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

THIAGO YURI EVARISTO DE SOUZA

AVALIAÇÃO DE DESEMPENHO NO ARMAZENAMENTO DE DADOS PARA UM SISTEMA DE PROCESSAMENTO DE IMAGENS DE SATÉLITE

CAMPINA GRANDE - PB

2022

THIAGO YURI EVARISTO DE SOUZA

AVALIAÇÃO DE DESEMPENHO NO ARMAZENAMENTO DE DADOS PARA UM SISTEMA 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 : Thiago Emmanuel Pereira da Cunha Silva

CAMPINA GRANDE - PB

2022

THIAGO YURI EVARISTO DE SOUZA

AVALIAÇÃO DE DESEMPENHO NO ARMAZENAMENTO DE DADOS PARA UM SISTEMA 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:

Thiago Emmanuel Pereira da Cunha Silva

Orientador – UASC/CEEI/UFCG

Andrey Elisio Monteiro Brito

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 02 de Setembro de 2022.

CAMPINA GRANDE - PB

RESUMO

SAPS é um sistema que executa algoritmos para aquisição, pré-processamento e processamento de imagem de satélite. Durante a execução dos algoritmos, arquivos são compartilhados entre os processos que compõem cada um dos estágios (aquisição, pré-processamento e processamento). Atualmente, o SAPS utiliza um sistema de arquivos NFS centralizado como solução para compartilhamento de dados. O NFS é uma solução conhecida por apresentar alguns problemas, tais como implicar em ponto único de falha e não ser escalável. Existem outras abordagens mais recentes que resolvem os problemas do NFS. Uma dessas, o ObjectiveFS é um sistema de arquivos em Cloud que permite a escalabilidade do storage possuindo uma performance similar ao armazenamento via NFS. Neste trabalho projetarei e implementarei mudanças no projeto do SAPS permitindo o uso de uma nova abordagem que possibilite resolver os problemas do NFS. A solução escolhida lida com o uso de uma SDK desenvolvida para lidar com a nuvem do Openstack e um dos seus serviços de armazenamento, o Object Storage (Swift). Este trabalho gerou uma melhoria no design do SAPS, porém seu desempenho não atendeu as expectativas de ser superior, existindo um trade-off entre ambas as versões.

Avaliação de desempenho no armazenamento de dados para um sistema de processamento de imagens de satélite

Thiago Yuri Evaristo de Souza
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
thiago.souza@ccc.ufcg.edu.br

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

Resumo

SAPS é um sistema que executa algoritmos para aquisição, pré-processamento e processamento de imagem de satélite. Durante a execução dos algoritmos, arquivos são compartilhados entre os processos que compõem cada um dos estágios (aquisição, pré-processamento e processamento). Atualmente, o SAPS utiliza um sistema de arquivos NFS centralizado como solução para compartilhamento de dados. O NFS é uma solução conhecida por apresentar alguns problemas, tais como implicar em ponto único de falha e não ser escalável. Existem outras abordagens mais recentes que resolvem os problemas do NFS. Uma dessas, o ObjectiveFS é um sistema de arquivos em Cloud que permite a escalabilidade do storage possuindo uma performance similar ao armazenamento via NFS. Neste trabalho projetarei e implementarei mudanças no projeto do SAPS permitindo o uso de uma nova abordagem que possibilite resolver os problemas do NFS. A solução escolhida lida com o uso de uma SDK desenvolvida para lidar com a nuvem do Openstack e um dos seus serviços de armazenamento, o Object Storage (Swift). Este trabalho gerou uma melhoria no design do SAPS, porém seu desempenho não atendeu as expectativas de ser superior, existindo um trade-off entre ambas as versões.

Palavras-chave

remote sensing, cloud, storage, file systems

1. Introdução/Motivação

Evapotranspiração (ET) é uma termo de referência aos processos combinados pelos quais a água se move da superfície da Terra para a atmosfera. Estimar ET ajuda a entender o ciclo da água desempenhando um importante papel na gestão hídrica e irrigação agrícola [9].

O cálculo feito para estimativa de evapotranspiração por meio de imagens de sensoriamento remoto como dados de satélite iniciou na década de 1980 [1]. Entretanto algumas outras informações são necessárias tais como dados meteorológicos, indicadores de relevo e outros.

Alguns métodos utilizados no cálculo de ET são comuns no estado da arte, tais como SEB (Surface Energy Balance) e métodos empíricos [1]. Essas abordagens utilizam dados espaciais de resolução média, a exemplo de imagens de satélite providas pela família Landsat com cerca de 30 metros.

O SAPS (SEB Automated Processing Service) é um sistema que permite aos seus usuários, por meio de uma interface gráfica (GUI), acesso ao serviço de processamento automatizado de métodos SEB [2]. Esse sistema gerencia a execução de algoritmos, armazenamento dos resultados e os disponibiliza, além de atender a novos pedidos de processamento em diferentes datas e locais.

O processamento é dividido em 3 fases: aquisição [4], pré-processamento [5] e processamento [6]. Sendo, a aquisição responsável por obter os dados de entrada (dados de satélite, meteorológicos e outros), pré-processamento para preparar esses dados e gerar outras informações e por fim, processamento para geração do ET por meio de um método (p.ex. SEB).

Durante a execução dos algoritmos, arquivos são compartilhados entre os processos que compõem cada um dos estágios, sendo alguns na ordem de 2GB a depender do tipo de satélite [1]. Para uma alta demanda de processamento, o SAPS pode ficar limitado a uma certa vazão de acordo com o mecanismo de armazenamento temporário desses dados.

Atualmente, o SAPS utiliza um sistema de arquivos NFS centralizado como solução para compartilhamento entre as fases, no qual é configurado em uma máquina com um volume montado para comportar os dados temporários. Essa solução tem alguns problemas, sendo os principais apresentar um ponto único de falha e não ser escalável.

Este trabalho busca projetar e implementar uma nova versão do código do SAPS utilizando uma nova abordagem em cloud para resolver os problemas vistos no sistema de arquivos NFS.

Muitos serviços de nuvem possuem soluções para storage de arquivos com práticas de montagem em instâncias/vms do mesmo provedor. Exemplos dessa abordagem são o AWS S3 e o Google Cloud Storage.

Algumas aplicações como o s3fs permitem Linux, macOS e FreeBSD montarem um bucket S3 via FUSE [10,11]. FUSE (Filesystem in USERspace) é uma interface de software para SO's Unix que permite que usuários não privilegiados criem seus sistemas de arquivos sem editar o kernel [11].

Para o nosso caso em particular, observamos os serviços em cloud disponibilizados pelo Openstack, como Object Store (Swift) [12, 13]. Essa limitação do escopo de busca se dá para uma experimentação da solução proposta na nuvem do Laboratório de Sistemas Distribuídos.

Este trabalho gerou uma melhoria no design do SAPS, porém seu desempenho não atendeu as expectativas de ser superior, existindo um trade-off entre ambas as versões.

Nas próximas seções deste trabalho serão apresentadas: a visão geral da arquitetura atual utilizando NFS (seção 2), a nova arquitetura e solução proposta além de resultados obtidos no

experimento (seção 3), experiências e lições aprendidas (seção 4) e conclusão e trabalhos futuros sugeridos (seção 5).

2. Visão geral da arquitetura atual

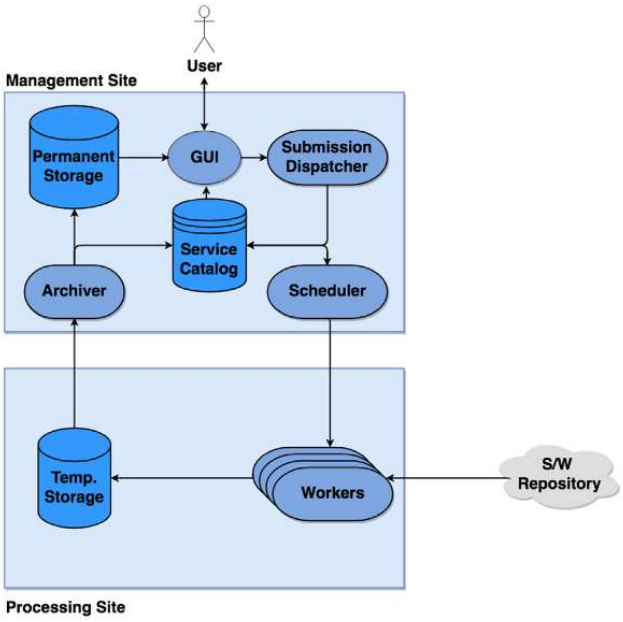

Fig 1. Principais componentes da arquitetura do SAPS

O SAPS é um sistema distribuído sendo composto de alguns componentes (Fig 1), que são: *GUI* [3], *Catalog* [14], *Dispatcher* [15], *Scheduler* [7], *Archiver* [16], *Temporary Storage* [17], Permanent Storage, Workers e *S/W Repository*.

GUI é a interface gráfica do usuário, ou também nomeada de *dashboard* [3]. É responsável por fornecer, aos usuários, as features do SAPS, tais como: processar, acompanhar e recuperar processamentos de scripts realizadas no sistema.

Catalog [14] é o componente responsável pelo banco de dados, sendo o core do SAPS. É extremamente importante seu estado e sua disponibilidade, pois os outros componentes enxergam-o e o consultam para decidir e prosseguir.

Dispatcher [15], ou também *Submission Dispatcher,* é a API do SAPS no qual o frontend se comunica via REST API (utilizando Restlet framework [20]).

Scheduler [7] é o escalonador de processamento de tarefas do SAPS. Esse componente lida com a demanda de diversos usuários fazendo uso de algum algoritmo de escalonamento, tais como: round-robin [21].

Archiver [16] é o armazenador de processamentos finalizados no SAPS, ou seja, realiza a transferência dos arquivos do ambiente temporário para o permanente. Outra função secundária é realizar a limpeza do armazenamento temporário [17]. *Temporary storage* [17] é o local temporário utilizado pelos workers para armazenar dados referentes aos estágios de processamento (inputdownloading [4], preprocessing [5] e processing [6]).

Permanent Storage é um armazenamento com diversas configurações para segurança, replicação e outros fatores para todos os dados dos processamentos dos scripts em todos os estágios. Atualmente contém duas opções de storage: *Swift* para soluções provisionadas em Openstack cloud, ou *Apache + NFS* para provisionamento em qualquer ambiente.

Workers são as unidades de execução dos scripts do SAPS. Atualmente, são gerenciadas por um outro sistema chamado *Arrebol* [18]. E finalmente, *S/W* (Software) *Repository* que agrupa um conjunto de scripts do workflow (inputdownloading [4], preprocessing [5] e processing [6]) para execução nos workers.

2.1 Árvore de diretório de um tarefa SAPS

```
/
-- /<task1>
---- /inputdownloading
------ <inputdownloading files>
---- /preprocessing
------ <preprocessing files>
---- /processing
------ <processing files>
-- ...
-- /<taskN>
---- /inputdownloading
---- /preprocessing
---- /processing
```

Fig 2. Árvore de diretórios das tarefas SAPS

O SAPS adota uma simples árvore de diretório para organizar suas tarefas (Fig 2). Cada task possui um id gerado no momento da criação feito pelo componente *Dispatcher*. Esse identificador (*taskId*) é único e será utilizado como nome do diretório da tarefa, por exemplo, o diretório *task1* tem o identificador *taskId* da tarefa 1, enquanto o diretório *taskN* tem o identificador *taskId* da tarefa N.

Cada tarefa possui, em seu diretório, três subdiretórios padrões: inputdownloading, preprocessing e processing, referentes às fases de aquisição, pré-processamento e processamento mencionadas anteriormente. Em cada uma delas estão contidos os arquivos resultantes de cada etapa de processamento, respectivamente.

Todas as tarefas bem sucedidas possuem os três subdiretórios com seus respectivos dados. Entretanto, aquelas que falham no processamento podem ausentar alguns desses diretórios. O fato disso ocorrer é de que falharam em estágios anteriores, por exemplo, se uma tarefa falhou na etapa de preprocessing (pré-processamento), então só há dados da etapa de inputdownloading (aquisição).

2.2 Fluxo de execução de um novo processamento

Uma nova submissão (Fig 3) vinda do frontend, comunica um lote (batch) de novas tarefas para a API do *Dispatcher*. Neste pedido são informados: a região, o intervalo temporal e os scripts.

A região indica ao SAPS por meio de coordenadas (latitude e longitude), o local no qual deseja-se realizar o processamento. Porém para suprir múltiplas regiões, é passado um par de

coordenadas para determinar um polígono de região de processamento.

Os scripts são atrelados a uma fase de processamento (inputdownloading, preprocessing ou processing). Foi definida uma estrutura básica para os scripts [19], assegurando (a aqueles que o respeitarem) o correto funcionamento no workflow, permitindo também que novas contribuições externas surjam.

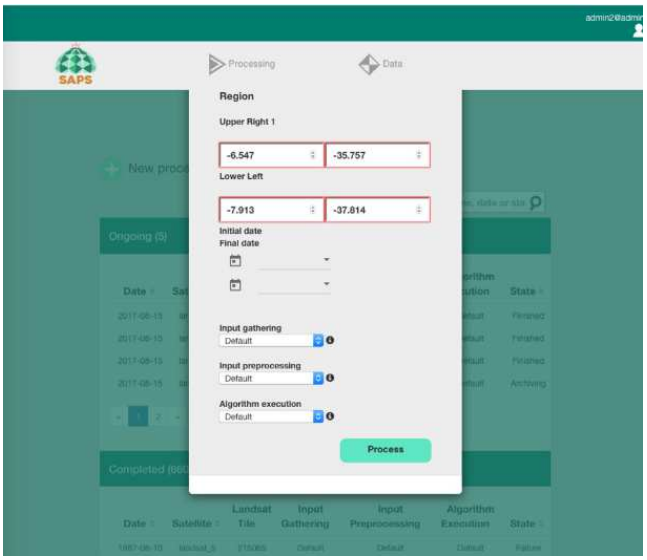

Fig 3. Novo pedido de processamento no SAPS via dashboard

2.3 Máquina de estados das tarefas SAPS

O SAPS possui uma máquina de estados para suas tarefas de processamento (Fig 4). Cada estado dita qual componente irá computar a task e passá-la para o estado seguinte.

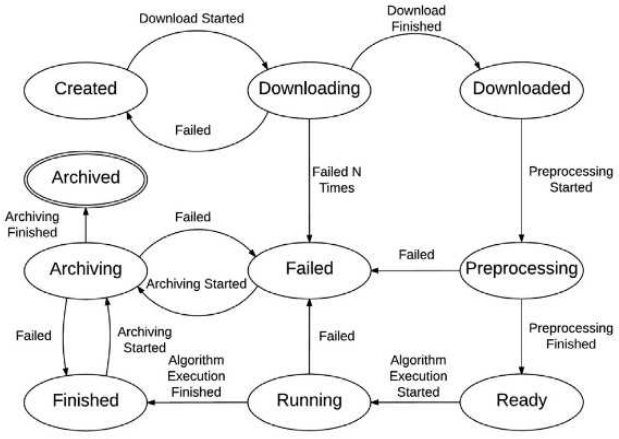

Fig 4. Máquina de estados da task do SAPS

As novas tarefas SAPS são iniciadas no estado *created* no Catalog inseridas pelo *Dispatcher* após processar o novo batch de processamento. Após isso, o *Scheduler* enxerga o novo estado do banco de dados com as novas tarefas e aloca um worker para cada task selecionada para processamento, mudando para um estado intermediário *downloading*.

Em um maior nível de detalhe, o *Scheduler* não lida com a alocação de um worker, na prática é feita uma comunicação com um serviço de escalonamento de trabalhos (jobs) chamado *Arrebol*.

O Arrebol possui um conjunto de workers (virtual machines, ou então VMs) que são gerenciados via Docker, além de fornecer uma API para seus usuários conseguirem solicitar e acompanhar a execução de seus pedidos.

O *Scheduler* após enviar as requisições de processamento das tarefas selecionadas em estado *created* ao *Arrebol*, periodicamente consulta-o para verificar seu status aguardando a finalização. Após a conclusão de algum subconjunto delas, o *Scheduler* repassa o estado de cada uma das tarefa para *downloaded*, caso bem sucedidas.

Para os próximos passos, o componente *Scheduler* realiza os mesmos steps das tarefas no estado *created* para aquelas em *downloaded* e *ready*. Em resumo, seleciona tarefas em *downloaded/ready*, aloca workers mudando para um estado intermediário *preprocessing/running* e após a finalização bem-sucedida, altera para o estado *ready/finished*.

O fato importante no *Scheduler* é o escalonador que, de forma gulosa, procura selecionar e finalizar as tarefas em estágios mais avançados, ou seja, uma tarefa em *ready* tem maior prioridade do que aquelas em *downloaded/created*, e assim por diante.

O último passo é dado pelo *Archiver*, que aguarda por tasks no estado *finished*, ou seja, que finalizam com sucesso todas as fases de processamento. Com isso, o componente move os resultados de cada tarefa do *temporary storage* para o *permanent storage*, mudando o estado para *archiving* durante o envio e para *archived* quando bem sucedido.

Em contrapartida, durante qualquer um dos estados ativos da tarefa (*downloading*, *preprocessing*, *running* e *archiving*) podem ser levados ao estado de falha, *failed*. Para lidar com possíveis problemas externos, tais como, network, indisponibilidade e outros, foi adotado uma abordagem simples de *retry* para contorná-los.

Os scripts de inputdownloading, preprocessing e processing são executados nos estados de *downloading*, *preprocessing* e *running* respectivamente.

2.4 Sobre o uso do NFS

Uma decisão importante é dada pelo *Scheduler* para priorizar as tarefas com estados mais avançados, pois permite com que os dados no armazenamento temporário (*temporary storage*) sejam liberados após tratadas pelo *Archiver*.

O armazenamento temporário utiliza o NFS (Network File System) para manter os dados dos processamentos. Essa solução funciona com um servidor NFS implantado em uma instância (VM) compartilhando, via rede, um diretório exportado do seu sistema de arquivos. Com isso, seus clientes NFS (outras instâncias/VMs) podem montar o diretório exportado e realizar operações sobre os arquivos.

2.5 Problemas da solução com NFS

Os principais problemas no uso do NFS para armazenamento temporário são apresentar um ponto único de falha (p.ex uma falha na VM que hospeda o NFS, implica em bloquear todo o sistema) e escalabilidade (NFS server executando em uma VM com storage limitado).

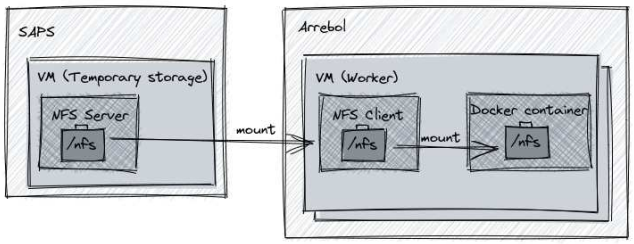

Fig 5. Adaptação do Arrebol para o SAPS utilizando NFS

Além de que, cada worker do *Arrebol* deve se tornar um NFS client para que cada job venha a enxergar o servidor. Com isso, foi feita uma versão exclusiva do *Arrebol* para uso no SAPS, fazendo com que o diretório montado no worker também seja montado no container docker gerenciado pelo Arrebol (Fig 5).

Esse diretório visualiza toda a árvore de diretório do SAPS (vista em 2.1). Uma vulnerabilidade dada por esse compartilhamento é que uma task X pode atrapalhar (deletar, mover, modificar, …) os dados de outra task Y.

3. Arquitetura e Projeto da Solução

Na nova arquitetura (Fig 6), o *temporary storage* utilizará o *Swift* ao invés do *NFS*. O serviço de Object Storage do Openstack (Swift) é um software de armazenamento em nuvem que permite armazenar e recuperar lotes de dados por meio de uma API simples.

A ideia consiste em manter os dados temporários em um local mais escalável. Para isso, os workers coletam os dados necessários para o processamento do script atual, e ao final, arquiva os novos dados resultantes no *temporary storage*.

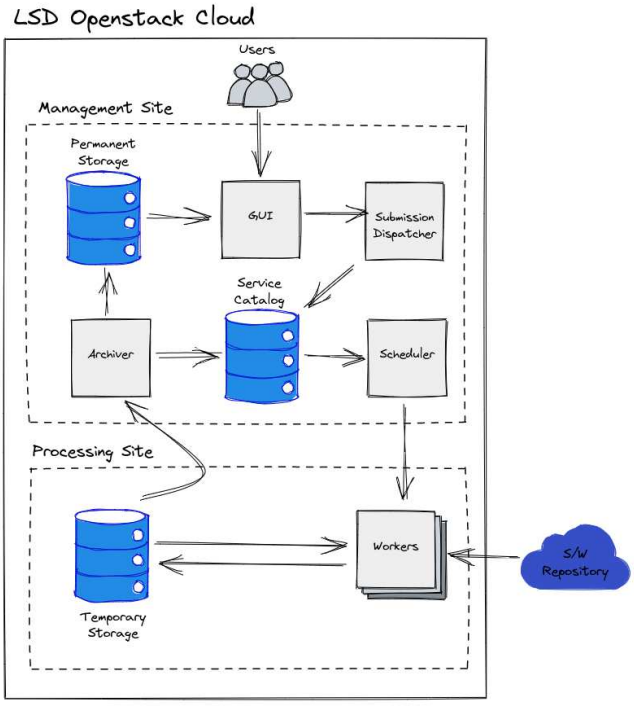

Fig 6. Arquitetura do SAPS igualmente vista na Fig 1

Nessa nova solução, o *Archiver* terá uma implementação para cada tipo de *temporary storage*, para a solução atual do *NFS* e para nova abordagem utilizando *Swift*. A escolha do tipo de deploy será feito via arquivos de configuração.

O *Arrebol* gerencia workers que não deveriam manter estados de processamentos passados para novos jobs. Nessa nova solução, os workers iniciam o processamento do script sem nenhum conhecimento prévio (montagem de um diretório fixo do host).

Para poder coletar e arquivar os dados, será utilizado uma SDK criada pela própria Openstack, o openstacksdk [8].

3.1 Sobre a ferramenta openstacksdk

O openstacksdk é um projeto do OpenStack que visa fornecer um kit completo de desenvolvimento de software para os programas que compõem sua comunidade. É uma biblioteca Python com documentação, exemplos e ferramentas correspondentes lançadas sob a licença Apache 2.

Seu principal objetivo é se comunicar com qualquer nuvem OpenStack. Para fazer isso, é necessário um arquivo de configuração (Fig 7), mas também pode usar variáveis de ambiente.

```yaml
clouds:
  mordred:
    region_name: Dallas
    auth:
      username: 'mordred'
      password: XXXXXXX
      project_name: 'demo'
      auth_url: 'https://identity.example.com'
```

Fig 7. Exemplo de arquivo de configuração (YAML)

Após a instalação e importação do pacote openstack (obtido via pip), é possível iniciar uma conexão com a cloud (Fig 8). Com a conexão estabelecida, a SDK fornece algumas funções que realizam operações nos serviços do Openstack cloud [22].

```python
import openstack

# Initialize and turn on debug logging
openstack.enable_logging(debug=True)

# Initialize connection
conn = openstack.connect(cloud='mordred')

# List the servers
for server in conn.compute.servers():
    print(server.to_dict())
```

Fig 8. Iniciando conexão com Openstack cloud usando SDK

3.2 Mudança na máquina de estados das tarefas SAPS

O novo fluxo da tarefa SAPS (Fig 9) é igual como em 2.3, porém com algumas mudanças sutis.

Nos passos 6, 10 e 14, os resultados dos passos de inputdownloading, preprocessing e processing, respectivamente, são armazenados no *temporary storage* utilizando o openstacksdk.

Por outro lado, nos passos 9 e 13, os resultados anteriormente armazenados são recuperados para realizar o processamento.

Contudo, no passo 16 (*archiving*), o *Archiver* irá realizar a transferência dos dados da tarefa contida no *temp storage* para o *permanent storage*. Após ter êxito na operação, ocorre o passo 17 (*archived*).

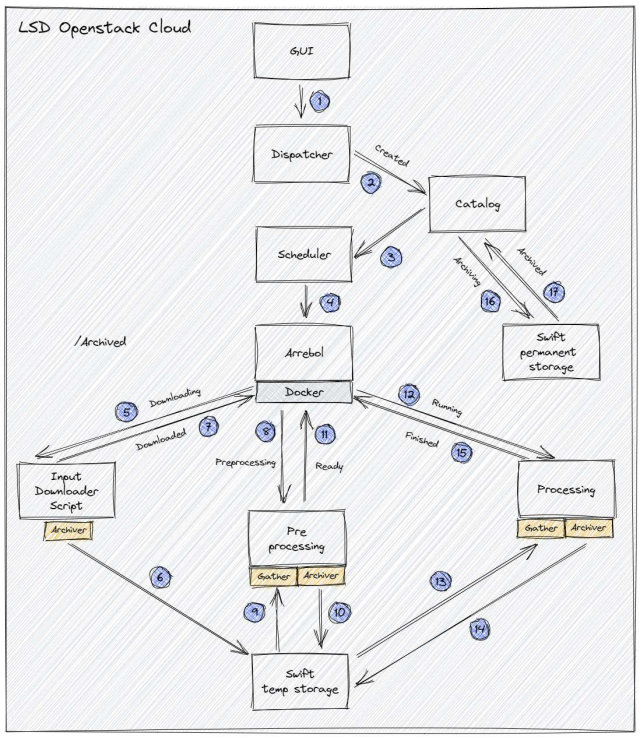

Fig 9. Máquina de estados das tarefas SAPS na nova arquitetura

3.3 Prós e contras da nova arquitetura

Com a nova solução, alguns trade offs são vistos quando comparado a solução NFS (Tabela 1).

Prós	Contras
Maior escalabilidade	Dependência de uma solução openstack cloud com suporte ao serviço de Object Storage (Swift)
Cópias dos dados originais de uma tarefa, não ocorrendo o compartilhamento dos dados de outra tarefa	Adição de um tempo para coleta e envio dos dados dos estágios anteriores e atuais
Armazenamento distribuído com replicação e outras funções gerenciadas pelo Swift	
Tolerância a falhas	

Tabela 1. Trade offs da solução Openstack comparada com o NFS

A escalabilidade antes definida pela capacidade em disco da instância/volume do diretório exportado do NFS Server, agora é flexibilizada pela cota do Swift. Na prática, para aumentá-la ou diminuí-la basta modificarmos o parâmetro desse recurso em cloud.

Os dados de uma tarefa não serão visíveis a outras tarefas como na abordagem do NFS. Cada tarefa recupera e envia seus dados para o Swift, não conseguindo modificar as outras, além do serviço manter as informações replicadas.

Entretanto, a nova arquitetura é operante somente em cloud Openstack com suporte ao serviço de Object Storage (Swift). Além disso, como visto anteriormente na máquina de estados das tarefas SAPS (seção 3.2), temos a adição de um tempo referente a coleta e envio dos dados sobre a task, isso ocorre porque os workers *Arrebol* são *stateless* (não contém/guardam informações prévias).

3.4 Projeto da nova arquitetura

O componente *Scheduler* se comunica com um serviço de escalonamento de jobs denominado *Arrebol*. Nessa comunicação, é solicitado a criação de um processamento que irá executar uma lista de comandos. Essa execução é feita em um container Docker utilizando uma imagem previamente registrada no Docker Hub [23].

A solução consiste em compor comandos, com auxílio do openstacksdk, que busquem/enviem os dados necessários/resultantes, fazendo assim, com que a execução do script de inputdownloading/preprocessing/processing seja bem sucedida.

A SDK fornece funções para criar, listar, deletar e recuperar dados no Swift (serviço do Openstack), no qual serão utilizados durante o processamento das tarefas SAPS nos workers do *Arrebol* e no fluxo do *Archiver*.

Foi desenvolvida uma versão simples para o *Arrebol* [24] suportar envio de variáveis de ambiente para um novo job, e dessa forma, sendo utilizado para a configuração do openstacksdk permitindo uma conexão com a cloud do Openstack LSD.

No *Archiver* haverá uma nova implementação [25] que lida com a abordagem do *temporary storage* utilizando *Swift* através do uso de interface em nível de código. Com isso, esse componente conseguirá transferir de um container *Swift* (podendo estar na mesma ou em outra cloud Openstack) representado pelo *temporary storage* de um *processing site* qualquer do SAPS para outro container Swift do *permanent storage* referente ao *management site* (visto anteriormente na Fig 1).

O *Scheduler* também receberá uma nova implementação [26] gerenciada via interface em nível de código para suportar tanto a versão antiga do armazenamento temporário NFS quanto a nova Openstack/Swift. A diferença básica entre as implementações é dado pelos comandos executados e a configuração feita pelos workers do Arrebol (visto na Fig 10).

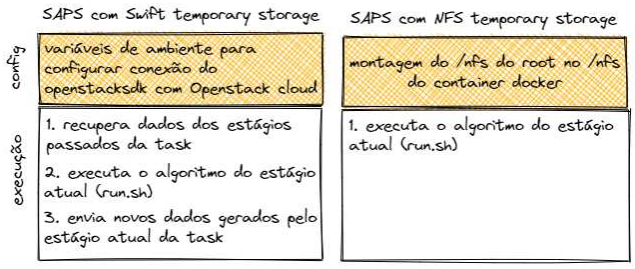

Fig 10. Diferenças na configuração e execução do worker do Arrebol

Todas as implementações vistas acima foram *forks* feitos dos repositórios originais do SAPS e adotou-se a branch *tcc* para commits realizados para adaptação das soluções.

3.5 Avaliação de desempenho

Para verificar o funcionamento da nova solução, foi realizado um experimento em ambas as versões, a antiga (NFS) e a nova (Openstack/Swift). O objetivo principal é avaliar o impacto de sobrecarga de read/write no armazenamento temporário, ou seja, para a solução NFS seria estressar o diretório compartilhado pelo NFS Server. Entretanto, para a solução Openstack/Swift será testada pelo acesso/requisição simultânea ao serviço em Cloud.

Para cada versão foi preparada uma instância com 2 VCPUs, 4 GB RAM e 80GB de espaço em disco. Na versão Swift foram levantados os componentes do SAPS (catalog, scheduler e archiver) e o Arrebol. Na versão NFS, foram levantados os mesmos componentes, além da configuração do NFS Server.

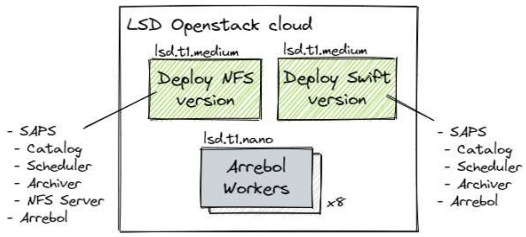

Fig 11. Organização das instâncias para experimento na cloud do Openstack LSD

Além das instâncias referente a cada versão, também foram levantadas 8 VMs responsáveis pelo processamento do *Arrebol*, ou seja, seus workers.

Os scripts dos estágios das tarefas SAPS utilizadas no experimento foram criados e podem ser acessados no docker hub [27, 28, 29]. Basicamente o script de inputdownload baixa 3 arquivos de 362MBs cada de um site, o script de preprocessing copia os 3 arquivos baixados no estágio de inputdownloading como seus resultados e o script de processing copia os 3 arquivos copiados no estágio de preprocessing (visto na Fig 12).

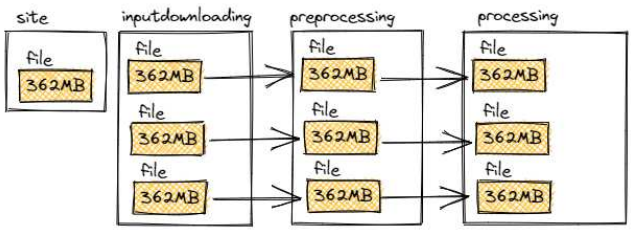

Fig 12. Fluxo dos dados de uma tarefa SAPS do experimento utilizando os scripts criados

Os seguintes cenários foram estabelecidos para verificarmos o desempenho das implementações (Tabela 2):

Cenário	Versão do temporary storage	Quant. de tasks SAPS	Nº workers
1	NFS	1	1
2		2	2
3		4	4
4		8	8
5	Swift	1	1
6		2	2
7		4	4
8		8	8

Tabela 2. Cenários do experimento

Os cenários foram montados de tal forma que cada worker se responsabilize por uma única tarefa SAPS e que todos estejam ocupados realizando uma sobrecarga de read/write durante a execução paralela.

A diferença entre as versões se dá no ponto de sobrecarga. Na versão Swift acontece nas operações realizadas nas API do Swift na cloud do Openstack. Entretanto, a versão NFS acontece nas manipulações de leitura e/ou escrita que ocorrem no diretório compartilhado do NFS Server.

No caso básico (cenários 1 e 5), haverá somente 1 worker com sua tarefa SAPS para execução. Em detalhes temos que

1. Na versão NFS:
   1.1. no estágio de inputdownloading: terá somente 1 acesso de escrita (no ato de download) concorrentemente ao armazenamento temporário do NFS;
   1.2. nos estágios seguintes de preprocessing/processing: terá somente 1 acesso de leitura/escrita (no ato de copiar).
2. Na versão Swift:
   2.1. no estágio de inputdownloading: terá somente 1 requisição de escrita (no ato de salvar os dados resultantes do download) concorrentemente ao armazenamento temporário no container Swift do Openstack;
   2.2. nos estágios seguintes de preprocessing/processing: terá um conjunto de requisições de leitura/download (no ato

de recuperar os dados salvos da task) concorrentemente no container Swift do Openstack e outro conjunto de requisições de escrita (no ato de salvar os dados resultantes do preprocessing/processing) concorrentemente ao armazenamento temporário no container Swift do Openstack.

No caso mais complexo (cenários 4 e 8), haverá 8 workers processando cada uma de suas tarefas SAPS, ou seja, mais sobrecarga tanto de read/write no diretório compartilhado do NFS Server quanto de requisições a API do Swift na cloud do Openstack.

Cada cenário apontado na tabela foi executado 5 vezes para obtermos métricas como média e desvio padrão. Uma breve explicação da tabela é de que os cenários 1, 2, 3 e 4 da versão NFS são os mesmos dos cenários 5, 6, 7 e 8 da versão Swift respectivamente.

3.5.1 Resultados

Cada cenário foi realizado isoladamente e os resultados foram salvos no repositório do GitHub e podem ser acessados aqui.

Diferente do que esperávamos, a nova versão para o armazenamento temporário com Swift foi mais lenta em todos os cenários quando comparada a versão antiga do NFS (visto na Fig 13), considerando a carga projetada no experimento.

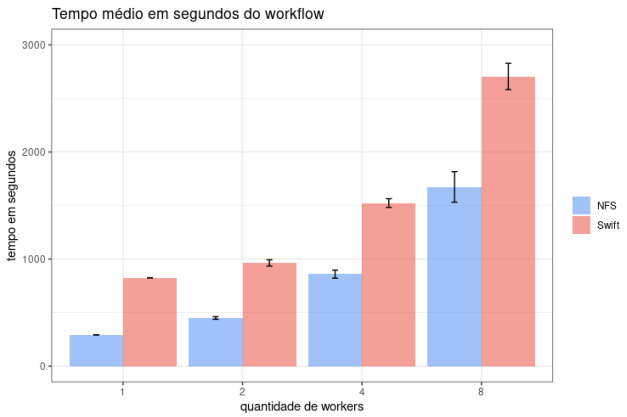

Fig 13. Tempo médio em segundos do workflow para cada versão de armazenamento temporário

Um fator importante não levado em consideração foi o fato da API do Openstack/Swift sofrer grandes impactos com a sobrecarga. Por ser uma interface que lida com armazenamento de um volume grande de dados, espera-se que saiba lidar com acesso concorrente de maneira mais escalável, assim como AWS S3 e outros serviços conhecidos.

O experimento buscava mostrar que a sobrecarga no NFS ocorreria, como observado no gráfico, mas na versão Swift iria ser algo constante, ou seja, quanto no melhor ou pior cenário teria resultado equivalente. Com isso, a solução do Swift seria indicada para um cenário com múltiplos workers.

O tempo médio foi calculado com a seguinte fórmula:

$$TM = (LAST\_TMP - FIRST\_TMP) + ARCHIVING\_TIME$$

onde,

- *LAST_TMP* é o timestamp unix do término do último estágio (processing) da última tarefa SAPS do cenário;
- *FIRST_TMP* é o timestamp unix do início do primeiro estágio (inputdownloading) da primeira tarefa SAPS do cenário;
- *ARCHIVING_TIME* é o tempo total para arquivagem dos dados de todas as tarefas SAPS (feito de 1 a 1) do armazenamento temporário para o armazenamento permanente.

Abaixo na tabela 3, temos as métricas de média e desvio padrão para cada cenário:

Nº de workers / tarefas SAPS	Swift		NFS	
	Tempo médio	Desvio padrão	Tempo médio	Desvio padrão
1	824.400	-	292.000	-
2	963.600	29.543	449.600	12.116
4	1523.000	40.786	858.800	37.858
8	2705.400	122.680	1673.600	142.756

Tabela 3. Métricas de média e desvio padrão do tempo em segundos do workflow das versões de armazenamento temporário

A razão do tempo médio do Swift sobre o da versão NFS é cerca de 2.8 vezes pior no cenário simples, porém conforme aumenta a complexidade do cenário melhor fica o resultado para a nova versão. No entanto, não sabemos se o resultado se torna equivalente em algum desses cenários mais complexos.

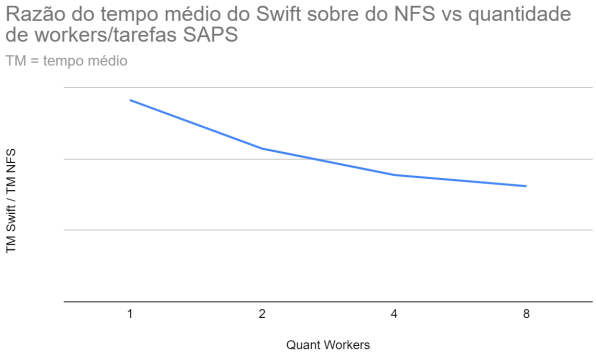

Fig 14. Razão do tempo médio do Swift sobre do NFS nos diversos cenários do experimento

4. Principais desafios para escolha da solução

Nosso principal objetivo para esse trabalho é encontrar soluções de acordo com os seguintes requisitos (em ordem de prioridade):

1. adequabilidade de implantação na infraestrutura do Laboratório de Sistemas Distribuídos (LSD);
2. popularidade no uso, o que pode indicar confiabilidade, qualidade e robustez da solução;

Dentre esses requisitos, o mais determinante foi o primeiro. Consideramos as opções de armazenamento nativas da cloud do LSD. As opções de armazenamento presentes na distribuição Openstack do LSD são Swift (blob) e block storage (volumes).

Diversas soluções em cloud foram impossibilitadas com o uso somente dessas, principalmente as presentes nas novas versões do Openstack.

5. Conclusão e trabalhos futuros

Apesar dessa nova solução não conseguir ser equivalente em desempenho, alguns outros problemas foram resolvidos (visto na tabela 1). A ferramenta openstacksdk ainda está em fase de desenvolvimento durante esse trabalho (atualmente na versão 0.100.1.dev21), o que indica que podem haver mudanças no processo em suas versões futuras, possibilitando resultados diferentes dos vistos aqui.

Um dos principais problemas importantes resolvidos foi a questão dos workers do *Arrebol* serem *stateful*, ou seja, ter conhecimento prévio dos dados da tarefa SAPS, antes mesmo de executar nenhum comando. Com isso, o SAPS se torna dependente de uma versão especial *Arrebol*, embora devesse ser facilmente portável para outro serviço de execução de jobs (semelhantes ao *Arrebol*).

Embora a nova solução só resolva o problema para deploy em uma cloud Openstack com suporte ao serviço do Swift, ainda remove essa dependência.

Alguns trabalhos futuros podem ser feitos como: refazer teste em uma cloud Openstack na sua versão mais atual, realizar o experimento em cenários com mais workers ou com arquivos maiores, adicionar a portabilidade da nova solução para suporte a docker/kubernets e realizar implementação utilizando volumes (block storage) para investigar trade offs entre as opções de armazenamento existentes no LSD Openstack Cloud.

6. Agradecimentos

Quero agradecer a todos os professores do curso de bacharelado em Ciência da Computação da Universidade Federal de Campina Grande (UFCG), por oferecer uma excelente ementa das disciplinas ofertadas no plano de curso, além da ministração das aulas de forma didática, apesar do regime remoto ocorrido devido a pandemia da covid-19.

Não esquecendo de agradecer a minha família, em especial, meus pais, José Evaristo de Souza (pai) e Vaneocleide de Souza (mãe), e minha avó materna, Maria José Elias de Souza, que sempre me apoiaram desde meu ensino fundamental, depositando créditos de confiança no meu potencial educacional no qual venho a concluir minha graduação.

7. Referências

[1] Cunha, J., “A high-throughput shared service to estimate evapotranspiration using Landsat imagery”, Computers and Geosciences, vol. 134, 2020. doi:10.1016/j.cageo.2019.104341.

[2] "SAPS engine". GitHub. 16 de Março, 2022. Site: https://github.com/ufcg-lsd/saps-engine

[3] "SAPS front-end". GitHub. 16 de Março, 2022. Site: https://github.com/ufcg-lsd/saps-dashboard

[4] "SAPS input download stage implementations". GitHub. 16 de Março, 2022. Site: https://github.com/ufcg-lsd/saps-scripts-inputdownload

[5] "SAPS preprocessing stage implementations". GitHub. 16 de Março, 2022. Site: https://github.com/ufcg-lsd/saps-scripts-preprocessing

[6] "SAPS processing stage implementations". GitHub. 16 de Março, 2022. Site: https://github.com/ufcg-lsd/saps-scripts-processing

[7] "SAPS scheduler component implementation". GitHub. 16 de Março, 2022. Site: https://github.com/ufcg-lsd/saps-scheduler

[8] “Openstacksdk”. Openstack. 02 de Maio de 2022. Site: https://docs.openstack.org/openstacksdk/latest

[9] “Evapotranspiration”. Wikipédia. 17 de Março de 2022. Site: https://en.wikipedia.org/wiki/Evapotranspiration

[10] "s3fs". GitHub. 17 de Março, 2022. Site: https://github.com/s3fs-fuse/s3fs-fuse

[11] "FUSE". Wikipédia. 17 de Março, 2022. Site: https://en.wikipedia.org/wiki/Filesystem_in_Userspace

[12] "Swift". Openstack. 17 de Março, 2022. Site: https://wiki.openstack.org/wiki/Swift

[13] "Openstack Wiki". Openstack. 17 de Março, 2022. Site: https://wiki.openstack.org/wiki/Main_Page

[14] "SAPS catalog". GitHub. 16 de Março, 2022. Site: https://github.com/ufcg-lsd/saps-catalog

[15] "SAPS dispatcher". GitHub. 16 de Março, 2022. Site: https://github.com/ufcg-lsd/saps-dispatcher

[16] "SAPS archiver". GitHub. 16 de Março, 2022. Site: https://github.com/ufcg-lsd/saps-archiver

[17] "Temporary storage". GitHub. 16 de Março, 2022. Site: https://github.com/ufcg-lsd/saps-archiver#temporary-storage

[18] “Arrebol”. GitHub. 24 de Maio, 2022. Site: https://github.com/ufcg-lsd/arrebol

[19] “SAPS scripts template”. 24 de Maio, 2022. Site: https://github.com/ufcg-lsd/saps-scripts-template

[20] “Restlet Framework”. 24 de Maio, 2022. Site: https://restlet.talend.com/

[21] “Round-robin scheduling”. 24 de Maio, 2022. Site: https://en.wikipedia.org/wiki/Round-robin_scheduling

[22] “Openstack SDK connection”. 02 de Maio de 2022. Site: https://docs.openstack.org/openstacksdk/latest/user/connection.html#openstack.connection.Connection

[23] “Docker Hub”. 30 de Maio de 2022. Site: https://hub.docker.com/

[24] “Arrebol com implementações do TCC”. 15 de Agosto de 2022. Site: https://github.com/thiagoyeds/tcc-arrebol/tree/tcc

[25] “SAPS Archiver com implementações do TCC”. 15 de Agosto de 2022. Site: https://github.com/thiagoyeds/tcc-saps-archiver/tree/tcc

[26] “SAPS Scheduler com implementações do TCC”. 15 de Agosto de 2022. Site: https://github.com/thiagoyeds/tcc-saps-scheduler/tree/tcc

[27] “Docker hub do script de inputdownload utilizado no experimento do TCC”. 15 de Agosto de 2022. Site: https://hub.docker.com/r/thiagoyeds/inputdownloader

[28] “Docker hub do script de preprocessing utilizado no experimento do TCC”. 15 de Agosto de 2022. Site: https://hub.docker.com/r/thiagoyeds/preprocessor

[29] “Docker hub do script de processing utilizado no experimento do TCC”. 15 de Agosto de 2022. Site: https://hub.docker.com/r/thiagoyeds/worker