Escalonamento de Aplicações em Instâncias Preemptivas Sujeitas a Falhas Temporais

Luan Teylo[1], Lúcia Maria de A. Drummond[1], Luciana Arantes[2], Pierre Sens[2]

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

[2]Laboratoire d'Informatique de Paris 6 – Sorbonne Université

{luanteylo, lucia}@ic.uff.br, {luciana.arantes, pierre.sens}@lip6.fr

Abstract. *By contracting Virtual Machines on Amazon EC2, the user can opt for on-demand instances, which have high availability, or instances spot (preemptive), which are idle resources offered at a lower price, whose availability may vary throughout the run. Despite the economic advantage, if the demand for resources increases, the preemptive instances can be revoked by the provider without warning, and put in hibernation until the demand decreases. This work presents initial results and the formulation of the scheduling problem which considers hibernation-prone preemptive instances to minimize the monetary cost of execution.*

1. Introdução

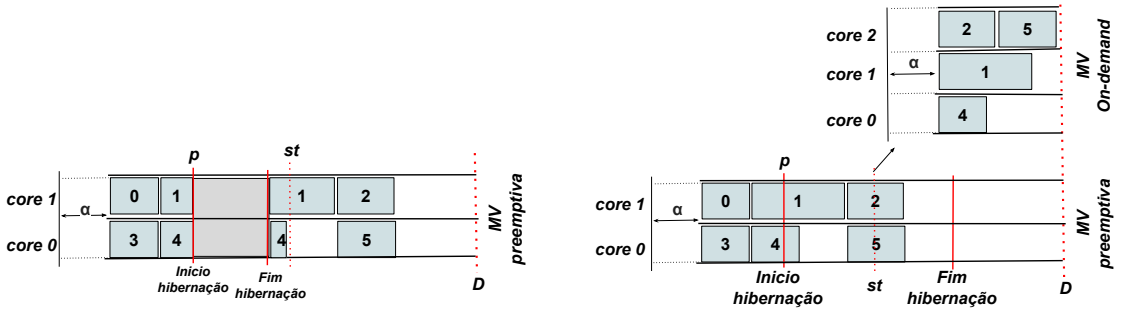
Os provedores do tipo IaaS (do inglês, *Infrastructure-as-a-Service*) permitem que os usuários contratem máquinas virtuais (MVs) com diferentes recursos a um preço fixo por unidade de tempo. Atualmente, esses provedores oferecem diversas classes de MVs, com diferentes garantias em relação a sua disponibilidade e volatilidade. Na Amazon EC2, por exemplo, o usuário pode escolher entre três classes de MVs[1]: *reserved*, que tem preço total pré-definido, e disponibilidade de longo prazo; *on-demand*, que são instâncias alocadas para períodos de tempo específicos, cuja disponibilidade é garantida durante toda a contratação; e *spot*, que são recursos ociosos vendidos com descontos de até 90% em seu preço, porém sem garantia de disponibilidade [Agmon Ben-Yehuda et al. 2013]. A disponibilidade das instâncias *spot*, também chamadas de instâncias preemptivas, flutua conforme a demanda de recursos da nuvem. Conforme representado na Figura 1a, quando a demanda aumenta, o provedor hiberna a MV, salvando sua memória e contexto. Caso a demanda diminua, a instância é iniciada novamente a partir do ponto de hibernação.

Este trabalho apresenta os resultados iniciais da proposta de uso de instâncias preemptivas sujeitas a hibernação para minimizar o custo de execução de aplicações BoT (do inglês, *Bag-of-Task*), sujeitas a um *deadline*. Note que uma MV hibernada pode, ou não, retornar a tempo para que a aplicação cumpra o *deadline*. Caso a MV não retorne a tempo, dizemos que ela sofreu uma *falha temporal* e, neste caso, instâncias *on-demand* são utilizadas para que a aplicação cumpra o *deadline*. O objetivo é demonstrar as vantagens econômicas do uso de instâncias preemptivas para esse tipo de aplicação e que, do ponto de vista econômico, esperar pode ser a melhor decisão em relação a hibernação.

[1]https://aws.amazon.com/ec2/instance-types/

2. Definição do Problema

Seja M o conjunto das MVs que podem ser utilizadas para executar a aplicação BoT, B o conjunto de tarefas desta aplicação e $T = [1, \ldots, D]$ um conjunto discreto que representa o tempo válido de execução, no qual D é o *deadline* definido pelo usuário. Uma MV $j \in M$ pode ser *on-demand* ou preemptiva e tem seu custo de alocação dado por c_j, que é o preço que o usuário paga por um Ciclo de contratação (Cc), que em geral é de 60 minutos. Além disso, toda MV $j \in M$ tem uma capacidade de memória m_j, dada em gigabytes, e um conjunto de *cores* virtuais VC_j de tamanho $|VC_j|$. Por sua vez, cada tarefa $i \in B$ requisita uma quantidade de memória rm_i (também em gigabytes), que deve estar disponível ao longo de toda a sua execução. Além disso, assumimos que cada tarefa será executada em apenas um *core* $k \in VC_j$ da MV, não sendo permitido concorrência de tarefas por *cores*. As Figuras 1a e 1b apresenta esse modelo de escalonamento. Como pode ser visto, como não há memória suficiente para atender as tarefas 1 e 5 ao mesmo tempo, o *core 0* permanece ocioso até que a tarefa 1 termine, e a memória seja liberada. Note que, como não há concorrência por *cores* virtuais, as tarefas alocadas em um mesmo *core* k são executadas uma depois da outra.



(a) Hibernação sem falha temporal **(b)** Hibernação com falha temporal

Figura 1. Cenários de alocação de tarefas em instâncias preemptivas suscetivas a falhas temporais

Com isso, podemos definir o problema de escalonamento como um problema de otimização combinatória, utilizando a formulação da mochila multi-dimensional [Kou and Markowsky 1977]. Para isso, considere a variável binária X^t_{ijk}, que indica se uma tarefa $i \in B$ começará executar ($X^t_{ijk} = 1$), ou não ($X^t_{ijk} = 0$), no core $k \in VC_j$ no tempo $t \in T$. Sabendo que e_{ij} é o tempo de execução da tarefa i na MV j, podemos definir as restrições de memória e de CPU (Equações 1 e 2). Além dessas restrições também é necessário definir as restrições de alocação e de *deadline*. A restrição apresentada na Equação 3, determina que toda tarefa $i \in B$ deve ser alocada e iniciada apenas uma vez, em uma única MV e em um único core. Já a restrição da Equação 4 garante que o deadline D não será violado. Por fim, definimos o objetivo de minimizar o custo da execução, conforme a Equação 5.

$$\sum_{i \in B} \sum_{p}^{t} X^p_{ijk} * rm_i \leq m_j, \quad \forall j \in M, \forall t \in T \text{ e } \forall k \in VC_j, \text{ onde } p = \max(t - e_{ij}, 1) \tag{1}$$

$$\sum_{i \in B} \sum_{p}^{t} X_{ijk}^{p} \leq |VC_j|, \quad \forall j \in M, \forall t \in T \text{ e } \forall k \in VC_j, \text{ onde } p = \max(t - e_{ij}, 1) \tag{2}$$

$$\sum_{j \in M} \sum_{t \in T} X_{ijk}^{t} = 1, \forall i \in B \tag{3}$$

$$X_{ijk}^{t} * (t + e_{ij}) \leq S_j \leq D \quad \forall i \in B, \forall j \in M \text{ e } t \in T \tag{4}$$

$$\min \sum_{j \in M} X_{i_j}^{t} * (\lceil \frac{S_j}{Cc} \rceil * c_j) \tag{5}$$

Em um ambiente onde as MVs são suscetíveis a hibernações, podem ocorrer os seguintes cenários: i) a MV hibernada retorna a tempo para que a aplicação cumpra o *deadline* (Figura 1a); ou ii) a MV não retorna a tempo para que o *deadline* seja cumprido (Figura 1b). O segundo cenário representa uma *falha temporal* na instância e, neste caso, é necessário migrar as tarefas alocadas nessa instância para MVs *on-demand*. Desse modo definimos o problema de falhas temporais seguidas de migração. Sejam $j \in M$ um instância que hibernou no tempo $p \in T$ e $Q_{jp} \subset B$ o conjunto das tarefas que devem ser migradas se j não retornar a tempo de cumprir o *deadline*. O conjunto Q_{jp} é composto pelas tarefas que estavam executando na MV j no momento da hibernação (tempo p) e pelas tarefas que estavam esperando para serem executadas nessa instância. Assim, podemos definir Q_{jp}, também chamado de grupo de migração, conforme a Equação 6. Onde $Suc_{ij} \subset B$ é o conjunto formado pelas tarefas sucessoras de i, isto é, tarefas alocadas no mesmo core que i e que só serão executadas após o seu término, e $Par_{jp} \subset B$ é o conjunto que contém as tarefas que estão executando na MV j no tempo $p \in T$.

Além das tarefas que deverão ser migradas, também é necessário definir as instâncias *on-demand* que serão utilizadas caso uma falha ocorra. Seja $K \subset M$ o conjunto formado por essas instâncias, considerando que as MVs em K levam um tempo de *overhead* α para ficarem prontas para execução, e sabendo que o número de períodos necessários para executar as tarefas em Q_{jp} utilizando essas MVs é dado por $rt(Q_{jp}, K)$, podemos determinar um tempo limite de espera $st \in T$ (Equação 7), de modo que, caso j não retorne antes de st o processo de migração deverá ser iniciado. Dessa forma, o problema da falha temporal se resume em encontrar um conjunto de instâncias $K \subset M$, tal que $st + rt(Q_{jp}, K) \leq (D - \alpha)$. Note que a solução de migração só será valida se $p \leq st$, já que o processo deve iniciar depois da hibernação. Com isso, podemos definir um limite superior para $rt(Q_{jp}K)$, chamado de restrição de migração (Equação 8).

$$Q_{jp} = \bigcup_{i \in (Par_{jp})} Suc_{ij} \cup \{i\} \tag{6}$$

$$st = (D - \alpha) - rt(Q, K) \tag{7}$$

$$rt(Q_{jp}, K) \leq (D - \alpha) - p \tag{8}$$

3. Resultados Experimentais

Essa Seção apresenta as avaliações iniciais do custo monetário das MVs preemptivas sujeitas a hibernação, considerando a formulação apresentada na Seção 2. Os testes foram realizados utilizando as informação das MVs oferecidas pela Amazon EC2, com os preços em dólar das instâncias *spot* (preemptivas) e *on-demand* no mês de setembro de 2018 na região *us-east-1*. Foram utilizadas as MVs dos tipos m4.large, c3.large, c4.large, m4.2xlarge e m4.4xlarge. Já as aplicações BoTs utilizadas, foram obtidas em [Reiss et al. 2011], uma base de dados que contém as informações de execução de uma série de aplicações submetidas ao servidores do Google durante todo o mês de março de 2011. Para os testes foram utilizados 4 aplicações desse banco de dados, sendo que cada aplicação foi avaliada utilizando o menor *deadline* possível (definido iterativamente através de incrementos de 1 hora). A estratégia de *Migração Com Espera* foi comparada com uma abordagem que utiliza apenas MVs *on-demand* (*Apenas On-demand*), e com uma abordagem que não considera a possibilidade de retorno das MVs hibernadas (*Migração Imediata*). Para a avaliação dos custo de execução, foram considerados os dois cenários possíveis para a hibernação: com falha temporal; e sem falha temporal. O tempo de inicio da hibernação em ambos os cenários foi fixado no período de 2 horas, e a duração da hibernação para o cenário sem falha temporal foi de 3 horas, já no caso do cenário com falha temporal a duração da hibernação foi de 1000 horas.

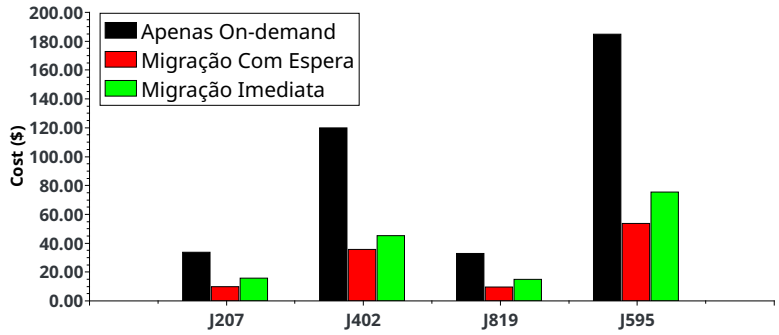
A Figura 2 apresenta o custo médio das execuções em ambos os cenários. No cenário sem falha temporal (Figura 2a), a solução de (*Migração Com Espera*) apresentou uma redução de custo monetário de 28,09% e 70,69% em relação as abordagens de *Migração Imediata* e *Apenas On-demand*, respectivamente. Esse resultado é esperado uma vez que de acordo com a política de preços da Amazon EC2, o usuário não é cobrado pelo tempo que instância permanece hibernada. Porém, note que mesmo a estratégia de *Migração Imediata* foi em média 59,24% mais barata que a estratégia *Apenas On-Demand*. Isso acontece, pois uma parte considerável das tarefas são executadas durante as duas primeiras horas já que, devido ao baixo custo das instâncias preemptivas, MVs de maior poder computacional podem ser alocadas nessa fase do processamento. Já no cenário onde a hibernação gerou uma falha temporal, Figura 2b, os custo monetário das abordagens de *Migração Imediata* e *Com Espera* são iguais, já que o planejamento de migração é o mesmo nas duas abordagens. Assim, as estratégias de migração foram 59,24% mais baratas do que a abordagem *Apenas On-demand*.

4. Conclusão e Trabalhos Futuros

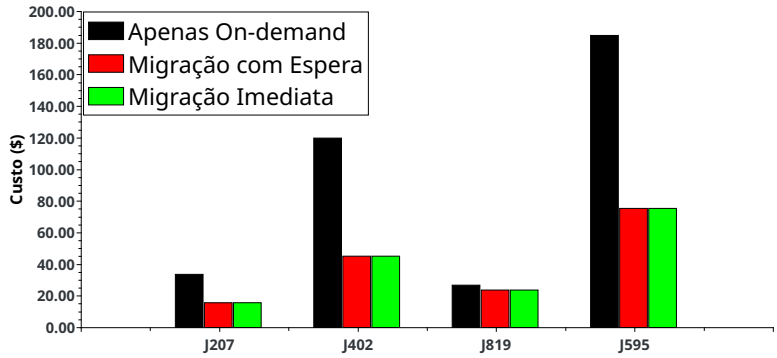
Este trabalho apresentou o problema de escalonamento de aplicações em nuvens computacionais utilizando MVs preemptivas sujeitas a falhas temporais. Foram apresentados a formulação matemática do problema, e a avaliação em termos de custo monetário considerando diferentes estratégias para lidar com a hibernação.

Os testes mostraram que o uso dessas instâncias é economicamente viável, sendo que, mesmo no cenário com ocorrência de falhas temporais, os custos de execução das aplicações avaliadas foram menores quando comparados a execução utilizando apenas instâncias *on-demand* (59,24%, em média).

Como trabalho futuro, será desenvolvido uma proposta de escalonamento que associa técnicas de tolerância a falhas, como *checkpoints* e reexecução em ambientes suscetivos a falhas temporais. Adicionalmente, serão criados mais cenários de testes para a avaliação da proposta, considerando modelos estatísticos para simular hibernações e falhas temporais.



(a) Cenário sem falhas temporais



(b) Cenário com falhas temporais

Figura 2. Comparação do custo de execução nos cenários com e sem falhas temporais

Referências

Agmon Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., and Tsafrir, D. (2013). Deconstructing amazon ec2 spot instance pricing. *ACM Transactions on Economics and Computation*, 1(3):16.

Kou, L. T. and Markowsky, G. (1977). Multidimensional bin packing algorithms. *IBM Journal of Research and development*, 21(5):443–448.

Reiss, C., Wilkes, J., and Hellerstein, J. L. (2011). Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, pages 1–14.