Incluindo Previsão num Gerenciador Elástico de Memória*

Gustavo M. Moreira, André F. Gonçalves, Cristina Boeres, Vinod E. F. Rebello

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brazil

{gustavomuller,andrefg}@id.uff.br,{boeres,vinod}@ic.uff.br

Abstract. *Cloud computing offers users relatively cheap and easy access to resources and services with sufficient capacity to meet their response times and performance requirements. To be cost effective, cloud provider's have to optimize resource utilization to meet the demands of the largest number of clients using the fewest resources. This work aims to improve decisions made by MEC, a component of the MEMiC tool to manage memory elasticity of VMs. A new predictive performance model is used to adjust VM preemption and scheduling to reduce application execution times and increase resource utilization. Initial experimentation confirms the benefits of the proposed proactive approach.*

Resumo. *Nuvens computacionais oferecem recursos e serviços a seus usuários, que por sua vez desejam que suas requisições sejam atendidas rapidamente, com baixos custos de utilização. Do lado do provedor, é desejado maximizar o uso de recursos do ambiente para atender um maior número de clientes. Este trabalho incorpora a MEC, um controlador de recursos pertencente à ferramenta MEMiC, que realiza decisões de preempção de máquinas virtuais, um modelo preditivo para guiar tais decisões objetivando a minimização do tempo de serviço e maximizando o uso dos recursos de memória. Resultados experimentais mostram as vantagens de abordagem proativa proposta.*

1. Introdução

Um dos objetivos de Computação em Nuvem é o de facilitar o acesso a recursos computacionais através da *Internet*, sendo que o gerenciamento desses recursos não tem ação direta do usuário, mas dos provedores de serviços que cobram pelo seu uso de alguma forma. Para sua utilização, o usuário deve especificar a quantidade de recursos necessários para que a sua aplicação execute da maneira desejada. Se essa quantidade for sub ou superestimada, devido ao dinamismo de consumo de recursos das aplicações, tal execução pode ser afetada negativamente ou recursos podem ser desperdiçados. Dessa forma, mecanismos vêm sendo desenvolvidos para aprimorar o gerenciamento de recursos em nuvem, tendo como objetivo principal o atendimento das necessidades das aplicações submetidas pelo usuário dinamicamente, evitando o desperdício de recursos ou falta deles.

O foco desta pesquisa é de não somente maximizar a utilização do sistema, mas também minimizar o tempo de execução dos *jobs*, atendendo assim qualidade de serviço tanto do lado do provedor quanto do usuário. Note que, o provedor de serviços têm

*Esta trabalho foi apoiado pelo projeto REMATCH (Processo nº 88887.310261/2018-00) do Programa Institucional de Internacionalização (PrInt) da CAPES e os alunos G. Moreira e A. Gonçalves por bolsas de Iniciação Científica de CNPq/PIBIC 2019-20.

grande interesse no uso eficaz dos recursos, maximizando sua utilização em relação às demandas, e minimizando o desperdício. A base do trabalho é o *Memory Elasticity Management in Clouds* (MEMiC) [Valencia et al. 2018], que explora a elasticidade vertical da memória [Sawamura et al. 2016] (que consiste em aumentar ou diminuir a quantidade de memória disponível para uma VM, durante sua execução de acordo com sua demanda) e controla esquemas de preempção com o objetivo de fazer melhor uso dos recursos. Neste artigo, é proposto um modelo preditivo que correlaciona informações coletadas das execuções e serve para auxiliar políticas preemptivas do MEMiC. O propósito é minimizar o comportamento reativo de MEMiC em relação ao gerenciamento dos requisitos de memória, tornando-o um pouco proativo, considerando o impacto das decisões preemptivas em cada máquina virtual que executa um *job*. Neste artigo, serão introduzidos o MEMiC e seu funcionamento, as novas funcionalidades e a nova abordagem para o gerenciamento das VMs. Os experimentos são ferramentas para a análise da utilização de memória, do impacto da proposta, comparando com o MEMiC original.

2. O Gerenciador MEMiC

O MEMiC é um *framework* que expande as funcionalidades do *Memory Elastic Controller* (MEC), proposto em [Sawamura et al. 2016], combinando políticas de preempção (pausa, suspensão e migração) e explorando elasticidade vertical de memória, em um ambiente virtual que ”se espalham”em múltiplas máquinas físicas ou *hosts*. O MEMiC tem dois níveis de gerenciamento (Figura 1a), onde na base encontra-se o MEC, que controla dinamicamente a quantidade de memória alocada às máquinas virtuais (VMs, do inglês *virtual machines*) em um único *host*, manipulando os estados dessas VMs de acordo com informações coletadas. Seu principal propósito é melhorar o desempenho do sistema evitando o uso de *swap*. Em um nível superior, encontra-se o *MEMiC Manager* que recebe submissões de novos *jobs*, solicita o escalonamento de novas requisições nos *hosts* e identifica a necessidade de migração de VMs para atender suas demandas.

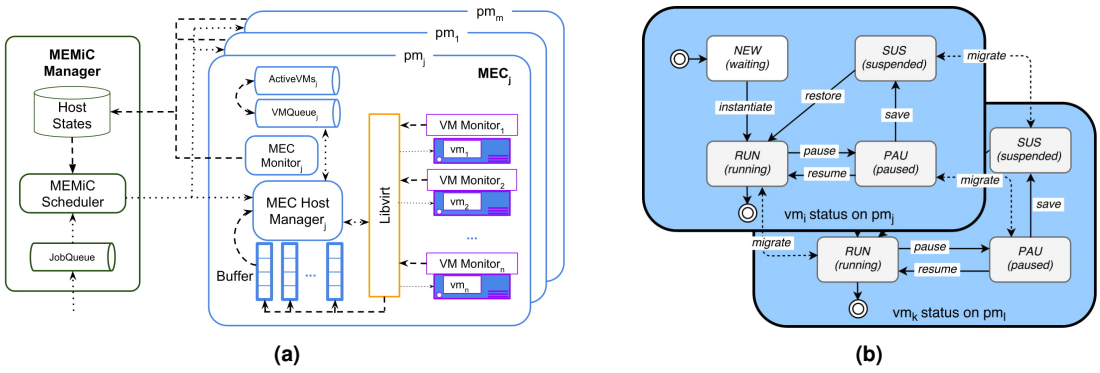


Figura 1. (a) Arquitetura do MEMiC e (b) Estados de VMs sob controle de MEC

A Figura 1b apresenta as transições de estados que compõem o ciclo de vida de uma VM [Valencia et al. 2018]. Uma VM instanciada com sucesso encontra-se no estado RUN. Durante sua execução, dependendo da disponibilidade de memória no *host* e dos requisitos do *job* na VM, este pode necessitar de mais memória. Porém, pode não mais haver memória disponível no *host* e *swap* ser acessado, e então páginas que foram colocadas em *swap*, se necessárias, serão trazidas de volta causando *uma sequência (quantidade)*

de eventos de swap out e swap in, denotado aqui como **siso**. Para evitar a perda de desempenho de execução e o uso indevido de recursos de memória devido a **siso**, mecanismos de preempção são efetivados, de acordo com decisões do MEC. Uma VM pode passar para o estado pausado (PAU), continuando alocada em memória, mas o acesso a área de *swap* é interrompido. Se as demandas de memória são ainda mais críticas, VMs pausadas podem ser suspensas (SUS) – uma cópia da imagem é gravada em disco, liberando toda memória antes alocada. Em relação à migração de VM, tanto o MEMiC manager quanto o MEC estão envolvidos, visto que o MEMiC atesta a necessidade da migração pareando os *hosts* e o MEC escolhe a VM a ser migrada ou nega a migração. Finalmente, as ações *resume* ou *restore* são efetivadas quando as demandas de memória são supridas, ou seja, se a VM estiver em PAU ou SUS, respectivamente, passa para RUN.

A elasticidade vertical de memória está baseada em informações individuais das VMs e na disponibilidade de memória no *host*. O MEC monitora cada VM e a partir das informações coletadas, tenta suprir suas necessidades previstas de forma dinâmica. MEC leva em consideração a taxa de consumo de memória do *job* entre os momentos atual t e anterior $t - \alpha$, sendo α, o período de tempo entre dois eventos de gerenciamento, e faz uma projeção de consumo para o momento futuro. Assim, a cada intervalo de gerenciamento, ajustes (retirada, incremento ou manutenção) são feitos na quantidade de memória alocada à VM, baseando-se (reativamente) na situação atual desta.

3. Um Mecanismo Proativo para Elasticidade Vertical de Memória

O MEC realiza o controle dos estados das VMs de forma reativa, pois está baseado em métricas coletadas até o momento atual. Ainda, as decisões são tomadas em relação a cada VM, somente com uma certa ponderação sobre o impacto no restante do sistema virtual. Tal abordagem faz com que o MEC frequentemente tome decisões de escalonamento e provisionamento de recursos que parecem benéficas no momento da análise, mas que podem trazer consequências negativas futuramente. Gostaríamos de reduzir a probabilidade de tomar decisões ruins de longo prazo. A ideia inicial desse estudo é prover um modelo preditivo que seja uma ferramenta a qual permita ao MEC tomar decisões considerando uma expectativa de término de execução de cada *job*. Ainda, modificações pontuais nos algoritmos de preempção foram efetivados para que o MEC passe a tomar decisões gradualmente, ou seja, mais cuidadosamente e não tão drasticamente.

Serão utilizadas neste trabalho instâncias com duas aplicações sintéticas *memory bound* (fazem acesso intenso à memória durante toda a execução), denotadas por $J1$ e $J2$. Cada aplicação aloca um vetor, cujo tamanho é usado para definir o pico máximo de memória alocada da aplicação, e acessa repetidamente os elementos desse vetor. A diferença entre $J1$ e $J2$ está na forma de acesso ao vetor: em $J1$ todo o vetor é acessado sequencialmente, e repetidamente, uma determinada quantidade de vezes, enquanto que em $J2$ o vetor é dividido em partes de igual tamanho, sendo cada parte acessada sequencialmente, também repetidamente. Para cada requisição de execução de uma aplicação Ji, o MEMiC cria uma instância que será executada em uma única VM (que por sua vez, executa unicamente Ji), sendo denotada aqui como $Ji.j$, onde j é o número da instância.

3.1. Modelo preditivo baseado em *swap in* – MPSi

Para a elaboração de suposições futuras úteis sobre as VMs, é possível utilizar modelos de previsão que tentam correlacionar as informações coletadas sobre as VMs. O uso da área

de *swap*, e o acontecimento de *swap in* e *swap out* são os principais eventos que conduzem as decisões do MEC, já que ocorrências relacionadas à *swap* são as que podem mais afetar negativamente o desempenho de execução das aplicações. Na verdade, o acesso à área de *swap* e *swap out* (que pode ser conduzido em paralelo com a execução de uma aplicação) não são fontes de atraso por si só, mas a ocorrência de *swap in* leva a um impacto negativo. Sendo assim, o modelo construído propõe-se estabelecer matematicamente uma relação de causalidade entre *swap in* e o atraso produzido na execução.

Na especificação do modelo preditivo, as aplicações sintéticas foram consideradas. Dados de modelagem e teste foram gerados considerando situações em que suspensão e migração não ocorreram, devido a demanda das aplicações e a configuração do ambiente. O experimento consiste na execução de duas instâncias do $J1$, denotadas como $J1.1$ e $J1.2$ (duas VMs distintas) que necessitam de 4,5 GiB de memória cada, sendo que $J1.2$ é submetido 120 segundos após $J1.1$. Desta forma, existe tempo hábil para que a $J1.1$ receba toda a memória necessária, alocada sobre demanda de acordo com a política de MEC. Para cada execução dessas duas instâncias, é definida uma máquina *host* com M de memória disponível, onde M varia de 5,5 até 8,5 GiB em incrementos de 0,5 GiB. Para cada configuração com M de memória, cinco execuções foram repetidas.

A Figura 2 mostra uma execução das duas instâncias e seus estados, considerando o *host* com $M = 7,5$ GiB de memória: $J1.1$ fica em RUN durante toda a sua execução (o ideal) e não há consumo de *swap*. Já $J1.2$ começa a consumir *swap* visto que não há memória suficiente disponível para ele. Mas quando $J1.2$ começa fazer **siso**, que prejudica seu desempenho, o MEC decide pausá-lo. Somente quando $J1.1$ termina e é disponibilizada sua memória, $J1.2$ volta para RUN com a memória necessária.

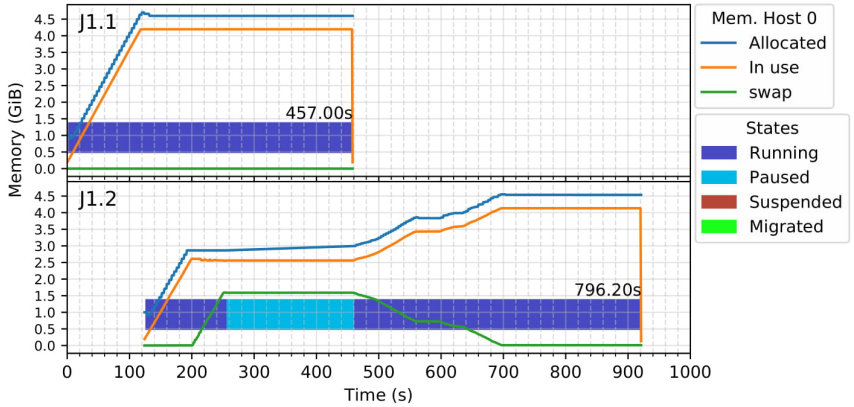


Figura 2. Estados de duas VMs com instâncias $J1.1$ e $J1.2$ em uma máquina com 7,5 GiB de memória sob gerenciamento do MEC

Os dados coletados para a modelagem e teste estão disponíveis nos *logs* produzidos por MEC. Desses dados, foram obtidos aqueles relativos à quantidade de *swap in* realizados durante a execução de $J1.2$. Pode-se derivar o atraso $\delta_{si}(Ji.j)$ na execução de $Ji.j$ devido à quantidade de *swap in* $si(Ji.j)$ sofrida durante sua execução, através de:

$$\delta_{si}(J1.2) = te_{\text{total}}(J1.2) - te_{\text{preemp}}(J1.2) - te_{\text{ideal}}(J1.2) \quad (1)$$

onde $te_{\text{total}}(Ji.j)$ e $te_{\text{ideal}}(Ji.j)$ são o tempo atual e ideal de execução da instância, respectivamente, e $te_{\text{preemp}}(Ji.j)$ é tempo em que a VM ficou em estados diferentes de RUN.

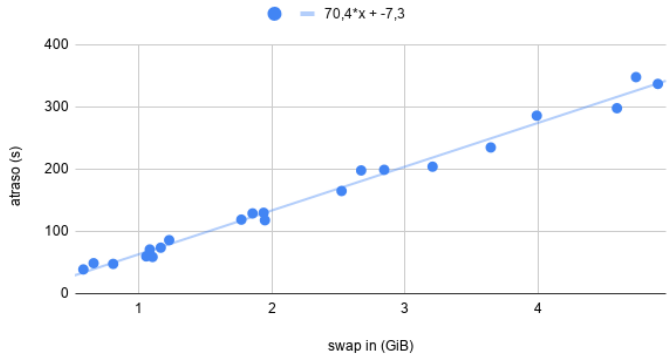


Figura 3. Modelo preditivo que correlaciona uso de *swap in* (em GiB) e atraso de execução (em segundos)

Analisando a relação entre $si(Ji.j)$ e $\delta_{si}(Ji.j)$, pode-se derivar um modelo a partir de uma regressão linear simples, como visualizado na Figura 3, sendo expresso na Equação (2).

$$\delta_{si}(Ji.j) = 70,4 \times si(Ji.j) - 7,3 \tag{2}$$

O que efetivamente está sendo estimado é o custo de recuperar dados do *swap* de volta para memória do *host* em uso.

Para criar o modelo, dividiu-se a massa de dados entre *modelagem* e *teste*. Para *modelagem* utilizou-se, aleatoriamente, três execuções de cada grupo de cinco execuções, sendo o restante utilizado para *teste*. Duas métricas foram estabelecidas para a avaliação do modelo: erro percentual absoluto médio (*mean absolute percentage error*) ou $MAPE$ e a média das diferenças absolutas (MDA) [Kotz et al. 2004]. Considerando todas as execuções de *modelagem* e *teste*, tem-se:

	m	$MAPE$	MDA
modelagem	21	$6,72\%$	8,33s
teste	14	$7,45\%$	8,48s

Como os valores de erro, tanto percentual, quanto absoluto são estatisticamente pequenos, considerando o tempo de execução ideal, pode-se concluir que a equação (2) consegue modelar com precisão a correlação existente entre *swap in* e atraso. Isso é verdade mesmo para uma quantidade reduzida de dados, devido a linearidade da relação.

Ao aplicar o modelo na previsão do tempo restante de execução de uma instância $Ji.j$ no momento t, supondo que exista memória suficiente, denotado por $tre(Ji.j, t)$, deseja-se assim prever a quantidade de "trabalho" restante de $Ji.j$, ou seja, em quanto tempo a aplicação terminaria supondo que ela, a partir do momento do cálculo t, executasse em cenário ideal em relação à disponibilidade de memória. O cálculo é refeito à cada intervalo t de gerenciamento do MEC da seguinte forma:

$$tre(Ji.j, t) = te_{\text{ideal}}(Ji.j) - te_{\text{real}}(Ji.j, t) - te_{\text{preemp}}(Ji.j, t) - \delta_{si}(Ji.j, t) \tag{3}$$

onde: $te_{\text{ideal}}(Ji.j)$ é tempo total de execução ideal da aplicação quando há memória suficiente; te_{real} é o tempo de execução corrente de $Ji.j$ (até o tempo de relógio atual t); $te_{\text{preemp}}(Ji.j, t)$ é tempo em que a VM ficou em estados diferentes de RUN até t, e; $\delta_{si}(Ji.j, t))$ é o atraso estimado devido a quantidade de *swap in* realizado até t.

3.2. Política proativa de preempção

Na nova versão denotada por MEC_PRO, as decisões de preempção passam a considerar $tre(Ji.j, t)$ juntamente com informações coletadas sobre as VMs, como a quantidade de *swap* usada. Ainda, na proposta de MEC_PRO, modificou-se a política preemptiva de pausa de VMs como descrito a seguir. Para diminuir tanto a sensibilidade do MEC quanto à percepção de **siso** e evitar falsos positivos que podem levar a preempções desnecessárias, a *política de preempção de pausa* em MEC_PRO considera a nova condição: a VM está em **siso** se a relação entre a quantidade de *swap in* $si(Ji.j)$ e *swap out* $so(Ji.j)$ é tal que $\beta_1 < \frac{si(Ji.j)}{so(Ji.j)} < \beta_2$, isto é, $si()$ e $so()$ têm magnitude similar, sendo β_1 e β_2 parâmetros de configuração. Desta forma, pequenas flutuações dessas métricas não levam à preempção da VM.

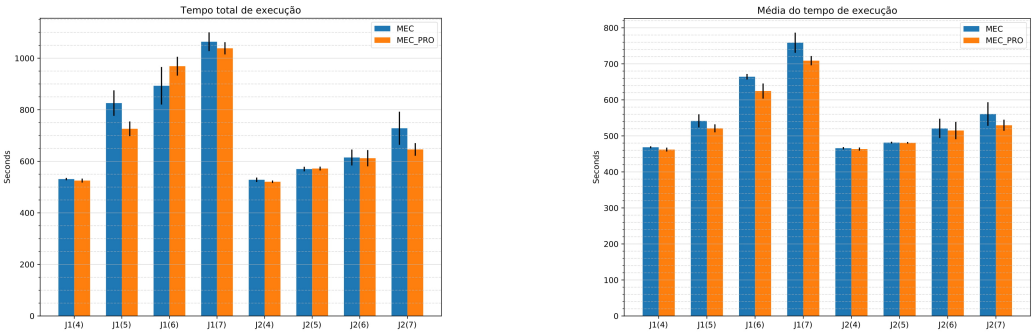
Seja $cPAU$ o conjunto de VMs locais que estão em RUN, mas estão em **siso**. O MEC calcula $tre(Ji.j, t)$ para cada VM em $cPAU$ e cria a lista $cPAU_{ord}$ ordenada não-decrescentemente por $tre(Ji.j, t)$. Na política preemptiva, a primeira VM em $cPAU_{ord}$, vm_1, não é pausada, continuando em RUN, enquanto as VMs restantes vão para PAU. Apesar de vm_1 estar em **siso**, a motivação é que continue em execução mesmo sofrendo atraso, mas não é afetada pela concorrência das outras VMs acessando *swap*. Considerando uma boa previsão, a referida VM terminará primeiro que as outras pausadas e a memória será liberada para que MEC restaure alguma VM em PAU para RUN.

Em relação à politica de suspensão, o MEC pode decidir suspender mais de uma VM em um mesmo intervalo de gerenciamento. Considerando que a suspensão é um processo demorado e varia de acordo com a quantidade de memória sendo consumida pela VM, decidiu-se em MEC_PRO suspender somente uma VM, a menor, por vez, liberando memória mais gradativamente, porém mais rapidamente.

4. Resultados

A comparação entre os resultados produzidos por MEC [Valencia et al. 2018] e MEC_PRO, que considera o escalonamento proativo baseado em MPSi (Seção 3.1), é apresentada a seguir. Os testes foram executados em um único *host* com $M = 22\ GiB$, utilizando $J1$ e $J2$ que idealmente necessitam de $4,5\ GiB$ de memória alocada, cada. Foram quatro conjunto de testes com $n = 4, 5, 6, 7$ VMs respectivamente, (um *job* por VM), submetidos em intervalos de 20 segundos. Em relação ao MEC_PRO, os valores $\beta_1 = 0,5$ e $\beta_2 = 2,0$ foram definidos experimentalmente. Uma análise mais extensa da relação entre esses parâmetros será realizada em trabalhos futuros. Note que no MEC, β_1 e β_2 não são utilizados – qualquer que seja o **siso**, a VM já é considerada para preempção. A notação $Ji(n)$ denota a execução de um job Ji em cada uma de n VMs.

Como é possível observar na Figura 4a, o MEC_PRO reduziu os tempos totais de execução de cada conjunto de VMs, em média, em 1,10%, 12,09%, -8,5%, 2,39% (no caso de $J1$, $n = 4, 5, 6, 7$) e 1,45%, -0,35%, 0,44%, 11,28% (para o caso de $J2$, $n = 4, 5, 6, 7$). Vale apontar que nos casos de $J1(4)$ e $J2(4)$, a melhora no tempo total de execução não deve ser atribuída à nova abordagem, mas sim a uma flutuação natural, já que há memória suficiente para que as quatro VMs sejam atendidas. Nos demais casos essa melhora é devido às mudanças implementados no MEC_PRO, que diminuem o número de preempções desnecessárias, como exibido na Tabela 1.



(a) Tempo total de execução **(b) Média do tempo de execução**

Figura 4. Comparação entre MEC e MEC_PRO com $J1$ e $J2$ em $n = 4, 5, 6, 7$ VMs

Tabela 1. Número médio de preempções de MEC e MEC_PRO

	$J1(4)$	$J1(5)$	$J1(6)$	$J1(7)$	$J2(4)$	$J2(5)$	$J2(6)$	$J2(7)$
MEC	0	$0,9$	$3,4$	$6,75$	0	0	$1,05$	$2,2$
MEC_PRO	0	$0,1$	$1,45$	$2,75$	0	0	$0,25$	$0,1$

Analisando mais detalhadamente $J1(6)$ da Figura 4a em que o tempo total de execução piorou em relação ao MEC, percebe-se que a maior tendência preemptiva do MEC produziu uma melhor distribuição do atraso oriundo das preempções entre as execuções das VMs, enquanto o MEC_PRO, reduz o número de preempções e prioriza *jobs* que potencialmente terminaram primeiro. No MEC_PRO, os últimos *jobs* admitidos no sistema, por terem maior $tre()$, são "sacrificados" em prol dos admitidos anteriormente e, por isso, o tempo total de execução é maior do que em MEC. Entretanto, MEC_PRO visa alcançar um tempo médio de execução menor, de cada *job*, em relação aos tempos produzido pelo MEC, como pode ser observado na Figura 4b. Os resultados atestam a melhoria na média do tempo individual de execução foi de 1,37%, 3,80%, 5,96%, 6,56% no caso de $J1$, $n = 4, 5, 6, 7$, e de 0,50%, 0,13%, 1,12%, 5,57% no caso de $J2$, $n = 4, 5, 6, 7$. Em outras palavras, no caso de $J1$ com $n = 7$ *jobs*, um total de $348,6$ segundos de utilização de recursos (núcleos de CPU e memória RAM) foram poupados. No caso de $J2$, a melhora não foi devido MPSi, mas sim aos aprimoramentos implementadas em MEC_PRO como a inclusão de β_1 e β_2.

5. Trabalhos Relacionados

A disponibilidade de memória é fundamental para o desempenho de execução de aplicações, sejam estas online [Farokhi et al. 2016], ou em lote [Sawamura et al. 2015]. [Moltó et al. 2013] explorou elasticidade vertical com VEM mantendo uma porcentagem de memória livre nas VMs que deve ser definida pelo usuário, o que pode ser uma difícil tarefa – valores errôneos podem prejudicar o desempenho das VMs. [Sawamura et al. 2016] mostrou que MEC consegue um desempenho melhor do que VEM por gerenciar a utilização *swap* com um uso inteligente de preempção.

Elasticidade está também sendo aplicada em contêineres e [Al-Dhuraibi et al. 2018] coordena elasticidade vertical de VMs e contêineres dentro do mesmo ambiente de nuvem. [Chen et al. 2020] implementam preempção de tarefas em contêineres como uma forma de explorar a elasticidade vertical, onde recursos

podem ser tirados imediatamente ou gradativamente de contêineres que executam tarefas longas. Memória é liberada quando ocorre a preempção e o uso de *swap* também é monitorado como uma forma de orientar as decisões. A necessidade da decisão de preempção como política de escalonamento de VMs inspirou o trabalho aqui desenvolvido.

6. Conclusões

Devido a sua natureza dinâmica e compartilhada, o gerenciamento dos recursos da Nuvem é uma tarefa complexa. Este trabalho propõe apontar os benefícios de uma abordagem proativa para guiar decisões de escalonamento a partir da análise e estabelecimento de correlações entre informações associadas ao desempenho de execução. O modelo preditivo proposto correlaciona *swap in* com o atraso no tempo de execução na VM, sendo incorporado ao gerenciador, o MEC_PRO, juntamente com aprimoramentos nas decisões sobre preempção de VMs. O resultado dessas modificações é um gerenciador capaz de ponderar sobre as informações coletadas e a partir delas tomar decisões que, para a versão anterior MEC, seriam classificadas como prejudiciais à execução da aplicação. Como trabalhos futuros, mais análises com diferentes parâmetros, configurações e aplicações *memory bound* são necessárias para melhor traçar os benefícios da proposta. Ainda, a previsão do impacto de eventos como suspensão e do tempo de fim dos *jobs* nas VMs serão pesquisados. Em suma, esforços futuros são para munir o MEMiC e o MEC de ferramentas para possibilitar a criação de um escalonamento proativo efetivo, elaborando algoritmos de escalonamento que tentam abarcar a maior quantidade de cenários possíveis.

Referências

Al-Dhuraibi, Y., Zalila, F., Djarallah, N. B., and Merle, P. (2018). Coordinating Vertical Elasticity of both Containers and Virtual Machines. In *8th Int. Conf. Cloud Computing and Service Sciences (CLOSER)*.

Chen, W., Zhou, X., and Rao, J. (2020). Preemptive and low latency datacenter scheduling via lightweight containers. *IEEE Trans. on Par. and Dist. Systems*, 31(12):2749–2762.

Farokhi, S., Jamshidi, P., Lakew, E., Brandic, I., and Elmroth, E. (2016). A hybrid cloud controller for vertical memory elasticity: A control-theoretic approach. *Future Generation Comp Sys*, 65:57–72.

Kotz, S., Read, C. B., Balakrishnan, N., Vidakovic, B., and Johnson, N. L. (2004). *Encyclopedia of Statistical Sciences*. Wiley-Interscience.

Moltó, G., Caballer, M., Romero, E., and Alfonso, C. (2013). Elastic memory management of virtualized infrastructures for applications with dynamic memory requirements. *Procedia Comp Sci.*, 18:159–168.

Sawamura, R., Boeres, C., and Rebello, V. E. F. (2015). Evaluating the Impact of Memory Allocation and Swap for Vertical Memory Elasticity in VMs. In *2015 27th Int. Sym. on Computer Arc. and High Performance Computing (SBAC-PAD)*, pages 186–193.

Sawamura, R., Boeres, C., and Rebello, V. E. F. (2016). MEC: The Memory Elasticity Controller. *23rd IEEE Int. Conf. on High Performance Comp. (HiPC)*, pages 111–120.

Valencia, J., Boeres, C., and Rebello, V. E. F. (2018). Combining VM preemption schemes to improve vertical memory elasticity scheduling in clouds. In *IEEE/ACM 11th Int. Conf. on Utility and Cloud Computing (UCC)*, pages 53–62.