Pontifícia Universidade Católica
do Rio de Janeiro

Pedro Mendonça Pinto Rocha

Melhoria de tempo na execução de workflows científicos distribuídos baseada na localização informada de arquivos

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio.

Orientadora: Profa. Noemi de La Rocque Rodriguez

Rio de Janeiro
Abril de 2018

PUC-Rio - Certificação Digital Nº 1512343/CA

Pontifícia Universidade Católica
do Rio de Janeiro

Pedro Mendonça Pinto Rocha

Melhoria de tempo na execução de workflows científicos distribuídos baseada na localização informada de arquivos

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Profa. Noemi de La Rocque Rodriguez
Orientadora
Departamento de Informática – PUC-Rio

Prof. Roberto Ierusalimschy
Departamento de Informática – PUC-Rio

Prof. Markus Endler
Departamento de Informática – PUC-Rio

Prof. Márcio da Silveira Carvalho
Coordenador Setorial do Centro Técnico Científico – PUC-Rio

Rio de Janeiro, 25 de Abril de 2018

PUC-Rio - Certificação Digital Nº 1512343/CA

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador.

Pedro Mendonça Pinto Rocha

Graduado em Ciência da Computação pela Universidade Federal do Rio de Janeiro em 2012

PUC-Rio - Certificação Digital Nº 1512343/CA

Ficha Catalográfica

Mendonça Pinto Rocha, Pedro

Melhoria de tempo na execução de workflows científicos distribuídos baseada na localização informada de arquivos / Pedro Mendonça Pinto Rocha; orientador: Noemi de La Rocque Rodriguez. – Rio de Janeiro: PUC-Rio, Departamento de Informática, 2018.

v., 50 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Workflow científico distribuído;. 3. FUSE;. 4. localização de arquivo;. 5. transferência antecipada de arquivos;. I. de La Rocque Rodriguez, Noemi. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

PUC-Rio - Certificação Digital Nº 1512343/CA

A todos os alunos que ainda terei.

Agradecimentos

Primeiramente, agradeço à minha orientadora Professora Noemi Rodriguez, por toda sua paciência, dedicação ao longo dessa jornada e por sempre acreditar em mim, mesmo quando eu mesmo não acreditava. Não fosse sua sensibilidade, compreensão e liderança, eu certamente não teria conseguido chegar aqui.

Agradeço aos meus pais, que me sempre me deram todo o apoio que eu precisei, principalmente nesta reta final, foram fundamentais para tornar o impossível possível.

Agradeço à minha amiga Carol, que teve um papel fundamental para que eu não desistisse, e me apoiou nas horas fundamentais. Sua amizade, sem dúvida nenhuma, foi uma dos melhores presentes que esse mestrado me trouxe.

Agradeço à Cris, que no momento fundamental, me permitiu ter a paz de espírito e a tranquilidade para conseguir terminar.

Agradeço ao amigo Rodrigo, sempre me fazendo companhia nas longas horas até tarde na PUC.

Agradeço aos amigos Daniel, Felipe, Paulo e Nino, por me acolherem desde o início, por todas as horas dedicadas a PAA, e inúmeras outras matérias.

Agradeço a todos os meus amigos e amigas maravilhosos que tenho, pela paciência, compreensão e, acima de tudo, apoio, nesse meu momento de isolamento.

Agradeço à equipe de suporte do DI, por toda paciência e presteza com minhas inúmeras requisições.

Agradeço ao Tecgraf por me dar todas as oportunidades para que eu pudesse obter esse título.

Agradeço à PUC-Rio e seus excelentes professores, pela excelente formação acadêmica e por me dar a liberdade de aproveitar toda sua infra-estrutura dentro da minha disponibilidade de tempo.

Agradeço à CAPES pelo auxílio financeiro durante minha pesquisa.

PUC-Rio - Certificação Digital Nº 1512343/CA

Resumo

Mendonça Pinto Rocha, Pedro; de La Rocque Rodriguez, Noemi. **Melhoria de tempo na execução de workflows científicos distribuídos baseada na localização informada de arquivos**. Rio de Janeiro, 2018. 50p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Na execução de workflows científicos distribuídos, a principal forma de passar os dados entre os nós de execução do workflow é utilizando arquivos. Quando os arquivos são grandes, uma parte considerável do tempo de execução do workflow é gasto transferindo os arquivos entre o servidor de armazenamento compartilhado e os nós de execução. Este trabalho propõe uma estratégia de transferência de arquivos diretamente entre os nós de execução, antecipando as necessidades da próxima etapa do workflow e diminuindo a sobrecarga com tráfego dos arquivos para os servidores de armazenamento. O trabalho analisa cenários em que a estratégia se mostra vantajosa e em quais não.

PUC-Rio - Certificação Digital Nº 1512343/CA

Palavras-chave

Workflow científico distribuído; FUSE; localização de arquivo; transferência antecipada de arquivos;

Abstract

Mendonça Pinto Rocha, Pedro; de La Rocque Rodriguez, Noemi (Advisor). **Lowering the execution time of scientific distributed workflows based on informed file location**. Rio de Janeiro, 2018. 50p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

For distributed scientific workflows the main method of sharing data between the execution nodes is through files. When those files are large, a substantial portion of the workflow's execution time is spent transferring the files between the storage server and the execution nodes. This work proposes a strategy for transfering the files directly between the execution nodes, antecipating the requirements of the next step of the workflow and lowering the overhead from transfering the files to and of the storage server. This dissertation analises scenarios in which this strategy shows to be advantajous and in which it doesn't.

PUC-Rio - Certificação Digital Nº 1512343/CA

Keywords

Distributed scientific workflow; FUSE; file location; antecipated file transfer;

Sumário

1 Introdução **13**
1.1 Objetivos 14
1.2 Organização da dissertação 14

2 Workflows científicos e padrões de acessos de dados **15**
2.1 Padrão de acesso de dados 16
2.1.1 Sequencial (Pipeline) 16
2.1.2 Distribuição (Broadcast) 17
2.1.3 Partição (Scatter) 17
2.1.4 Coleta (Gather) 17
2.2 Exemplos de workflows científicos 17
2.2.1 Montage 18
2.2.2 CyberShake 18
2.2.3 Broadband 19
2.3 Oportunidades de melhoria 20
2.3.1 Análise preliminar 21

3 Trabalhos relacionados **23**
3.1 WOSS: Workflow Optimized Storage System 23
3.2 pNFS e Lua 24
3.3 CONFUGA 25
3.4 AME e AMFS 26

4 Sistema de Transferência Antecipada - STA **27**
4.1 Estratégia 27
4.2 Modelo 28
4.3 Funcionamento 28
4.4 Implementação 29
4.4.1 FUSE – Filesystem in Userspace 30
4.4.1.1 Arquitetura FUSE 30
4.4.2 Fila de tarefas e Pool de Threads 31
4.4.3 Identificando o destino dos arquivos 32
4.4.4 Leitura com espera 32
4.5 Comentários adicionais 33

5 Testes e Resultados **35**
5.1 Testes de escrita 35
5.2 Padrões de acesso a dados 37
5.2.1 Sequencial 38
5.2.2 Coleta 39
5.2.3 Transmissão 40
5.2.4 Partição 41
5.2.5 Conjunção dos Padrões 42
5.3 Testes de processamento 42

PUC-Rio - Certificação Digital Nº 1512343/CA

5.4 Comentários finais 44

6 Conclusão e Trabalhos Futuros **46**

Referências Bibliográficas **47**

A Testes de viabilidade do FUSE **49**

PUC-Rio - Certificação Digital Nº 1512343/CA

Lista de figuras

2.1 Pipeline 16
2.2 Broadcast 17
2.3 Scatter 17
2.4 Gather 18
2.5 Representação do Montage Workflow. A figura destaca os padrões *gather*, *scatter* e *pipeline*. Imagem obtida de `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator` em Março/2018 19
2.6 Representação do CyberShake Workflow. Imagem obtida de `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator` em Março/2018 20
2.7 Representação do Broadband Workflow. Imagem obtida de Juve et al.(13), p. 687 20

4.1 Arquitetura FUSE. Imagem obtida de `https://en.wikipedia.org/wiki/File:FUSE_structure.svg` em Março/2018 30

PUC-Rio - Certificação Digital Nº 1512343/CA

Lista de tabelas

2.1 Cópia e leitura de um arquivo de 3GB 21
2.2 Cópia e leitura de um arquivo de 3GB em estações distintos 21

5.1 Resultados de escrita 36
5.2 Resultados de escritas com múltiplas cópias concorrentes 36
5.3 Resultados dos testes do padrão sequencial 38
5.4 Resultados dos testes do padrão de coleta 39
5.5 Resultados dos testes do padrão de transmissão 40
5.6 Resultados dos testes do padrão de transmissão 41
5.7 Resultados das variações dos tempos de processamento 43

PUC-Rio - Certificação Digital Nº 1512343/CA

PUC-Rio - Certificação Digital Nº 1512343/CA

Educação é uma descoberta progressiva de nossa própria ignorância.

Voltaire.

1 Introdução

Aplicações científicas de áreas como astronomia, genética e sismologia se utilizam da execução de complexos *workflows*, muitas vezes lidando com volumes de informações grandes demais para serem processados por um único computador. Como consequência, novos desafios surgiram, tais como execução paralela desses esquemas complexos ou movimentação eficiente dos dados, em busca de diminuir o tempo computacional necessário para obtenção de resultados, minimizando custos e acelerando a possibilidade de novas descobertas.

Para esse fim, inúmeras ferramentas e soluções foram criadas com o objetivo de auxiliar as pesquisas que dependem da execução de *workflows* científicos. Tais ferramentas podem ser classificadas, de forma geral, em dois grupos: os gerenciadores de *workflows*, responsáveis por definir e orquestrar a execução de um desses esquemas; e os sistemas de arquivos distribuídos focados em execução de *workflows*.

Contudo, o compartilhamento dos dados entre os nós de execução de um *grid* ou um *cluster*, assim como o transporte dos dados gerados pelas tarefas do *workflow* ainda possuem problemas que não estão totalmente solucionados. Embora o compartilhamento, propriamente, dos dados com os nós de execução não seja mais uma dificuldade — o protocolo NFS, que já faz parte do *kernel* Linux desde o início dos anos 2000, se tornou um padrão de fato para compartilhamento de arquivos em uma rede —, ainda há um gargalo que ocorre quando há inúmeros processos lendo e escrevendo dados remotamente. Dado que *workflows* são usualmente executados em ambientes paralelos e distribuídos, acabam sendo, em muitos casos, bastante afetados por esse gargalo. Se todas as tarefas do *workflow* precisam ler seus arquivos de um servidor remoto, e tem de escrever seus resultados também neste servidor, a cada operação de leitura ou escrita é necessário que a tarefa fique esperando o dado ser transferido pela rede, gerando um atraso nas execuções e diminuindo sua eficiência.

Nesse trabalho, propomos uma estratégia para minimizar esse atraso e a avaliamos utilizando um protótipo que desenvolvemos para isto.

1.1

PUC-Rio - Certificação Digital Nº 1512343/CA

Objetivos

Esse trabalho tem como propósito analisar as potenciais reduções no tempo de execução de um *workflow* científico distribuído, através da utilização dos discos locais aos nós de execução como armazenamento temporário dos dados gerados por cada passo executado. Ao conseguir fazer as operações de leitura e escrita em um disco local em vez de um remoto, uma tarefa pode ser seu tempo de execução reduzido por não precisar esperar o fim da transferência dos dados a cada operação. No entanto, para que seja possível se beneficiar de leituras local, também é necessário antecipar a transferência dos dados de entrada de uma tarefa, para que no início de sua execução, ao menos parte deles já estejam disponíveis no disco local do nó de execução destinado a ela.

Visando alcançar esse objetivo, nós propomos uma estratégia que consiste em antecipar as transferências dos arquivos intermediários, efetuando-as paralelamente à execução da tarefa produtora dos dados, conforme os dados forem gravados no disco local ao nó de execução. Para decidir o destino das transferências, a estratégia depende que seja fornecida uma configuração contendo as informações de distribuição de cada arquivo, que podem ser obtidas através da identificação dos padrões de acesso aos dados, existentes na cadeia de dependência do *worfklow*.

A fim de avaliar os potenciais ganhos da estratégia apresentada, propomos e apresentamos um protótipo do Sistema de Transferência Antecipada, um sistema de arquivos distribuídos que se encarrega de fazer as transferências, paralelamente à execução das tarefas produtoras, dos arquivos para seus nós de execução destino onde servirão de entrada para os próximos passos da execução. Em seguida foi realizada uma bateria de testes, explorando os potenciais ganhos para cada padrão de acesso aos dados, e também uma análise dos resultados.

PUC-Rio - Certificação Digital Nº 1512343/CA

1.2 Organização da dissertação

O presente trabalho está organizado da seguinte forma: no Capítulo 2 apresentamos uma definição mais formal de *workflow* científico, definimos quais são os padrões de acesso a dados explorados e apresentamos alguns exemplos de *workflows* científicos existente. No Capítulo 3 mostramos os trabalhos relacionados que foram utilizados como referência. No Capítulo 4 descrevemos a estratégia proposta e apresentamos o protótipo do Sistema de Transferência Antecipada, explicando seu funcionamento e detalhando alguns aspectos de sua implementação. No Capítulo 5 apresentamos os testes efetuados e os resultados obtidos, assim como uma análise em cima destes resultados. Por fim, no Capítulo 6 apresentamos conclusões e trabalhos futuros.

2
Workflows científicos e padrões de acessos de dados

De acordo com Silva et al.(5)(2017, p. 1, tradução nossa), "Tipicamente, *workflows* científicos são descritos como um grafo direcionado acíclico (*directed acyclic graph*, ou DAG), cujos vértices representam tarefas de um *workflow* que são conectadas através de arestas de fluxo de dados, estabelecendo, assim, uma execução sequencial ou paralela dos vértices"[1]

Dentro do ambiente da pequisa científica há inúmeros trabalhos que consistem em executar uma sequência complexa de algoritmos sobre um volume grande de dados para se alcançar o resultado pretendido, descrevendo, assim, *workflows* científicos.

Dentro deste contexto, é necessário se fazer distinção entre três conceitos distintos: o gerenciador de execução de *workflows* científicos, o *workflow* científico e uma instância de um *workflow* científico. O *workflow* científico se refere ao conjunto de metadados que definem o modelo da ordem de execução dos algoritmos e procedimentos que devem ser efetuados sobre a massa de dados; a instância de um *workflow* científico se refere a um conjunto de dados específicos a serem processados e também as definições exatas de quantos algoritmos e procedimentos devem ser instanciados para uma execução; e o gerenciador de execução de *workflows* se refere às ferramentas necessárias para de fato instanciar as tarefas e processar os dados de uma instância. Um *workflow* científico pode possuir inúmeras instâncias distintas, variando o número de ocorrências de cada uma das tarefas necessárias, mas de forma geral, a ordem de execução e a lista de dependências de cada tarefa se mantém a mesma em todas as instâncias.

Com o crescimento e popularização dos paradigmas de computação paralela e distribuída, a adoção de *workflows* tem se tornado uma alternativa comum no campo da computação científica para abordar problemas que possuem estruturas complexas e precisam tratar dados massivos.

A utilização de *workflows* apresenta inúmeras vantagens. Pelo fato da comunicação entre as tarefas tipicamente se dar através de arquivos padrão, é possível a utilização de programas legados com algoritmos sofisticados, mas

[1]Typically, scientific workflows are described as a directed-acyclic graph (DAG), whose nodes represent workflow tasks that are linked via dataflow edges, thus prescribing serial or parallel execution of nodes.

PUC-Rio - Certificação Digital Nº 1512343/CA

que já sejam utilizados rotineiramente em suas áreas de conhecimento, e que comumente obtêm seus dados de entrada e geram seus dados de saída por arquivos. Também torna fácil ter um ambiente de execução compatível com o de desenvolvimento. Eleva, ainda, a tolerância a falhas das execuções, já que cada estágio do *workflow* gera um arquivo temporário que pode persistir enquanto seu conteúdo ainda não tiver sido todo consumido. Todas essas características tornam o modelo de *workflow* muito atrativo para aplicações de computação científica que precisam tratar grandes volumes de dados e cujo tempo de processamento é grande o bastante para que suas execuções sejam mais suscetíveis a falhas e interrupções não previstas. Neste sentido, esforços que visem diminuir esses tempos de processamento são relevantes para permitirem que mais dados sejam processados num mesmo tempo e que as execuções fiquem menos sujeitas a falhas.

2.1 Padrão de acesso de dados

De forma geral, um *workflow* pode ser representado por um grafo direcionado acíclico (*directed acyclic graph*, ou DAG), ou um não-DAG(16). O foco deste trabalho é em aplicações de *workflow* científico que possam ser representadas por um DAG. Ao analisar o grafo que representa um *workflow*, é possível observar como os dados fluem entre as tarefas. Certos arquétipos de como os dados passam de uma tarefa à outra são padrões que se repetem em inúmeros *workflows* científicos (4, 17). A seguir citamos alguns.

2.1.1 Sequencial (Pipeline)

No padrão sequencial, ou *pipeline*, o dado flui de uma tarefa para a outra sequencialmente, ou seja, o arquivo de saída de uma tarefa vira o arquivo de entrada da próxima. A figura 2.1 contém uma representação gráfica deste padrão.

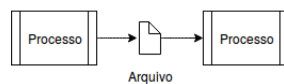

Figura 2.1: Pipeline

Esse é o padrão mais comum, pois descreve basicamente qualquer interação entre dois processos através de arquivos.

PUC-Rio - Certificação Digital Nº 1512343/CA

2.1.2
Distribuição (Broadcast)

No padrão de multitransmissão, ou *broadcast*, o mesmo arquivo gerado por uma tarefa é consumido por tarefas distintas e independentes entre si. A figura 2.2 contém uma representação gráfica deste padrão.

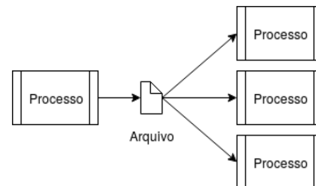

Figura 2.2: Broadcast

2.1.3
Partição (Scatter)

No padrão de partição, ou *scatter*, uma tarefa gera um conjunto de arquivos e cada arquivo é utilizado como entrada de uma nova tarefa. Neste padrão em geral cada arquivo gerado será lido por uma nova tarefa distinta e independente. A figura 2.3 contém uma representação gráfica deste padrão.

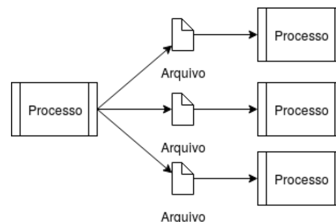

Figura 2.3: Scatter

PUC-Rio - Certificação Digital Nº 1512343/CA

2.1.4
Coleta (Gather)

No padrão de coleta, ou *gather*, um conjunto de tarefas gera um conjunto de arquivos, sendo o mais comum cada tarefa gerando um arquivo, que então são lidos por uma outra a tarefa ou um conjunto de tarefas trabalhando coletivamente. A figura 2.4 contém uma representação gráfica deste padrão.

2.2
Exemplos de workflows científicos

Nesta seção são descritos alguns *workflows* científicos.

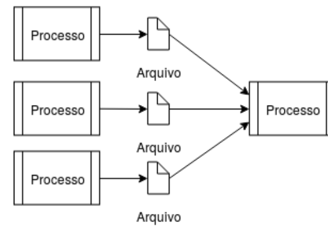

Figura 2.4: Gather

2.2.1 Montage

Na figura 2.5 pode-se ver a representação de uma instância do *Montage Workflow* (3). O *Montage* é uma aplicação de astronomia que utiliza inúmeras fotografias do espaço para compor uma visão uniformizada da região fotografada. A estrutura do *workflow* muda para acomodar variações o no número de arquivos de entrada, variando a quantidade de tarefas computacionais instanciadas.

Na figura 2.5 é possível identificar ocorrências dos padrões *pipeline*, *scatter* e *gather*. Uma análise feita por Juve et al.(13) demonstra que esse *workflow* gasta uma boa parte de sua execução com operações de entrada e saída. Nesta análise, os autores perceberam que essa aplicação, embora tenha passos com uso intensivo de CPU, as tarefas que duram mais tempo durante a execução são justamente as que fazem mais operações de entrada e saída mas não são as que tem maior uso de CPU. O passo que consome 27% do tempo tem uma utilização média de apenas 8% da CPU, mas são as tarefas que mais efetuam operações de leitura e escrita de arquivo, com leituras de aproximadamente 1GB em média e escritas de aproximadamente 0,8GB em média. Pode-se dizer que esse *workflow* possui é intensivo de operações de entrada e saída.

PUC-Rio - Certificação Digital Nº 1512343/CA

2.2.2 CyberShake

Na figura 2.6 pode-se ver a representação de uma instância do *CyberShake Workflow* (9). O *CyberShake* é utilizado pelo *SCEC (Southern California Earthquake Center)* para fazer uma projeção dos perigos de um terremoto em uma região. Apesar de sua estrutura mais simples, o *CyberShake* é utilizado para realizar quantidades significativas de computação em conjuntos de dados massivos.

Na análise realizada por Juve et al.(13), a aplicação se mostrou ter um uso intensivo de CPU e, embora o volume de dados lidos seja grande, não há um número alto de operações de entrada e saída. Mais de 97% do tempo da aplicação é gasto em um passo que tem uma utilização de CPU média de

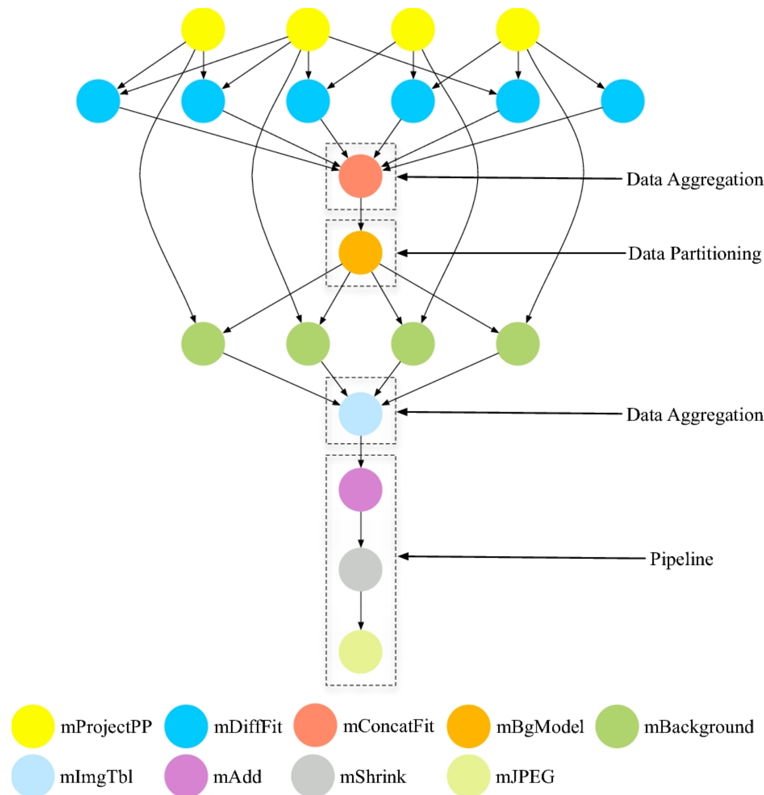

Figura 2.5: Representação do Montage Workflow. A figura destaca os padrões *gather*, *scatter* e *pipeline*. Imagem obtida de `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator` em Março/2018

PUC-Rio - Certificação Digital Nº 1512343/CA

92%, uma leitura média de 547MB, mas uma escrita média de apenas 0,02MB. Embora haja passos que gastem mais tempo com operações de entrada e saída, e tenham tempos de execução altos, estes têm menor peso no gasto total de tempo.

2.2.3 Broadband

Na figura 2.7 pode-se ver a representação de uma instância do *Broadband Worfklow*(14). O *Broadband* é uma plataforma criada para gerar dados realistas de terremotos para engenheiros que estudam o fenômeno, de forma a facilitar análises dos dados de terremotos sem que seja necessário coletar dados de um.

Na análise realizada por Juve et al.(13), a aplicação se mostrou de uso intensivo tanto de CPU quanto de operações de entrada e saída, sendo a tarefa que concentra o maior uso de tempo de CPU (55% do tempo) também é a que gera mais dados de saída, gerando arquivos de resultado com tamanho médio de 1,8GB por tarefa.

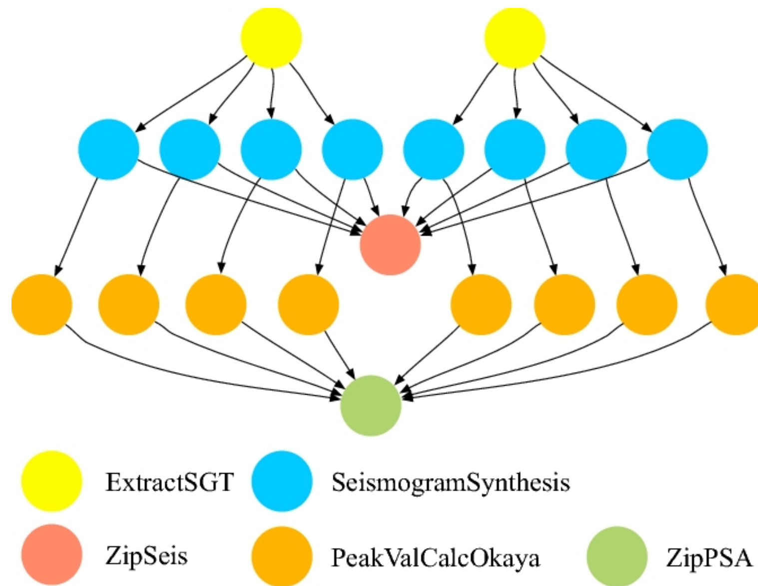

Figura 2.6: Representação do CyberShake Workflow. Imagem obtida de `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator` em Março/2018

PUC-Rio - Certificação Digital Nº 1512343/CA

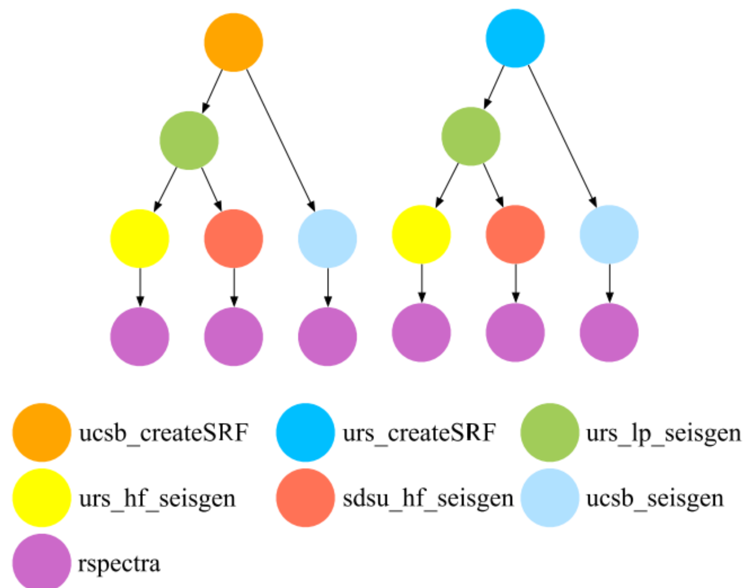

Figura 2.7: Representação do Broadband Workflow. Imagem obtida de Juve et al.(13), p. 687

2.3 Oportunidades de melhoria

Para *workflows* distribuídos cuja comunicação se dá através da escrita e leitura de arquivos, uma das questões enfrentadas é como compartilhar esses dados entre os nós de execução. Mesmo que fosse possível gravar localmente em

cada nó de execução uma cópia completa dos dados iniciais, conforme as tarefas terminassem os novos arquivos que fossem gerados localmente não estariam automaticamente disponíveis para os outros nós de execução. Para automatizar esse processo, a solução é utilizar algum sistema de arquivos distribuídos que possa obter e gravar os dados remotamente e que seja visível a todos os nós de execução. A escolha do sistema de arquivos distribuídos varia de acordo com o ambiente escolhido para a executar uma instância de um *workflow*. No entanto, essa abordagem pode gerar um gargalo para sistemas que forem muito intensivos em operações de entrada e saída. Se a cada passo o arquivo tiver de ser lido de um servidor remoto e escrito de volta também remotamente, a execução do *workflow* fica limitada pela velocidade de transferência da rede utilizada.

2.3.1 Análise preliminar

Simulações preliminares explicitam melhor essas diferenças. Os testes foram rodados o primeiro com todas as leituras e escritas locais e o segundo com todas as leituras e escritas remotas. Num cenário simples de cópia de um arquivo que em seguida é lido, foram obtidos os resultados apresentados na tabela 2.1.

PUC-Rio - Certificação Digital Nº 1512343/CA

Tabela 2.1: Cópia e leitura de um arquivo de 3GB

	Localmente	Remotamente
Tempo médio (s)	179,3	897,2
Desvio padrão (s)	5,3	1,69

Pode-se observar que ao efetuar todas as operações de leitura e escrita remotamente, o tempo de execução aumenta em 5 vezes.

Em seguida, avaliamos um cenário onde cada processo é executado numa estação distinta, e o arquivo copiado é sempre lido inicialmente de uma estação remota. No primeiro cenário, a cópia é gravada localmente na estação que executa a cópia, para então ser lida remotamente pelo processo seguinte, e no segundo cenário, a cópia já é gravada remotamente na estação que fará a leitura, permitindo que o processo seguinte faça apenas uma leitura local. Os resultados são apresentados na tabela 2.2.

Tabela 2.2: Cópia e leitura de um arquivo de 3GB em estações distintos

	Cópia local / Leitura remota	Cópia remota / Leitura local
Tempo médio (s)	626,6	514,0
Desvio padrão (s)	3,7	2,4

Os resultados demonstram uma redução de 30% do tempo de execução do primeiro cenário em comparação às operações feitas todas remotamente, e o segundo demonstra uma redução de 42%.

Baseado nisso, propomos uma estratégia que consiste em tentar antecipar as necessidades de execução de cada tarefa de forma que quando ela comece a ser executada, o arquivo já esteja disponível localmente na estação de execução. A seção 4.1 possui uma descrição melhor da estratégia. No capítulo 3 são descritos outros trabalhos que propuseram outras abordagens para diminuir o tempo de processamento de um *workflow*.

PUC-Rio - Certificação Digital Nº 1512343/CA

3
Trabalhos relacionados

Diversos trabalhos exploram e elaboram sobre padrões de acesso de dados de *workflows* e apresentam algumas propostas e abordagens para otimização. Neste capítulo descrevemos os trabalhos que serviram como ponto de partida para a pesquisa desenvolvida, e discorremos sobre como cada um deles diferem do trabalho apresentado.

3.1
WOSS: Workflow Optimized Storage System

Al-Kiswany et al(1) propõem o WOSS: Workflow optimized storage system — ou sistema de armazenamento otimizado para *workflows* em tradução livre —, um sistema de arquivos distribuído que utiliza o espaço ocioso do disco local de nós de execução pertencentes a um *grid* ou a um *cluster* para armazenar os arquivos e que foi planejado para ser utilizado na execução de *workflows* distribuídos. Por utilizar os discos locais dos nós de execução, o *WOSS* já oferece uma vantagem com relação a outras abordagens que só possuem alocação dos dados em servidores remotos, uma vez que, estatisticamente, ocorrerá de uma tarefa requisitar um arquivo que já está alocado localmente. Além disso, através da utilização de *tags* metadados, o sistema permite que o gerenciador do *workflow* consulte a localização física de um determinado arquivo e também forneça ao sistema dicas sobre os padrões de acessos a cada arquivo, de forma que seja possível alocar uma tarefa diretamente onde o dado já está sendo salvo. Então, no momento da criação de um arquivo, o gerenciador do *workflow* pode indicar qual o padrão de acesso a esse arquivo, e no momento de alocar uma tarefa, o gerenciador pode consultar onde o arquivo está localizado e usar essa informação na alocação. Neste caso, quem decide onde o arquivo será armazenado é o sistema de arquivos, e a ideia é que o gerenciador de *workflows* apenas use essa informação na hora de fazer a alocação de tarefas.

Essa abordagem assume que o sistema de arquivos já conhece alguns padrões de dados e apenas aplica o padrão conhecido indicado pela *tag* metadado. Desta forma, o gerenciador de *workflow* fica limitado às configurações já implementadas pelo sistema de arquivos, o que nem sempre pode ser suficiente para atender a todas as necessidades da instância do *workflow*

PUC-Rio - Certificação Digital Nº 1512343/CA

sendo executada, uma vez que um determinado padrão pode não ter sido implementado, impedindo uma oportunidade de otimização.

O trabalho também apresenta alguns resultados de testes, tanto sintéticos quanto com aplicações reais. Nos testes sintéticos, os autores testaram os ganhos de tempo para cada um dos padrões de dados descritos (*pipeline, scatter, broadcast* e *reduce*[1]), e observaram ganhos substanciais em todos os casos: 10x mais rápido se comparado com o mesmo teste utilizando NFS para os padrões *pipeline* e *scatter*, e 4x mais rápidos para os padrões *broadcast* e *reduce*. Nos testes reais, o trabalho apresenta um ganho de 40% na execução do BLAST Workflow (2) e um ganho de 30% na execução do Montage Workflow.

Embora essa abordagem possibilite trazer ganhos, como é o sistema de arquivos que define o destino de cada arquivo, e não o gerenciador do *workflow*, nem sempre o nó de execução escolhido para armazenar um arquivo atenderá eventuais requisitos computacionais — quantidade mínima de memória, número de processadores, programas instalados — necessários para executar a tarefa que irá consumir o respectivo arquivo. Desta forma, nem sempre o gerenciador de *workflow* será capaz de alocar uma tarefa no nó de execução onde seu arquivo de entrada está armazenado, limitando, assim, as oportunidades de ganhos no tempo de execução.

Apesar do trabalho de Al-Kiswany et al(1) possuir muitos paralelos com a pesquisa aqui apresentada, não foi possível utilizá-lo como base para darmos continuidade por impossibilidade de obter uma base de código funcionando, mesmo após tentativa de contato com os autores.

3.2 pNFS e Lua

Grawinkel et al(10) propõem uma abordagem análoga à citada na seção anterior, no sentido que são usados para definir políticas específicas de alocação para arquivos. Em vez de utilizar tags metadados, o trabalho propõe que seja possível o cliente do sistema definir, para cada arquivo, seu leiaute interno de alocação entre os nós disponíveis. Isso permite que seja feito um controle fino de alocação dos arquivos, oferecendo bastante flexibilidade ao cliente que estiver utilizando o sistema de arquivos. A abordagem descrita sugere uma implementação do protocolo *pNFS*[2] como base do sistema de arquivos e a utilização de um módulo Lua para o *kernel* do sistema operacional(15).

Embora o trabalho de Grawinkel et al não fale especificamente do cenário para execução de *workflows*, sua proposta permite que o gerenciador de um

[1] O padrão *reduce* é similar ao padrão de coleta.

[2] *pNFS* é uma versão paralelizada do *Network File System* (*NFS*), definida na versão 4.1 (RFC 5661 – `https://tools.ietf.org/html/rfc3010`), em janeiro de 2010.

PUC-Rio - Certificação Digital Nº 1512343/CA

workflow tenha controle total sobre a forma como os dados serão armazenados e distribuídos dentro dos nós de armazenagem do sistema de arquivos, ao injetar *script* Lua no *kernel* do sistema operacional que especifica qual o layout de armazenamento dos arquivos utilizados, tendo a oportunidade de decidir a localização específica de cada arquivo baseada nos requisitos computacionais da tarefa que irá consumir o conteúdo do respectivo arquivo.

O trabalho apresentado, no entanto, oferece apenas um estudo de viabilidade da ideia, não possuindo nenhuma implementação concreta de sua proposta. Os únicos testes apresentados eram relativos à viabilidade de execução de *scripts* Lua dentro do *kernel*. Além disso, apesar do padrão *pNFS* já existir há alguns anos, sua adoção tem sido lenta e não foi possível encontrar nenhuma solução de código aberto que implementasse o padrão completamente para que pudéssemos tentar aplicar a ideia apresentada. Apesar de ser um caminho válido, ter de implementar um sistema de arquivos no padrão *pNFS* do zero implicaria em ter de resolver inúmeras questões internas de alocação de dados, o que nos distanciaria do nosso foco da execução de *workflows*.

PUC-Rio - Certificação Digital Nº 1512343/CA

3.3 CONFUGA

Donnelly & Thain(8) e desenvolveram uma solução chamada CONFUGA, que incorpora a alocação de tarefas e o sistema de arquivos distribuídos em uma só ferramenta. Na alocação da tarefa, o gerenciador do *workflow* deve informar também quais arquivos serão consumidos como entrada e quais serão gerados como saída. Com base nessa informação, a ferramenta é capaz de identificar quais arquivos serão necessários em quais servidores de execução e, portanto, é capaz de alocar o dado e a tarefa na mesma localidade.

Donnelly & Thain fizeram uma análise de possíveis reduções no tempo de execução de *workflows* ao se balancear estratégias de distribuição dos dados entre os nós de execução entre sob demanda (ou *pull*) e antecipadamente (ou *push*). Os testes demonstram que é possível se obter ganhos entre 48% e 77% se comparados com a utilização de só uma das duas estratégias, mas não apresenta nenhum teste em comparação com outros sistemas de arquivos distribuídos.

Ao integrar o gerenciador do *workflow* e o sistema de arquivos distribuídos em uma só ferramenta, as oportunidades de se otimizar automaticamente a execução do *workflow* são maximizadas, uma vez que se tem controle total dos recursos disponíveis. No entanto, essa solução dificulta a adoção de pesquisas que já possuam seus *worfklows* definidos em outras ferramentas de execução. Por existirem inúmeras pesquisas que utilizam outros gerenciadores de *workflow*, é importante explorar possibilidades que possam ser acopladas a essa base de

código existente, e é justamente esse contexto que nossa pesquisa explora.

3.4 AME e AMFS

Zhang et al(17) propõem uma solução que abrange desde o gerenciador do *workflow*, através das ferramentas AME e AMFS, passando pelo alocador de tarefas, até o sistema de arquivos distribuídos. No entanto, diferentemente do *Confuga*, o AME, que é o gerenciador e executor de *workflows*, e o AMFS, que é o sistema de arquivos distribuídos, podem ser usados separadamente. Por ter controle total da execução de todas as etapas do processamento, a ferramenta tem oportunidades de aplicar otimizações em diferentes camadas e momentos da execução. Além de poder identificar os padrões de acesso a dados do *workflow* sendo executado, é possível tomar decisões sobre a forma de replicação de um dado, seja através da cópia do mesmo, seja duplicando a tarefa que gera aquele dado.

Apesar da ferramenta apresentar vantagens como a possibilidade de automatização da tomada de decisão sobre cada otimização a ser aplicada, ela abre mão da compatibilidade *POSIX*[3], limitando seu uso por dificultar a utilizar de aplicações já existentes que se utilizem desta interface padrão para acesso a arquivos.

Por propor uma solução completa, todas as análises de desempenho levam em consideração não só o desempenho do sistema de arquivos, como também do gerenciador e executor de *workflows*. Além disso, também não oferecem nenhum comparativo com a utilização de outros sistemas de arquivos distribuídos, o que torna os resultados difíceis de serem analisados no contexto da pesquisa aqui apresentada.

[3] *Portable Operating System Interface* (*POSIX*), ou interface portável entre sistemas operacionais em tradução livre, é um conjunto de normas definidas pela IEEE 1003.1 (`http://standards.ieee.org/findstds/standard/1003.1-2017.html`) para manutenção e compatibilidade entre sistemas operacionais.

PUC-Rio - Certificação Digital Nº 1512343/CA

4
Sistema de Transferência Antecipada - STA

Neste capítulo vamos apresentar em detalhes o trabalho desenvolvido que chamamos de Sistema de Transferência Antecipada, ou STA, desde a definição de sua arquitetura até alguns detalhes de implementação que forem relevantes.

4.1
Estratégia

Como já descrito anteriormente, há alguns trabalhos, que, como o nosso, tentam abordar o gargalo gerado pela transferência de arquivos entre os nós de execução e o local onde o dado está armazenado de fato. Como todos os nós de execução precisam ter acesso aos arquivos que serão processados, uma solução comum é obtê-los pela rede, em geral se utilizando de sistemas de arquivos remotos ou distribuídos, sendo a solução mais trivial a utilização de um NFS que aponte para um servidor de arquivos. No entanto, essa abordagem trivial se mostra muitas vezes ineficiente, uma vez que todas as operações de entrada e saída acabam ficando sujeitas à capacidade da banda de rede para fazer a transferência destes arquivos.

Algumas abordagens tentam minimizar essas transferências com sistemas de arquivos distribuídos que utilizam os próprios discos das máquinas de execução como locais de armazenamento. Essa abordagem, porém, não garante que o arquivo de entrada necessário a uma tarefa já esteja armazenado no nó de execução designado a esse processamento. Então embora estatisticamente essa coincidência possa ocorrer, resultando em alguma redução no tempo de execução, entendemos que ainda há espaço para melhorias.

Em seguida, direcionamos nossos esforços para permitir que, quando uma tarefa inicie, seus arquivos de entrada já estejam presentes no disco local do nó de execução designado, diminuindo o tempo de leitura do arquivo, uma vez que não seria mais necessário obtê-lo de um servidor remoto. Nosso objetivo é avaliar se essa abordagem resulta em um ganho real de execução, se comparada à alternativa trivial.

Para tanto, nossa ideia consiste em tentar direcionar o arquivo de saída de uma tarefa para o próximo nó de execução onde esse dado será necessário para a tarefa seguinte, de acordo com o *workflow*. Para isso funcionar, é necessário que

PUC-Rio - Certificação Digital Nº 1512343/CA

o gerenciador de *worfklow* seja capaz de determinar antecipadamente em quais nós de execução as próximas tarefas serão executadas para que os arquivos sejam corretamente direcionados. Essa abordagem requer gerenciadores de *workflow* capazes de planejar pelo menos dois passos a frente, pois assim que uma tarefa inicia, ela pode já começar a gerar seu arquivo de saída e isso exige que seu destino já tenha sido definido.

4.2 Modelo

Para o contexto deste trabalho estamos considerando as instâncias de *workflow* que rodam em ambientes nos quais os computadores utilizados para execução possuem discos locais e a escrita local é mais rápida que a escrita remota. Essa é uma característica recorrente em laboratórios com estações de trabalho comuns, então todos os grupos que utilizem um ambiente de execução com esta configuração poderão, potencialmente, se beneficiar com o trabalho aqui apresentado.

Também estamos considerando apenas os *workflows* que não fazem escritas sobrepostas nos arquivos gerados, ou seja, que nenhum trecho de um arquivo de saída seja reescrito enquanto estiver sendo gerado. Por fazermos cópias assíncronas e paralelas por múltiplas threads, não há como garantir que a mesma ordem de escrita seja reproduzida nas cópias, o que poderia causar uma alteração no resultado final do arquivo caso algum trecho seja reescrito enquanto ele estiver sendo transferido.

PUC-Rio - Certificação Digital Nº 1512343/CA

4.3 Funcionamento

Considerando um contexto em que todos os nós de execução tem uma visão consistente do mesmo sistema de arquivos distribuído, decidimos que o melhor caminho para alcançar nossos objetivos seria efetuar a transferência dos dados pelo sistema de arquivos, de forma que isso ficasse transparente para as tarefas a serem executadas. Nesta camada é possível redirecionar a escrita do arquivo para um ponto de montagem remoto fazendo com que o arquivo seja gravado efetivamente já no nó de execução destino. Contudo, para evitar que a tarefa produtora fique presa esperando o término da transferência, primeiro é feita uma escrita num cache no disco local, e em paralelo um conjunto de *threads* se encarrega de fazer a transferência do dado salvo neste cache. Aqui assumimos que configuração dos padrões de acesso aos dados associados a cada arquivo, e também quais os nós de execução disponíveis, se encontram num arquivo que é consultado pelo STA para realizar as escritas remotas antecipadas.

Para os testes realizados, o arquivo de configuração consiste de uma tabela Lua que contém uma lista de destinos por nome de arquivo.

A arquitetura descrita depende que dois pré-requisitos sejam atendidos. Primeiro, é necessário que todos os nós de execução possuam uma capacidade de armazenamento local grande o suficiente para armazenar os arquivos intermediários. Também é necessário que todos os nós de execução sejam capazes de acessar os outros nós através da rede. Como estamos supondo um cenário de utilização de um laboratórios com estações de trabalho comuns funcionando como um *grid* de execução, ou ainda da utilização de *clusters* que possuam discos locais, essas duas características são comuns de se encontrar.

4.4 Implementação

Para efetuar testes e validar a estratégia proposta nesse trabalho, foi desenvolvido um protótipo do STA na linguagem C, utilizando FUSE (*File System in User Space*) (11) para implementar a camada de sistema de arquivos, a biblioteca *pthreads* para implementar a *pool* de *threads* encarregada de efetuar as tarefas de cópias dos arquivos e a linguagem Lua para consulta dinâmica sobre os locais de escrita de cada arquivo. Além disso, todas as partições remotas foram montadas utilizando SSHFS (12), por não necessitar de privilégios de administrador, ao contrário de partições NFS. Há uma sobrecarga potencial pelo uso do SSHFS em vez do NFS, mas buscamos identificar o ganho comparativo, então essa sobrecarga não prejudica os resultados.

O fluxo do STA se dá da seguinte forma: quando uma tarefa efetua a abertura de seu arquivo para escrita, o STA consulta a configuração para descobrir qual deve ser o destino ou os destinos do arquivo, cria a cópia do cache local e também já cria todas as cópias remotas; num segundo momento, quando chegam os comandos de escrita, o sistema primeiro escreve o dado localmente e em seguida cria tarefas internas numa fila para serem executadas por um conjunto de *threads* e que efetuarão a cópia do arquivo propriamente; por fim, ao receber o pedido de fechamento do arquivo, um marcador é sinalizado indicando a solicitação e o arquivo é fechado quando a última tarefa de cópia é executada.

Esse fluxo pode fazer com que, ao final de uma tarefa, embora o gerenciador do *workflow* entenda que já pode iniciar a tarefa seguinte que vai ler o dado recém gerado, o arquivo ainda não tenha sido completamente copiado para a máquina onde essa tarefa será executada. Para garantir que nenhuma tarefa identificasse o final do arquivo sem que ele tivesse sido completamente copiado, foi necessário implementar também uma lógica de leitura com espera.

PUC-Rio - Certificação Digital Nº 1512343/CA

PUC-Rio - Certificação Digital Nº 1512343/CA

4.4.1 FUSE – Filesystem in Userspace

FUSE é uma interface para que programas no espaço de usuário possam exportar um sistema de arquivos para o *kernel* do sistema operacional. O FUSE possui um módulo *kernel* e uma biblioteca no espaço de usuário. É através desta biblioteca que o programa do usuário acessa o módulo *kernel*, permitindo que o sistema operacional reconheça aquela aplicação como um sistema de arquivos. FUSE permite que qualquer usuário implemente seu próprio sistema de arquivos customizado, sem precisar fazer nenhuma alteração ao *kernel*. Isso é possível pois os sistemas operacionais utilizam um sistema de arquivos virtual, ou *VFS* — *Virtual File System*, e esta camada se encarrega de direcionar qualquer chamada à instância de sistema de arquivos designada para gerir o conteúdo do diretório solicitado. Ao montar um diretório usando FUSE, todas as chamadas que manipulam os arquivos deste diretório são redirecionadas pelo VFS para o módulo FUSE, que então se encarrega de chamar as funções definidas pelo usuário em sua implementação das interfaces FUSE.

4.4.1.1 Arquitetura FUSE

Para se montar um sistema de arquivos implementado em cima do FUSE não se utiliza diretamente o comando *mount*. O programador escreve um programa utilizando a API FUSE fornecida e gera um binário executável que, ao ser executado, efetua a montagem do sistema. Isso é necessário pois esse processo é que ficará responsável por atender as requisições feitas ao sistema de arquivos que utiliza o FUSE. A figura 4.1 ilustra o cenário descrito.

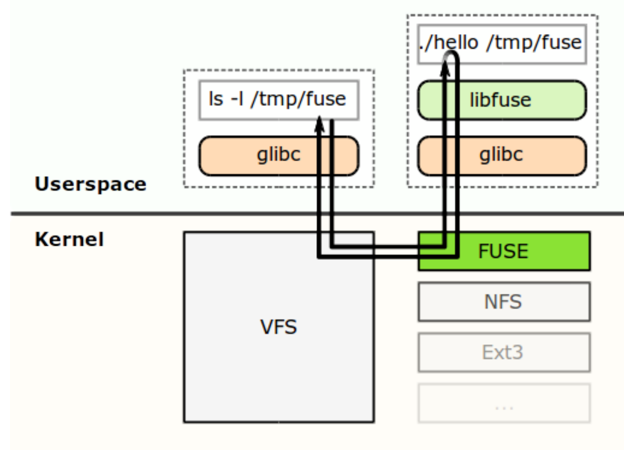

Figura 4.1: Arquitetura FUSE. Imagem obtida de `https://en.wikipedia.org/wiki/File:FUSE_structure.svg` em Março/2018

No exemplo da Figura 4.1, o comando *ls* faz uma chamada ao sistema operacional, que o delega ao *VFS*. Esse então identifica que o diretório requisitado é um sistema de arquivos que utiliza *FUSE* e então repassa a chamada ao programa feito pelo usuário, que a processa e responde. A resposta é então devolvida até retornar ao comando original. Fica evidente neste cenário que há uma sobrecarga grande nesta abordagem em comparação a um sistema de arquivos que fique dentro do *kernel*, que é agravado pela mudança de contexto entre o *kernel* e a aplicação do usuário para que a chamada seja tratada. No entanto, dado o benefício da versatilidade de poder implementar um sistema de arquivos com todos os recursos disponíveis do espaço de usuário, que permite a elaboração de regras e comportamentos complexos sem as restrições de se programar dentro do *kernel*, além da vantagem de ser possível montar o sistema sem a necessidades de permissões de administrador, esse mecanismo tem sido amplamente adotado na implementação de sistemas de arquivos distribuídos e paralelos(7). Em testes de viabilidade realizados no início deste trabalho também foi possível verificar que em alguns casos, a sobrecarga é desprezível, como pode ser verificado no Apêndice A.

PUC-Rio - Certificação Digital Nº 1512343/CA

4.4.2 Fila de tarefas e Pool de Threads

Com o objetivo de minimizar os gargalos, decidimos que todas as operações de escrita que seriam feitas em uma partição remota passariam a ser assíncronas. A ideia é que a operação de escrita não precise esperar que o dado seja transferido pela a rede para ser concluída, bastando que o dado seja escrito num cache local, enquanto uma *pool* de *threads* efetua as escritas remotas consumindo tarefas inseridas em uma fila. Após a escrita no cache local, é inserida na fila de execução uma tarefa de cópia deste dado para a partição remota e, paralelamente, a *pool* de *threads* se encarrega de consumir e executar as tarefas da fila. A primeira abordagem adotada consistiu em utilizar um cache em memória RAM para maximizar a velocidade de resposta do processamento dos comandos de escrita, permitindo que a tarefa prosseguisse com seu processamento rapidamente. No entanto, essa estratégia não se mostrou viável pois a capacidade de transferência da rede não era suficiente para dar vazão aos dados gerados pela tarefa, fazendo com que o STA consumisse toda a memória RAM disponível no nó de execução. Com isso, a abordagem foi alterada para utilizar o disco local do nó de execução como cache, que apesar de mais lento do que utilizar a memória RAM, permite o término da execução da tarefa sem consumir todo o recurso de memória disponível.

Por utilizarmos o SSHFS para fazer a transferência de fato, a implemen-

tação desta funcionalidade é feita com comandos de escritas em arquivos, e o controle de concorrência é todo feito pelo sistema de arquivos utilizado. Para tanto, os métodos de escrita e leitura utilizados para efetuar a cópia do arquivo não fazem uso de cursor implícito, garantido que não incorra em problemas de corrida entre as *threads* executoras ao redefinir a posição do cursor a todo instante, e garantindo também que a ordem de execução das tarefas de cópia não influencie no resultado. Basicamente, cada escrita feita no STA resultará numa escrita igual feita numa partição SSHFS.

4.4.3 Identificando o destino dos arquivos

Como mencionado em 4.1, estamos supondo que o local de execução das tarefas consumidoras de um arquivo é definido anteriormente à execução da tarefa que produz o respectivo arquivo. Como não está no escopo deste trabalho avaliar estratégias de orquestração das tarefas de execução de um *workflow*, optamos por gerar um arquivo que já contém todas as definições de quais arquivos devem ser escritos em quais nós de execução, de forma que quando uma tarefa é iniciada em seu respectivo nó de execução, seu arquivo de entrada já se encontra disponível localmente. Contudo, entendemos que esse cenário utilizado para os testes do STA pode não atender às necessidades em todos os casos, então optamos por uma implementação que fosse possível redefinir a estratégia de alocação a cada execução, sem a necessidade de alterar a programação do binário executável. Decidiu-se por utilizar Lua para atribuir esse dinamismo ao código que implementa a API FUSE.

A utilização de uma linguagem de *script* também permite que o STA seja estendido e passe a implementar lógicas complexas e dinâmicas para a definição ou descoberta de destinos dos arquivos gerados a cada execução. Além disso, a escolha de Lua também se deu por ser uma linguagem que possui integração nativa com C, e por possuir desempenho comparado a linguagens compiladas.

4.4.4 Leitura com espera

A escrita assíncrona gera uma defasagem potencial entre o momento do término da tarefa e o momento em que termina a transferência do arquivo gerado. Isso permite que uma tarefa que se inicie logo em seguida a uma outra comece a ler um arquivo que ainda não está completo. Para evitar que essa leitura esgotasse o arquivo local identificando erroneamente seu final, foi necessário também fazer uma alteração na operação de leitura para que isso não ocorresse.

PUC-Rio - Certificação Digital Nº 1512343/CA

O problema só ocorre quando uma tarefa faz uma requisição de leitura ao sistema de arquivos e a operação retorna menos *bytes* lidos que os solicitados, pois a tarefa entende que a leitura atingiu o final do arquivo. Era necessário, portanto, que, nesses casos, a operação de leitura fosse capaz de identificar que o arquivo sendo requisitado ainda não havia sido completamente copiado. Para resolver esse problema, assim que uma tarefa requisita a criação de um arquivo, no mesmo diretório é criado um outro arquivo de mesmo nome com sufixo *".wait"*, que só é removido quando a última tarefa de transferência da fila de tarefas for executada. No momento da leitura por parte da tarefa consumidora, o STA, antes da leitura, verifica se existe o arquivo de espera. Em seguida, o STA tenta ler do arquivo os dados solicitados pela tarefa. Se a operação de leitura conseguir ler todos os bytes solicitados, o STA então retorna os dados ao usuário e encerra, caso contrário, o STA repete o processo todo até que a leitura obtenha todos os bytes desejados ou o arquivo de espera não esteja mais lá, indicando assim que a cópia do arquivo se encerrou.

4.5 Comentários adicionais

Como descrito na seção 4.1, a estratégia proposta supõe que o gerenciador de *workflow* seja capaz de planejar dois passos à frente, para garantir que já seja possível definir o destino de um arquivo no momento de sua criação. Contudo, uma outra abordagem possível seria determinar em qual nó de execução deve ser iniciada a próxima tarefa em função de onde o arquivo foi salvo fisicamente, supondo neste caso a utilização de um sistema de arquivos distribuído que use alguma heurística interna para decidir onde o dado ficará salvo efetivamente. Em nossos testes optamos por utilizar a primeira abordagem, mas poucas alterações seriam necessárias no código do STA para apoiar testes com a segunda abordagem, pois bastaria adicionar a lógica de decisão da localidade do arquivo e passar a utilizar os *scripts* Lua como meio para que o gerenciador de *workflow* pudesse identificar onde cada arquivo foi salvo para poder tomar uma decisão informada sobre onde alocar cada tarefa. O resultado final, contudo, seria essencialmente o mesmo, pois teríamos de qualquer forma sempre a tarefa executando no mesmo lugar onde o arquivo já está, que é justamente o que pretendemos analisar neste trabalho, tornando indiferente a solução escolhida para as simulações.

Outro ponto de análise é a escolha do momento de transmissão dos dados, se deve ocorrer à posteriori, ou seja, se deve começar a ser feita a partir do início da tarefa que irá consumir dados, ou à priori, na ocasião de sua geração. Nas duas abordagens, a tarefa produtora se beneficia das vantagens de escrever

PUC-Rio - Certificação Digital Nº 1512343/CA

seus dados localmente, assim como em ambos os casos, é possível que a tarefa consumidora precise ficar esperando os dados serem transferidos para continuar sua execução. No entanto, no caso à priori há uma possibilidade de, ao término da tarefa produtora, a maior parte dos dados já ter sido transferida, permitindo que a tarefa consumidora já possa começar sua execução sem nenhuma espera, e ainda há a possibilidade do restante dos dados serem transferidos ainda antes de serem requisitados, uma vez que a tarefa pode ainda estar ocupada com os dados que já estavam disponíveis. Na abordagem à posteriori, mesmo que a transferência seja mais rápida que a capacidade da tarefa de consumir os dados, esta sempre terá uma espera inicial em virtude da transferência dos dados iniciais. Desta forma, optamos pela abordagem à priori pois acreditamos que possui um potencial maior de ganho.

PUC-Rio - Certificação Digital Nº 1512343/CA

5
Testes e Resultados

Neste capítulo apresentamos os testes efetuados para avaliar a efetividade da estratégia descrita na seção 4.1. Iniciamos com testes de escrita para avaliar os impactos do STA sobre o tempo de execução da tarefa que está produzindo o arquivo. Em seguida, na seção 5.2 descrevemos os testes efetuados para avaliar os potenciais ganhos de tempo em cada um dos padrões de acessos a dados descritos na seção 2.1. Depois em 5.3 analisamos como o impacto do tempo intensivo de processamento da tarefa sobre os tempos de execução das tarefas seguintes. Na seção 5.4 tecemos comentários finais sobre os testes realizados, com análises do ambiente disponível para sua realização, assim como outras análises dos resultados obtidos.

Todas os testes foram realizados em computadores com as seguintes configurações:

- CPU: i7-4790 @ 3.60GHz, 8 núcleos
- RAM: 8GB
- Disco: 7200 rpm
- Rede: Ethernet 100 mbps

PUC-Rio - Certificação Digital Nº 1512343/CA

5.1
Testes de escrita

Com o intuito de tentar entender qual a influência das camadas adicionadas tanto pelo STA quanto pelo SSHFS, uma bateria de testes de escrita foi realizada para medir as diferenças de tempo em cada cenário.

Foram estabelecidos 4 cenários de escrita para testes comparativos, e cada um deles foi executado 30 vezes. Para esses testes foi criado um programa que cria e preenche um arquivo até ter 5gb de tamanho.

(A) Escrita no disco local

(B) Escrita no diretório do STA, mas sem efetuar nenhuma cópia

(C) Escrita no diretório do STA, efetuando 1 cópia

(D) Escrita no diretório montado com SSHFS

Tabela 5.1: Resultados de escrita

	(A)	(B)	(C)	(D)
Tempo médio (s)	150,7	157,3	184,6	462,4
Desvio padrão (s)	0,76	1,69	28,18	1,17

A tabela 5.1 exibe o resultado dos primeiros testes de escrita realizados. Nela, podemos ver que mesmo quando não há cópia, já há um aumento de 5% no tempo de execução, que é a soma das sobrecargas da utilização do FUSE como também a consulta nas configurações dos arquivos para identificar quais as regras de cópia do arquivo criado. Em seguida, podemos analisar que há um impacto ainda maior quando uma cópia é efetuada. A natureza concorrente entre escrita do arquivo pelo programa produtor e as leituras das *threads* do STA encarregadas de executar as tarefas de cópia geram uma sobrecarga que diminui a eficiência da escrita, além de deixá-la mais instável, como é possível observar na alta variação de tempo entre as execuções. No entanto, mesmo significando um aumento médio de $22,5\%$ no tempo de escrita se comparando com os tempos utilizando o disco diretamente, utilizar o STA ainda representa uma redução de aproximadamente 60% se comparando com o tempo de escrita direto no servidor remoto.

PUC-Rio - Certificação Digital Nº 1512343/CA

Após verificar essa sobrecarga em decorrência da cópia do arquivo, decidimos efetuar testes para investigar mais a fundo quais os impactos de mais cópias serem feitas. No entanto, diferentemente dos testes anteriores em que o programa utilizado apenas fazia simulação de processamento e escrita, para esse contexto utilizamos o programa que foi desenvolvido para os testes do padrão sequencial. Nosso entendimento é que esse programa representa um cenário mais realista com relação ao uso de recursos da máquina do que o programa anterior e, portanto, acreditamos que sua utilização nos aproxima de situações reais.

Foram elaborados 4 cenários para esses testes, variando o número de cópias a serem feitas do arquivo gerado, todos eles lendo um arquivo de 3gb do disco local e escrevendo em um diretório controlado pelo STA, e simulando 5 ciclos de processamento a cada conjunto de dados lido do arquivo, o que representa um total médio de 48 segundos do tempo total de simulação a cada rodada.

Tabela 5.2: Resultados de escritas com múltiplas cópias concorrentes

# Cópias	6	5	4	3
Tempo médio (s)	125,9	127,3	128,6	132,9
Desvio padrão (s)	2,6	2,6	2,7	3,3

Ao contrário do que seria esperado, houve um impacto menor no tempo

de execução de cada programa com um número maior de cópias. Isso pode ter se dado pelo fato de que, ao possuir mais cópias, as *threads* do STA fazem mais leituras do mesmo trecho do arquivo em cache para ser copiado, favorecendo otimizações do sistema de arquivos nativo. Havendo um número menor de cópias a serem feitas, há menos tarefas designadas para copiar cada trecho do arquivo, aumentando o número de trechos distintos concorrentes sendo lidos.

5.2 Padrões de acesso a dados

Nesta seção serão detalhados os testes realizados para avaliar a possibilidade de ganhos para cada um dos padrões de acesso a dados. A ideia de cada um dos testes é tentar isolar somente o potencial de ganho especificamente de cada um dos padrões descritos. Para isso, foram elaborados alguns programas cujo objetivo é simular uma única tarefa de um *workflow*. Todos eles consomem um ou mais arquivos, simulam um processamento em cima do dado lido e em seguida produzem um ou mais arquivos, dependendo do padrão associado. Além disso, para medir o tempo não só da tarefa produtora, mas também da que irá consumir o arquivo produzido, todos os testes executam 2 passos, com o primeiro arquivo sendo sempre lido de um servidor remoto. Ao final, as últimas tarefas tiveram seus resultados de escritas descartados (redirecionando para */dev/null*), de forma que fosse possível isolar somente os ganhos com a leitura dos arquivos já pré-transferidos.

Para cada padrão realizamos duas baterias de testes com 30 repetições cada. A bateria (A) sem utilização do STA, com todas as leituras e escritas sendo feitas por um servidor remoto, e a outra bateria (B) utilizando o STA desenvolvido, com os arquivos sendo redirecionados no momento de suas criações e edições. A expectativa era que, em todos os casos, os testes da bateria utilizando o STA conseguisse executar em menos tempo do que os testes da bateria com a leitura e escrita por servidores remotos. Em todos os casos o mesmo arquivo de 3gb foi utilizado como entrada inicial para todos os testes.

Para avaliar cada um dos padrões foram desenvolvidos programas cuja finalidade é emular uma tarefa de um *workflow* científico. Todos os programas fazem leitura e escrita de um ou mais arquivos, dependendo do padrão que estiver representando, e, além disso, todos os programas fazem uma simulação de processamento. Essa simulação consiste em iterar sobre todas as posições do *buffer* de leitura, para garantir simulação de todos os dados acessados, e para cada posição são feitas operações matemáticas apenas para emular o processamento desse dado. Por fim, é feita uma escrita na posição lida, também para emular que os dados estão sendo alterados. Chamamos essa iteração pelo

PUC-Rio - Certificação Digital Nº 1512343/CA

buffer de um ciclo de simulação. O padrão para os testes a seguir foi utilizar sempre três ciclos de simulação.

Em todos os testes a seguir as medidas foram feitas a partir do tempo real total de execução de cada processo ou conjunto de processos, em segundos. Para casos em que havia mais de um programa sendo executado em sequência, o tempo medido foi o da execução do *script* desenvolvido para rodar aquele conjunto de programas. Os tempos de processamento foram medidos com o somatório da passagem de tempo real logo antes e imediatamente após cada passo da simulação de processamento.

5.2.1 Sequencial

O programa criado para simular o padrão sequencial foi também utilizado em todos os testes dos outros padrões. Seu funcionamento consiste em ler um arquivo, em blocos de 5MB, simular um processamento dos dados lidos, com leitura e escrita em todas as posições de memória no *buffer* onde os dados ficavam temporariamente contidos, e em seguida, o bloco era escrito em um novo arquivo, criado pelo programa.

PUC-Rio - Certificação Digital Nº 1512343/CA

Para essa simulação, foi elaborado um *script* que efetuava a chamada da primeira instância deste programa sequencial, que fazia sua leitura de um servidor remoto, através do SSHFS, e em seguida fazia a escrita do conteúdo deste arquivo, no diretório do STA ou no remoto, de acordo com a respectiva bateria. Em seguida o *script* inicia um segundo programa sequencial em um outro nó de execução, que fará a leitura de seu arquivo de entrada do servidor remoto, ou localmente do diretório controlado pelo STA, de acordo com a bateria sendo executada. A expectativa é que, no caso (B), ao fazer uma escrita num cache local, a primeira instância do programa sequencial irá terminar antes do que no caso (A), por não precisar esperar que todos os dados sejam transferidos a um diretório remoto a cada escrita, e, na sequência, a próxima instância do programa sequencial seja capaz de se beneficiar dos dados que já estiverem presentes localmente, diminuindo a espera na obtenção dos dados que, inevitavelmente, tem de vir remotamente, seja do nó de execução anterior, seja de um servidor remoto de dados. A tabela 5.3 contém os resultados.

Tabela 5.3: Resultados dos testes do padrão sequencial

30 repetições	(A)	(B)
Tempo médio (s)	$600,4$	$407,0$
Desvio padrão	$74,4$	$19,4$

Em todos os casos, o tempo de processamento simulado em cada passo

foi de aproximadamente 47 segundos em média, totalizando 94 segundos, sendo o resto do tempo sendo gasto com a leitura e a escrita do arquivo.

O resultado demonstra uma redução média de aproximadamente 32% do tempo de execução total das duas tarefas em sequência, o que está de acordo com as expectativas. Se for considerar que em ambas as baterias houve um tempo gasto médio só com processamento, ou seja, que não é possível de ser reduzido com a estratégia proposta, podemos tentar isolar só os outros tempos, subtraindo esse valor gasto da conta original, o que chega a uma redução média de aproximadamente 38% do tempo de execução total das tarefas.

5.2.2 Coleta

O funcionamento do programa criado para simular o padrão de coleta consiste em ler o conteúdo de 4 arquivos, e escrever tudo em um único arquivo de saída, simulando processamento dos dados após cada passo de leitura, de forma análoga ao programa sequencial. Diferentemente do teste anterior, no entanto, cada arquivo inicial tem 1/4 to tamanho original do arquivo utilizado anteriormente, fazendo com que o arquivo final possua o mesmo tamanho que nos outros testes.

Para essa simulação foi elaborado um *script* que roda paralelamente e concorrentemente 4 programas sequenciais, e cada um deles, em um nó de execução distinto, lê um arquivo diferente do mesmo servidor remoto, e escreve sua saída, de acordo com cada bateria, ou de volta no servidor remoto, ou no diretório controlado pelo STA, que então se encarrega de escrever desse arquivo em seu diretório remoto destino. Todos os 4 arquivos gerados são neste momento redirecionados para o mesmo destino, de onde o programa de coleta lê os arquivos alternadamente, simulando processamento após cada chamada ao comando de leitura e em seguida efetua a escrita dos dados lidos no arquivo destino passado como parâmetro — que, no caso, é */dev/null*. Na bateria (A), nesse último passo o programa de coleta faz a leitura dos arquivos todos de um mesmo servidor remoto, enquanto que na bateria (B), a leitura será feita do diretório controlado pelo STA, que se encarregará de garantir que o programa aguarde, se necessário, a cópia do conteúdo dos arquivos sendo lidos. A tabela 5.4 contém os resultados.

Tabela 5.4: Resultados dos testes do padrão de coleta

30 repetições	(A)	(B)
Tempo médio (s)	$581,1$	$275,4$
Desvio padrão	$3,1$	$15,4$

PUC-Rio - Certificação Digital Nº 1512343/CA

Por processarem arquivos menores, o tempo médio de processamento simulado dos programas sequenciais foi de 12 segundos, enquanto que o tempo médio de processamento simulado do programa de coleta foi de 47 segundos, equivalente ao caso anterior por se tratar do mesmo volume de dados, totalizando 59 segundos, sendo o resto gasto com a leitura e a escrita dos arquivos.

O resultado demonstra uma redução média de aproximadamente 53% do tempo total de execução das tarefas em sequência, o que novamente está de acordo com as expectativas. Analogamente, desconsiderando o tempo gasto somente com processamento, que é proporcional ao volume de dados processados e não é afetado por eventuais esperas das transferências de dados, a redução média chega a 59% do tempo de execução total das tarefas.

5.2.3 Transmissão

Para testar o padrão de transmissão não foi necessária a criação de nenhum programa específico, pois, individualmente do ponto de vista da tarefa, ela pode simplesmente se comportar como o programa sequencial. A diferença está no segundo passo, onde em vez de uma, são executadas algumas instâncias do programa sequencial. Todas as tarefas do segundo passo têm de ler o mesmo arquivo gerado pela tarefa do primeiro passo, caracterizando-se assim, o padrão de transmissão.

Para a simulação de transmissão foi elaborado um *script* que roda uma instância do programa sequencial e após seu término, inicia a execução de 4 novas instâncias, cada uma em um nó de execução distinto, também do programa sequencial. No primeiro passo é feita a leitura de um arquivo obtido de um servidor remoto, e no segundo passo, no caso (A), o primeiro arquivo de saída é escrito de volta no servidor remoto, com os 4 processos seguintes lendo desse arquivo gerado; e no caso (B), o primeiro arquivo de saída é escrito localmente no diretório controlado pelo STA, que então efetua a cópia para os nós de execução que ficarão encarregados de executar as tarefas do segundo passo, permitindo que façam a leitura desses arquivos do diretório local controlado pelo STA, que se encarrega de garantir a leitura completa do arquivo em cada nó de execução. A tabela 5.5 contém os resultados.

Tabela 5.5: Resultados dos testes do padrão de transmissão

30 repetições	(A)	(B)
Tempo médio (s)	$1372,0$	$1090,9$
Desvio padrão	$103,2$	$1,3$

Por processarem o mesmo volume de dados, o tempo médio de processa-

PUC-Rio - Certificação Digital Nº 1512343/CA

mento foi o mesmo do caso sequencial, 47 segundos, aproximadamente, em cada passo, totalizando 94 segundos, sendo o resto do tempo gasto com operações de leitura e escrita dos arquivos.

O resultado demonstra uma redução média de aproximadamente 20% do tempo total de execução das tarefas em sequência. Se considerado somente o tempo gasto com a transferência dos arquivos, descartando o tempo gasto com simulação de processamento, esse valor sobe para 22%.

5.2.4 Partição

Para o teste de partição foi necessário criar um programa específico, com comportamento similar ao de coleta, porém inverso, ou seja, o programa lê um arquivo e gera outros 4, simulando processamento após cada operação de leitura, todos com cerca de 1/4 do tamanho do arquivo de entrada.

Para essa simulação foi elaborado um *script* que roda uma instância do programa de partição, e após o término, da mesma forma que o teste de transmissão, inicia a execução de 4 instâncias do programa sequencial, cada uma em um nó de execução distinto. No primeiro passo, o programa de partição faz a leitura do arquivo de entrada, obtido de um servidor remoto e em seguida gera 4 novos arquivos, cada um contendo 1/4 do tamanho do arquivo de entrada. Na bateria (A) esses arquivos são salvos de volta no servidor remoto, e no caso (B) esses arquivos são salvos no diretório controlado pelo STA, onde serão redirecionados aos nós de execução para serem processados pelo próximo passo da simulação. Ao término da execução do programa de partição, os programas sequenciais são devidamente instanciados em seus nós de execução respectivos, onde já poderão a começar a ler os arquivos gerados — no caso (A) são obtidos do servidor remoto e no caso (B) são obtidos do diretório local controlado pelo STA. A tabela 5.6 contém os resultados.

Tabela 5.6: Resultados dos testes do padrão de transmissão

30 repetições	(A)	(B)
Tempo médio (s)	$555,2$	$376,9$
Desvio padrão	$48,6$	$33,4$

Analogamente ao caso da coleta, por processarem arquivos menores, o tempo médio de processamento simulado dos programas sequenciais foi de 12 segundos, enquanto que o tempo médio de processamento simulado do programa de coleta foi de 47 segundos, equivalente ao caso anterior por se tratar do mesmo volume de dados, totalizando 59 segundos, sendo o resto gasto com a leitura e a escrita dos arquivos.

PUC-Rio - Certificação Digital Nº 1512343/CA

O resultado demonstra uma redução média de 34% do tempo total de execução das tarefas. Se considerado somente o tempo gasto com a transferência dos arquivos, descartando o tempo gasto com simulação de processamento, esse valor sobe para 38%.

5.2.5 Conjunção dos Padrões

Para tentar simular um cenário mais próximo de um teste real, foram elaborados testes que faziam uma conjunção dos padrões apresentados anteriormente. Elaboramos uma simulação inspirada no Montage Workflow, tentando seguir sua sequência de padrões, como demonstrado no grafo apresentado na figura 2.5.

Da mesma forma como anteriormente, foram definidas duas baterias de testes, uma com todos os arquivos sendo lidos e escritos de um mesmo servidor remoto, e outra com a utilização do STA e a estratégia proposta.

Embora as primeiras execuções tenham se mostrado muito promissoras, com resultados iniciais até superiores aos vistos nos testes dos padrões individualmente, por ser um teste que demandava muitos recursos do laboratório, tanto em quantidade de máquinas quanto número de horas consecutivas necessárias para finalizar cada um dos testes, não foi possível completar um número de execuções bem sucedidas o suficiente para podermos apresentar um resultado que fosse estatisticamente relevante. Sem uma boa quantidade de resultados, não é possível afirmar, por exemplo, que as diferenças de tempo entre as execuções não se deram somente por uma instabilidade na rede.

PUC-Rio - Certificação Digital Nº 1512343/CA

5.3 Testes de processamento

Outro ponto relevante a ser avaliado é qual o possível impacto que o STA poderia ter no tempo de execução das tarefas do ponto de vista do uso do processador. Para *workflows* que sejam mais intensivos em processamento, qualquer impacto poderia gerar uma diferença no tempo final.

Para investigar o potencial impacto, foram executados testes variando a quantidade dos ciclos das simulações de processamento. Foram feitos testes com 3, 20 e 30 ciclos simulados e com o STA efetuando 3 cópias do arquivo gerado, com baterias de 10 testes cada. Os primeiros resultados não indicaram nenhum efeito, mas levantamos a hipótese de ser apenas uma consequência dos nós de execução possuírem inúmeros núcleos de processamento, e do fato das simulações terem sido feitas mono-processadas.

Para aprofundarmos a investigação, modificamos os programas de simulação para utilizar OpenMP(6), paralelizando os ciclos de processamento simulado e repetimos os testes. Novamente, nenhum impacto foi identificado, mas esse resultado não é totalmente inesperado, uma vez que as *threads* do STA que se encarregam de fazer a cópia dos arquivos são todas intensivas somente de entrada e saída, não consumindo muito tempo do processador, afinal.

Contudo, a realização desses testes criou uma oportunidade de observar quais as consequências da variação do tempo de processamento de cada tarefa. Ao aumentar o número de ciclos de simulação, as tarefas passaram a ter um intervalo maior entre uma escrita e outra no arquivo sendo produzido. Isso possibilita que haja mais tempo para a cópia do arquivo ser feita através da rede, fazendo com que mais dados sejam transferidos ao longo da execução da tarefa corrente. Como consequência, quando a próxima tarefa começar, já haverá um pedaço maior do arquivo disponível localmente, e em alguns casos, era possível ocorrer a transferência quase completa do arquivo, diminuindo muito o tempo que o passo seguinte teria de ficar esperando caso os dados ainda estivessem sendo transferidos.

Desta forma, elaboramos uma nova bateria de testes para verificar como isso se dá. Repetimos os mesmos testes realizados com variação do número de ciclos simulados, mas agora também simulando que o arquivo produzido em cada teste seja devidamente consumido por uma tarefa conseguinte. Realizamos 10 testes com 3, 20 e 30 ciclos simulados, e em todos os casos, o arquivo gerado foi configurado para ser copiado para 3 nós de execução destino. Para calcular o tempo total, somamos o tempo de execução do primeiro programa com o maior tempo de execução dentre as 3 tarefas que efetuam a leitura de cada uma das cópias.

A tabela 5.7 exibe os resultados obtidos. Para complementar os dados apresentados, é preciso considerar que 3 ciclos de simulação de processamento consomem, em média, 48 segundos, 20 ciclos consomem, em média, 323 segundos e 30 ciclos consomem, em média, 477 segundos.

Tabela 5.7: Resultados das variações dos tempos de processamento

	3 ciclos	20 ciclos	30 ciclos
Média do tempo de execução do primeiro passo (s)	114,1	378,7	533,7
Média do tempo de execução do segundo passo (s)	693,9	429,9	479,3
Média do tempo de execução total (s)	808,4	809,2	1014,4

Embora haja um aumento considerável no tempo total de execução da primeira tarefa de 3 para 20 ciclos, o tempo total de execução dos dois passos se manteve essencialmente constante. O aumento do tempo de execução do

PUC-Rio - Certificação Digital Nº 1512343/CA

primeiro passo acaba sendo compensado com um segundo passo mais ágil, mantendo o tempo total de execução constante. A partir dessa observação, fica evidente que o gargalo desse processamento sequencial está justamente na transferência do arquivo para execução da tarefa seguinte. Ao analisar os tempos obtidos nas tarefas com 30 ciclos simulados, o mesmo tempo constante não se mantém, mesmo sendo o mesmo tamanho de arquivo. No entanto, ao analisar o tempo total de processamento do segundo passo, é possível observar que é essencialmente só o necessário para completar o processamento, um indicativo de que o gargalo, nesse caso, deixou de ser a transferência do arquivo e passou a ser o processamento. Esses testes são um bom indicativo dos benefícios dessa estratégia, pois demonstram que nos aproximamos de diminuir a sobrecarga de tempo com transferência dos dados ao máximo possível, dado que não há como executar a tarefa num outro nó de execução sem transferir os dados de entrada para lá ao menos uma vez.

5.4 Comentários finais

Nesse capítulo analisamos o potencial de ganho da estratégia proposta. Também foi possível ter um vislumbre do elevado grau de complexidade do problema, com um grande número de variáveis envolvidas, sendo que não conseguimos elaborar sobre inúmeros outros aspectos que podem ter impacto direto no desempenho dos processos. Estabilidade e capacidade da rede e a utilização de caches por parte do sistema operacional ou sistemas de arquivos utilizados de forma auxiliar, são, por exemplo, algumas dessas variáveis que podem ter impacto nos resultados.

Um aspecto que foi observado, mas que acaba ficando mascarado, é que em algumas baterias rodadas, haviam certas zonas de estabilidade nos resultados, muitas vezes após algumas baterias já tendo sido executadas. Em todos os testes dos padrões de acesso a dados, a eliminação de uns poucos valores muito discrepantes reduzem o desvio padrão a 1% ou menos, além de evidenciar ganhos marginalmente melhores nas reduções dos tempos de execução das tarefas. Além disso, a natureza repetitiva dos testes permitia otimizações de cache por parte dos sistemas utilizados, gerando distorções que não aconteceriam num caso real, onde não é o mesmo arquivo que é acessado todas as vezes. Nas baterias de teste feitas sobre o SSHFS, foi possível notar que após algumas rodadas as chamadas já não estavam mais sendo feitas remotamente, diminuindo bastante esse tempo de execução desse que deveria ser o teste base para comparativo com a estratégia apresentada. Sendo assim, é possível que um potencial de ganho tenha sido mascarado por essa anomalia.

PUC-Rio - Certificação Digital Nº 1512343/CA

Outro aspecto que pode impactar nos resultados, mas que acabou não tendo prioridade em nossas análises, é o número de *threads* levantadas para se encarregar de fazer as transferências dos arquivos. Em todos os testes realizados, havia sempre 5 *threads* ativas no STA.

Por fim, conseguimos identificar que até mesmo as características internas dos processos que estão rodando no *workflow* são capazes de influenciar no potencial de ganhos. Uma aplicação que faça primeiro todo seu processamento e por fim, faça todas suas operações de entrada e saída, pode acabar não se beneficiando tanto da estratégia, pois não fornece ao sistema boas oportunidades de fazer a transferência enquanto o programa ainda está efetuando seu processamento.

PUC-Rio - Certificação Digital Nº 1512343/CA

6
Conclusão e Trabalhos Futuros

Neste trabalho apresentamos e avaliamos uma estratégia para diminuir o tempo de execução de *workflows* científicos distribuídos. A estratégia consiste em diminuir transferências dos dados desnecessárias, já transferindo o dado para onde ele será necessário no momento de sua criação. Apesar de haver outros esforços nessa mesma direção, ainda é um problema aberto sem resposta definitiva.

Apresentamos também o STA, que implementa essa estratégia, validando as vantagens e os ganhos potenciais de se utilizá-la. Uma série de testes demonstra que é possível ter reduções substanciais no tempo de execução de 30% ou mais, mas também evidencia limitações e possibilidades de melhorias.

O STA não é totalmente capaz de abordar algumas situações, como, por exemplo, o salvamento de partes de um mesmo arquivo em nós de execução distintos, ou ainda, outras estratégias de cópias múltiplas que consigam desonerar as conexões ponto a ponto da rede, ao tentar trafegar todo o volume de dados de uma só vez.

Embora tenham sido apresentados resultados promissores, essa estratégia ainda está longe de poder ser adotada amplamente em um ambiente de produção. Ainda há alguns pontos a serem resolvidos, como a identificação automática dos padrões de acesso aos arquivos e a integração com as ferramentas de execução e gerência de *workflows*.

Desta forma, há espaço para melhorias e novas pesquisas sobre o tema, como explorar alternativas que possam de fato rodar dentro do *kernel*, para diminuir as sobrecargas causadas pela utilização de FUSE e também evoluir o STA para que seja um sistema de arquivos auto-suficiente, deixando de ser apenas uma espécie de *proxy* para os outros sistemas de arquivos instalados.

Além disso, também poderíamos explorar mais a fundo todos os impactos que as características das tarefas de um *workflow* têm sobre a estratégia, possivelmente elaborando uma classificação nova, com a qual ficaria facilmente identificável em que casos a adoção da estratégia se mostra muito promissora e em qual casos talvez não valha o esforço.

PUC-Rio - Certificação Digital Nº 1512343/CA

Referências bibliográficas

[1] AL-KISWANY, S.; VAIRAVANATHAN, E.; COSTA, L. B.; YANG, H. ; RIPEANU, M.. **The case for cross-layer optimizations in storage: A workflow-optimized storage system**. arXiv preprint arXiv:1301.6195, jan 2013.

[2] ALTSCHUL, S. F.; GISH, W.; MILLER, W.; MYERS, E. W. ; LIPMAN, D. J.. **Basic local alignment search tool**. Journal of Molecular Biology, 215(3):403–410, oct 1990.

[3] BERRIMAN, G. B.; GOOD, J. C.; CURKENDALL, D. W.; JACOB, J. C.; KATZ, D. S.; PRINCE, T. A. ; WILLIAMS, R.. **Montage: An On-Demand Image Mosaic Service for the NVO**. Astronomical Data Analysis Software and Systems XII ASP Conference Series, 295:343–346, 2003.

[4] BHARATHI, S.; CHERVENAK, A.; DEELMAN, E.; MEHTA, G.; SU, M.-H. ; VAHI, K.. **Characterization of scientific workflows**. In: 2008 THIRD WORKSHOP ON WORKFLOWS IN SUPPORT OF LARGE-SCALE SCIENCE, p. 1–10. IEEE, nov 2008.

[5] DA SILVA, R. F.; CALLAGHAN, S. ; DEELMAN, E.. **On the use of burst buffers for accelerating data-intensive scientific workflows**. In: PROCEEDINGS OF THE 12TH WORKSHOP ON WORKFLOWS IN SUPPORT OF LARGE-SCALE SCIENCE - WORKS '17, p. 1–9, New York, New York, USA, 2017. ACM Press.

[6] DAGUM, L.; MENON, R.. **OpenMP: an industry standard API for shared-memory programming**. Computational Science & Engineering, IEEE, 5(1):46–55, 1998.

[7] DEPARDON, B.; LE MAHEC, G. ; SÉGUIN, C.. **Analysis of Six Distributed File Systems**. Research report, Feb. 2013.

[8] DONNELLY, P.; THAIN, D.. **Balancing push and pull in Confuga, an active storage cluster file system for scientific workflows**. Concurrency and Computation: Practice and Experience, may 2016.

PUC-Rio - Certificação Digital Nº 1512343/CA

[9] GRAVES, R.; JORDAN, T. H.; CALLAGHAN, S.; DEELMAN, E.; FIELD, E.; JUVE, G.; KESSELMAN, C.; MAECHLING, P.; MEHTA, G.; MILNER, K.; OKAYA, D.; SMALL, P. ; VAHI, K.. **CyberShake: A physics-based seismic hazard model for southern california**. Pure and Applied Geophysics, 168(3):367–381, Mar 2011.

[10] GRAWINKEL, M.; SUSS, T.; BEST, G.; POPOV, I. ; BRINKMANN, A.. **Towards dynamic scripted pNFS layouts**. Proceedings - 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, SCC 2012, p. 13–17, 2012.

[11] HENK, C.; SZEREDI, M.. **FUSE: Filesystem in userspace**. Online at https://github.com/libfuse/libfuse, 2018.

[12] HOSKINS, M. E.. **SSHFS: Super easy file access over SSH**. Linux J., 2006(146):4–, June 2006.

[13] JUVE, G.; CHERVENAK, A.; DEELMAN, E.; BHARATHI, S.; MEHTA, G. ; VAHI, K.. **Characterizing and profiling scientific workflows**. Future Generation Computer Systems, 29(3):682–692, 2013.

[14] SILVA, F.; CALLAGHAN, S. ; JORDAN, T. H.. **SCEC Broadband Platform: System architecture and software implementation**. Seismological Research Letters, 2011.

[15] VIEIRA NETO, L.; IERUSALIMSCHY, R.; DE MOURA, A. L. ; BALMER, M.. **Scriptable operating systems with Lua**. Proceedings of the 10th ACM Symposium on Dynamic languages - DLS '14, p. 2–10, 2014.

[16] YU, J.; BUYYA, R.. **A taxonomy of scientific workflow systems for grid computing**. ACM SIGMOD Record, 34(3):44, sep 2005.

[17] ZHANG, Z.; KATZ, D. S.; WOZNIAK, J. M.; ESPINOSA, A. ; FOSTER, I.. **Design and analysis of data management in scalable parallel scripting**. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, p. 85:1—-85:11, 2012.

PUC-Rio - Certificação Digital Nº 1512343/CA

A
Testes de viabilidade do FUSE

PUC-Rio - Certificação Digital Nº 1512343/CA

Como prova de conceito, decidimos implementar um protótipo de nossa solução utilizando FUSE. Primeiramente, apenas com objetivo de tentar avaliar qual a sobrecarga adicionada pelo uso do FUSE, fizemos um teste simples de redirecionamento dos arquivos. Ao montar nosso sistema de arquivos, indicamos um outro diretório para ser o ponto de armazenamento destino dos arquivos e todas as chamadas ao nosso sistema simplesmente repassa e reproduz os comandos neste diretório. Com isso, conseguimos fazer testes para avaliar qual o impacto do uso do FUSE como intermediário dessas operações. Rodamos algumas baterias de testes, medindo seus tempos. O primeiro teste consistiu somente em escrever arquivos de 30gb, com blocos de 1kb por escrita. Avaliamos escritas com as funções *write* e *fwrite*, no sistema de arquivos padrão do sistema operacional e depois no nosso protótipo. Avaliamos também a habilitação da propriedade *big_writes* do FUSE, destinada a situações de escritas de grandes volumes de dados. Para cada caso foram feitas 30 execuções.

fwrite 1kb	sem fuse	fuse	fuse bw
média (s)	168,27	176,52	175,49
desvio padrão (s)	6,03	8,11	9,01

write 1kb	sem fuse	fuse	fuse bw
média (s)	188,89	356,26	365,75
desvio padrão (s)	9,61	14,43	16,41

Com blocos de escritas de 1kb, a utilização da função *fwrite* apresenta um desempenho melhor em todos os casos. Acreditamos que seja por conta de utilização de *buffers* que diminuem a necessidade na chamada de sistema como ocorre com a função *write*. No entanto, para a função *write* há uma mudança clara de desempenho. Acreditamos que enquanto o sistema operacional consiga empregar o uso de *buffers* ou caches quando a escrita é feita, ao delegar para o FUSE cada chamada do *write*, a sobrecarga começa a ser mais evidente. Então os testes sugerem que é importante avaliar o emprego de *buffers* e caches na solução. Por outro lado, os testes também mostram que, embora haja um impacto pelo uso do FUSE, a diferença de tempo médio é inferior a 5%, então

com uma expectativa de ganhos finais da ordem de 15%, ainda temos um ganho real.

Decidimos em seguida repetir os testes, mas com blocos de escritas de 1mb em vez de 1kb, diminuindo o número de chamadas de sistema. Neste cenário já não observamos uma grande diferença entre os tempos médios de execução. Esse resultado é muito promissor, pois demonstra que a sobrecarga pelo uso de FUSE não é necessariamente significativa em todos os casos. Quando a escrita é feita minimizando as chamadas de sistema, o impacto pelo uso do FUSE é mínimo. Além disso, nesse caso a utilização da opção *big_writes* surtiu efeito e praticamente anulou a sobrecarga do FUSE.

fwrite 1mb	sem fuse	fuse	fuse bw
média (s)	171,35	174,52	174,83
desvio padrão (s)	7,20	6,63	7,36

write 1mb	sem fuse	fuse	fuse bw
média (s)	172,25	177,22	172,61
desvio padrão (s)	5,96	7,32	6,00

Outro teste que fizemos foi o de duplicar os dados escritos pelo programa em dois diretórios destino distintos. Isso nos permite implementar regras como replicação de um arquivo para ser consumido por processos presentes em diferentes localidades. Os testes foram bem sucedidos, mas sem utilização de escrita assíncrona, essa abordagem dobra o tempo de execução do programa, como já seria esperado que acontecesse.

PUC-Rio - Certificação Digital Nº 1512343/CA