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

VINICIUS BARBOSA DA SILVA

ARREBOL: SISTEMA ELÁSTICO DE PROCESSAMENTO EM LOTE DE APLICAÇÕES SINGLE-NODE

CAMPINA GRANDE - PB

2022

VINICIUS BARBOSA DA SILVA

ARREBOL: SISTEMA ELÁSTICO DE PROCESSAMENTO EM LOTE DE APLICAÇÕES SINGLE-NODE

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 : Francisco Vilar Brasileiro

CAMPINA GRANDE - PB

2022

VINICIUS BARBOSA DA SILVA

ARREBOL: SISTEMA ELÁSTICO DE PROCESSAMENTO EM LOTE DE APLICAÇÕES SINGLE-NODE

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:

Francisco Vilar Brasileiro

Orientador e Professor da Disciplina TCC – UASC/CEEI/UFCG

Dalton Dario Serey Guerrero

Examinador – UASC/CEEI/UFCG

Reinaldo Cézar de Morais Gomes

Examinador – UASC/CEEI/UFCG

Trabalho aprovado em: 02 de Setembro de 2022.

CAMPINA GRANDE - PB

ABSTRACT

With the advancement of computers and their increasing use in many different areas, research that uses computing intensively to obtain results, also known as e-researches, have become increasingly present in academia and in companies that need to run applications that process large amounts of data. For a long time, the most common environment for running these applications was the supercomputers, which are infrastructures with high installation and maintenance costs and are mostly present in specific locations, such as large companies and academic institutions. Therefore, this type of infrastructure is not always accessible to all researchers. Supercomputers have previously allocated computational resources (CPU, RAM, storage) that if not properly used, either because of an inefficient scheduling policy or low demand, end up being wasted. With this in mind, we propose Arrebol: a cloud-based batch processing system that takes advantage of the elasticity of this environment and allocates resources on demand. Arrebol monitors the cluster's workload and tries to balance it with the available resources. Whenever possible, it tries to optimize resource utilization by keeping it as small as possible. The present work continues the Arrebol development by implementing the resource management server (Resource Manager).

Arrebol - Sistema elástico de processamento em lote de aplicações *single-node*

Aluno: Vinicius Barbosa da Silva

Orientador: Francisco Vilar Brasileiro

Unidade Acadêmica de Sistemas e Computação, Universidade Federal de Campina Grande - UFCG
R. Aprígio Veloso, 882 - Universitário, Campina Grande - PB, 58428-830, Brasil

4 de setembro de 2022

Resumo: Com o avanço dos computadores e seu crescente uso nas mais diferentes áreas, pesquisas que usam computação de forma intensa para obter resultados, também conhecidas por e-researches, vêm se tornando cada vez mais presentes na academia e em empresas que necessitam executar aplicações que processam grande quantidade de dados. Durante muito tempo, o ambiente mais comum para execução dessas aplicações eram os supercomputadores, que são infraestruturas de elevado custo de instalação e manutenção e que em sua maioria estão presentes em locais específicos, como grandes empresas e instituições acadêmicas. Portanto, esse tipo de infraestrutura nem sempre é acessível para todos os pesquisadores. Os supercomputadores dispõem de recursos computacionais (CPU, RAM, armazenamento) previamente alocados que se não forem usados adequadamente, seja por uma política de escalonamento ineficaz ou por uma baixa demanda, acabam sendo desperdiçados. Com isso em mente, propomos o Arrebol: um sistema de processamento em lote baseado em nuvem, que tira proveito da elasticidade desse ambiente e aloca recursos sob demanda. O Arrebol monitora a carga de trabalho do cluster e procura equilibrá-la com os recursos disponíveis. Sempre que possível, tenta otimizar a utilização de recursos mantendo a menor possível. O presente trabalho dá continuação ao desenvolvimento Arrebol, implementando o servidor de gerenciamento de recursos (Resource Manager).

Palavras chave: Computação na nuvem, sistemas em lote, jobs, escalonador, gerenciamento de recursos.

Repositórios:
https://github.com/viniciusbds/arrebol-pb
https://github.com/viniciusbds/arrebol-pb-worker
https://github.com/viniciusbds/arrebol-pb-resource-manager

1 Introdução

Uma aplicação em lote (ou *job*) é um grupo de tarefas com um conjunto de entradas que podem ser executadas sem interação com o usuário [1]. Como essas aplicações geralmente não têm a necessidade de serem executadas instantaneamente, o gerenciamento e escalonamento dos jobs pode ser facilitado, uma vez que é possível escolher o melhor momento para executá-los. Uma maneira comum de implementar um sistema em lote é usar uma fila para armazenar os jobs que chegam no sistema. À medida em que recursos do sistema são liberados, a fila de jobs é consumida via alguma política de escalonamento, como por exemplo, a *First-In-First-Out* (FIFO). Essa abordagem requer alguns cuidados que podem impactar no tempo de espera das aplicações, por exemplo: se os recursos forem insuficientes, o tempo de espera dos jobs pode crescer bastante. Por outro lado, se os recursos forem muito superiores à demanda, ocorrerá desperdício.

Os supercomputadores, por boa parte do tempo, foram os ambientes mais comuns para execução desse tipo de aplicação. O seu planejamento de capacidade é feito antecipadamente durante sua idealização, sendo definida uma capacidade máxima que não pode ser ultrapassada sem a adição de mais recursos computacionais. Por outro lado, ambientes baseados em computação na nuvem provêm recursos sob demanda, tornando o planejamento de capacidade dinâmico e passível de ser modificado ao longo do tempo. Com

isso em mente, Smaneoto *et. al*. desenvolveram o Arrebol [1], um sistema de processamento em lote que ao invés de usar uma infraestrutura fixa como os supercomputadores, usa a nuvem e com sua elasticidade aumenta ou diminui os recursos consumidos conforme a demanda oscila. O sistema tenta acompanhar a carga atual no servidor principal, a fim de prover disponibilidade sem desperdiçar recursos.

O Arrebol foi projetado para possuir três componentes principais: o Servidor Principal, os Workers e o Resource Manager. Para ser fácil de integrar com os clientes, conta com interfaces RESTful para fornecer acesso ao núcleo do sistema. Além disso, o servidor principal funciona usando um modelo de comunicação passiva, apenas respondendo requisições e não iniciando. Dessa maneira ele não precisa salvar o estado dos demais componentes e funciona de maneira independente.

1.1 Problema e Solução

Apesar dos supercomputadores serem úteis para aplicações que requerem alto desempenho, processamento paralelo de dados e uma rede de alta velocidade, eles podem não ser a melhor opção de infraestrutura para alguns tipos de aplicações de processamento em lote. Esse é o caso, por exemplo, de jobs do tipo single-node, que com apenas um nó conseguem realizar seu trabalho em tempo hábil sem necessidade de usar a rede de alta velocidade para trafegar dados de um nó para outro. Ao usar supercomputadores para executar esse tipo de aplicação, desperdiça-se recursos de rede da infraestrutura.

Além disso, os recursos computacionais desse tipo de infraestrutura são fixos e, uma vez montados, se não forem utilizados em sua capacidade máxima, ocorrerão desperdícios. Ou ainda, caso a demanda seja maior que a capacidade, não é tão simples e barato escalar o sistema para suportar a execução de todos os jobs. Por outro lado, usar recursos da nuvem como infraestrutura aparenta ser uma boa tentativa, uma vez que esse ambiente é flexível e pode ser dimensionado conforme a demanda. Como as aplicações consideradas neste trabalho são single-node, a utilização de uma rede de menor desempenho nessa infraestrutura não acarreta problemas. Porém, essa solução requer alguns cuidados. Em particular, é preciso escolher os melhores planos de oferta de serviços, a fim de minimizar o custo, além de fazer a alocação e liberação dinâmica de recursos para minimizar o tempo de espera em fila, ao mesmo tempo que se maximiza a utilização dos recursos alocados.

Atualmente, o Arrebol possui algumas limitações, como a inexistência do serviço Resource Manager e a ausência de uma estratégia mais elaborada para a escolha dos planos de oferta de serviço dos provedores de computação na nuvem (*Resource Provider*). Para resolver o primeiro problema, esse trabalho propõe uma implementação do componente de gerenciamento de recursos, o Resource Manager. Esse componente será o responsável por alocar recursos de nuvem sob demanda, buscando o equilíbrio entre a carga atual (jobs) e a quantidade de recursos.

2 Arquitetura e Projeto da Solução

Essa seção apresenta uma visão geral da arquitetura do Arrebol, detalha a arquitetura do componente Resource Manager, bem como seus protocolos de comunicação. Por fim, são mencionadas as tecnologias usadas para desenvolver a solução.

2.1 Arquitetura Geral

O Arrebol é um sistema de processamento em lote para aplicações em lote single-node que se aproveita da natureza de elasticidade da nuvem para alocar recursos sob demanda. Ele suporta a criação e o gerenciamento de múltiplas filas de jobs, permitindo assim uma divisão semântica no qual podem ser consideradas características dos jobs bem como uma política de escalonamento individual para cada fila. Sua arquitetura conta com três partes principais: um **servidor principal**, um conjunto de **workers** e um **ResourceManager** (RM).

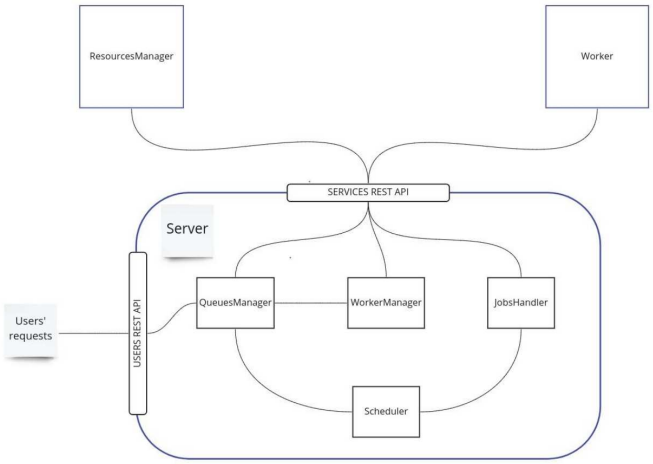

Figura 1: *Arquitetura do Arrebol*

O servidor principal é o coração do sistema, ele expõe uma API REST que atende as requisições dos Workers e do RS. Essa comunicação é feita de maneira unidirecional, de forma que o servidor principal apenas recebe e responde requisições das outras partes do sistema. Isso foi decidido com o objetivo de deixar o servidor principal independente dos demais. Os Workers são os responsáveis pela a execução das tarefas e cadastro dos seus resultados. Cada worker está associado a apenas uma fila, mas uma fila pode ter mais de um worker associado.

O RM é o componente responsável por controlar o número de workers no sistema, os adicionando e removendo conforme a demanda. Ele se comunica em um intervalo de tempo definido com o servidor principal para obter um resumo do estado geral do sistema (**State Summary**). Este resumo contém informações sobre a workload atual e o número de workers disponíveis por fila. Com o State Summary em mãos o RM calcula e define um número ideal de Workers para cada fila, se o número de Workers atual for menor do que o ideal é disparado um evento de criação de workers, caso seja maior, o RM dispara o evento de remoção de workers. A remoção de um Worker não é instantânea. Para evitar uma possível remoção e recriação precoce, o Worker só é removido de fato após um tempo mínimo de espera.

Tabela 1: *Exemplo de um State Summary*

ID da fila	n° workers associados	n° tarefas prontas para rodar
1	3	9
2	1	2
3	2	6

O RM conta com um ou mais provedores de recursos que fornecem máquinas virtuais (nós) para a execução dos Workers. Cada nó pode alocar um ou mais Workers, caso tenha recursos suficientes para atendê-los. Portanto, na criação de um novo Worker, o RM tenta alocá-lo em um dos nós disponíveis, requisitando a criação de um novo nó ao Resource Provider apenas se não houver recursos disponíveis.

A Figura 2 ilustra um exemplo de um nó com 20 cores de CPU e 32 Gbytes RAM, sendo ocupado por três Workers, totalizando juntos um consumo de 10.000 milicores de CPU e 17 Gbytes de RAM. Esse nó, portanto, ainda seria capaz de hospedar um ou mais Workers com os recursos restantes de 10000 milicores de CPU e 15 Gbytes de RAM.

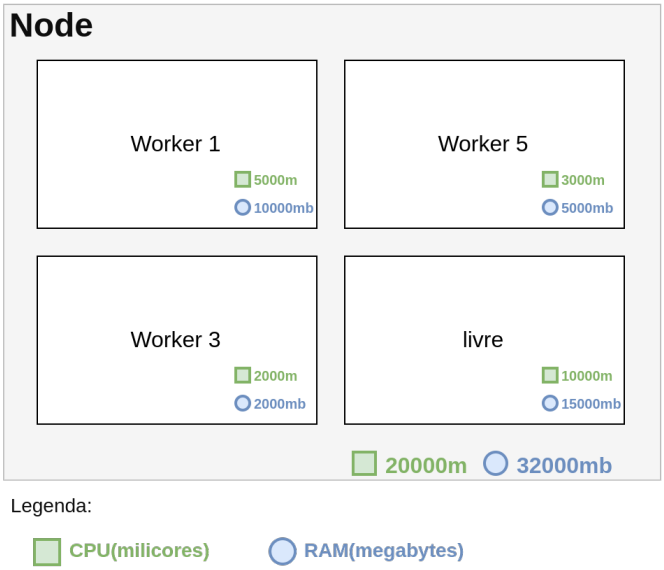

Figura 2: *Exemplo detalhado de um nó.*

Após a consolidação da remoção de um worker de um nó, o Resource Manager verifica a possibilidade de hospedar todos os workers restantes desse nó em outros nós, com objetivo de otimizar o uso de recursos. A próxima seção detalha a arquitetura interna do componente desenvolvido neste trabalho.

2.2 Resource Manager

A Figura 3 ilustra a arquitetura do ResourceManager, que é composto internamente pelos submódulos: Autoscaler, StateManager e Work Launcher e os módulos externos Resource Provider e servidor principal.

2.2.1 Autoscaler

Componente principal do Resource Manager, verifica em um intervalo de tempo definido um possível desbalanceamento entre os recursos disponíveis e a carga atual do sistema. A carga atual do sistema é definida e retornada pelo servidor principal por meio de um State Summary com informações detalhadas de cada fila: número de tarefas prontas para rodar e número de workers por fila. Com o State Summary em mãos, o Autoscaler verifica a necessidade de adicionar ou remover recursos usando uma **política de autoscale**.

Se a carga atual de uma determinada fila for superior à quantidade de workers disponíveis para ela, o componente solicita a criação de um ou mais workers. Caso o Autoscaler não consiga alocar o novo worker

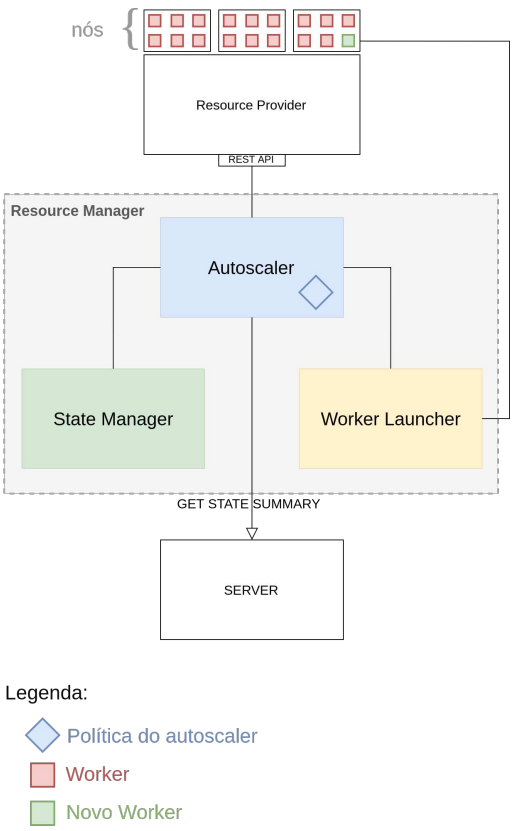

Figura 3: *Arquitetura do Resource Manager.*

em um dos nós disponíveis, ele solicita a criação de mais um nó ao provedor de recursos. Essa aquisição é feita de forma preguiçosa (lazy acquisition [3]), a fim de otimizar o uso de recursos e requisitar apenas as máquinas virtuais quando necessário. Por outro lado, se a carga atual de uma fila for inferior a sua quantidade de workers, o RM solicita a remoção dos excedentes e verifica a possibilidade de otimizar os recursos nos nós dos workers removidos. Como os Workers são stateless, após a remoção de um Worker de um nó, o RM verifica se outros workers desse mesmo nó podem ser removidos e criados em outros, a fim de manter a menor quantidade de recursos em uso possível.

2.2.2 Política de Autoscale

A política de autoscale decide o número ideal de Workers por fila. A criação de uma política de Autoscale mais elaborada e otimizada está fora do escopo deste trabalho. Por padrão usaremos a política mais simples que é de criar os Workers sob demanda e removê-los caso fiquem ociosos por um determinado período de tempo. Essa criação é n para n, ou seja: serão criados n workers para n tarefas.

2.2.3 State Manager

O State Manager é a entidade que registra quais recursos estão disponíveis e em qual nó cada Worker está alocado, armazenando as informações, assim como o servidor principal, em memória.

2.2.4 Worker Launcher (WL)

É o responsável por inicializar os workers nos nós selecionados pelo Autoscaler, isso inclui a cópia do código fonte e arquivos de configuração bem como a sua inicialização.

2.3 Protocolos

Essa seção descreve os protocolos de comunicação do RM com os demais componentes do Arrebol e com o provedor de recursos. Os protocolos incluem: balance checking protocol, resource creation protocol e removal resource protocol. Em todos os três protocolos as requisições são iniciadas pelo RM e contam com uma etapa de segurança para garantir a autenticidade. Todas as requisições são feitas com uma mensagem escrita com a chave privada do RM e o servidor principal é capaz de validar essa mensagem, já que possui a chave pública do RM configurada em tempo de implantação. Uma vez autenticado, as requisições são liberadas.

2.3.1 Balance Checking Protocol

Esse protocolo descreve como o RM verifica o balanceamento entre a carga atual do sistema (número de tarefas prontas para rodar) e os recursos computacionais. Em uma frequência definida, o RM checa esse balanceamento e aciona o processo de criação de recursos caso exista mais carga do que recursos disponíveis, ou o processo de remoção de recursos, quando a carga seja menor do que os recursos. Esse protocolo possui duas etapas principais:

1. O RM solicita ao servidor principal o estado do sistema, que é retornado por meio de um State Summary (Tabela 1.1). O State Summary contém informações como o número de tarefas prontas para rodar por fila, bem como o número de workers a ela associados. O número de workers associados a uma fila é a soma dos workers que já estão em execução e os que estão em processo de criação.

2. Com o State Summary em mãos, o Resource Manager verifica um possível desbalanceamento entre a carga das filas e os recursos disponíveis. Nesse

cenário, existem três possibilidades. Na primeira, o RM identifica que o número de tarefas prontas para rodar na fila é maior que o número de workers associados a ela. Nesse cenário é ativado o processo de criação de recursos. Uma segunda possibilidade é que o número de workers associados a uma fila seja superior à quantidade de atividades prontas para rodar. Nesse segundo caso, o Resource Manager ativa o processo de remoção de recursos. No terceiro e último cenário, o RM verifica que o número de workers associados é igual ao número de tarefas prontas para rodar. Nesse caso, nenhum evento é iniciado.

O diagrama de fluxo (Figura 4) resume o protocolo apresentado

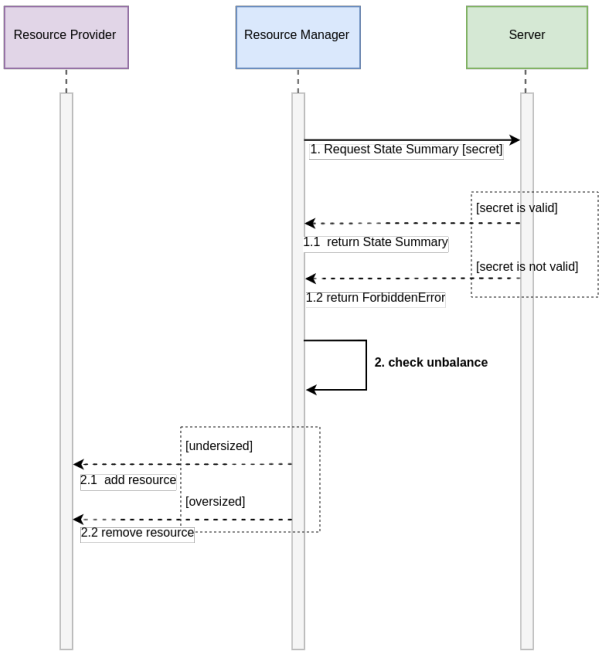

Figura 4: *Diagrama de fluxo do protocolo de checagem de balanceamento*

2.3.2 Resource Creation Protocol

Esse protocolo define como o RM adiciona novos workers no sistema. A fim de otimizar recursos do Resource Provider, o RM aloca um ou mais workers em um mesmo nó sempre que é possível. Por fins de segurança, o servidor principal gera, em tempo de implantação, uma lista de workerIDs disponíveis para criação, que são consumidos pelo RM neste protocolo. Podemos dividi-lo em três etapas:

1. O RM requisita ao servidor principal o próximo workerID disponível da lista. Como foi citado anteriormente, essa requisição é autenticada através de uma mensagem criptografada com a chave privada do RM, dessa forma o servidor principal, que possui a chave pública do RM, consegue validar a mensagem e assim retornar o ID.

2. O RM com o workerID em mãos, verifica a possibilidade de alocá-lo em um dos nós existentes. Caso não seja possível, solicita ao Resource Provider a criação de mais um nó para hospedar o novo worker.

3. Inicialização do Worker. Com o recurso pronto para uso, o Resource Manager envia os scripts de inicialização e um arquivo de configuração para o nó provisionado na etapa anterior e os executa. Com isso, temos o worker pronto para receber tarefas.

O diagrama de fluxo (Figura 5) resume o protocolo apresentado:

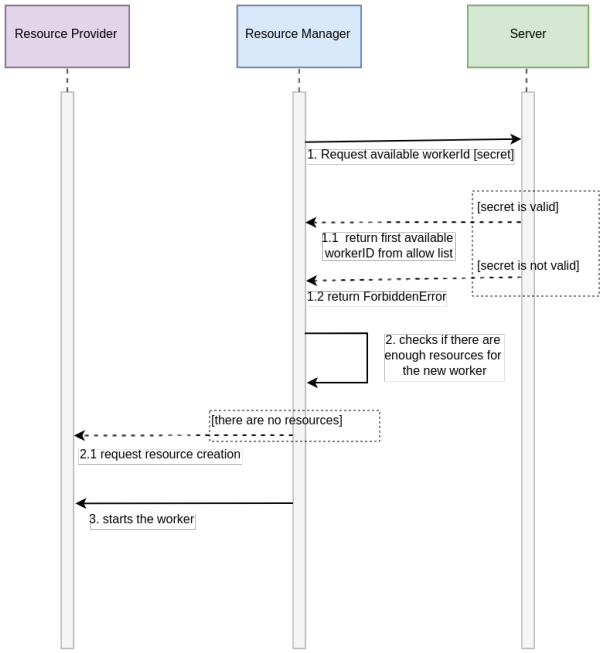

Figura 5: *Diagrama de fluxo do protocolo de criação de recursos*

2.3.3 Resource Removal Protocol

Uma vez que o RM verifica que há mais workers do que tarefas prontas para rodar em uma fila, ele dispara o evento de remoção. Essa remoção só é realizada de fato após um período mínimo de espera. O protocolo de remoção descreve como um worker é removido do sistema. As etapas são as seguintes:

1. O RM solicita a remoção do worker ao servidor principal, que após a autenticação, remove o estado do worker e o desconsidera.

2. O RM interrompe o processo do worker.

3. O RM verifica a possibilidade de remover outros workers do mesmo nó e criá-los em outros nós, a fim de manter o menor número de nós em uso possível.

O diagrama de fluxo (Figura 6) resume o protocolo apresentado:

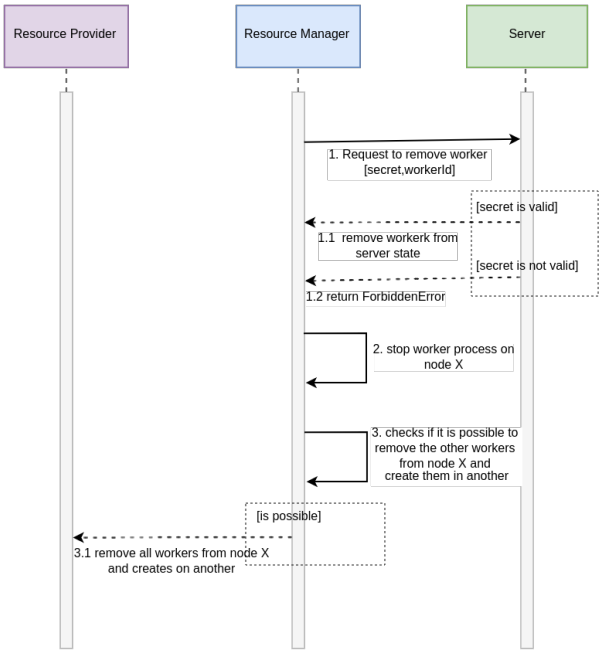

Figura 6: *Diagrama de fluxo do protocolo de remoção de recursos*

2.4 Tecnologias

Para o desenvolvimento do gerenciador de recursos, usamos a linguagem de programação Golang [2] e a plataforma de conteinerização Docker. As razões pela escolha da linguagem Golang foram, primeiramente, manter o padrão com os demais componentes do sistema escritos em Go. O segundo motivo se deu pelo fato da linguagem facilitar a comunicação entre processos por meio de canais. Esse tipo de comunicação facilitou a manipulação dos processos (Goroutines) dos workers inicializados pelo Worker Launcher. Usamos canais para realizar a comunicação dos processos de execução dos Workers, a fim de poder matá-los facilmente quando preciso.

O foco deste trabalho foi o desenvolvimento do Resource Manager, portanto para facilitar esse processo, usamos a ferramenta VirtualBox como Resource Provider. A solução implementada faz uso de interfaces, para facilitar a troca do Resource Provider para um ambiente na nuvem real no futuro.

3 Experiência e Lições Aprendidas

Essa seção discutirá lições aprendidas no decorrer do desenvolvimento do gerenciador de recursos Arrebol Resource Manager, desafios e limitações da solução e por fim possíveis trabalhos futuros.

3.1 Processo de Desenvolvimento

O processo de desenvolvimento consistiu em ciclos de desenvolvimento durante 15 semanas, com a participação em reuniões com os orientadores em algumas dessas semanas. Para desenvolver a solução, foram utilizadas as ferramentas: golang, Docker, VS-Code e o sistema operacional linux Ubuntu 20.04.

3.2 Principais desafios

Um dos desafios que atrasou um pouco o desenvolvimento nas semanas iniciais foi devido a pouca documentação existente no sistema Arrebol. Com isso, foi gasto um bom tempo para configurar as variáveis de ambiente, integrar branches e corrigir bugs de autenticação. Para evitar esse problema para trabalhos futuros, incluímos uma documentação detalhada do que é preciso e como executar o servidor principal e o Resource Manager.

3.3 Limitações

Atualmente o Resource Manager não realiza a otimização de um nó quando um worker é removido, como é sugerido nos protocolos. Essa otimização não foi implementada por que alguns cuidados precisam ser tomados antes de simplesmente parar o processo dos workers de um nó e criá-los em outro. Podemos imaginar, por exemplo, um cenário no qual uma atividade de longa duração está executando e está próxima de sua finalização, nesse caso talvez não seria viável interromper essa atividade e iniciá-la do zero em outro nó somente para manter um número mínimo de nós em uso.

3.4 Trabalhos futuros

O foco deste trabalho foi desenvolver o Resource Manager. Apesar de citarmos o provedor de recursos (Resource Provider) em alguns momentos, pouco foi desenvolvido sobre ele. Antes de fornecer recursos para o Resource Manager, o Resource Provider precisa escolher entre um ou mais planos de computação na nuvem. Atualmente existem algumas classes de planos, as principais são as instâncias sob demanda, instâncias reservadas e instâncias spot.

A primeira classe, sob demanda, compõe instâncias que permitem pagar a capacidade computacional por tempo consumido, sem compromissos de longo prazo. Dessa maneira, o cliente fica livre dos custos e complexidades de planejamento, aquisição e manutenção de hardware [5]. O lado ruim dessa solução é o preço que geralmente é mais elevado que os demais. A segunda classe, a de instâncias reservadas, por sua vez, compõem as instâncias de longo prazo. Nesse tipo de serviço o cliente se compromete a reservar um número de instâncias por um período de vigência de um ou de três anos [4]. A vantagem dessa solução é a redução do preço comparado com o plano sob demanda. Por fim, a classe spot são de instâncias ociosas na nuvem que são oferecidas aos clientes com um custo muito reduzido. [6] estima que é possível conseguir descontos entre 70% e 90% escolhendo uma instância spot comparando com uma instância sob demanda. O lado negativo para algumas situações é o fato de que nesse tipo de classe, não há nenhuma garantia que o cliente tenha esse recurso 100% do tempo, dessa forma o provedor pode recuperar esse recurso a qualquer momento.

Dito isto, um possível trabalho futuro seria um estudo aprofundado acerca de quais os melhores planos atenderiam o Arrebol. Esse estudo poderia ser feito através do monitoramento do uso de aplicações batch em exemplos reais.

Outro trabalho futuro possível seria o desenvolvimento de uma política de autoscale mais elaborada, que estime o número ideal de Workers por fila, possivelmente levando em conta cargas de trabalho antigas.

4 Agradecimentos

Agradeço primeiramente aos meus pais, José Valdomiro e Célia Maria Barbosa e à minha irmã Sabrina Barbosa pela confiança e apoio durante a minha caminhada. Agradeço a todos os professores por oferecerem uma formação acadêmica de qualidade, especialmente aos meus orientadores, Francisco Brasileiro e Thiago Emmanuel, que muito contribuíram para minha formação, através deste trabalho e dos três anos que passei no Laboratório de Sistemas Distribuídos. Por fim, agradeço aos meus amigos, Rich Elton, Samuel Vasconcelos, Igor Silveira, Matheus Santana, Yuri Souza, Lucas Andrade, José Davi e Levi Gomes, pela parceria durante a graduação.

5 Referências

[1] R. M. Smaneoto, T. Emmanuel Pereira and F. V. Brasileiro, "A Cloud-Based Batch Processing System for Loosely-Coupled Applications,"2021 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW), 2021, pp. 47-52, doi: 10.1109/SBAC-PADW53941.2021.00018.

[2] Russ Cox, Robert Griesemer, Rob Pike, Ian Lance Taylor, and Ken Thompson. 2022. The Go programming language and environment. Commun. ACM 65, 5 (May 2022), 70–78. https://doi.org/10.1145/3488716

[3] Kircher, Michael. (2001). Lazy Acquisition.

[4] Opções de compra de instância Amazon AWS https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGu purchasing-options.html

[5] Definição de preço sob demanda https://aws.amazon.com/pt/ec2/pricing/on-demand

[6] Reduza seus custos em EC2 com instâncias spot https://cleancloud.io/reduza-seus-custos-em-ec2-com-instancias-spot