profundidade e apresentados como trabalhos futuros:

- Elaboração de um atuador de E/S, possibilitando assim um que o Controlador de QoS se adapte melhor a cargas de trabalho que tem operações de E/S intensivas. Acreditamos que a elaboração deste tipo de atuador irá melhorar os resultados vistos na Seção 6.3.3, onde avaliamos operações tanto de leitura como de escrita no banco de dados.
- Atualmente, a utilização de *containers* está se popularizando tanto em ambientes de nuvem através do Docker [Doc], quanto em ambientes de alto desempenho, como no caso do Yarn [Yar]. Desta forma seria interessante a criação mecanismos de integração entre a arquitetura elástica apresentada neste trabalho e estes ambientes.
- Criação de um mecanismo global de controle para a utilização de múltiplos *containers* em múltiplos nodos físicos, possibilitando a utilização desta arquitetura em ambientes complexos.

REFERÊNCIAS BIBLIOGRÁFICAS

[AAB+08] Agrawal, R.; Ailamaki, A.; Bernstein, P. A.; Brewer, E. A.; Carey, M. J.; Chaudhuri, S.; Doan, A.; Florescu, D.; Franklin, M. J.; Garcia-Molina, H.; et al.. “The claremont report on database research”, *ACM SIGMOD Record*, vol. 37–3, 2008, pp. 9–19.

[AEADE11] Agrawal, D.; El Abbadi, A.; Das, S.; Elmore, A. J. “Database scalability, elasticity, and autonomy in the cloud”. In: Database Systems for Advanced Applications, 2011, pp. 2–15.

[AETE12] Ali-Eldin, A.; Tordsson, J.; Elmroth, E. “An adaptive hybrid elasticity controller for cloud infrastructures”. In: Network Operations and Management Symposium (NOMS), 2012, pp. 204–212.

[Arm09] Armbrust, M. “Above the clouds: A berkeley view of cloud computing.”, Relatório Técnico, EECS Department, University of California, Berkeley, 2009.

[ASS+09] Aboulnaga, A.; Salem, K.; Soror, A. A.; Minhas, U. F.; Kokosielis, P.; Kamath, S. “Deploying database appliances in the cloud.”, *IEEE Data Eng. Bull.*, vol. 32–1, 2009, pp. 13–20.

[AUT] “Amazon autoscaling”. Acessado: 02/10/2014, Capturado em: http://aws.amazon.com/autoscaling.

[AWS] “Amazon web services (aws) - serviços de computação em nuvem”. Acessado: 02/010/2015, Capturado em: http://aws.amazon.com/pt/.

[Azu] “Azurewatch: Monitoring and autoscaling features for windows azure”. Acessado: 10/01/2015, Capturado em: http://www.paraleap.com/azurewatch.

[BDF+03] Barham, P.; Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Neugebauer, R.; Pratt, I.; Warfield, A. “Xen and the art of virtualization”, *ACM SIGOPS Operating Systems Review*, vol. 37–5, 2003, pp. 164–177.

[BK12] Bosing, A.; Kaufmann, E. R. “Virtualização de servidores e desktops”, *Unoesc & Ciência-ACET*, vol. 3–1, 2012, pp. 47–64.

[BMVO12] Beernaert, L.; Matos, M.; Vilaça, R.; Oliveira, R. “Automatic elasticity in openstack”. In: Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management, 2012, pp. 2:1–2:6.

[Car08] Carissimi, A. “Virtualização: da teoria a soluções”, *Minicursos do Simpósio Brasileiro de Redes de Computadores–SBRC*, vol. 2008, 2008, pp. 173–207.

[CCS10] Cui, Y.; Chen, Y.; Shi, Y. “Scaling oltp applications on commodity multi-core platforms”. In: Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on, 2010, pp. 134–143.

[CCW+08] Chaudhary, V.; Cha, M.; Walters, J.; Guercio, S.; Gallo, S. “A comparison of virtualization technologies for hpc”. In: Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on, 2008, pp. 861–868.

[CLO] “Amazon cloudwatch: Serviço de monitoramento de nuvem e rede”. Acessado: 02/10/2014, Capturado em: http://aws.amazon.com/cloudwatch.

[CMKS09] Chieu, T.; Mohindra, A.; Karve, A.; Segal, A. “Dynamic scaling of web applications in a virtualized cloud computing environment”. In: e-Business Engineering, 2009. ICEBE ’09. IEEE International Conference on, 2009, pp. 281–286.

[CSGS13] Coutinho, E.; Sousa, F. R.; Gomes, D. G.; Souza, J. “Elasticidade em computaçao na nuvem: Uma abordagem sistemática”, *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2013)-Minicursos*, 2013.

[CVKB12] Calheiros, R. N.; Vecchiola, C.; Karunamoorthy, D.; Buyya, R. “The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds”, *Future Gener. Comput. Syst.*, vol. 28–6, Jun 2012, pp. 861–870.

[dac] “Dacapo benchmark suite”. Acessado: 10/01/2015, Capturado em: www.dacapobench.org.

[Doc] “Docker”. Acessado: 15/01/2015, Capturado em: http://www.docker.com.

[DTM12] Dawoud, W.; Takouna, I.; Meinel, C. “Elastic virtual machine for fine-grained cloud resource provisioning”. In: *Global Trends in Computing and Communication Systems*, Krishna, P.; Babu, M.; Ariwa, E. (Editores), Springer Berlin Heidelberg, 2012, *Communications in Computer and Information Science*, vol. 269, pp. 11–25.

[EBMD10] Emeakaroha, V. C.; Brandic, I.; Maurer, M.; Dustdar, S. “Low level metrics to high level slas-lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments”. In: High Performance Computing and Simulation (HPCS), 2010 International Conference on, 2010, pp. 48–54.

[Fis06] Fisher-Ogden, J. “Hardware support for efficient virtualization”, *University of California, San Diego, Tech. Rep*, 2006.

[FPG10] Fitó, J. O.; Presa, Í. G.; Guitart, J. “Sla-driven elastic cloud hosting provider”. In: 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, 2010, pp. 111–118.

[GdB12] Galante, G.; de Bona, L. “A survey on cloud computing elasticity”. In: Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on, 2012, pp. 263–270.

[GGW10] Gong, Z.; Gu, X.; Wilkes, J. “Press: Predictive elastic resource scaling for cloud systems”. In: Network and Service Management (CNSM), 2010 International Conference on, 2010, pp. 9–16.

[GND+07] Govindan, S.; Nath, A. R.; Das, A.; Urgaonkar, B.; Sivasubramaniam, A. “Xen and co.: Communication-aware cpu scheduling for consolidated xen-based hosting platforms”. In: Proceedings of the 3rd International Conference on Virtual Execution Environments, 2007, pp. 126–136.

[GOG] “Gogrid”. Acessado: 10/01/2015, Capturado em: http://www.gogrid.com.

[Gol73] Goldberg, R. P. “Architecture of virtual machines”. In: Proceedings of the workshop on virtual computer systems, 1973, pp. 74–112.

[Goo] “Google app engine: Platform as a service”. Acessado: 13/10/2014, Capturado em: https://cloud.google.com/appengine/docs.

[Hel09] Helsley, M. “Lxc: Linux container tools”, *IBM devloperWorks Technical Library*, 2009.

[HGG+14] Han, R.; Ghanem, M. M.; Guo, L.; Guo, Y.; Osmond, M. “Enabling cost-aware and adaptive elasticity of multi-tier cloud applications”, *Future Generation Computer Systems*, vol. 32, 2014, pp. 82–98.

[Hin11] Hindman, Benjamin e Konwinski, A. e. Z. M. e. G. “Mesos: A plataform for fine-grained resources sharing in the data center”, *Usenix Symposium on Networked Systems Design and Implementation*, vol. 3–1, Mar 2011.

[Hon03] Honeycutt, J. “Microsoft virtual pc 2004 technical overview”, *Microsoft, Nov*, 2003.

[KCH09] Kalyvianaki, E.; Charalambous, T.; Hand, S. “Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters”. In: Proceedings of the 6th International Conference on Autonomic Computing, 2009, pp. 117–126.

[KHvKR11] Kuperberg, M.; Herbst, N.; von Kistowski, J.; Reussner, R. “Defining and quantifying elasticity of resources in cloud computing and scalable platforms”. KIT, Fakultät für Informatik, 2011.

[Kop12] Kopytov, A. “Sysbench manual”, *MySQL AB*, 2012.

[LBCP09] Lim, H. C.; Babu, S.; Chase, J. S.; Parekh, S. S. “Automated control in cloud computing: Challenges and opportunities”. In: Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds, 2009, pp. 13–18.

[LCXDR13] Lange, T.; Cemim, P.; Xavier, M.; De Rose, C. “Optimizing the management of a database in a virtual environment”. In: Computers and Communications (ISCC), 2013 IEEE Symposium on, 2013, pp. 000594–000599.

[LM08] Laureano, M. A. P.; Maziero, C. A. “Virtualização: Conceitos e aplicações em segurança.”, *PUCPR*, 2008.

[Lom] Lomet, D. “Letter from the editor-in-chief bulletin related items”, *Data Engineering*, vol. 51, pp. 1.

[LP10] Lotti, L. P.; Prado, A. B. “Sistemas virtualizados–uma visão geral”, 2010.

[LWN] “Lwn -linux info from the source : Namespaces in operation”. Acessado: 22/10/2014, Capturado em: http://lwn.net/Articles/531114/.

[MBL+13] Morais, F.; Brasileiro, F.; Lopes, R.; Araújo, R.; Macedo, A.; Satterfield, W.; Rosa, L.; Grande-Brasil, C. “Um arcabouço para provisionamento automático de recursos em provedores de iaas independente do tipo de aplicaç ao”, 2013.

[Mel11] Mell Peter, T. G. “The nist definition of cloud computing”, *Special Publication 800-145*, 2011.

[MKF10] Marshall, P.; Keahey, K.; Freeman, T. “Elastic site: Using clouds to elastically extend site resources”. In: Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, 2010, pp. 43–52.

[MLS10] Meng, S.; Liu, L.; Soundararajan, V. “Tide: Achieving self-scaling in virtualized datacenter management middleware”. In: Proceedings of the 11th International Middleware Conference Industrial Track, 2010, pp. 17–22.

[MSA] “Azure: A plataforma de computação em nuvem da microsoft”. Acessado: 02/10/2014, Capturado em: http://azure.microsoft.com/pt-br/.

[NIM] “Nimbus project”. Acessado: 10/01/2014, Capturado em: http://www.nimbusproject.org, Agosto.

[OPE] “Openstack open source cloud computing software”. Acessado: 02/10/2014, Capturado em: http://www.openstack.org/.

[PHA] “Nimbus phantom documentation”. Acessado: 02/10/2014, Capturado em: http://www.nimbusproject.org/doc/phantom/.

[RBA11] Raveendran, A.; Bicer, T.; Agrawal, G. “A framework for elastic execution of existing mpi programs”. In: Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, 2011, pp. 940–947.

[RED] “Redhat: Guia de gerenciamento de recursos”. Acessado: 16/10/2014, Capturado em: https://access.redhat.com/documentation/.

[RG11] Redkar, T.; Guidici, T. “Windows Azure Platform”. Springer, 2011, vol. 1.

[Rig] “Rightscale: Cloud portfolio management”. Acessado: 15/01/2015, Capturado em: http://www.rightscale.com.

[Rig13] Righi, R. R. “Elasticidade em cloud computing: conceito, estado da arte e novos desafios”, *Revista Brasileira de Computação Aplicada*, vol. 5–2, 2013, pp. 2–17.

[RUB] “Rubis benchmark: Rice university bidding system”. Acessado: 10/01/2014, Capturado em: http://rubis.ow2.org/.

[Sca] “Scalr: Enterprise cloud management platform”. Acessado: 10/01/2015, Capturado em: http://www.scalr.net.

[SdO07] Silva, R. F.; de Oliveira, F. B. “virtualização de sistemas operacionais”, 2007.

[SMI05] SMITH, J. E.; NAIR, R. “The architecture of virtual machines. computer”, *IEEE Computer Society Press*, vol. 38–5, May 2005, pp. 32–38.

[SMM09] Sousa, F. R.; Moreira, L. O.; Machado, J. C. “Computação em nuvem: Conceitos, tecnologias, aplicações e desafios”, *II Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI)*, 2009, pp. 150–175.

[SMMM10] Sousa, F. R.; Moreira, L. O.; Macêdo, J.; Machado, J. C. “Gerenciamento de dados em nuvem: Conceitos, sistemas e desafios”. In: SBBD, 2010, pp. 101–130.

[SSGW11] Shen, Z.; Subbiah, S.; Gu, X.; Wilkes, J. “Cloudscale: Elastic resource scaling for multi-tenant cloud systems”. In: Proceedings of the 2Nd ACM Symposium on Cloud Computing, 2011, pp. 5:1–5:14.

[SSRA+11] Sobeslavsky, P.; Sardes, T.; Rhône-Alpes, I.; De Palma, N.; Boyer, F.; Dillenseger, B. “Elasticity in cloud computing”, *Master’s thesis, Joseph Fourier University, ENSIMAG, Grenoble, France*, 2011.

[TAU] “Ibm developerworks: Elasticiade em cloud computing”. Acessado: 10/12/2014, Capturado em: https://www.ibm.com/developerworks.

[TPC] “Tpc: Transaction processing performance concil”. Acessado: 10/12/2014, Capturado em: http://www.tpc.org.

[USC+08] Urgaonkar, B.; Shenoy, P.; Chandra, A.; Goyal, P.; Wood, T. “Agile dynamic provisioning of multi-tier internet applications”, *ACM Trans. Auton. Adapt. Syst.*, vol. 3–1, Mar 2008, pp. 1:1–1:39.

[VMW] “Vmware virtualization”. Acessado: 10/01/2014, Capturado em: http://www.vmware.com.

[VR13] Verderami, B. M.; Rosa, R. “Avaliando o uso da computação em nuvem na ti para pequenas e médias empresas brasileiras”, *Revista Computação Aplicada-UnG*, vol. 2–1, 2013, pp. 05–14.

[VV09] Velte, A.; Velte, T. “Microsoft virtualization with Hyper-V”. McGraw-Hill, Inc., 2009.

[XDODPDR14] Xavier, M. G.; De Oliveira, I. C.; Dos Passos, R. D.; De Rose, C. A. “Towards better manageability of database clusters on cloud computing platforms”. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing, 2014, pp. 366–367.

[XEN] “Xentop : Linux fundation colaborative projects”. Acessado: 10/10/2014, Capturado em: http://wiki.xen.org/wiki/Xentop.

[XNR+13] Xavier, M.; Neves, M.; Rossi, F.; Ferreto, T.; Lange, T.; De Rose, C. “Performance evaluation of container-based virtualization for high performance computing environments”. In: Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on, 2013, pp. 233–240.

[XOR+15] Xavier, M.; Oliveira, I.; Rossi, F.; Dos Passos, R.; Mateussi, K.; De Rose, C. “A performance isolation analysis of disk-intensive workloads on container-based clouds”. In: Parallel, Distributed and Network-Based Processing (PDP), 2015 23st Euromicro International Conference on, 2015, pp. Aceito para Publicação.

[Yar] “Apache hadoop”. Acessado: 16/12/2014, Capturado em: http://hadoop.apache.org.

[ZWS06] Zhu, X.; Wang, Z.; Singhal, S. “Utility-driven workload management using nested control design”. In: American Control Conference, 2006, 2006, pp. 6 pp.–.