Estratégias para Alocação Dinâmica de Recursos em um Ambiente Híbrido de Computação em Nuvem

Paulo Antonio Leal Rego[1], Emanuel Ferreira Coutinho[1,2], José Neuman de Sousa[1]

[1]Mestrado e Doutorado em Ciência da Computação (MDCC)
Universidade Federal do Ceará (UFC) – Fortaleza, CE – Brasil

[2]Instituto UFC Virtual
Universidade Federal do Ceará (UFC) – Fortaleza, CE – Brasil

pauloalr@ufc.br, emanuel@virtual.ufc.br, neuman@ufc.br

Abstract. *With the rise of cloud computing, many applications are migrating to such environments. Increasingly resources are needed to fulfill the applications SLA requirements. These resources are rarely static, because they vary over time due the workload changes. The purpose of this work is to present a solution for dynamic and autonomic resource allocation in a cloud computing environment. This work describes strategies, based on the creation of virtual machines and dynamic allocation of resources, to meet SLA requirements and ensure the applications quality of service in a hybrid cloud computing environment. The solution leverages the FairCPU architecture to dynamically adjust the amount of CPU according to thresholds pre-defined by the user. Adding to that, several experiments were conducted to compare three different strategies.*

Resumo. *Com a ascensão da Computação em Nuvem, muitas aplicações estão migrando para este tipo de ambiente. Cada vez mais recursos são necessários para atender os requisitos de SLA das aplicações e esses recursos são raramente estáticos, pois variam com o tempo, dada as mudanças na carga de trabalho. O objetivo deste trabalho é apresentar uma solução para alocação dinâmica e autonômica de recursos em um ambiente de Computação em Nuvem. Este trabalho descreve estratégias, baseadas na criação de máquinas virtuais e alocação dinâmica de recursos, para atender os requisitos de SLA e garantir a qualidade de serviço das aplicações em um ambiente híbrido de Computação em Nuvem. A solução aproveita a arquitetura FairCPU para ajustar dinamicamente a quantidade de CPU conforme limites pré-estabelecidos pelo usuário. Além disso, experimentos foram conduzidos para comparar três estratégias.*

1. Introdução

Atualmente, muitos ambientes de Computação em Nuvem hospedam várias aplicações com diferentes características. É natural que tais aplicações possuam diferentes Acordos de Nível de Serviço (*Service Level Agreement* - SLA).

Nuvens computacionais são tipicamente *datacenters* virtualizados em larga escala, que hospedam milhares de servidores. Apesar de existirem diversas vantagens ao se utilizar infraestruturas virtualizadas, como escalabilidade de recursos sob demanda, existem também questões que impedem sua utilização generalizada. Em particular, para o sucesso

comercial desse paradigma, as infraestruturas de Computação em Nuvem precisam prover uma melhor Qualidade de Serviço (*Quality of Service* - QoS) [Wu et al. 2011].

Alocar recursos da infraestrutura para que esta atenda ao SLA das aplicações não é uma tarefa fácil. Muitos parâmetros podem ser utilizados como critérios para a alocação dos recursos. Algumas das métricas utilizadas são geralmente relacionadas ao consumo de CPU e memória, mas qualquer fator pode ser utilizado como entrada para tomada de decisão na alocação de recursos.

A quantidade de recursos necessária para as aplicações são raramente estáticas, pois variam com o resultado de mudanças em cargas de trabalho global, e aplicações internas. O sub-provisionamento de recursos pode causar violações dos Objetivos de Nível de Serviço (*Service Level Objective* - SLO), que são frequentemente associados com significativas penalidades financeiras. O sobre-provisionamento desperdiça recursos que podem ser utilizados em outros casos. Para evitar ambos problemas, a quantidade de recursos alocada para aplicações pode ser ajustada dinamicamente [Gong et al. 2010].

Este artigo tem como objetivo trabalhar sobre o desafio de prover recursos computacionais sob demanda para garantir QoS das aplicações executadas em máquinas virtuais (MVs). Como resultado, uma solução autonômica foi desenvolvida (estendendo a arquitetura FairCPU [Rego et al. 2011]) para modificar dinamicamente a quantidade de recursos de CPU e instanciar/remover MVs. Além disso, experimentos foram realizados em uma nuvem híbrida, utilizando um balanceador de carga para distribuir as requisições de uma aplicação *web*, alinhada a diferentes estratégias para a alocação dinâmica de recursos.

O trabalho está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados; a Seção 3 descreve a solução proposta, detalhando os componentes da arquitetura; na Seção 4 é descrita a metodologia de avaliação; a Seção 5 apresenta os experimentos e discute os resultados; finalmente, a conclusão e trabalhos futuros são apresentadas na Seção 6.

2. Trabalhos relacionados

Diversos trabalhos consideram múltiplos aspectos do ambiente para alocação de recursos, baseando-se nesses parâmetros para a tomada de decisão. A maioria dos trabalhos executa a coleta de informações dos ambientes e os utilizam para decidir a alocação dos recursos.

Em [Zhang et al. 2010], uma abordagem para o gerenciamento de recursos da infraestrutura utilizando um *loop* de controle adaptativo foi proposta. Foram utilizados dois *loops* de controle: um para alocar os recursos e outro para otimizar o consumo. Seus experimentos utilizaram o Xen e trabalharam no nível de PaaS (*Platform as a Service*). Este trabalho é genérico para se adequar a qualquer parâmetro ou aspecto, como CPU e memória, porque ele monitora métricas de desempenho e pode modificá-la sempre que necessário.

Em [Heo et al. 2009] foi executado um experimento que verifica como recursos de memória se comportam na consolidação de um ambiente, e verifica como memória e CPU se comportam em diferentes aplicações. Foi desenvolvido um controle para alocar dinamicamente memória e garantir que as MVs possuam memória suficiente para executar suas aplicações. Foi utilizado o Xen como hipervisor nos experimentos.

Em [Emeakaroha et al. 2011], uma heurística para a alocação de recursos conside-

rando diversos aspectos do ambiente é apresentada. Esta heurística inclui um mecanismo para o balanceamento de carga para a distribuição da execução de uma aplicação e uma estratégia para criação de novas máquinas virtuais. Os experimentos utilizaram o simulador CloudSim [Calheiros et al. 2011] para cargas de trabalho heterogêneas.

No trabalho de [Pokluda et al. 2010], um *framework* para o gerenciamento dinâmico de memória foi desenvolvido baseado nos conceitos de sensores e atuadores. Páginas de memória foram utilizadas e ações foram tomadas quando necessário. Neste trabalho, o hipervisor OpenVZ foi utilizado.

[Oliveira 2012] utilizou uma estratégia para alocação de recursos baseada em CPU. A solução apresenta um monitor, para o monitoramento da utilização de CPU de máquinas físicas, e um gerente, que utiliza a informação coletada para alocar ou desalocar recursos, adicionando ou removendo MVs. O Xen foi o hipervisor utilizado neste trabalho.

[Wu et al. 2011] propôs algoritmos para a alocação de recursos com o objetivo de minimizar o custo da infraestrutura e violações de SLA. As necessidades do cliente são mapeadas para parâmetros da infraestrutura, como tempo de resposta. O CloudSim foi utilizado para simular o ambiente de Computação em Nuvem e executar os algoritmos propostos para a alocação dos recursos.

A Tabela 1 apresenta alguns trabalhos relacionados. Nota-se que a maioria dos trabalhos utilizou CPU ou memória como parâmetros para a alocação dos recursos. A abordagem proposta neste artigo utiliza variação de CPU e o hipervisor KVM, diferente da maioria dos outros trabalhos, que utilizam o Xen. Além disso, a arquitetura FairCPU permite a criação de diferentes estratégias, sendo que três delas foram avaliadas nesse trabalho.

Tabela 1. Comparação entre trabalhos relacionados (M=medição, S=simulação, MA=modelagem analítica).

Artigo	Assunto	Mecanismo de Alocação	Técnica
[Zhang et al. 2010]	Aplicação *Web* (TomCat)	*Threads* do servidor *web* e memória	M-MA
[Heo et al. 2009]	Aplicação *Web* (Apache)	CPU e memória	M-MA
[Emeakaroha et al. 2011]	Aplicação *Web* e HPC / CloudSim	Quantidade de aplicações que as MVs suportam	S
[Pokluda et al. 2010]	Aplicação *Web* (Apache)	Memória	M
[Oliveira 2012]	Aplicação *Web* (TomCat) e banco de dados	Uso de CPU	M
[Wu et al. 2011]	CloudSim	Tempo de resposta e custo	S

3. Solução Proposta

De acordo com [Kephart and Chess 2003], um sistema autonômico compreende um conjunto de elementos autonômicos. Um elemento autonômico é um componente responsável para o gerenciamento de seu próprio comportamento, em conformidade com políticas definidas, e para interação com outros elementos autonômicos que provêm ou consomem serviços computacionais.

O elemento básico é o *loop* de controle, exibido na Figura 1. Este *loop* atua

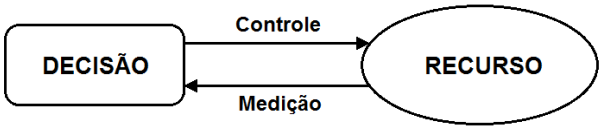

Figura 1. *Loop* de controle.

como um gerenciador de recursos através de ações de monitoramento, análise e tomada de decisão, seguindo políticas particulares.

Um *loop* de controle inteligente é capaz de prover capacidades de computação autonômica, tais como ciclos de processamento adicionais quando necessário, atualizações de software, reinicialização da execução de uma aplicação após falhas e criação de *backups* após o processamento diário [Hariri et al. 2006].

A solução proposta estende a arquitetura FairCPU [Rego et al. 2011], pela adição do conceito de sensor e *loop* de controle [Kephart and Chess 2003]. Assim, o ambiente pode ser monitorado e a alocação dos recursos de CPU pode ser realizada para garantir a QoS das aplicações.

A FairCPU utiliza o conceito de Unidade de Processamento (UP) para alocar os recursos de CPU para as MVs de maneira a garantir um desempenho homogêneo, independente da máquina física subjacente e do modelo de seu processador. Desta maneira, a arquitetura permite aos usuários configurar o poder de processamento das MVs utilizando uma métrica uniforme.

A UP é a abstração utilizada para representar o poder de processamento, similar a ECU[1] (*Elastic Computing Unit*) da Amazon EC2. Ela pode ter um valor conhecido em termos de GFLOPS (bilhões de operações de ponto flutuante por segundo), ou outra métrica (como MIPS, MHz, etc) que possa ser utilizada para representar o poder de processamento de cada máquina física ou virtual.

A relação entre a solução proposta e a arquitetura FairCPU pode ser visualizada na Figura 2. A abordagem está dividida em duas partes: Agente e Controlador Autonômico. O Agente Autonômico é o componente presente em cada MV, que é responsável por coletar dados e monitorar as aplicações. O Controlador Autonômico gerencia e analisa a informação coletada pelo Agente em tempo real. Esta informação é armazenada e utilizada para aumentar ou reduzir dinamicamente os recursos alocados, afim de garantir o SLA.

O Controlador Autonômico utiliza a interface disponível pelo componente Cliente da FairCPU, no qual está disponível a função de ajustar dinamicamente a quantidade de recursos alocados para a MV. Um importante componente da arquitetura FairCPU é o Gerenciador de Limites, que é o módulo responsável por configurar a quantidade de recursos de CPU que podem ser utilizados pelas MVs. A implementação atual do Gerenciador de Limites pode configurar o uso de CPU para os hipervisores Xen e KVM. Mais informação sobre os módulos da arquitetura FairCPU podem ser encontradas em

[1]Uma ECU fornece a capacidade de CPU equivalente à de um processador 2007 Opteron ou 2007 Xeon de 1,0-1,2 GHz [Amazon 2013].

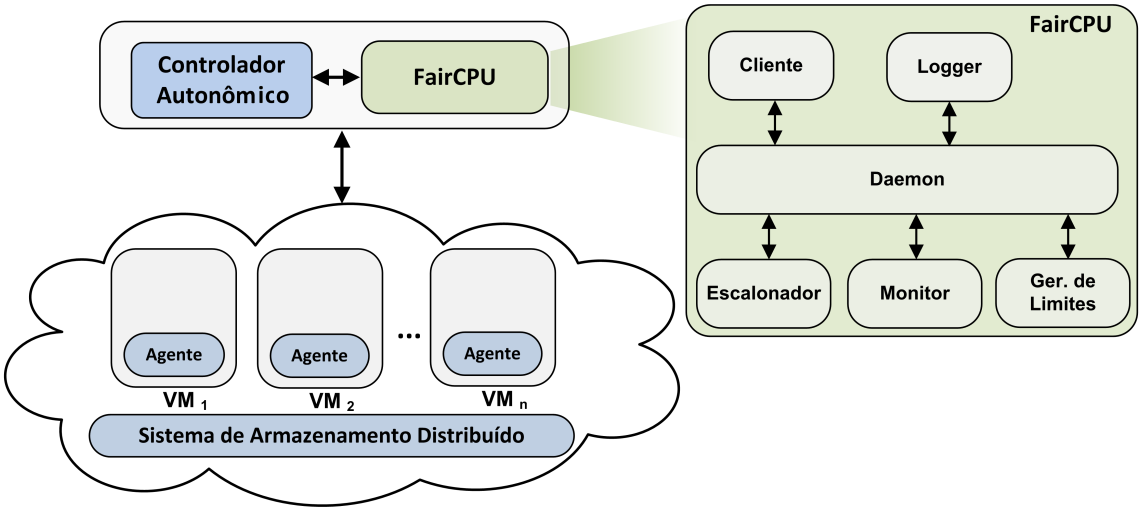

Figura 2. Visão geral da solução proposta.

[Rego et al. 2011].

O Agente e o Controlador Autonômico foram desenvolvidos com a linguagem de programação Ruby, assim como a arquitetura FairCPU e seu módulo Cliente. Em um trabalho prévio, esta solução foi utilizada para alocar dinamicamente recursos de CPU às MVs considerando apenas a utilização atual de CPU e o tempo de resposta das aplicações [Rego et al. 2012].

Neste ponto, a solução permite ajustar a memória da MV em tempo de execução e criar novas instâncias de MVs. Além disso, um arquivo XML com informações de SLA pode ser utilizado para configurar o Controlador para tomada de decisões. Um exemplo do arquivo de SLA pode ser visualizado na Figura 3.

Como exemplo, a segunda linha do arquivo XML significa: a métrica tempo de resposta (*response time*) deve ser monitorada e caso seu valor seja maior do que 2 segundos, a operação da FairCPU **setpu+** deve ser chamada para aumentar a quantidade de Unidades de Processamento em 1.

Com as novas funcionalidades implementadas, diferentes estratégias de alocação dinâmica de recursos podem ser desenvolvidas. Primeiro, é possível aumentar/reduzir a quantidade de CPU e memória das máquinas virtuais para lidar com diferentes cargas de trabalho. Neste caso, a quantidade total de CPU e memória da máquina física subjacente é o limite superior. Segundo, é possível criar novas instâncias de MVs e utilizar um

```
<sla>
    <metric name="responsetime" if="greater" threshold="2.0" op="setpu+">1</metric>
    <metric name="responsetime" if="less" threshold="0.2" op="setpu-">1</metric>
    <metric name="memory" if="greater" threshold="90%" op="setmem+">10%</metric>
    <metric name="memory" if="less" threshold="50%" op="setmem-">10%</metric>
    <metric name="cpu" if="greater" threshold="90%" op="setpu+">1</metric>
    <metric name="cpu" if="less" threshold="45%" op="setpu-">1</metric>
    <metric name="responsetime" if="greater" threshold="2.0" op="newvm" template="2">1</metric>
    <metric name="responsetime" if="less" threshold="0.2" op="removevm" template="2">1</metric>
</sla>
```

Figura 3. Exemplo do arquivo de configuração XML.

balanceador de carga para dividir a carga de trabalho entre elas. Esta estratégia é mais adequada para aplicações que utilizam um paradigma cliente/servidor, mas também é bastante interessante para aplicações distribuídas de uso intensivo de CPU, como criar um novo nó em um cluster virtual.

A segunda estratégia pode ser utilizada para criar novas instâncias de MVs tanto com recursos locais (nuvem privada), quanto com recursos de provedores pagos de Computação em Nuvem (nuvem pública). A abordagem mais comum é utilizar recursos locais primeiramente. Entretanto, uma vez que todos os recursos locais estejam sendo utilizados ou dependendo da funcionalidade requisitada, é possível instanciar MVs em nuvens públicas, como a Amazon EC2. Neste ponto, a solução proposta pode utilizar qualquer uma dessas estratégias.

3.1. Políticas de Alocação

De acordo com [White et al. 2004], uma política de alocação especifica quais medidas devem ser tomadas sempre que o sistema alcança determinado estado. Ela pode ser na forma ”SE (condição) ENTÃO (ação)”, onde uma condição caracteriza um estado ou um conjunto de estados. Um exemplo de política de ação é ”SE (tempo de resposta médio do servidor *web* for maior que 500 ms) ENTÃO (adicionar um novo servidor *web*)”. O problema com políticas de ação é que quando uma série de políticas de ação é especificada, isto dificulta a identificação de conflitos que podem surgir entre as políticas.

Por exemplo, em um sistema multi-camadas, se um servidor *web* e uma aplicação requerem recursos adicionais e os recursos disponíveis são insuficientes para ambos, um conflito surge. Para lidar com tais conflitos, o sistema necessita de políticas de resolução de conflitos, como dar uma prioridade maior para o servidor. Entretanto, como conflitos geralmente surgem em tempo de execução, é difícil especificar políticas de resolução.

Decidir qual é a melhor política de resolução de conflitos não está no escopo deste trabalho. O objetivo deste trabalho é apresentar uma arquitetura e estratégias de alocação dinâmicas. Considerando um ambiente híbrido, existem três possíveis estratégias: (1) elasticidade vertical, pelo aumento/redução dos recursos alocados; (2) elasticidade horizontal, pela criação de MVs em uma nuvem privada; e (3) elasticidade horizontal, com a criação de MVs em um ambiente de nuvem pública.

4. Metodologia de Avaliação

A metodologia para avaliação da solução utilizada neste trabalho segue [Jain 1991]. A Tabela 2 detalha as características relacionadas ao planejamento dos experimentos, assim como métricas e cargas de trabalho.

Uma nuvem privada heterogênea criada com o gerenciador de nuvem OpenNebula[2] versão 3.8 foi utilizada como *testbed*, consistindo de 1 máquina do tipo Ci5, 2 máquinas do tipo Ci7 e 1 máquina do tipo X4, todas conectadas por uma rede Gigabit Ethernet. As configurações das máquinas físicas são apresentadas na Tabela 3.

Todas as máquinas físicas utilizaram o sistema operacional Ubuntu Server 12.04 64 bits e o hipervisor KVM. Para todos os experimentos, as MVs com os geradores de carga foram executados na máquina do tipo Ci5 com 4 UPs, a MV com o balanceador de

[2]http://opennebula.org

Tabela 2. Projeto de experimentos

Características	Descrição
Sistema	Ambiente privado de Computação em Nuvem (Open Nebula) e provedor de nuvem pública (Amazon EC2)
Métricas	Tempo de resposta, tempo do experimento, requisições perdidas
Parâmetros	CPU, memória, MVs, instâncias da Amazon do tipo (*xsmall*)
Fatores	UP, configuração do httperf (quantidade de requisições, taxa de requisições, tempo)
Técnica de Avaliação	Medição
Carga de Trabalho	Carga de trabalho do httperf
Projeto de Experimentos	O experimento consiste da geração da carga de trabalho com o httperf no servidor *web* Apache para que dinamicamente se adéque à quantidade de CPU ou às MVs. Este experimento será executado em uma nuvem privada e em um provedor público de nuvem (solução híbrida)
Análise dos Dados	Interpretação dos resultados descritos nos gráficos
Apresentação dos Resultados	Gráfico de barras

Tabela 3. Configuração das máquinas físicas.

Tipo de Máquina	Processador	Memória
X4	Intel Xeon X3430 @ 2.4 GHz	8 GB
Ci5	Intel Core i5-750 @ 2.66 GHz	4 GB
Ci7	Intel Core i7-930 @ 2.8 GHz	24 GB

carga foi executada na máquina do tipo X4 com 4 UPs, e as MVs com a aplicação PHP e o servidor *web* Apache foram executadas nas máquinas do tipo Ci7 com um número variável de UPs. Todas as MVs possuíam 2 VCPUs, 1 GB de memória e o sistema operacional Ubuntu Server 12.04 64 bits.

Para simplificar os experimentos, a UP foi definida como 50% de um núcleo do processador da máquina física do tipo Ci7. Assim, 1 UP é igual a 50% da capacidade da CPU, 2 UPs equivalem a 100% da capacidade[3] da CPU, e assim por diante.

5. Experimentos

Os experimentos foram projetados para serem executados em dois ambientes: nuvem privada construída com o OpenNebula e nuvem pública da Amazon. Para ambos os ambientes, as mesmas ferramentas foram utilizadas. O servidor *web* Apache foi instalado e configurado em uma MV e uma pequena aplicação *web* em PHP foi desenvolvida para gerar o consumo de recursos. A ferramenta httperf[4] foi utilizada como gerador de cargas de trabalho sobre essa aplicação. O balanceador de carga Nginx[5] e o Agente Autonômico

[3]Considerando que um processador com 4 núcleos tem capacidade de 400%.
[4]http://www.hpl.hp.com/research/linux/httperf
[5]http://nginx.org

foram instalados em outra MV.

O tempo de resposta da aplicação para cada requisição foi coletado e armazenado para tomada de decisão posterior. Um *script*, escrito em Ruby, lê o arquivo de *log* do Nginx a cada 5 segundos para coletar detalhes sobre as requisições.

O gráfico na Figura 4 descreve o comportamento da carga de trabalho utilizada pelo httperf. Pode-se verificar que cada taxa foi aplicada por 2 minutos e inicia com 2 requisições por segundo, incrementando em 1 até 6 requisições por segundo, quando inicia a decair até completar 14 minutos de experimento. A mesma carga de trabalho foi aplicada aos três experimentos, num total de 3480 requisições por experimento.

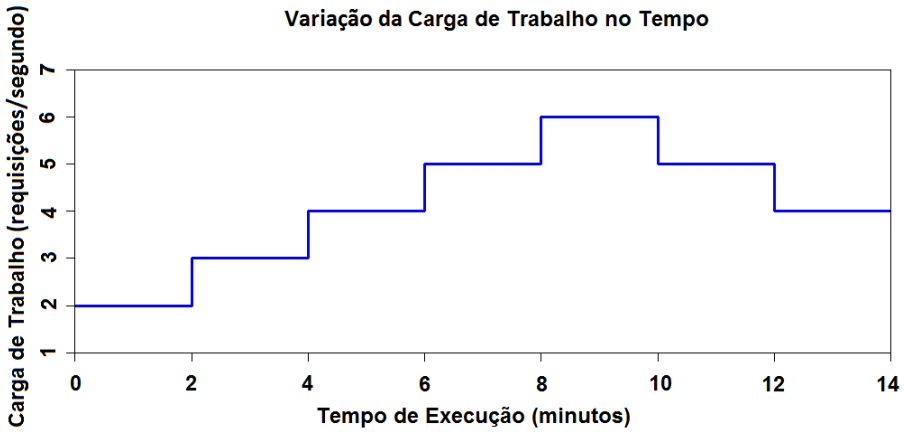

Figura 4. Variação da carga de trabalho do httperf com o tempo.

5.1. Experimento 1 - Nuvem Privada e Elasticidade Vertical

Neste experimento o tempo de resposta foi utilizado para a tomada de decisão. Foi estabelecido como limite superior o tempo de resposta de 2 segundos, como pode ser visto no XML da Figura 5. Inicialmente a quantidade de UPs é igual a 1 e quando o limite for excedido, o número de UPs deve ser aumentado em 1.

```
<sla>
    <metric name="responsetime" if="greater" threshold="2.0" op="setpu+">1</metric>
    <metric name="responsetime" if="less" threshold="0.2" op="setpu-">1</metric>
</sla>
```

Figura 5. Arquivo de configuração XML para o primeiro experimento.

A Figura 6 apresenta o tempo de resposta da aplicação sobre o tempo. Este tempo de resposta aumenta entre os minutos 6 e 8 pois a taxa do httperf atinge 5 requisições/segundo. A estratégia selecionada incrementa rapidamente a quantidade de UPs e o tempo de resposta estabiliza, retornando ao valor esperado. Para este experimento, 13 requisições foram perdidas até o sistema se adaptar.

5.2. Experimento 2 - Nuvem Privada com Elasticidade Horizontal

Neste experimento, não deve haver variação na quantidade de CPU alocada para as MVs. Quando o limite for excedido, uma nova MV é instanciada e o balanceador de carga

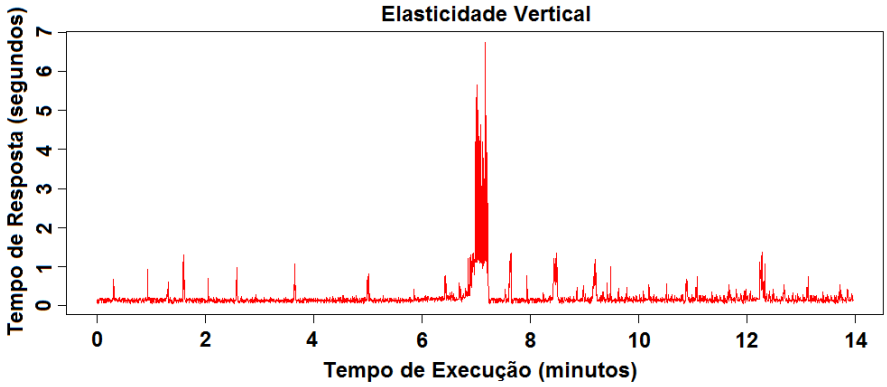

Figura 6. Tempo de resposta para o experimento com elasticidade vertical.

inclui o IP da nova MV na sua lista de redirecionamento. Assim, a nova MV pode receber requisições.

A Figura 7 apresenta o arquivo XML de configuração para este experimento. A MV que será criada deve ser do tipo 2 (número que identifica o modelo da MV que deve ser criada pelo OpenNebula). Este modelo de MV foi criado previamente.

```
<sla>
    <metric name="responsetime" if="greater" threshold="2.0" op="newvm" template="2">1</metric>
    <metric name="responsetime" if="less" threshold="0.2" op="removevm" template="2">1</metric>
</sla>
```

Figura 7. Arquivo de configuração XML para o segundo experimento.

A Figura 8 apresenta o tempo de resposta da aplicação com o tempo. Diferentemente do experimento anterior, o tempo de resposta foi maior que o ideal, devido ao fato do tempo de criação da MV ser maior que tempo necessário para incrementar a quantidade de UPs. Neste experimento, 218 requisições foram perdidas.

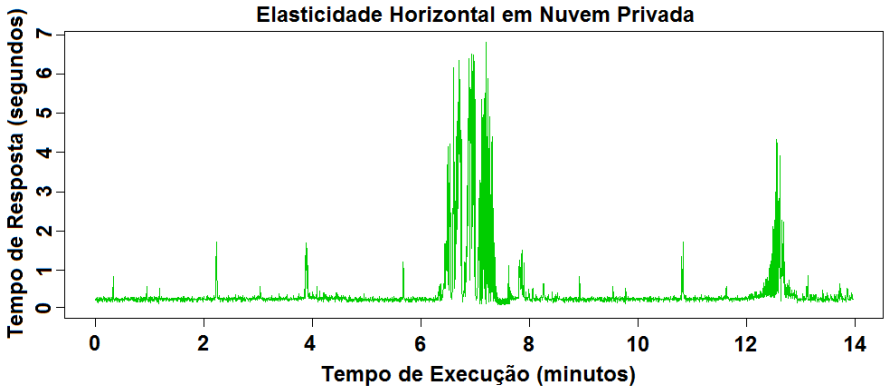

Figura 8. Tempo de resposta para o experimento com elasticidade horizontal em recursos locais.

5.3. Experimento 3 - Nuvem Híbrida e Elasticidade Horizontal

Para este experimento, uma nuvem privada com o OpenNebula também foi utilizada com uma MV com o httperf, outra MV para o Nginx e outra para a aplicação PHP. Quando o

limite para o tempo de resposta é ultrapassado, uma nova MV deve ser inicializada em uma nuvem pública e o balanceador de carga deve incluir seu endereço IP na lista de redirecionamento, para que ela possa receber requisições.

O provedor público de nuvem escolhido foi a Amazon EC2. A instância utilizada foi do tipo *xsmall*, com 1 VCPU (1 ECU) e 1.7 GB de memória. O arquivo XML de configuração (Figura 9) indica que a MV deve ser criada. O modelo da MV a ser criada tem o identificador 7 e foi configurado previamente no OpenNebula, com a identificação da AMI (*Amazon Machine Image*), o tipo de instância, região e zona da Amazon.

```
<sla>
    <metric name="responsetime" if="greater" threshold="2.0" op="newvm" template="7">1</metric>
    <metric name="responsetime" if="less" threshold="0.2" op="removevm" template="7">1</metric>
</sla>
```

Figura 9. Arquivo de configuração XML para o terceiro experimento.

A Figura 10 apresenta o tempo de resposta sobre o tempo. Para este experimento, 671 requisições foram perdidas devido ao fato da criação da MV na Amazon demorar mais de 2 minutos.

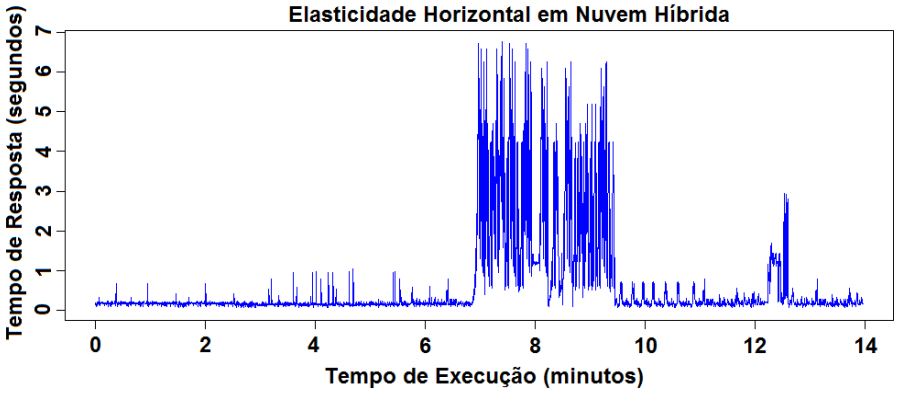

Figura 10. Tempo de resposta para o experimento com elasticidade horizontal em recursos públicos.

5.4. Discussão dos Resultados

Comparando as Figuras 6, 8 e 10, é possível verificar um comportamento similar na variação do tempo de resposta. Os três experimentos iniciaram com um baixo tempo de resposta (menor que 1 segundo) e aproximadamente na metade do experimento (entre o sexto e o oitavo minuto) há um grande pico no tempo de resposta. Este aumento no tempo de resposta se deve ao acréscimo da taxa de requisições por segundo, que atingiu uma carga de trabalho que não era suportada pelo servidor. Neste momento, dependendo da estratégia do experimento, uma ação era tomada no ambiente.

O Experimento 1 obteve o melhor resultado, pois a alteração dinâmica da quantidade de CPU da máquina virtual é feita de forma rápida. Os experimentos 2 e 3 obtiveram resultados piores, pois uma nova MV precisou ser criada em ambos. O Experimento 2 obteve um resultado melhor do que o Experimento 3, pois a MV é criada mais rápido na infraestrutura local.

A quantidade de pacotes perdidos no Experimento 3 é bem maior, uma vez que, com o aumento da carga de trabalho, o servidor ficou sobrecarregado e os pacotes foram considerados perdidos quando o tempo de resposta estava acima de 7 segundos. Como a criação da MV extra foi mais demorada na nuvem pública, o servidor ficou sobrecarregado por mais tempo, e, por isso, mais pacotes foram perdidos. Pode-se notar também que, depois do minuto 12, há um pequeno pico no tempo de resposta, mais notório nos experimentos 2 e 3. Esses picos são resultados da remoção da MV extra e reconfiguração do balanceador de carga. Como a taxa de requisições caiu, o tempo de resposta também caiu e uma ação foi disparada. Uma vez que tempo de resposta não ultrapassou 7 segundos, não houve perda de pacotes nesse momento.

6. Conclusão

Este trabalho apresenta estratégias para alocação de recursos em ambientes de Computação em Nuvem baseados na alocação dinâmica de CPU e a criação de MVs (elasticidade vertical e horizontal). Uma solução baseada na arquitetura FairCPU foi utilizada para testar diferentes formas de garantir a qualidade do serviço de aplicações web.

Este trabalho explorou diferentes ambientes, em especial uma nuvem híbrida implementada com o gerenciador de nuvem OpenNebula, para que fosse possível criar MVs numa infraestrutura local e na nuvem da Amazon. A utilização de uma solução que automaticamente (1) incrementa e decrementa recursos de uma MV e (2) cria MVs, seja na nuvem privada ou na nuvem pública, foi uma atividade muito importante. Esta atividade nos permitiu explorar a arquitetura FairCPU e suas funcionalidades, bem como vislumbrar diferentes formas de adaptar os recursos de uma aplicação às mudanças de cargas de trabalho.

Um trabalho futuro vislumbrado seria misturar a elasticidade vertical e horizontal para criar uma estratégia híbrida. Uma ideia interessante é utilizar escalabilidade vertical durante a criação das MVs (seja na nuvem privada, seja na nuvem pública) e assim evitar violações do SLA durante esse intervalo de tempo. Tão logo a MV seja criada, os recursos adicionados podem ser removidos. Além disso, utilizar uma técnica preditiva para a tomada de decisões pode ajudar a diminuir a quantidade de pacotes perdidos, uma vez que a tomada de decisão seria feita antes do serviço sobrecarregar. Outros possíveis trabalhos futuros estão relacionados à utilização de outros provedores de nuvens públicas, como a HP Cloud Services, além de realizar experimentos mais complexos, com a utilização de outras métricas para análise de desempenho, e a formulação de um modelo de custo para avaliar a solução híbrida.

Referências

Amazon (2013). Amazon web services. http://aws.amazon.com/pt/. Online; acessado em março-2013.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50.

Emeakaroha, V., Brandic, I., Maurer, M., and Breskovic, I. (2011). Sla-aware application deployment and resource allocation in clouds. In *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*, pages 298 –303.

Gong, Z., Gu, X., and Wilkes, J. (2010). Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 9 –16.

Hariri, S., Khargharia, B., Chen, H., Yang, J., Zhang, Y., Parashar, M., and Liu, H. (2006). The autonomic computing paradigm. *Cluster Computing*, 9(1):5–17.

Heo, J., Zhu, X., Padala, P., and Wang, Z. (2009). Memory overbooking and dynamic control of xen virtual machines in consolidated environments. In *Integrated Network Management, 2009. IM '09. IFIP/IEEE International Symposium on*, pages 630 –637.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons, INC, 1st edition.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

Oliveira, C. (2012). An architecture for experimentation with multi-tier internet applications in virtualized server environments. In *II Workshop de Sistemas Distribuídos Autonômicos*.

Pokluda, A., Keller, G., and Lutfiyya, H. (2010). Managing dynamic memory allocations in a cloud through golondrina. In *Systems and Virtualization Management (SVM), 2010 4th International DMTF Academic Alliance Workshop on*, pages 7 –14.

Rego, P. A. L., Coutinho, E. F., de Carvalho Sousa, F. R., and de Souza, J. N. (2012). Alocação autonômica de recursos para máquinas virtuais baseada em características de processamento. In *II Workshop de Sistemas Distribuídos Autonômicos*.

Rego, P. A. L., Coutinho, E. F., Gomes, D. G., and de Souza, J. N. (2011). Architecture for allocation of virtual machines using processing features. In *1st International Workshop on Cloud Computing and Scientific Applications (CCSA)*.

White, S., Hanson, J., Whalley, I., Chess, D., and Kephart, J. (2004). An architectural approach to autonomic computing. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 2 – 9.

Wu, L., Garg, S., and Buyya, R. (2011). Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 195 –204.

Zhang, Y., Huang, G., Liu, X., and Mei, H. (2010). Integrating resource consumption and allocation for infrastructure resources on-demand. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 75 –82.