Implementando Elasticidade no Nível do Sistema Operacional

Valquíria Prestes Belusso[1], **Guilherme Galante**[1]

[1]Ciência da Computação
Universidade Estadual do Oeste do Paraná (UNIOESTE)
Caixa Postal 711 – 85.819-110 – Cascavel-PR

val.prestes2000@gmail.com, guilherme.galante@unioeste.br

Abstract. *Currently two elasticity approaches are most commonly employed. In the first approach, an elasticity controller uses monitoring data from the virtualized environment or application to make decisions about the scalability of available resources. In the second approach, an elasticity controller is integrated directly into the application's source code, allowing the application itself to perform actions to allocate and release resources. Both approaches have their limitations: External monitors are not always suitable for certain applications, and the programming-level approach requires restructuring or modifying the application's source code. In this context, we present an elasticity controller integrated into a GNU-Linux operating system. It enables dynamic and automatic allocation of processing and memory resources in a fast and effective manner. The solution is validated using synthetic benchmarks and three scientific applications.*

Resumo. *Atualmente, duas abordagens de elasticidade são amplamente utilizadas. Na primeira abordagem, um controlador de elasticidade faz uso de dados de monitoramento do ambiente virtualizado ou da própria aplicação para tomar decisões relacionadas à escalabilidade dos recursos disponíveis. Na segunda abordagem, o controlador de elasticidade é diretamente incorporado ao código-fonte da aplicação, permitindo que a aplicação em si execute as ações de alocação e desalocação de recursos. Ambas as estratégias apresentam suas limitações: os monitores externos nem sempre são adequados para determinados tipos de aplicações, e a abordagem em nível de programação exige modificações e reestruturações no código-fonte da aplicação. Nesse contexto, apresenta-se um controlador de elasticidade integrado a um sistema operacional GNU-Linux para fornecer alocação dinâmica e automatizada de recursos de processamento e memória de forma rápida e efetiva. A solução é validada por meio de benchmarks sintéticos e três aplicações científicas.*

1. Introdução

A elasticidade é uma das principais características da computação em nuvem e pode ser definida como a capacidade de um sistema de adicionar ou remover dinamicamente recursos computacionais utilizados por uma determinada aplicação ou usuário de acordo com a demanda [Herbst et al. 2013]. Na prática, a elasticidade é implementada em nuvens utilizando tecnologias de virtualização, que permitem que recursos virtualizados possam ser fornecidos rapidamente e de acordo com a carga de trabalho. Os recursos podem incluir desde CPUs virtuais (vCPUs), memória, e armazenamento, até máquinas virtuais completas.

A elasticidade oferecida pelos mecanismos atuais baseia-se em duas abordagens [Barnawi et al. 2020]. A primeira abordagem baseia-se no monitoramento das solicitações externas ou no uso de recursos (carga do processador, uso de memória, solicitações de E/S) que podem variar amplamente ao longo do tempo. Na segunda abordagem, o controle da elasticidade é feito em nível de programação, ou seja, o controlador de elasticidade é incorporado ao código-fonte da aplicação, permitindo que as ações de alocação e desalocação de recursos partam da própria aplicação. No entanto, ambas as abordagens possuem limitações. O uso de monitores externos não são apropriados para algumas classes de aplicações e nem sempre conseguem coletar as informações necessárias de forma efetiva e rápida, ocasionando a alocação ineficiente dos recursos. Por outro lado, a abordagem em nível de programação, necessita que a aplicação seja reestruturada para a inclusão das ações de elasticidade, exigindo conhecimento profundo da aplicação e o acesso ao seu código-fonte, que nem sempre é possível.

Sob este contexto, este trabalho apresenta um controlador de elasticidade integrado a um sistema operacional (SO) GNU-Linux para fornecer alocação dinâmica e automatizada de recursos de processamento e memória. Considerando que o SO tem controle sobre a execução das aplicações e gerencia os recursos do sistema computacional, pode-se coletar as informações necessárias para controlar a elasticidade de modo rápido e acurado, e gerenciar a alocação de recursos de modo elástico e transparente. A solução é validada utilizando *benchmarks* sintéticos e aplicações científicas.

O restante do trabalho é organizado da seguinte forma. A Seção 2 apresenta alguns trabalhos relacionados. Na Seção 3 apresenta-se o controlador de elasticidade desenvolvido. Na Seção 4 realiza-se a avaliação experimental. Por fim, a Seção 5 conclui este trabalho.

2. Trabalhos Relacionados

No contexto da exploração da elasticidade de modo geral, os trabalhos de [Barnawi et al. 2020] e [Saif et al. 2021] apresentam uma visão abrangente da área, revisando diversas soluções propostas. Restringindo ao escopo de sistemas operacionais, há alguns poucos trabalhos explorando a alocação elástica de recursos. Nestes, algum aspecto do sistema operacional é projetado para gerenciar dinamicamente os recursos do *hardware* e compartilhá-los entre as aplicações.

[Youseff et al. 2012] abordam os requisitos para a exploração de elasticidade em serviços de sistema operacional, de modo a atender às demandas em constante mudança em ambientes de computação *multicore* e em nuvem. Além disso, o trabalho apresenta a implementação de um sistema de arquivos elástico no SO fos. Os resultados experimentais mostram que a solução é capaz de alcançar desempenho comparável a um sistema com excesso de recursos, no entanto, consumindo menos recursos do sistema.

[Ababneh et al. 2018] apresentam o sistema operacional ElasticOS. O SO proposto permite a um processo ou *thread* expandir automaticamente seus recursos associados através de várias máquinas, aumentando ou reduzindo conforme a demanda, sem exigir que a aplicação seja reprojetada ou configurada com uma combinação complexa de ferramentas e estruturas adicionais. A implementação é feita usando como base um *kernel* Linux versão 2.6.38.8.

Mais recentemente, [Goodarzy et al. 2021] descrevem o SmartOS, um sistema operacional baseado em aprendizado por reforço para automaticamente ajustar a alocação de memória, CPU e E/S. O SmartOS utiliza o aprendizado por reforço para coletar continuamente informações do ambiente e fazer alterações nos parâmetros de recursos expostos pelo *kernel* do Linux para melhorar a experiência do usuário.

3. Controlador de Elasticidade

Nesta seção apresenta-se uma solução de elasticidade baseada nas informações do SO para a alocação dinâmica de CPU e memória. Considerando que o SO tem informações sobre todas as aplicações que estão sendo executadas e suas demandas por recursos, pode-se coletar as informações necessárias para controlar a elasticidade de modo rápido e acurado, e gerenciar a alocação elástica de recursos de modo transparente sem a necessidade de modificar o código-fonte das aplicações.

A ideia central consiste em coletar informações sobre o uso de CPU e memória da máquina virtual (VM), processar as demandas e solicitar a alocação dinâmica dos recursos para o hipervisor subjacente, como ilustrado na Figura 1. Para fins de teste de conceito, desenvolveu-se um controlador de elasticidade para máquinas virtuais Xen, o qual opera como um serviço no sistema operacional convidado, funcionando como um *daemon*. O hipervisor foi escolhido por permitir a alocação de CPUs (via *hotplug*) e memória (via *memory ballooning*), além de permitir a comunicação bidirecional, via Xenstore, entre a máquina virtual (DomU, na nomenclatura do Xen) e o domínio de controle do hipervisor (Dom0).

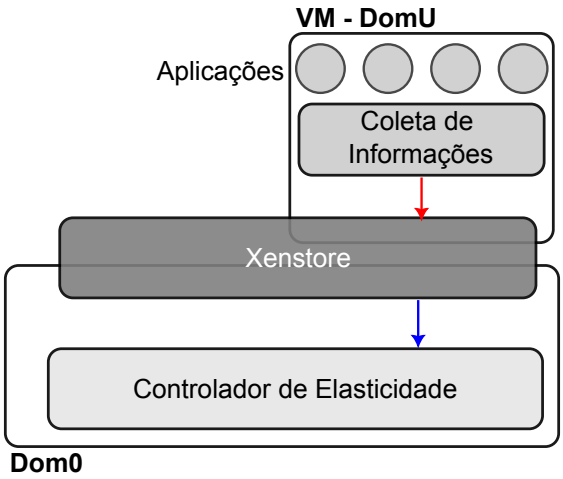

Figura 1. Controlador de Elasticidade.

O funcionamento do controlador de elasticidade é ilustrado no diagrama de sequência da Figura 2. O processo inicia-se com o Coletor, o qual solicita dados de uso de CPU ou memória para o SO, que responde com essas informações. O Coletor calcula as demandas e armazena no Xenstore os dados de alocação, que são lidos pelo Controlador externo à VM. Caso haja modificações no Xenstore, o Controlador realiza o envio de comandos ao Xen para a alocação de recursos adicionais. Este processo é realizado repetidamente, enquanto a VM estiver em execução. Os códigos-fonte estão disponíveis no GitHub[1]

[1] `https://github.com/ValquiriaBelusso/SO_Elastico_Xen`

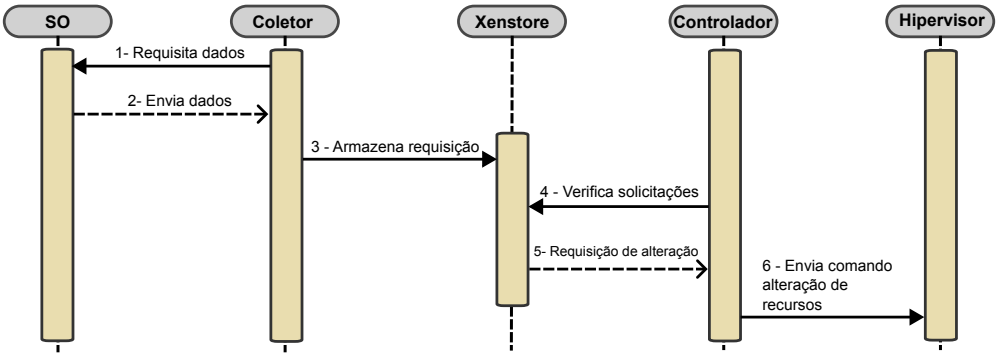

Figura 2. Diagrama de sequência do controlador de elasticidade.

3.1. Elasticidade de memória

A capacidade de modificar dinamicamente a memória de uma máquina virtual em tempo de execução representa um recurso importante para aplicações com requisitos dinâmicos de memória. Este recurso permite manter memória suficiente para alocar os dados da aplicação e, consequentemente, preservar o desempenho da aplicação (evitando o uso de *swapping* em memória secundária).

A coleta de dados de memória foi implementada utilizando um *script* Systemtap[2] para coletar as informações das chamadas das funções da libc.so.6. Quando o Systemtap detecta alguma chamada às funções de alocação de memória, realiza a requisição de memória, se necessário. Até o presente momento, não foi implementado a função de desalocação de memória, uma vez que não há forma de coletar a quantidade de memória liberada com a função *free* da biblioteca.

3.2. Elasticidade de CPU

Para controle de alocação de CPUs foi desenvolvido um *script* em Python que usa informações coletadas por meio do comando *mpstat*[3] para tomar decisões sobre alocar ou desalocar CPUs, dependendo de certos critérios de carga.

Com base nos dados coletados, é calculada uma média móvel do percentual livre das CPUs (últimas 5 coletas, com intervalo de 1 segundo entre elas). Se a média estiver abaixo do limite inferior estabelecido (80%) e houver CPUs disponíveis na máquina física, uma nova vCPU é alocada.Por outro lado, se a média de uso de CPU de uma determinada vCPU for baixa, uma vCPU é retirada da máquina virtual.

4. Experimentos e Resultados

Nesta seção apresentam-se os experimentos realizados para avaliar o controlador de elasticidade implementado. Na Seção 4.2 são apresentados os experimentos com *benchmarks* sintéticos e nas Seções 4.3 a 4.5 são apresentados experimentos tendo como base aplicações científicas executando na VM. As aplicações foram selecionadas por possuírem código-fonte disponível, e de modo que cada uma delas apresentasse um comportamento distinto em termos de uso de memória e CPU.

[2]`https://sourceware.org/systemtap/documentation.html`
[3]`https://man7.org/linux/man-pages/man1/mpstat.1.html`

4.1. Ambiente Computacional

Para a realização dos testes, foi utilizado um ambiente computacional com uma única máquina física equipada com um processador Intel Core i3-7100 contendo 2 núcleos físicos e 4 *threads*. O sistema operacional utilizado foi o GNU/Linux Ubuntu 22.04 LTS. A máquina possui 16 GB de memória RAM DDR4 e um SSD de 500 GB. Para o hipervisor Xen (Dom0), foi criada uma máquina virtual com 1 núcleo de processamento, 1 GB de memória RAM e o sistema operacional GNU/Linux Ubuntu 22.04 LTS.

4.2. Benchmarks Sintéticos

Para a validação individualizada do controlador de elasticidade, em termos de memória, desenvolveu-se uma aplicação sintética que realiza seis alocações de 500 MB, a cada 30 segundos, totalizando 3 GB. Os resultados podem ser visualizados na Figura 3. Inicialmente, a VM tem cerca de 700 MB de memória, e conforme a demanda da aplicação cresce, o controlador realiza o incremento da memória da VM. No gráfico da direita há duas curvas, sendo que a curva em azul representa o total de memória observado pelo hipervisor, enquanto que a linha vermelha indica a quantidade de memória observada internamente pela VM. Mesmo após pesquisa, não foi possível identificar o motivo da divergência.

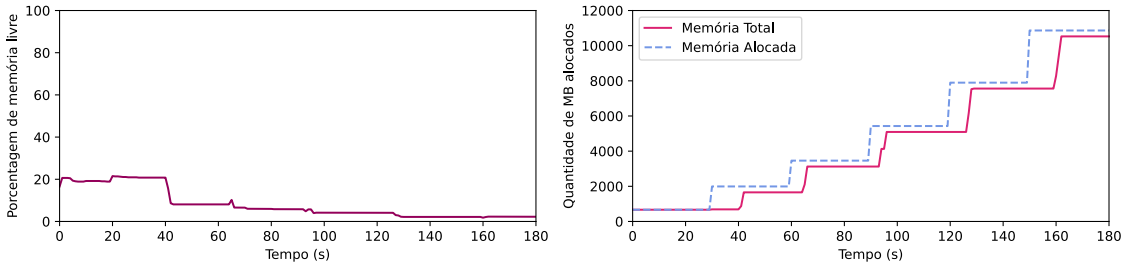

Figura 3. *Benchmark* sintético: Alocação dinâmica de Memória.

Para os testes individuais de CPU, utilizou-se uma aplicação escrita em C e paralelizada com OpenMP, que calcula a quantidade de números primos dentro de um intervalo estabelecido. O resultado obtido com a execução da aplicação com 3 *threads* podem ser visualizados na Figura 4.

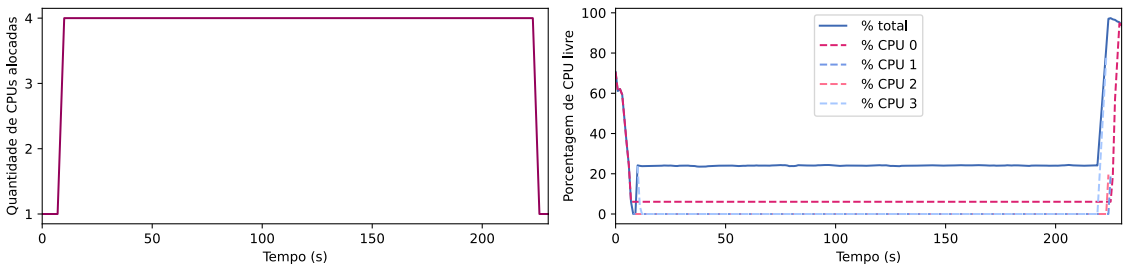

Figura 4. *Benchmark* sintético: Alocação dinâmica de CPU.

Com a execução da aplicação com 3 *threads*, e todas elas consumindo os recursos de computação por completo, 3 vCPUs são alocadas logo no início. Uma vez que os 3 núcleos estão ocupados, uma quarta vCPU é adicionada. Ao final da execução, 3 vCPUs são desalocadas, uma vez que não há demanda por processamento.

4.3. Aplicações Científicas: Transferência de Calor/Gradiente Conjugado

O problema de transferência de calor consiste em determinar a temperatura em qualquer ponto interno de uma placa em um dado instante de tempo. A solução implementada consiste na montagem de um sistema de equações e sua resolução (via Gradiente Conjugado) para cada passo de tempo de simulação. A aplicação é paralelizada com OpenMP e foi executada com 2 *threads*. O código fonte está disponível no GitHub[4]. Para essa aplicação e demais aplicações científicas, os controladores de memória e CPU são executados em conjunto. Os resultados do experimento são apresentados na Figura 5.

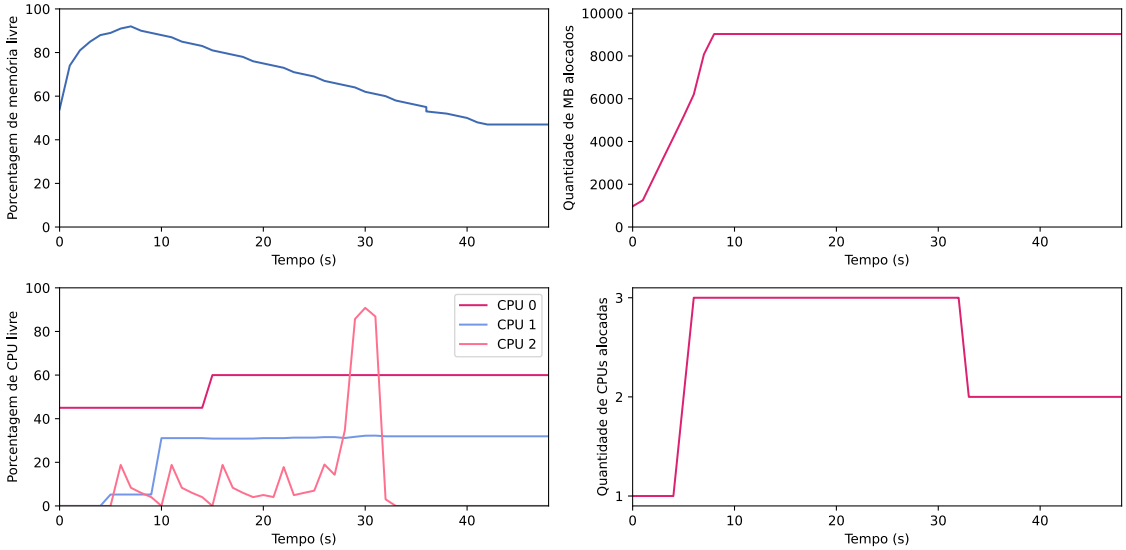

Figura 5. TC-CG: Alocação dinâmica de CPU e Memória.

Considerando as demandas de memória, o controlador conseguiu garantir a memória necessária para a execução da aplicação, alcançando cerca de 9 GB de memória alocada para a VM. Em temos de CPU, alocou-se 3 vCPUs no pico de demanda, e quando a necessidade de processamento foi reduzida, uma vCPU foi desalocada. A análise de CPU é mais complexa de ser realizada pois as *threads* são escalonadas em vCPUs diferentes ao longo da execução. Para uma análise mais precisa seria necessário implementar um mecanismo dinâmico de afinidade para as *threads* da aplicação e vCPUs.

4.4. Aplicações Científicas: LULESH

O *Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics* (LULESH) é um código de simulação desenvolvido para estudar e analisar o comportamento de materiais em condições extremas, como explosões e alto impacto [Karlin et al. 2013]. O código-fonte da aplicação está disponível no GitHub[5]. Os resultados do experimento executando o LULESH com 4 *threads* são apresentados na Figura 6. Os resultados mostram uma alocação inicial de cerca de 8 GB memória, necessária no início da execução da aplicação. No entanto, logo essa memória é desalocada, fazendo que a VM fique com memória excedente, uma vez que o controlador implementado não realiza a desalocação. No caso da alocação de vCPUs, mesmo considerando que a aplicação é executada com 4 *threads*,

[4] https://github.com/ValquiriaBelusso/SO_Elastico_Xen/
[5] https://github.com/LLNL/LULESH

apenas 3 vCPUs são alocadas dinamicamente. Neste caso, apenas as CPUs 0 e 1 são efetivamente usadas, e a vCPU 2 tem cerca de 50% de sua capacidade utilizada. Portanto, não houve a necessidade de alocar 4 vCPUs.

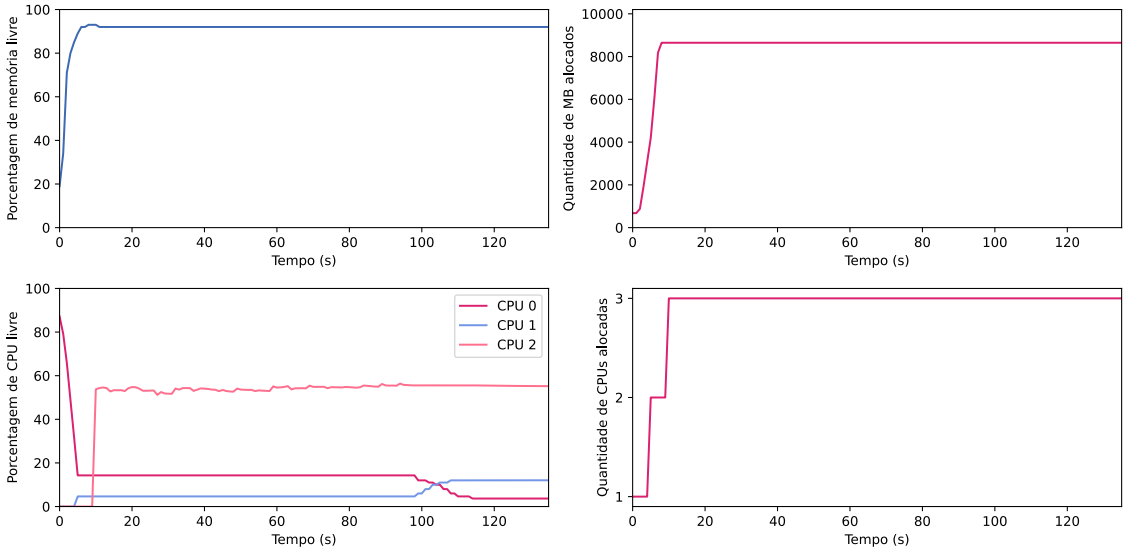

Figura 6. LULESH: Alocação dinâmica de vCPUs e Memória.

4.5. Aplicações Científicas: OmpSCR

O *benchmark* OmpSCR é composto por um conjunto de aplicações OpenMP escritas em C, C++ e Fortran. As aplicações deste *benchmark* utilizadas no experimento foram, decomposição LU, Mandelbrot, transformação rápida de Fourier, simulação dinâmica molecular e quicksort. As seis aplicações foram executadas na sequência, de modo que a demanda por recursos tivessem características diversas ao longo da execução. Todas as aplicações foram executadas com 4 *threads*. Os resultados são apresentados na Figura 7.

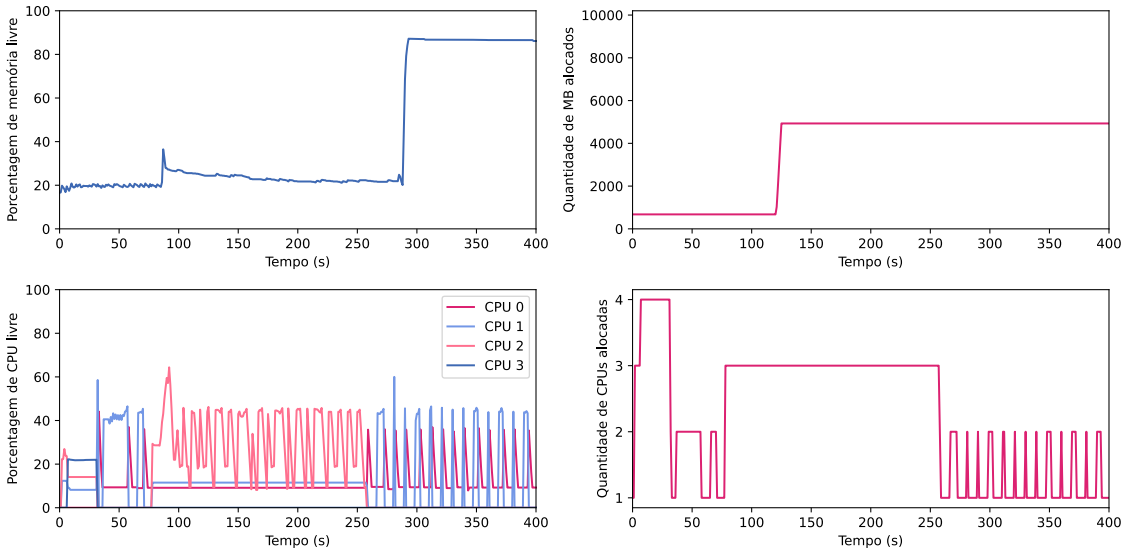

Figura 7. OmpSCR: Alocação dinâmica de vCPUs e Memória.

Em relação à memória, a quantidade inicial alocada para a VM foi suficente até aproximadamente 120 segundos de execução. Após, 4 GB foram alocados e quase que completamente utilizados até cerca dos 300 segundos de execução. No caso da alocação de CPUs, pode-se verificar que as demandas oscilaram bastante ao longo da execução, fazendo com que o número de vCPUs variasse consideravelmente. Aqui constata-se uma situação inconveniente após os 260 segundos de execução, onde há uma sequência de alocações e desalocações de vCPUs. Esse problema decorre da oscilação da carga de trabalho gerada pela aplicação e deve ser resolvida com o aprimoramento do algoritmo.

5. Conclusão

A principal contribuição do controlador de elasticidade é a sua capacidade de adaptar os recursos de acordo com mudanças na demanda de carga de trabalho. Essa adaptatividade é especialmente valiosa em ambientes dinâmicos, onde as cargas de trabalho podem variar de forma imprevisível. O controlador de elasticidade demonstrou ser eficaz na alocação dinâmica de recursos de acordo com a demanda de carga de trabalho na maioria dos casos.

Como trabalhos futuros pretende-se melhor os algoritmos de alocação de vCPUs e memória, de modo que monitorem e ajustem os recursos de forma mais precisa. O objetivo é evitar a alocação excessiva de vCPUs durante picos de carga e garantir a desalocação adequada quando as vCPUs não são mais necessárias. Além disso, é fundamental implementar estratégias avançadas de gerenciamento de memória, permitindo a alocação mais precisa e também a desalocação de memória não utilizada.

Referências

Ababneh, E., Al-Ali, Z., Ha, S., Han, R., and Keller, E. (2018). Elasticizing linux via joint disaggregation of memory and computation. *CoRR*, abs/1806.00885.

Barnawi, A., Sakr, S., Xiao, W., and Al-Barakati, A. (2020). The views, measurements and challenges of elasticity in the cloud: A review. *Computer Communications*, 154:111–117.

Goodarzy, S., Nazari, M., Han, R., Keller, E., and Rozner, E. (2021). Smartos: Towards automated learning and user-adaptive resource allocation in operating systems. In *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '21, page 48–55, New York, NY, USA. Association for Computing Machinery.

Herbst, N. R., Kounev, S., and Reussner, R. (2013). Elasticity in cloud computing: What it is, and what it is not. In *10th International Conference on Autonomic Computing (ICAC 13)*, pages 23–27, San Jose, CA. USENIX Association.

Karlin, I., Keasler, J., and Neely, R. (2013). Lulesh 2.0 updates and changes. Technical Report LLNL-TR-641973, Lawrence Livermore National Laboratory.

Saif, M. A. N., Niranjan, S. K., and Al-ariki, H. D. E. (2021). Efficient autonomic and elastic resource management techniques in cloud environment: Taxonomy and analysis. *Wirel. Netw.*, 27(4):2829–2866.

Youseff, L., Beckmann, N., Kasture, H., Gruenwald, C., Wentzlaff, D., and Agarwal, A. (2012). The case for elastic operating system services in fos. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, page 265–270, New York, NY, USA. Association for Computing Machinery.