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

VÉLMER OLIVEIRA ODON

SUPORTE PARA COMPARTILHAMENTO DE DADOS EM UM SERVIÇO DE PROCESSAMENTO DE IMAGENS DE SATÉLITE

CAMPINA GRANDE - PB

2019

VÉLMER OLIVEIRA ODON

SUPORTE PARA COMPARTILHAMENTO DE DADOS EM UM SERVIÇO DE PROCESSAMENTO DE IMAGENS DE SATÉLITE

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. Thiago Emmanuel Pereira da Cunha Silva.

CAMPINA GRANDE - PB

2019

O26s Odon, Vélmer Oliveira.
Suporte para compartilhamento de dados em um serviço de processamento de imagens de satélite. / Vélmer Oliveira Odon. – 2021.

9 f.

Orientador: Prof. Dr. Thiago Emmanuel Pereira da Cunha Silva.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Sistemas distribuídos. 2. E-ciência. 3. Processamento de imagem de satélite. 4. Satélite – serviço de processamento de imagem. 5. SAPS – Serviço de processamento de imagens de satélites. 6. Graphical User Interface – GUI. I. Silva, Thiago Emmanuel Pereira da Cunha. II. Título.

CDU:004.4(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

VÉLMER OLIVEIRA ODON

SUPORTE PARA COMPARTILHAMENTO DE DADOS EM UM SERVIÇO DE PROCESSAMENTO DE IMAGENS DE SATÉLITE

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. Thiago Emmanuel Pereira da Cunha Silva
Orientador – UASC/CEEI/UFCG

Professor Dr. Adalberto Cajueiro de Farias
Examinador – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 02 de julho de 2019.

CAMPINA GRANDE - PB

Suporte para Compartilhamento de Dados em um Serviço de Processamento de Imagens de Satélite

Vélmer Oliveira
velmer.odon@ccc.ufcg.edu.br
Universidade Federal de Campina Grande

Thiago Emmanuel Pereira
temmanuel@computacao.ufcg.edu.br
Universidade Federal de Campina Grande

RESUMO

O SAPS é um serviço de processamento de imagens de satélite que demanda alto poder computacional. Por exemplo, um dos algoritmos disponibilizados na plataforma leva cerca de 90 minutos para processar uma única imagem. Tipicamente, uma instância desse serviço é operada e mantida por um departamento ou instituto de pesquisa. Como múltiplas instâncias do serviço podem co-existir, é possível que as mesmas imagens sejam processadas por instâncias diferentes (quando, por exemplo, usuários das instituições têm interesses de pesquisa coincidentes). Nesse sentido, há desperdício de recursos como processamento, memória e armazenamento. Caso um usuário de uma instância do serviço pudesse aproveitar os dados processados anteriormente em outra instância, além de reduzir o desperdício de recurso, seu tempo de espera seria bastante reduzido. Para que tal aproveitamento de dados ocorra é necessário que, antes de processar uma submissão feita por um usuário, uma instância se comunique com outras a fim de saber se essas já processaram tal submissão. Neste trabalho, nós apresentamos a arquitetura, implementação e a prova de conceito do mecanismo de comunicação entre diferentes instâncias do SAPS, que permite o compartilhamento de dados entre as mesmas.

KEYWORDS

Sistemas distribuídos, e-ciência

1 INTRODUÇÃO

O SAPS é um serviço de processamento de imagens de satélite que demanda alto poder computacional. Por exemplo, considerando a execuao do algoritmo SEBAL [3, 4] em uma uma única imagem do satélite Landsat 5, de dimensões 170km x 185km, com 30 metros de resolução, a execução de um dos algoritmos disponibilizados no SAPS dura, em média, 90 minutos em uma máquina com um processador de 8 núcleos e 32GB de memória física. Ainda, o tamanho da saída gerada é da ordem de 1.8GB [2]. Tais valores de processamento e de armazenamento são altos, portanto, uma estratégia para maximizar o bom uso de recursos é essencial.

Tipicamente, uma instância do SAPS é administrada por um grupo de pesquisa. Nesse sentido, é possível que em contexto regional, no nordeste brasileiro, por exemplo, existam diversas instâncias em execução, cada uma administrada por uma universidade da região. Com isso, é possível que submissões redundantes, ou seja, que correspondem a mesma imagem de satélite, usando o mesmo algoritmo, sejam submetidas pelos usuários das várias instâncias da região.

Nosso trabalho visa integrar diferentes instâncias do SAPS, para habilitar o reuso dos recursos de processamento e armazenamento através do compartilhamento de resultados de submissões previamente processadas. Esse compartilhamento é possível porque a execução dos algoritmos de processamento de imagens é determinística, ou seja, os resultados serão iguais sempre que executados com os mesmos parâmetros de entrada. Estes parâmetros incluem a região geográfica que se quer processar, o intervalo de datas considerado para processamento e os algoritmos que serão empregados no processamento (dentre um conjunto de opções disponíveis).

Neste artigo nós apresentamos a arquitetura, implementação e a avaliação do mecanismo de comunicação entre diferentes instâncias do SAPS. Tal mecanismo provê uma maneira de definir, de forma independente em cada uma instâncias, uma rede de outras instâncias para as quais se pode requisitar dados. Dado que uma instância configurou sua rede de colaboração, antes de processar uma submissão localmente, a mesma deve descobrir quais vizinhas possuem dados que podem ser aproveitados referentes àquela submissão. Assim, o processamento local pode ser substituído pelo aproveitamento dos dados já processados.

Porém, considerando os dados de uma única imagem, seria preciso transferir algo em torno de 1.8GB, tornando a transferência custosa. Para viabilizar uma maneira efetiva de compartilhar esses dados, o mecanismo adota uma estratégia de transferência lógica. Ao invés de uma transferência física dos dados de uma instância para a outra, o mecanismo cria somente uma referência para os dados localizados em uma instância vizinha.

Com isso, nos resta definir como o usuário acessará os dados que foram aproveitados. Para acessar os dados localizados na própria instância, o usuário recebe uma URL para cada uma das imagens processadas. Assim, para a instância prover acesso a dados remotos, ela irá se comunicar com a instância que os gerou - através da referência aos dados - para obter a URL de acesso da imagens processadas por aquela. Nesse sentido, o mecanismo permite que o usuário

acesse os dados de forma transparente, sem a necessidade de saber em que instância os dados foram gerados.

O restante do artigo está estruturado da seguinte forma. Inicialmente, nós provemos uma fundamentação teórica com conceitos necessários para entendimento das seções posteriores (Seção 2). Em seguida, apresentamos a arquitetura da solução, discutindo as principais decisões arquiteturais (Seção 3). Posteriormente, apresentamos a implementação da solução através de uma prova de conceito (Seção 4). Seguidamente, apresentamos a avaliação da implementação (Seção 5). Por fim, compartilhamos as experiências adquiridas, detalhes sobre o processo de desenvolvimento, os desafios encontrados com suas respectivas soluções e o direcionamento para trabalhos futuros (Seção 6).

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção apresentamos os principais conceitos e informações fundamentais para o entendimento das seções posteriores.

Componentes de serviço e seu funcionamento

O SAPS tem sua arquitetura formada por componentes, cada um deles possuindo uma função bem definida. Cada um desses componentes é configurável no momento de sua implantação pelos seus responsáveis. Tais configurações definem como os componentes irão interagir entre si e com elementos externos. Os componentes que compõem a arquitetura do serviço, assim como suas interações, estão ilustrados na Figura 1.

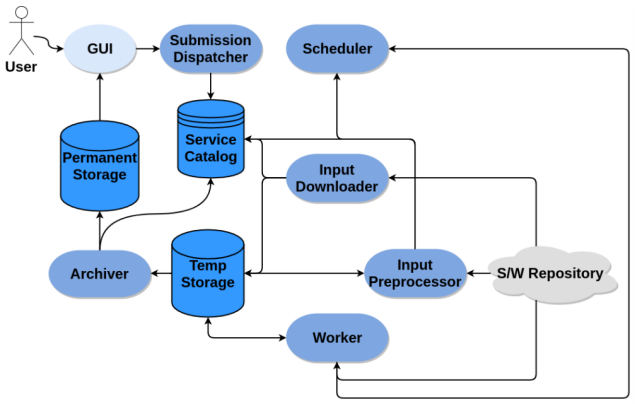

Figura 1: Principais componentes da arquitetura.

O usuário interage com o sistema através da Interface Gráfica do Usuário (Graphical User Interface - GUI), sendo esse componente o único que o usuário acessa diretamente, o qual serve como front-end para os demais componentes do sistema. Através da GUI, o usuário realiza submissões de processamento, fornecendo os parâmetros da mesma. Os parâmetros consistem da região geográfica que se quer processar, a qual pode ser formada por mais de uma imagem de satélite, o intervalo de tempo em que as imagens foram obtidas, assim como a fonte da imagem, o algoritmo de pré-processamento necessário (por exemplo, algoritmo para descartar imagens que tenham sofrido interferência pela presença de nuvens) e, por fim, o algoritmo que, de fato, processará a imagem de satélite.

Os parâmetros selecionados são submetidos, pela GUI, para o Submission Dispatcher através da sua interface REST. Esse componente tem a responsabilidade de validar os parâmetros de entrada e, caso sejam válidos, o mesmo irá criar uma entidade chamada Image Task para cada dia do intervalo de processamento. Essa entidade possui como principais atributos a data da imagem de satélite, a sua região e o satélite que a capturou. Uma Image Task possui um conjunto de estados bem definidos que a mesma pode assumir. Cada um deles representa em que fase do fluxo de execução aquela Image Task está, além disso, define qual componente deve atuar sobre a mesma naquele momento. A máquina de estados de uma Image Task é ilustrada na Figura 2.

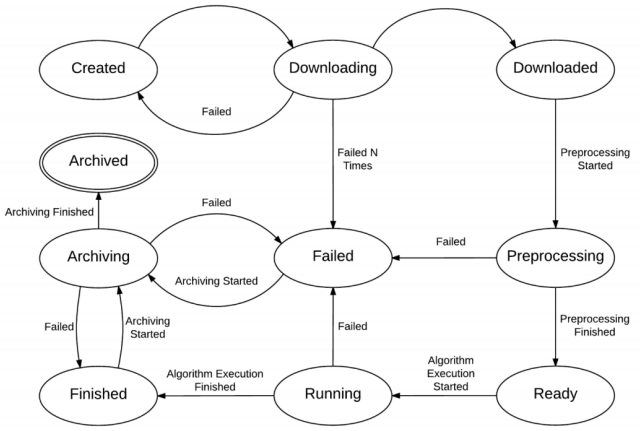

Figura 2: Máquina de estado de uma Image Task.

O Submission Dispatcher armazena Image Tasks no banco de dados referenciado como Service Catalog.

O Input Downloader realiza uma checagem por novas Image Tasks criadas, as quais assumem o estado Created. Encontrando, o Input Downloader inicia o download da imagem de satélite, atualizando o estado da Image Task para Downloading. Caso o download falhe, o estado é revertido para Created para que o download da imagem possa ser repetido uma quantidade fixa de vezes. Caso ultrapasse esse limite de tentativas, a Image Task assume o estado Failed e não será mais processada. Caso o download seja realizado com sucesso, a Image Task assume o estado Downloaded.

Similar ao Input Downloader, o Input Preprocessor busca no Service Catalog imagens que estão em um estado específico, nesse caso, aquelas que já foram baixadas, ou seja, que estão no estado Downloaded. Ao encontrar, o componente inicia a execução do algoritmo de pré-processamento. Diferente do Input Downloader, caso o algoritmo falhe uma única vez, a Image Task já tem seu estado alterado para Failed. Finalizando o pré-processamento com sucesso, o Input Preprocessor atualiza o estado da Image Task para Ready, o qual indica que a mesma está pronta para ser processada.

O processamento de uma imagem é realizado pelo Worker. Sua única responsabilidade é executar o algoritmo escolhido pelo usuário e armazenar os dados e metadados gerados no Temporary Storage. O componente Scheduler monitora o Service Catalog em busca de imagens a serem processadas. Em seguida, as encaminha para realização do processamento pelo Worker. Ao encaminhar uma Image Task para o Worker, o Scheduler atualiza o estado da mesma para Running e monitora toda a etapa de processamento, garantindo que em caso de falha a Image Task assuma o estado Failed e em caso de sucesso assuma o estado Finished.

Por fim, resta mover os dados e metadados do Temporary Storage para um ambiente mais confiável, o Permanent Storage. Essa é a responsabilidade do Archiver, o último componente do fluxo de processamento de uma imagem de satélite. Além disso, o mesmo deve atualizar a Image Task para Archived, o que representa que a imagem de satélite teve seu processamento finalizado, podendo ter sido bem-sucedido ou não.

Nesse momento, o usuário pode solicitar o acesso aos dados gerados pelo processamento. Esse acesso é realizado a partir de uma URL temporária gerada pelo SAPS.

3 ARQUITETURA DA SOLUÇÃO

Nesta seção discutimos quais decisões arquiteturais tomamos para garantir o funcionamento do mecanismo de compartilhamento de dados, além do impacto dessas mudanças na arquitetura do sistema. Primeiramente, definimos em que momento do fluxo de processamento o mecanismo irá atuar. Posteriormente, definimos como uma instância conhece suas vizinhas, formando uma rede de colaboração SAPS. Em seguida, apresentamos como é realizada a comunicação entre diferentes instâncias, assim como a identificação de submissões redundantes. Adiante, discutimos como os dados serão transferidos. Para finalizar as decisões arquiteturais, definimos como o usuário acessa os dados de processamentos realizados por outras instâncias. Por fim, discutimos quais mudanças são necessárias para que tais decisões sejam implementadas.

Decisões Arquiteturais

Inicialmente, é necessário definir quando o mecanismo irá atuar, ou seja, em qual ou quais etapas do fluxo de processamento o mecanismo executará. Como nossa solução visa evitar a duplicação de dados, nosso mecanismo deve intervir antes do fluxo de uma imagem alcançar o Input Downloader. Nesse sentido, é necessário impedir que o Submission Dispatcher processe qualquer submissão proveniente da GUI sem antes verificar se a mesma já foi processada por uma outra instância. Com isso, o mecanismo deve encaminhar ao Submission Dispatcher apenas a parcela da submissão que deve ser processada localmente, em outras palavras, a parcela que não foi processada por outras instâncias.

Porém, para nosso mecanismo verificar se uma submissão já foi processada é necessário que as instâncias do SAPS consigam se comunicar, e para isso uma instância precisa saber com quem ela pode se comunicar. Ou seja, é preciso configurar as instâncias que farão parte da rede de colaboração. é importante que essa configuração seja opcional, visto que o processamento pode ser realizado independente do compartilhamento de dados entre diferentes instâncias. Como essas funcionam de maneira independente, não é desejável utilizar um esquema de descoberta de vizinhos automática, visto que um grupo responsável por uma instância pode decidir não compartilhar seus processamentos. Decidimos adotar um modelo liberal, no qual o grupo responsável define quem são seus vizinhos, podendo ser, por exemplos, grupos de diferentes departamentos de uma mesma universidade ou de universidades distintas.

Para alcançar esse modelo, como nosso mecanismo atuará em conjunto do Submission Dispatcher, disponibilizamos um novo parâmetro de configuração no arquivo referente ao Submission Dispatcher, o qual armazena as URLs das instâncias vizinhas, formando a rede de colaboração. Como dito anteriormente, esse parâmetro é opcional, ou seja, essa rede pode ser vazia e ainda assim o fluxo de processamento será realizado.

Considerando que uma instância sabe com quem se comunicar, precisamos definir os dois elementos básicos de uma comunicação: pergunta e resposta. O propósito da comunicação é identificar se uma submissão é redundante. Nesse sentido, a instância que envia a pergunta deve enviar os parâmetros da submissão feito pelo usuário, para que a instância vizinha possa verificar se já processou ou não tais parâmetros. Como os resultados dos processamentos são determinísticos, podemos apenas utilizar um casamento de padrões entre os parâmetros recebidos com os parâmetros já processados. Assim, sabemos o que deve ser perguntado e como definir se a resposta será positiva ou negativa (a instância já processou e a instância não processou, respectivamente), assim, nos resta definir o que será enviado como resposta.

A instância que realiza a pergunta está interessada nos dados gerados pelo processamento da imagem, então seria interessante obter tais dados como resposta. Porém, transferir esses dados é inviável por dois motivos. Primeiro, transferir os dados significa realizar uma cópia dos dados de uma instância para outra, o que contrapõe o objetivo proposto, o qual é evitar a redundância e desperdício de recursos. Segundo, como discutido na Seção 1, o processamento gera muitos dados, então transferir essa quantidade de dados, entre sistemas que podem estar em redes diferentes, pode ser mais custoso e demorado do que gerá-los localmente, então a economia do tempo de processamento também seria descartada.

Nesse sentido, nossa estratégia é adicionar um nível de indireção para acessar tais dados, realizando uma transferência lógica dos mesmos. Ao invés da instância copiar os dados processados por outra, ela irá armazenar um apontador para esses. Esse indica que há dados gerados por um processamento de uma imagem de satélite que estão localizados em outra instância. Como discutido na Seção 2.1, uma Image Task é uma entidade que representa o registro de uma imagem de satélite sendo processada, a qual é associada aos dados gerados por tal processamento. Em outras palavras, podemos definir uma Image Task como um apontador para os dados do processamento de uma imagem. Nesse sentido, nossa estratégia usufrui dessa definição e utiliza uma Image Task como apontador para os dados em uma outra instância. Assim, essas entidades serão o conteúdo da resposta da comunicação.

A instância que realizou a pergunta recebe uma lista de Image Tasks e insere cada uma dessas em seu Service Catalog, para que as mesmas representem um apontador para os dados remotos. Trivialmente, caso essa lista seja retornada vazia, significa que a instância vizinha não processou os parâmetros especificados na pergunta.

Para diferenciar uma Image Task que aponta para dados locais de uma que aponta para dados remotos, nós definimos que uma Image Task pode assumir um novo estado - incrementado a máquina de estados da mesma -, o qual recebe o nome de Remotely Archived, o qual indica que a imagem de satélite teve seu processamento finalizado, mas que seus dados estão salvos em uma instância remota.

O estado é o único parâmetro que deverá ser diferente da Image Task que está armazenada no Service Catalog da instância vizinha, pois, alterar os outros parâmetros significaria alterar os parâmetros do processamento da imagem, o que tornaria a Image Task inconsistente. Além disso, dois parâmetros que não estão relacionados diretamente aos resultados, mas que devem ser mantidos são o identificador único da Image Task e a URL da instância em que a Image Task foi processada. Esses parâmetros serão utilizados para que a instância que reaproveitou a Image Task possa encontrar os dados na instância que processou a imagem.

A última das decisões arquiteturais necessárias para implementação do mecanismo consiste em definir o modo que o usuário irá acessar os dados gerados por uma instância diferente da que o mesmo acessou. Não seria interessante que o usuário tivesse necessidade de saber em quais instâncias o conjunto de imagens do qual ele deseja obter os dados foi processado. Caso o fizesse, ele teria que acessar cada um dos sites das instâncias que processaram pelo menos uma das imagens do conjunto solicitado, e requisitar as URLs dos dados gerados. Fazendo isso estaríamos complicando a experiência do nosso usuário. Nesse sentido, a estratégia deve permitir que o usuário acesse os dados das imagens de forma transparente, ou seja, sem ter a necessidade de saber em que instância aqueles dados foram gerados.

Dessa forma, nossa estratégia é automatizar esse trabalho de buscar as URLs em cada umas das instâncias que forneceram os dados de pelo menos uma imagem do conjunto de imagens solicitadas pelo usuário. Para tal, o mecanismo deve utilizar os parâmetros salvos nas Image Tasks e realizar uma uma requisição a instância que gerou os dados para os quais aquelas Image Tasks apontam, especificando os identificadores das mesmas. Com isso, o usuário receberá uma lista com as URLs das imagens processadas localmente juntamente com as URLs das imagens processadas remotamente.

Modificação na Arquitetura

Nesta seção nós discutimos quais mudanças devem ser aplicadas na arquitetura para tornar viável a implementação das decisões realizadas na seção anterior.

Inicialmente, devemos definir o interceptador das submissões feitas ao Submission Dispatcher. Esse interceptador, como discutido anteriormente, não deve influenciar no fluxo de processamento caso não existam instâncias vizinhas ou dados a serem aproveitados. Porém, sua responsabilidade é fortemente acoplada a do Submission Dispatcher. Assim, definimos que o nosso interceptador será um subcomponente do Submission Dispatcher, realizando a comunicação com as instâncias vizinhas e encaminhando o que deve ser processado localmente para o mesmo. Em outras palavras, esse interceptador tem o papel de gerenciar quais parcelas da submissão devem ser aproveitadas e quais devem ser processadas localmente. Dessa forma, o nomeamos de Submission Manager.

Portanto, a interface REST que antes encaminhava a submissão ao Submission Dispatcher passará a encaminhá-la ao Submission Manager. Esse, por sua vez, irá acessar as interfaces REST das instâncias vizinhas em busca de imagens que já foram processadas e casem com os parâmetros da submissão.

Para que o Submission Manager se comunique com as interfaces REST das outras instâncias é necessário que essas exponham um nova rota em sua API. A mesma deve receber

os parâmetros de uma submissão e retornar os apontadores - Image Tasks - para as imagens que a instância já processou.

Após receber as respostas dessa nova rota de cada uma das instâncias vizinhas, o Submission Manager deve armazenar no Service Catalog as Image Tasks recebidas e encaminhar para o Submission Dispatcher apenas a parcela restante de imagens de satélite que devem ser processadas localmente.

4 IMPLEMENTAÇÃO DA SOLUÇÃO

Nesta seção apresentamos a implementação de uma prova de conceito da solução, discutindo quais as tecnologias e serviços utilizados para implementar a arquitetura discutida na seção anterior.

A solução a ser implementada é uma extensão do serviço SAPS já existente, portanto as tecnologias utilizadas para implementá-lo serviram como base para implementação do nosso mecanismo. Dessa forma, o Submission Manager é implementado em Java dentro da arquitetura do framework que constitui a base do serviço: o Restlet.

As comunicações entre as interfaces REST das instâncias é feita através de requisições HTTP, as quais são realizadas e monitoradas através de recurso *client-side* chamado Client Resource, o qual é provido pelo Restlet [1]. Os dados transferidos nessas requisições e em suas respostas utilizam o formato JSON.

O Service Catalog é implementado utilizando o banco de dados PostgreSQL, nesse sentido as Image Tasks - apontadores que utilizamos para referenciar dados em outras instâncias - transferidas entre as instâncias são registros em tabelas desse banco.

Assim como a transferência de Image Tasks, a transferência das URLs - utilizadas pelos usuários para acessar os dados gerados pelo processamento das imagens - associadas a essas é feita através de requisições HTTP utilizando o recurso Client Resource.

5 AVALIAÇÃO DA SOLUÇÃO

Nesta seção apresentamos a avaliação da prova de conceito implementada, a qual foi discutida na seção anterior. Nossa avaliação consiste de uma verificação de corretude através de testes de aceitação, a qual deseja atestar que a implementação permite o compartilhamento de dados entre diferentes instâncias SAPS.

Ambiente de Teste

Nós configuramos uma implantação mínima de duas instâncias SAPS em um ambiente de teste. Cada uma dessas instâncias possuía seu Submission Dispatcher, consequentemente o subcomponente Submission Manager, implantado. O Service Catalog foi o outro componente configurado para ambas instâncias. Uma das instâncias, a qual nomeamos de instância A, possuía três Image Tasks com o estado Archived inseridas em seu Service Catalog, simulando que o serviço processou três imagens de satélite. Enquanto, a outra instância, a qual nomeamos de instância B, não possuía nenhuma Image Task inserida em seu Service Catalog. Além disso, configuramos a rede de colaboração da instância B para possuir a instância A, ou seja, a instância B pode requisitar o compartilhamento de dados à instância A.

Testes de Aceitação

Com os testes de aceitação nós desejamos confirmar que as Image Tasks armazenadas na instância A podem ser aproveitadas pela instância B. Dessa forma, estamos validando que uma instância SAPS pode manter uma referência para dados processados remotamente.

Inicialmente, nosso teste realiza asserções quanto às pré-condições estabelecidas, isto é, que a instância A e a instância B possuem, respectivamente, três e nenhuma Image Task em seu banco de dados.

Em seguida, o teste realiza uma submissão à instância B, simulando uma submissão feita pelo usuário através da GUI. Essa submissão possui os parâmetros que casam exatamente com os parâmetros das três Image Tasks armazenadas na instância A. Assim, é esperado que o Submission Manager da instância B se comunique com a instância A, perguntando se essa processou alguma imagem referente aos parâmetros fornecidos na pergunta. Por sua vez, a instância A deve retornar uma lista com as três Image tasks que possui, significando que sim, a mesma já processou aquela submissão.

Dessa forma, é esperado que a instância B não crie nenhuma Image Task local, visto que toda a submissão pode ser aproveitada da instância A. Após submeter os parâmetros à instância B, o teste verifica se essa agora possui três Image Tasks em seu banco de dados. Porém, isso não é suficiente, visto que essas poderiam ser Image Tasks criadas localmente, não provenientes da instância vizinha.

Assim, o teste verifica se todas Image Tasks estão com o estado Remotely Archived, significando que foram provenientes do compartilhamento de dados entre as instâncias A e B. Por fim, é verificado se os demais atributos das Image Tasks foram mantidos, comparando-os com os atributos das Image Tasks salvas na instância A.

Todos os critérios do teste de aceitação foram satisfeitos, portanto, podemos afirmar que duas instâncias vizinhas conseguem compartilhar entre si dados de processamentos de imagens de satélite.

6 EXPERIÊNCIA E LIÇÕES APRENDIDAS

Nesta seção apresentamos as experiências e lições aprendidas durante todo o processo deste trabalho. Detalhando o processo de desenvolvimento, os desafios encontrados no mesmo, assim como as soluções para esses desafios. Em seguida, realizamos um direcionamento para o que pode ser

feito no futuro para manter e evoluir o que foi produzido neste trabalho.

Processo de Desenvolvimento

O processo de desenvolvimento foi definido juntamente com o Prof. Thiago Emmanuel Pereira, orientador deste trabalho. Definimos e realizamos encontros semanais para discutir o que foi feito até o momento e o que ainda precisava ser feito para finalizarmos a implementação. Utilizamos o Git como gerenciador de versões do código, as quais eram divididas entre features macro. Para cada uma dessas features ser adicionada ao código existente, o código dessas deveria ser revisado e aceito por Thiago. Todo o código desenvolvido pode ser encontrado em https://github.com/velmer/saps-engine/tree/remote-sharing.

Desafios e Suas Soluções

O primeiro desafio surgiu no início do trabalho, visto que o SAPS era um serviço já existente, sobre o qual eu não tinha nenhum conhecimento, e o objetivo deste trabalho foi modificá-lo. Nesse sentido, foi desafiador entender a arquitetura do SAPS e como seus componentes se comportavam e interagiam entre si, para só então poder iniciar a modelagem da solução. Contudo, pude contar com o conhecimento sobre o SAPS e auxílio do Thiago, os quais foram essenciais para o entendimento e solução do problema.

Um outro desafio está relacionado ao SAPS necessitar de máquinas com alto poder computacional para executar, se comparadas com máquinas de especificações convencionais. Nesse sentido, não foi possível executar o SAPS em momento algum do processo de desenvolvimento na máquina utilizada nesse. Porém, com auxílio do Thiago Emmanuel e do colega Thiago Yuri - integrante da equipe de desenvolvimento do SAPS - foi possível acessar máquinas virtuais com especificações condizentes com as requeridas pelo SAPS. Dessa forma, conseguimos executar e validar o mecanismo implementado neste trabalho.

Trabalhos Futuros

Atualmente, a comunicação de uma instância com suas vizinhas ocorre de maneira sequencial, ou seja, a comunicação com uma vizinha irá começar apenas quando a comunicação com a vizinha anterior for encerrada. Nesse sentido, é interessante realizar a comunicação paralelamente, a fim acelerar a obtenção das respostas provenientes das instâncias vizinhas. Esse paralelismo é possível porque as respostas são independentes umas das outras, portanto não afetará o funcionamento do sistema.

Além disso, uma instância pode estar sobrecarregada com o processamento das suas imagens de satélite. Caso a mesma esteja presente em uma rede de colaboração, é possível que requisitem dados a essa instância enquanto ela está sobrecarregada. Assim, os recursos computacionais alocados para essa instância serão divididos entre o processamento de imagens e o processamento das requisições de instâncias vizinhas. Tal divisão afetará a performance da instância, fazendo com que o processamento das imagens demore mais que o esperado. Portanto, é interessante adicionar uma possibilidade de aceitar ou não colaborar com uma instância vizinha. A decisão de aceitar ou não pode ser feita automaticamente, analisando os níveis de utilização dos recursos computacionais (por exemplo, CPU, memória e disco) da instância. Quando os recursos estiverem sob uso intenso (por exemplo, com utilização acima de 80%) a instância estará indisponível para responder requisições de suas vizinhas, caso contrário, a mesma estará disponível e responderá normalmente às requisições.

REFERÊNCIAS

[1] [n.d.]. Acessado em 16 de Junho de 2019 de https://restlet.com/

[2] Esdras Pereira Iana Rufino Carlos Galvão Fernanda Valente John Cunha, Thiago Emmanuel Pereira and Francisco Brasileiro. 2018. SAPS: fast, automated execution of evapotranspiration workflows in infrastructure as a service clouds. Acessado em 15 de Junho de 2019 de http://saps.lsd.ufcg.edu.br/

[3] J. Wang Y. Ma J.F. Moreno G.J. Roerink W.G.M. Bastiaanssen, H. Pelgrum and T. van der Wal. 1998. A remote sensing surface energy balance algorithm for land (SEBAL). 2.Validation. *Journal of Hydrology* 1 (Dec. 1998), 212–213:213–229. https://doi.org/10.1016/S0022-1694(98)00254-6

[4] R.A. Feddes W.G.M. Bastiaanssen, M. Menenti and A.A.M. Holtslag. 1998. A remote sensing surface energy balance algorithm for land (SEBAL). 1. Formulation. *Journal of Hydrology* (Dec. 1998), 212–213:198–212. https://doi.org/10.1016/S0022-1694(98)00253-4