Escalonamento de Contêineres com Requisitos de Segurança Guiado por Aprendizado de Máquina

Kleiton Pereira[1], Renato Tanaka[1], Gustavo B. da Silva[1], Maurício A. Pillon[1], Marcelo Pasin[2], Guilherme P. Koslovski[1]

[1]LabP2D - Universidade do Estado de Santa Catarina, UDESC, Brasil

[2]University of Neuchâtel, Suíça

Resumo. *A utilização de contêineres em ambientes compartilhados requer mecanismos para prover segurança aos dados e operações. Neste contexto, este trabalho descreve um escalonador de contêineres com requisitos de segurança utilizando técnicas de aprendizado de máquina. Além de detalhes sobre a implementação, uma análise experimental destaca a eficiência do escalonador.*

1. Introdução e Motivação

O uso de contêineres para implementar sistemas distribuídos está em evidência devido a maleabilidade gerencial oferecida. Um diferencial de contêineres perante outras tecnologias é a variação em suas demandas de carga, ou seja, o requisitante informa valores mínimo e máximo para cada atributo (*e.g.* CPU, RAM). O compartilhamento de ambientes por múltiplos usuários remete, aos envolvidos, a preocupações na segurança da execução de suas aplicações. A criptografia dos dados tem sido aplicada a segurança de aplicações há décadas, porém, recentemente, processadores passaram a permitir o confinamento de dados e instruções em registradores (*Software Guard Extensions* (SGX)) [Costan and Devadas 2016]. Suportado pela criptografia em registradores, um de *framework* de gerenciamento de contêineres, com escalonador baseado no princípio *bin packing*, foi proposto por pesquisadores suíços [Vaucher et al. 2018].

O problema em questão é *NP-Difícil* e a solução com *bin packing* não é escalável. A escalabilidade em problemas de escalonamento já foi tratada em outros trabalhos com técnicas de Aprendizado por Reforço. Vengerov *et al.* propuseram um *framework* para a alocação dinâmica de recursos [Vengerov 2005] e Cheng *et al.* definiram um escalonador de tarefas para nuvens computacionais através do uso de *Deep Reinforcement Learning* (DRL) com foco em economia de energia [Cheng et al. 2018]. Neste contexto, o presente trabalho busca tornar a solução deste problema escalável através do uso de Aprendizado de Máquina (AM) e Redes Neurais (RN), acrescentando-se aos requisitos usuais o *Enclave Page Cache* (EPC) que é um componente do SGX.

2. Escalonador Proposto

2.1. Escalonamento por *Deep Reinforcement Learning*

O princípio de funcionamento do Escalonador DRL proposto é centrado em dois componentes: o agente e o ambiente. O agente é modelado como uma rede neural *Deep Q Network* (DQN) cujo objetivo é predizer um valor de desempenho conhecido como Q. Agente e ambiente interagem adequando, através do principio da ação-e-reação, os pesos da rede neural recalculado, a cada iteração, com base em uma recompensa fornecida pelo

ambiente. O ambiente, por sua vez, faz o gerenciamento dos contêineres de acordo com a fila de requisições, alocando ou desalocando-os de seus hospedeiros.

A cada iteração, a rede neural (agente) processa uma entrada constituída do estado atual dos N servidores (*e.g.* capacidade física do servidor) e a nova solicitação (duração estimada, capacidade mínima/máxima de recursos virtuais demandados - CPU_{min}, CPU_{max}, RAM_{min}, RAM_{max}, EPC_{min}, EPC_{max}). A saída deste agente é definição do servidor físico (baseado em uma função *softmax*) que a nova operação (alocação ou desalocação) deve ser aplicada. Todavia, a DQN não retorna a capacidade reservada ou residual do servidor, dificultando o processamento da próxima solicitação. Para resolver este problema, o escalonador proposto decompôs os servidores em faixas de recursos (subclasses). Solicitações são adequadas as faixas de recursos disponíveis, não mais a quantidade de recursos residuais. Portanto, o agente retorna ao ambiente a subclasse atribuída a solicitação e uma recompensa é atribuída a esta ação. A subclasse escolhida deve garantir a disponibilidade de recursos mínimos solicitados pela requisição.

2.2. Cálculo da Recompensa

A recompensa de requisição (r_{final}) é composta por três termos principais: a utilidade da alocação (r_{req}), a consolidação do DC (r_{cons}) e o eventual atraso no tempo de execução (r_{tempo}). Inicialmente, $r_{req} = \frac{1}{4} \cdot \left(\frac{CPU_{alloc}}{CPU_{max}}\right) + \frac{1}{4} \cdot \left(\frac{RAM_{alloc}}{RAM_{max}}\right) + \frac{1}{2} \cdot \left(\frac{EPC_{alloc}}{EPC_{max}+\varepsilon}\right)$ representa a função de utilidade indicando quanto a ação escolhida atendeu aos requisitos do usuário. CPU_{alloc}, RAM_{alloc} e EPC_{alloc} indicam os valores alocados, sendo que $\varepsilon > 0$ é necessário para evitar a divisão por zero em contêineres sem requisitos de segurança (EPC). Ainda, há pesos de importância para os recursos, dando maior impacto ao requisito de segurança. A recompensa quanto a consolidação do DC (r_{cons}), é calculada em função da desativação de servidores ociosos, ou seja, é atribuída com base na quantidade de servidores já em trabalho que poderiam comportar esse contêiner. Por sua vez, r_{tempo} é calculada em função do tempo de atraso, que analisa qual o impacto da alocação na fila de contêineres que possuem seu tempo de submissão igual ao momento atual. Por fim, r_{final} é definido como a média das recompensas individuais ($r_{final} = \frac{r_{req}+r_{cons}+r_{tempo}}{3}$).

3. Análise Experimental

Para o desenvolvimento tanto do ambiente quanto do agente foi utilizada a linguagem de programação Python (versão 3.7.2). Na implementação e no treinamento da rede neural foram utilizadas as bibliotecas Keras (versão 2.2.4) e TensorFlow (versão 1.12.0).

3.1. Métricas e Cenários Experimentais

A análise compara os resultados do escalonador proposto com o algoritmo *Round Robin* (RR), tradicionalmente utilizado pelos gerenciadores de contêineres Kubernetes e Swarm. Ainda, para definir a linha ótima, o cenário foi modelado como Programação Linear Inteira Mista (MILP), e solucionado usando a ferramenta IBM CPLEX. Para as comparações, foram selecionadas as métricas de atraso para um contêiner ser escalonado, a fragmentação do DC, e a utilidade (perspectiva do cliente).

Os dados de entrada provém do registro do Google Borg, previamente utilizado pela literatura [Vaucher et al. 2018]. Devido a abstração originalmente realizada, uma normalização ajustou as configurações ao *hardware* disponível. Os valores mínimos e

máximos foram inferidos dos valores inicialmente solicitados e a média utilizada. Os requisitos de segurança (EPC) não foram descritos nos dados originais e, para fins de validação, foram injetados de acordo com a quantidade de RAM. O percentual de EPC das solicitações variou entre 0% e 75%, possibilitando a análise do comportamento em 4 cenários (0%, 25%, 50% e 75%). Para todos os casos, foram considerados, 15 servidores divididos em duas classes. Na primeira, 9 servidores foram configurados com 8 cores, 64 GB RAM e 0 capacidade de EPC. Na segunda, 6 servidores possuem 8 cores, 8 GB RAM e 93, 5 MB de EPC.

3.2. Treinamento da Rede Neural

A RN foi treinada utilizando uma parte dos dados exclusiva para treinamento, assim evitando o *overfitting* (fenômeno memorização das respostas). Durante o treinamento, a RN armazena em memória: o valor da recompensa, o estado anterior do DC, a decisão do agente, o servidor e a subclasse escolhidos e, para finalizar, o novo estado do DC (aquele imediatamente após o provisionamento da última solicitação). O treinamento da RN acontece após o processamento de uma amostra de contêineres, utilizando os dados memorizados e a equação de Bellman com o intuito de encontrar os pesos adequados.

3.3. Discussão dos Resultados

Finalmente, os resultados são apresentados na Figura 1 e ordenados por percentual de EPC. Uma tendência geral é observada: o escalonador com DQN está mais próximo do ótimo quando comparado ao RR. Especificamente, no cenário sem requisições com EPC (Figura 1(a)) não ocorrem atrasados quando o algoritmo ótimo é usado. Porém, ao comparar com a métrica de utilidade é possível observar que tanto o RR quanto a DQN possuem maior desempenho. O algoritmo ótimo sempre seleciona os valores mínimos solicitados, enquanto DQN e o RR buscam atender o pedido máximo. Ainda, há consideravelmente menos atrasos nas alocações realizadas pela DQN quando comparado ao RR. A fragmentação mostra maior tendência da DQN em espalhar os contêineres ao longo dos servidores. Analisando o cenário com 25% de requisições EPC (Figura 1(b)), o mesmo padrão é observado em relação ao cenário anterior. Com 50% de requisições EPC (Figura 1(c)), uma variação é identificada. A fragmentação do algoritmo ótimo é similar ao do algoritmo RR enquanto a DQN distribui os contêineres aumentando a utilidade com menor atraso possível, visto que o número de servidores que possuem EPC é reduzido. Por fim, com 75% de requisições EPC (Figura 1(d)) o atraso da DQN é semelhante ao do RR devido ao grande número de contêineres solicitantes de um recurso escasso.

4. Conclusões

Um escalonador tem como tarefa alocar recursos para diferentes requisições, otimizando o DC. Sobretudo, em um cenário com múltiplos usuários compartilhando os recursos, o escalonador deve considerar requisitos de segurança nas requisições. O escalonador apresentado obteve resultados superiores aos algoritmos clássicos de escalonamento e se aproximou, em alguns casos, da solução ótima.

Referências

Cheng, M., Li, J., and Nazarian, S. (2018). Drl-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In *Proc. of the 23rd ASPDAC*, pages 129–134, Piscataway, NJ, USA. IEEE.