UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CAETANO BEZERRA CAVALCANTI ALBUQUERQUE

ESCALONADOR OPORTUNÍSTICO NA NUVEM COM QUALIDADE DE SERVIÇO

CAMPINA GRANDE - PB

2023

CAETANO BEZERRA CAVALCANTI ALBUQUERQUE

ESCALONADOR OPORTUNÍSTICO NA NUVEM COM QUALIDADE DE SERVIÇO

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : Professor Dr. Andrey Elísio Monteiro Brito

CAMPINA GRANDE - PB

2023

CAETANO BEZERRA CAVALCANTI ALBUQUERQUE

ESCALONADOR OPORTUNÍSTICO NA NUVEM COM QUALIDADE DE SERVIÇO

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Andrey Elísio Monteiro Brito

Orientador – UASC/CEEI/UFCG

Professor Dr. Reinaldo Cézar de Morais Gomes

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 28 de junho de 2023.

CAMPINA GRANDE - PB

RESUMO

O uso de computação oportunista na nuvem é amplamente adotado devido a sua capacidade de hospedar aplicações de processamento em lote a um custo reduzido. No entanto, esse modelo apresenta a desvantagem de não garantir disponibilidade, o que pode afetar a qualidade de serviço das aplicações. Além disso, devido à alta oscilação de preço, alcançar uma disponibilidade ideal com esse tipo de serviço requer despriorizar o potencial econômico oferecido. Este trabalho propõe uma solução na forma de um escalonador oportunista modular para Kubernetes, que busca equilibrar os custos dos nós oportunistas e os atrasos no processamento, garantindo assim a qualidade de serviço. Com base nos preços de 2021 da AWS, a solução demonstrou uma economia significativa, de até 50% em determinados dias, quando comparado aos custos com instâncias sob demanda, possibilitando o processamento em tempo quase real.

OPPORTUNISTIC AUTO-SCALING IN THE CLOUD GUARANTEEING QUALITY OF SERVICE

ABSTRACT

Opportunistic cloud computing is widely adopted due to its ability to host batch processing applications at a reduced cost. However, its primary drawback lies in the lack of availability guarantees, which can negatively impact application quality of service. Additionally, the high price fluctuations associated with this model make it challenging to achieve optimal availability without compromising the economic potential it offers. To address these challenges, this work presents a solution in the form of a modular opportunistic scaler for Kubernetes. The scaler is designed to strike a balance between the costs of opportunistic nodes and processing delays, ensuring a high quality of service for applications. Based on 2021 AWS pricing, the solution has demonstrated substantial cost savings of up to 50% on certain days when compared to On-Demand Instance costs, enabling near real-time processing.

Escalonador Oportunístico na Nuvem com Qualidade de Serviço

Caetano Bezerra Cavalcanti Albuquerque
Universidade Federal de Campina Grande
Rua Aprígio Veloso, 882, Bloco CO
CEP 58.429-900, Campina Grande, PB, Brasil
caetano.albuquerque@ccc.ufcg.edu.br

Andrey Elísio Monteiro Brito
Universidade Federal de Campina Grande
Rua Aprígio Veloso, 882, Bloco CO
CEP 58.429-900, Campina Grande, PB, Brasil
andrey@computacao.ufcg.edu.br

RESUMO

O uso de computação oportunista na nuvem é amplamente adotado devido a sua capacidade de hospedar aplicações de processamento em lote a um custo reduzido. No entanto, esse modelo apresenta a desvantagem de não garantir disponibilidade, o que pode afetar a qualidade de serviço das aplicações. Além disso, devido à alta oscilação de preço, alcançar uma disponibilidade ideal com esse tipo de serviço requer despriorizar o potencial econômico oferecido. Este trabalho propõe uma solução na forma de um escalonador oportunista modular para *Kubernetes*, que busca equilibrar os custos dos nós oportunistas e os atrasos no processamento, garantindo assim a qualidade de serviço. Com base nos preços de 2021 da AWS, a solução demonstrou uma economia significativa, de até 50% em determinados dias, quando comparado aos custos com instâncias sob demanda, possibilitando o processamento em tempo quase real.

Palavras Chave

Computação oportunista, escalonamento, processamento em lote, cloud, *Kubernetes*.

1. INTRODUÇÃO

A adoção de serviços de nuvem pública tem se tornado cada vez mais popular devido ao modelo IaaS (*infrastructure as a service*). Esse modelo permite aos usuários implantar suas aplicações na nuvem, com uma economia de esforços e custos operacionais. Os clientes têm acesso a uma ampla gama de instâncias disponíveis, nas quais podem hospedar seus serviços, levando em consideração seus requisitos específicos de rede, memória e poder computacional. Dessa forma, os usuários podem terceirizar as preocupações com a infraestrutura (tendo a garantia de uma alta disponibilidade e escalabilidade) e focar no desenvolvimento do seu produto, de maneira fornecer QoS (*Quality of Service*, i.e. qualidade de serviço) a seus clientes.

Os planos sob demanda são os mais tradicionais, nos quais o provedor aloca os recursos conforme solicitado pelo cliente e cobra com base no tempo de uso (por exemplo, por hora, por minuto ou até mesmo por segundo). Além desse modelo, os grandes provedores de nuvem também costumam oferecer o modelo de contratação oportunista (*Spot*). As instâncias *Spot* se diferenciam do plano mais tradicional, por oferecer ao cliente os recursos ociosos na nuvem, a um custo promocional (p. ex., 90% de desconto no caso da AWS [7]), porém sem garantia de disponibilidade. Dessa forma, a oferta do modelo oportunista varia de acordo com a procura por VMs em outros planos.

Devido à falta de garantia de disponibilidade, nem todas as aplicações podem se beneficiar dos descontos oferecidos pelas instâncias oportunistas. Aplicações que possuem requisitos de qualidade de serviço que exigem disponibilidade mínima de acesso ou prazos para o processamento de carga, não são compatíveis com o modelo *Spot*.

Lembrando também que a variação dos preços cobrados pelas instâncias oportunistas depende da disponibilidade de recursos ociosos. Para disponibilizar uma opção aos clientes de lidar com essa flutuação de preços, os provedores costumam permitir que os usuários definam um limite de preço, que indique o valor máximo que se deseja pagar por determinada VM, caso o preço limite seja atingido, os recursos serão deslocados. Portanto, para maximizar as chances de alocação, é necessário estar disposto a lidar com a oscilação de preços.

Com o objetivo de lidar com os desafios da computação oportunista e manter os custos baixos, este trabalho propõe uma arquitetura de um escalonador oportunista. Essa solução é projetada para escalonar aplicações de processamento em lote em um *cluster Kubernetes*, onde os nós responsáveis por hospedar as réplicas da aplicação são preferencialmente compostos por máquinas *Spot*.

O escalonador busca encontrar um equilíbrio entre o atraso no processamento e a economia de custos com os recursos computacionais. A alocação de recursos é realizada de forma a garantir que os termos de qualidade de serviço (QoS) da aplicação sejam atendidos. Dessa forma, é possível aproveitar as vantagens oferecidas pelas instâncias oportunistas, inclusive para aplicações com processamento em tempo quase real, isto é, prazos curtos próximos ao imediato, desde de que as mesmas possuam uma carga relativamente previsível. Além disso, o escalonador oferece controle de gastos ao evitar que os recursos sejam alocados em momentos nos quais o processamento da carga não está atrasado e o custo com a infraestrutura exceda o orçamento configurado. Isso significa que a solução proposta permite o uso eficiente de máquinas *Spot*, aproveitando os descontos oferecidos, ao mesmo tempo em que mantém o atraso no processamento dos dados em um nível aceitável.

No geral, a arquitetura proposta visa viabilizar o uso de instâncias oportunistas para aplicações de processamento em lote, proporcionando uma combinação de economia, controle na QoS da aplicação e nos custos.

Esse trabalho está organizado da seguinte forma: primeiramente é apresentada a motivação, mostrando o potencial de economia oferecido pelas instâncias oportunistas. Em seguida está descrita a

metodologia para a análise do histórico de preços. Logo depois é apresentada a solução proposta, detalhando cada um dos módulos que a compõem. A próxima seção descreve a forma que foi feita a avaliação da aplicação proposta e os resultados obtidos. E por fim é exposta a conclusão do trabalho e possíveis futuros trabalhos.

2. MOTIVAÇÃO

Atualmente, os provedores de nuvem oferecem duas principais formas de cobrança para a contratação de máquinas virtuais. O primeiro tipo são os planos com preços estáticos, que incluem as instâncias sob demanda e as reservadas. Nesses planos os custos são fixos e não sofrem com constantes oscilações ao longo do tempo. O segundo tipo de cobrança são os planos com custo dinâmico, sendo as instâncias *Spot* o principal representante dessa categoria [5].

Para os clientes que optam pelos planos oportunistas, é importante compreender a frequência e a magnitude das variações dos preços. Isso pode permitir a identificação dos momentos mais favoráveis para executar o processamento de suas aplicações, buscando minimizar os custos. Portanto, ter uma compreensão clara das flutuações de preços ao longo do tempo é de interesse para os contratantes desses planos.

2.1 Potencial de economia

Uma característica essencial das instâncias oportunistas é a sua variação de preços, que está diretamente relacionada ao nível de ociosidade dos recursos da infraestrutura do provedor. Mudanças nos valores cobrados pelas VMs podem ocorrer várias vezes ao longo do dia, refletindo a demanda e a disponibilidade dos recursos.

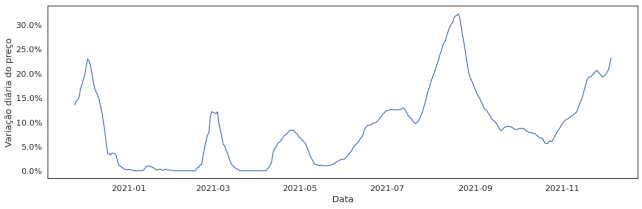

Figura 1: Variação diária no preço da instância *Spot c6g.4xlarge* da AWS, normalizada pelo preço no plano sob demanda

A figura 1 ilustra a variação diária do preço da instância *c6g.4xlarge* ao longo do ano de 2021 [1]. Cada dia é representado pela diferença entre o maior e o menor valor registrado para essa instância, sendo esse valor normalizado pelo custo da mesma VM no plano sob demanda. O gráfico revela que em certos dias ocorreram flutuações com magnitudes significativas, chegando a ultrapassar 30% do custo da instância no plano sob demanda.

Com base na variação diária dos preços das instâncias oportunistas, conforme ilustrado na figura 1, é possível identificar um potencial para aumentar a economia, escolhendo os momentos mais favoráveis para realizar o processamento dos dados. No entanto, para garantir a simplicidade e viabilidade dessa abordagem, é necessário analisar se existe algum padrão na oscilação dos preços ao longo do tempo. Isso permitiria aos usuários identificar os períodos com preços mais baixos e programar suas tarefas de processamento durante esses momento, maximizando assim a economia.

2.2 Dificuldade de previsão

Para analisar a existência de padrões nos valores cobrados pelas instâncias *Spot*, foram gerados gráficos que relacionam o custo da instância *c6g.4xlarge* da AWS com os dias da semana e os horários do dia. Esses gráficos permitem visualizar se as variações nos preços são consistentes.

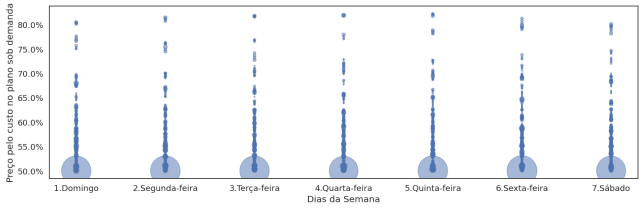

Figura 2: Preço da instância Spot *c6g.4xlarge* da AWS por dia da semana, normalizado pelo preço da instância no plano sob demanda.

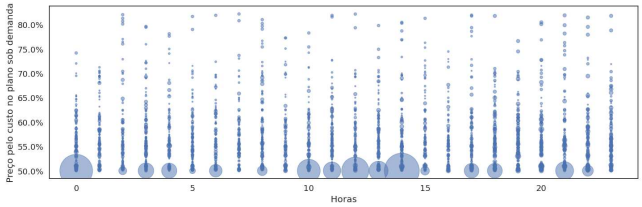

Figura 3: Preço da instância *Spot c6g.4xlarge* da AWS por horas do dia, normalizado pelo preço da instância no plano sob demanda.

As figuras 2 e 3 acima mostram que não existe nenhuma relação direta entre os preços cobrados e os dias da semana ou as horas do dia. Isso inviabiliza a estratégia de decidir o momento da alocação dos recursos com base nos momentos de menor custo registrados. Para conseguir ter um melhor controle sobre o custo com a infraestrutura, seria necessário decidir dinamicamente quando alocar uma instância, baseado em seu preço atual.

Finalmente é importante alinhar essa estratégia ao monitoramento da carga, garantindo que a alocação de instâncias seja feita de maneira eficiente e evitando a alocação desnecessária. Dessa forma, é possível atender aos termos de qualidade de serviço definidos, garantindo a satisfação do usuário e reduzindo os gastos com recursos computacionais.

2.3 Caso de uso: monitoramento não intrusivo de carga

O NIALM (*Non-Intrusive Appliance Load Monitoring*, i.e. monitoramento não intrusivo de carga de aparelhos [4]), é uma aplicação que pode se beneficiar de computação oportunística na nuvem. NIALM é um processo para deduzir os aparelhos que foram utilizados em dado momento, assim como também desagregar seus consumos individuais a partir de sinais energéticos agregados de um edifício. O uso de aplicações desse tipo para prover informações sobre o consumo está associado a uma economia de mais de 12% no uso de energia elétrica [2].

A aplicação escolhida como caso de uso, foi a implementação NIALM da LiteMe, uma empresa de inteligência energética. Essa aplicação usa de Redes Neurais Convolucionais para desagregar

dados [6]. O NIALM conta com componentes auxiliares, que são responsáveis por produzirem os itens de trabalho e publicarem os mesmos em uma fila de mensagens, de onde a aplicação irá resgatar os dados. Cada item de trabalho possui todas as informações necessárias para executar a desagregação (e.g., o medidor utilizado, os dados de energia, a topologia da rede, etc.). O NIALM pode ser escalado de forma horizontal (múltiplos *Pods* no *cluster Kubernetes*) para processar mais carga, caso seja necessário.

A LiteMe usa dos resultados gerados pelo NIALM, para prover serviços de inteligência energética a seus clientes, como o monitoramento dos dispositivos que causaram o maior consumo. Os clientes costumam acessar os serviços apenas esporadicamente ao longo do dia, o que permite pequenos atrasos no processamento dos dados. Além disso, os dados não são exibidos de forma instantânea, apenas nas granularidades de dia ou hora, isso com a finalidade de melhorar a acurácia da desagregação. Porém, caso o processamento esteja atrasado demais, clientes podem sofrer inconveniências, o que fere os termos de qualidade de serviço da aplicação.

3. METODOLOGIA

O objetivo principal deste trabalho é propor uma ferramenta capaz de aproveitar das oscilações nos custos das VMs *Spot*, e ao mesmo tempo não comprometer os termos de QoS da aplicação. Para isso, foram estudados o histórico dos preços das instâncias oportunistas da AWS, e imaginada uma arquitetura capaz de suprir os requisitos.

Para descrever melhor a metodologia, essa seção está dividida da seguinte forma: Inicialmente é exposta uma descrição dos dados e como eles foram usados na análise do histórico de preços. Posteriormente foi tratado sobre o processo de avaliação da solução.

3.1 Análise do histórico de preços

3.1.1 Dados originais

Para a análise do histórico de preços das instâncias oportunistas, foram utilizadas duas bases de dados diferentes. A primeira base de dados foi a que contém o histórico de preços das instâncias *Spot* [1], que contém informações relevantes sobre as instâncias *Spot*, como o tamanho da VM, o sistema operacional, a região de disponibilidade e o preço registrado para cada instância em um determinado momento.

A segunda base de dados usada foi fornecida pela a AWS em seu website e contém informações sobre os diferentes tamanhos de instâncias, incluindo a quantidade de memória, o número de vCPUs e a capacidade de armazenamento. Além disso, também fornece o preço no plano sob demanda para cada tamanho de instância.

Esses dados são essenciais para entender as oscilações nos preços das VMs *Spot* ao longo do tempo e para desenvolver estratégias de alocação eficiente, levando em conta os custos e os requisitos de QoS da aplicação.

3.1.2 Pré-processamento dos dados

Para analisar os dados do histórico de preços, foram realizados alguns passos de pré-processamento. Inicialmente, as informações que foram disponibilizadas no *website* da AWS, que estavam em texto plano, foram estruturadas e organizadas em planilhas separadas por região de interesse. Cada planilha contém as seguintes colunas: tamanho da instância, quantidade de memória, número de vCPUs, capacidade de armazenamento e o preço no plano sob demanda.

Em relação ao *dataset* com o histórico de preços das instâncias *Spot*, devido ao tamanho dos arquivos, foi necessário filtrar os dados por região, sistema operacional e tamanhos de interesse. Essa etapa de filtragem permite focar nas VMs relevantes para a análise.

Em seguida, foi feita a junção dessas duas bases de dados, usando como chave de ligação o tamanho da instância e a região de disponibilidade. Essa junção permite comparar os preços das VMs *Spot* com os custos da contratação no plano sob demanda, facilitando a análise de economia.

É importante ressaltar que, para simplificar a análise, foram considerados apenas os valores da região com o menor custo de cada instância no plano sob demanda. Isso evita uma falsa percepção positiva sobre a economia proporcionada pelas instâncias oportunistas.

3.1.3 Análise

Para entender melhor a magnitude e a frequência da oscilação dos preços das instâncias oportunista, foram criados gráficos usando a biblioteca *Seaborn* para *Python*. Para analisar a amplitude das variações diárias, foi traçado um gráfico de linhas, que ilustra essa flutuação ao decorrer do tempo. Já para verificar a frequência e a existência de padrões na oscilação foram feitos dois gráficos de dispersão, um que relaciona os preços cobrados com as horas do dia e outro que relaciona os custos com os dias da semana.

3.2 Avaliação

Foi realizada uma simulação para avaliar a eficiência da aplicação, utilizando os preços das instâncias oportunistas durante o ano de 2021. O experimento envolveu uma carga sintética baseada em uma estimativa futura da LiteMe, consistindo de 20 trabalhos por segundo. Além do cálculo estimado dos gastos com o uso de VMs *Spot* em conjunto com o escalonador, com o objetivo de comparar os custos entre os diferentes modelos de contratação, foram simulados também cenários com o uso das instâncias reservadas, das instâncias sob demanda e das máquinas oportunista sem a aplicação da solução proposta.

4. SOLUÇÃO

A solução desenvolvida conta com três componentes: o JobMonitor, responsável por gerar um relatório sobre o estado da fila de trabalhos a serem processados; o SpotSaver, que realiza a análise e sumarização dos preços atuais das instância oportunistas; e o WorkWise, responsável por requisitar as informações geradas pelas as outras duas aplicações e a partir desses dados, tomar decisão sobre a magnitude dos escalonamento de nós e *Pods* da aplicação. Apesar de atualmente as aplicações possuírem uma implementação simples, a decisão de separar em três módulos distintos foi feita para facilitar futuras melhorias em cada um dos componentes de forma independente.

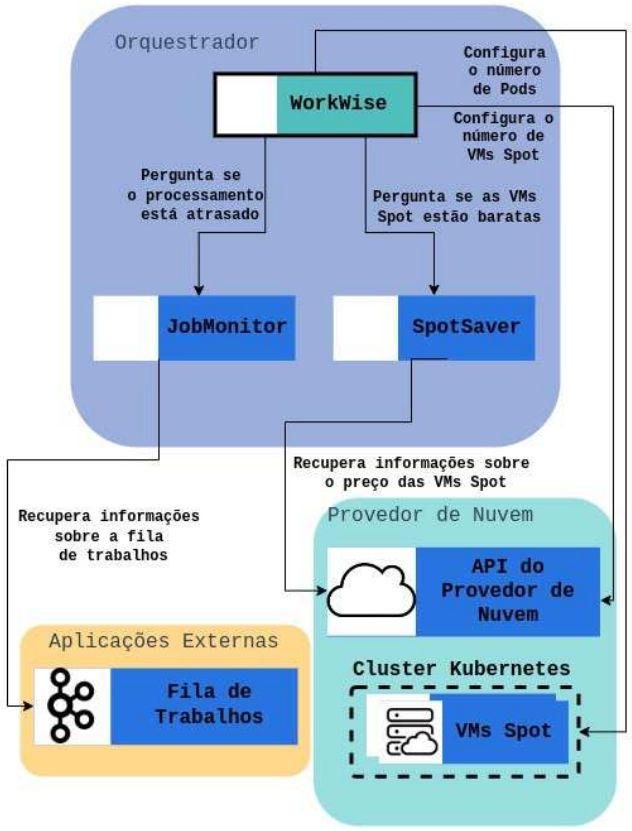

Figura 4: Diagrama da arquitetura proposta.

Essa seção está dividida em três subtópicos, nas quais cada um das três aplicações que compõem a solução será explicado em detalhes.

4.1 JobMonitor

O JobMonitor é uma API que, ao ser inicializada, requer a definição de um limiar, que indica o atraso máximo tolerável para não comprometer os termos de QoS da aplicação, e as credenciais para acessar a aplicação de fila de trabalhos. Quando acionado, o JobMonitor solicita à aplicação de fila de trabalhos informações sobre o dado mais antigo que ainda não foi processado e a quantidade de trabalhos pendentes. A partir desses dados, o JobMonitor consegue determinar se o atraso é considerado crítico, comparando o valor resgatado da fila com o threshold configurado. Em seguida, ele gera um relatório contendo o seguinte conteúdo:

- Tempo de atraso;
- Quantidade de trabalhos pendentes;
- Se o atraso é crítico (em relação ao SLA).

O funcionamento do JobMonitor se da seguinte maneira:

Pseudocódigo 1: JobMonitor

```
@rota("/jobs/delayed")
func get_status:
    fila = new FilaTrabalhos(CREDS)
    num_trabalhos, criação_primeira_mensagem = fila.get_data()

    tempo_atraso = now() - criação_primeira_mensagem
    atraso_crítico = tempo_atraso > ATRASO_MAX
    return {
        "tempo_atraso": tempo_atraso
        "num_trabalhos": num_trabalhos
        "atraso_crítico": atraso_crítico
    }
```

4.2 SpotSaver

O SpotSaver é uma API que no momento da sua inicialização precisa que seja definido um limiar que indica qual é o preço máximo que se deseja pagar por uma VM *Spot*, todos os dados necessários para realizar a requisição ao provedor de nuvem e a região do fornecedor de nuvem na qual se deseja hospedar a aplicação. Quando acionada, requisita para o provedor informações sobre o custo da instância nos planos sob demanda e oportunista em cada zona de disponibilidade da região de interesse. A partir desses dados, o SpotSaver consegue decidir se o custo da VM deve ser considerado barato, comparando o valor atual do plano *Spot* normalizado com o custo do sob demanda com o valor de threshold configurado, e gerar um relatório contendo o seguinte conteúdo:

- Tipo da instância;
- Zona de disponibilidade;
- Preço no plano oportunista;
- Se a VM *Spot* está disponível
- Se o custo deve ser considerado barato.

O funcionamento do SpotSaver se da seguinte maneira:

Pseudocódigo 2: SpotSaver

```
@rota("/price")
func get_spot_price:
    prov = new Provedor()
    preços_spot, preço_sob_demanda, spot_disponível =
            prov.get_preço(VM_TIPO)

    menor_preço_spot, zona = get_melhor_zona(preços_spot)
    razão_preço = menor_preço_spot / preço_sob_demanda
    está_barato = razão_preço <= PREÇO_LIMITE
    return {
        "spot_disponível": spot_disponível
        "está_barato": está_barato
        "razão_spot_sob_demanda": razão_preço
        "zona": zona
```

"vm_tipo": VM_TIPO

}

4.3 WorkWise

Por fim, o WorkWise é responsável por consultar periodicamente os outros dois serviços e com base nos seus retornos, tomar decisões sobre a quantidade de VMs e *Pods* que devem ser criados. Essa aplicação precisa que seja configurados os seguintes valores no momento da sua inicialização:

- Endereço para o JobMonitor e SpotSaver;
- Localização para as credenciais do *cluster Kubernetes*;
- Endereço para os arquivos do *TerraForm*[1], que indicam a configuração da infraestrutura;
- Credenciais para o provedor de nuvem;
- Intervalo entre as ações de escalonamento;
- Quantidade de *Pods* que cada VM suporta;
- Quantidade de trabalhos que cada *Pod* consegue processar durante o intervalo entre os escalonamentos;
- Informações do *Kubernetes* sobre a aplicação que está sendo escalonada, como *namespace* e nome do *Deployment*;
- Máximo de réplicas da aplicação que devem ser executadas simultaneamente.

O WorkWise funciona em ciclos, em cada iteração ele primeiramente requisita o JobMonitor e o SpotSaver informações sobre a fila de trabalho e o custo das instâncias oportunista no momento. Uma vez possuindo essas informações, a aplicação verifica se o custo das VMs está baixo, caso sim ele deve calcular a quantidade de *Pods* e nós para processar todos os dados que estão na fila até a próxima ação de escalonamento. Caso o custo das instâncias não esteja barato o suficiente, o WorkWise irá verificar se o atraso é crítico para a QoS da aplicação que está sendo escalonada, caso seja, as VMs serão escalonada no plano que estiver mais barato, numa magnitude que seja apenas suficiente para conseguir reverter a situação de atraso, para que seja possível esperar por um momento no qual o custo das VMs oportunistas esteja aceitável. Caso o atraso não seja considerado problemático, a quantidade de nós e *Pods* será definida como zero e o escalonador irá aguardar que o preço das instâncias reduza. Um último cenário é quando o atraso no processamento está comprometendo os requisitos de QoS da aplicação, mas o provedor não está ofertando VMs *Spot*, nesse caso será escalonado nós no plano sob demanda, como uma medida emergencial, o numero de maquinas que será definido é o suficiente apenas para reverter o atraso problemático.

O cálculo da quantidade de réplicas é feito dividindo a quantidade de trabalhos a serem processados durante o ciclo, pelo o valor, configurado na inicialização da aplicação, da quantidade de trabalhos que cada *Pod* da aplicação que está sendo escalonada consegue processar durante o tempo do ciclo. Já o número de VMs que serão criadas é dado pela divisão do número de réplicas da aplicação pelo valor, definido na inicialização do WorkWise, como a quantidade de *Pods* que cada máquina suporta, o resultado desse quociente é arredondado para o inteiro maior e mais próximo, isso para garantir que sempre será criada a quantidade de instâncias capaz de hospedar o número de instâncias da aplicação escalonada.

Para garantir o funcionamento da aplicação, o *cluster Kubernetes* precisa contar com um total de três *node pools*. O primeiro deles é constituído por uma VM do tipo sob demanda, que será responsável por hospedar as aplicações propostas neste trabalho, o JobMonitor, o SpotSaver e o WorkWise. Os outros dois *node pools* contarão com os nós responsáveis nos quais as réplicas da aplicação que está sendo escalonada serão instanciadas, um deles irá conter VMs no plano oportunista e o outro no modelo de contratação sob demanda.

O WorkWise, usa da ferramenta *TerraForm*, para configurar o número de VMs que serão alocadas durante os ciclos. Caso ocorra algum problema na criação das máquinas ou dos *Pods* e seja feita mais de uma vez a requisição para a geração de novos *Pods* ou máquinas, não existe o risco de que o número final seja maior do que o definido. Isso é consequência do fato de que tanto essa ferramenta, quanto o *Kubernetes* assumem uma abordagem declarativa, ou seja, as configurações aplicadas refletem um estado desejado e não uma mudança a ser aplicada sobre um estado anterior.

Pseudocódigo 1: WorkWise

```
while True:
  vm_barata, zona, vm_tipo, spot_disponível =
        get_relatório_SpotSaver()
  trabalhos_pendentes, tempo de atraso, atraso_critico =
        get_relatório_WorkWise()
  if vm_barata:
   num_pods = min( trabalhos_pendentes /
                TRABALHOS_POR_PODS, MAX_PODS )
  else if atraso_critico:
    num_trabalhos = (DURAÇÃO_CICLO * 1.5) *
                trabalhos_pendentes / tempo_atraso
    num_pods = min( num_trabalhos /
                TRABALHOS_POR_PODS, MAX_PODS )
  else:
    num_pods = 0
  end if
  num_vm = ⌈num_pods / PODS_POR_VM⌉
  if spot_disponível:
    terraform.set_spot_node(num_vm, zona, vm_tipo)
  else:
    terraform.set_sob_demanda_node(num_vm, zona, vm_tipo)
  end if
  kubernetes.set_num_pods(num_pods)
```

[1] O TerraForm é uma ferramenta de automação de provisionamento e gerenciamento de recursos de infraestrutura em provedores de nuvem.
https://www.terraform.io/

sleep(INTERVALO)

end while

---

5. AVALIAÇÃO

Para avaliar a eficiência da solução proposta, foram simulados os custos com a infraestrutura do NIALM utilizando diferentes tipos de instâncias: sob demanda, reservadas e oportunistas, com e sem o uso do escalonador. Cada réplica do NIALM tem uma capacidade de processar um trabalho a cada 1.5 segundos [3].

Considerando uma carga simulada equivalente a 1200 sensores monitorados, o que resulta em aproximadamente 20 trabalhos por segundo. Essa carga representa uma projeção futura, mas realista, da empresa. Para suportar uma demanda contínua, seriam necessárias cerca de 30 réplicas do NIALM operando constantemente. Os valores utilizados para a simulação foram os preços cobrados pela a AWS durante o ano de 2021 [1], por serem os dados mais recentes, que representam um ano completo, no momento em que as análises iniciaram. A instância escolhida foi a *c6g.4xlarge*, por ser capaz de executar 30 réplicas da aplicação escalonada.

Essa simulação permite avaliar os custos com a infraestrutura do NIALM e identificar a opção mais adequada em termos de custo-benefício, levando em consideração os diferentes tipos de VMs e o uso do escalonador.

Para garantir um processamento sem atrasos, o tamanho da VM escolhido é capaz de lidar com a carga simulada de forma eficiente e sem atrasos. Nos casos dos planos com preço constante, é possível calcular o custo diário das instâncias multiplicando o custo horário pelo número de horas em um dia. Dessa forma, obtém-se o valor total gasto diariamente.

No entanto, no caso da estimativa com o uso de máquinas oportunistas sem o escalonador, o cálculo da despesa diária é diferente. Para cada valor registrado ao longo do dia, ele é multiplicado pelo tempo em que esteve em vigência. Em seguida, todos esses produtos são somados ao longo do dia, resultando na despesa diária total.

Por fim, para calcular o custo com o uso do escalonador, foi adotada uma abordagem mais realista, utilizando dados sintéticos para simular a carga da aplicação. Os trabalhos a serem processados foram gerados segundo a segundo. O preço considerado aceitável pelo SpotSaver foi definido como 65% do preço da instância no plano sob demanda. Além disso, a tolerância máxima de atraso no processamento foi definida como 3 horas. A cada ciclo de 10 minutos, foi verificada a decisão que o escalonador tomaria com base no preço da instância e no atraso da carga da aplicação. Em cada ciclo, foram registrados o número de VMs que seriam utilizadas, juntamente com o preço vigente. Com esses dados, o custo diário foi calculado somando os valores que seriam gastos em cada ciclo ao longo do dia. Essa abordagem permitiu avaliar de forma mais precisa o custo com o uso do escalonador, considerando a dinâmica dos preços das instâncias e o atraso tolerável no processamento dos dados.

Como o objetivo é avaliar o nível de economia gerado pelo escalonador, os custos calculados para instâncias reservadas e oportunistas (com e sem o uso do escalonador) foram normalizados pelo custo do uso de VMs sob demanda.

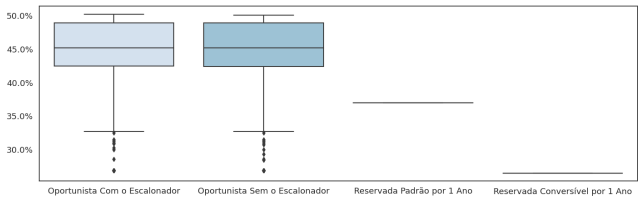

Figura 5: Economia diária em relação ao uso de instâncias no plano sob demanda

A figura 5 ilustra o percentual da economia diária obtida ao utilizar diferentes planos de instâncias em relação ao preço da VM sob demanda. Para as instâncias reservadas, que possuem um preço fixo ao longo do ano, é observado um único valor de economia diária. Para as máquinas reservadas padrões, a economia é de aproximadamente 37%, enquanto no plano conversível os descontos em relação ao plano sob demanda chegaram a um valor próximo de 26% (esta se diferencia do plano padrão por disponibilizar a possibilidade de troca da família da VM, do sistema operacional e opção de pagamento das instâncias).

Em média, o custo diário de uso de instâncias oportunistas (seja com ou sem o uso do escalonador) é consideravelmente menor do que o uso de VMs reservadas. Os gastos com instâncias *Spot*, tanto com quanto sem o escalonador, são praticamente iguais, com a maioria dos valores de economia diária variando entre 42% e 48%, mas podendo chegar a até 50%. No entanto, é importante ressaltar que o uso exclusivo das máquinas oportunistas não garante a disponibilidade desejada, como por exemplo em alguns momentosas VMs *Spot* podem não estar disponíveis, podendo assim ferir os termos de QoS da aplicação escalonada, sendo esse problema contornado pelo uso do escalonador.

6. CONCLUSÃO E TRABALHOS FUTUROS

O uso de instâncias oportunistas apresenta desafios e restrições em relação à qualidade de serviço que uma aplicação hospedada pode oferecer. No entanto, por meio da utilização do escalonador proposto, foi possível mitigar essas restrições, permitindo que aplicações com processamento em tempo quase real se beneficiem dos descontos oferecidos por essas instâncias, sem comprometer os termos de QoS.

Além disso, os resultados obtidos durante a simulação revelam uma economia significativa de até 50% em comparação com o uso de instâncias no plano sob demanda. Esses resultados destacam a eficácia do escalonador em reduzir os custos com recursos computacionais, tornando o uso de instâncias oportunistas uma opção viável para aplicações sensíveis à instabilidade da infraestrutura.

A solução apresentada, ainda possui algumas limitações, em especial quando se trata de aplicações que oferecem diferentes níveis de qualidade de serviços a seus contratantes, ou seja, diferentes limites máximos de atraso. Outro ponto de melhoria para o escalonador é uma evolução no módulo SpotSaver, para torná-lo capaz de usar alguma inteligência computacional para prever as oscilações dos preços cobrados nas instâncias

oportunistas e assim ser capaz de conseguir aproveitar ainda mais os baixos preços cobrados pelas VMs oportunista..

7. REFERENCES

[1] Ardi, C. (2022). Amazon EC2 Spot Price History: 2014-2015, 2017-2021. https://ant.isi.edu/~calvin/data/ec2-spot-price/.

[2] Carrie Armel, K., Gupta, A., Shrimali, G., and Albert, A. (2013) Is disaggregation the holy grail of energy efficiency? The case of electricity. Energy Policy, 52:213-234. Special Section: Transition Pathways to a Low Carbon Economy.

[3] Gama, D. A. (2022). Desagregação distribuída: evolução arquitetural do desagregador nialm da liteme. http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/29241.

[4] Hart, G. (1992). Nonintrusive appliance load monitoring. Proceedings of the IEEE, 80(12):18870-1891.

[5] Kumar., D. P., Baranwal, G., Raza, Z., and Vidyarthi, D. P. (2017). A survey on spot pricing in cloud computing. Journal of Network and System Management, 26:809-856.

[6] LiteMe (2022). LiteMe — InteligÊncia Energética. https://liteme.com.br/about.

[7] Services, A. W. (2022). Instâncias spot do Amazon EC2. https://aws.amazon.com/pt/ec2/spot/.

[8] Tamrakar, K., Yazidi, A., and Haugerud, H. (2017). Cost Efficient Batch Processing in Amazon Cloud with Deadline Awareness. In 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), pages 963-971.