UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ALEXANDRE K. S. MIYAZAKI

Otimizando StarVZ para carga de grandes volumes de dados

Monografia de Conclusão de Curso apresentada como requisito parcial para a obtenção do grau de Especialista em Big Data & Data Science

Orientador: Prof. Dr. Lucas Mello Schnorr

Porto Alegre
2019

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Miyazaki, Alexandre K. S.

Otimizando StarVZ para carga
de grandes volumes de dados / Alexandre K. S. Miyazaki. – Porto Alegre: PPGC da UFRGS, 2019.

32 f.: il.

Monografia (especialização) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2019. Orientador: Lucas Mello Schnorr.

1. Spark. 2. Hadoop. 3. StarVZ. I. Schnorr, Lucas Mello. II. Título.

“If I have seen farther than others,
it is because I stood on the shoulders of giants.”
— Sir Isaac Newton

AGRADECIMENTOS

TODO

RESUMO

TODO: Será escrito após a conclusão.

Palavras-chave: Spark. Hadoop. StarVZ.

Using LaTeX to Prepare Documents at II/UFRGS

ABSTRACT

TODO: Será escrito após a conclusão.

Keywords: Electronic document preparation, LaTeX, ABNT, UFRGS.

LISTA DE ABREVIATURAS E SIGLAS

BSP	Bulk-Synchronous Parallel
CSV	Comma-Separated Value
DAG	Directed Acyclic Graph
DFS	Distributed Filesystem
HPC	High-Performance Computing
MPI	Message Passing Interface
PVM	Parallel Virtual Machine
RDD	Resilient Distributed Dataset

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

TODO: Será escrito após a conclusão.

2 FUNDAMENTAÇÃO

Este Capítulo apresenta a fundamentação necessária para o entendimento do trabalho. Ele é separado em duas Seções, na primeira, são apresentadas as ferramentas de análise de desempenho de aplicações paralelas e, na segunda, contextualizam-se plataformas e arcabouços utilizados no trabalho.

2.1 Análise de desempenho de aplicações paralelas

Para melhor organização do trabalho, esta Seção foi dividida em duas partes. A primeira apresenta ferramentas que oferecem visualização de rastros de aplicações desenvolvidas no modelo *Bulk-synchronous parallel* (BSP), a segunda, contextualiza ferramentas voltadas para visualização de rastros de aplicações desenvolvidas no modelo orientado a tarefas.

2.1.1 Ferramentas de visualização clássicas

Essas ferramentas possuem o objetivo de prover visualizações de rastros para aplicações de HPC tradicionais. Estas eram desenvolvidas seguindo o modelo BSP, que consiste em uma série de super passos (computações, comunicações, sincronizações), executadas sobre um ambiente homogêneo. Como esta abordagem foi dominante durante muito tempo, suas necessidades balizaram o desenvolvimento da maior parte das ferramentas de análise de desempenho.

ViTE

ViTE (COULOMB et al., 2009) é uma ferramenta de visualização de rastros de código aberto. Suas entradas são arquivos na linguagem Pajé (STEIN et al., 2015) e para o processamento de grandes volumes ele conta com aceleração de Hardware e OpenGL.

Essa ferramenta exibe os recursos de forma hierárquica, onde oferece a visualização das tarefas (eixo vertical) em função de tempo (eixo horizontal), similar a um Gantt. Na análise de aplicações distribuídas, também é possível incluir indicadores de transferências de dados.

Paraver

Paraver (PILLET et al., 1995) também objetiva a visualização e análise de rastros de execução. Suas entradas são geradas por vários modelos de programação, pela ferramenta Extrae. Para lidar com grandes volumes de dados, ele realiza agregações com uma abordagem definida pelo usuário via arquivo de configuração.

Vampir

Vampir (KNÜPFER et al., 2008) é uma ferramenta proprietária de código fechado para fins de análise de rastros. Ela traz uma abordagem de cliente-servidor, onde o primeiro pode ser executado no hardware de experimentação e o segundo é o usualmente é o computador do usuário.

Suas entradas são arquivos OTF2 (Open Trace Format, version 2) (ESCHWEILER et al., 2012). Ele fornece múltiplas visualizações como gráficos de espaço-tempo e estatísticas de execução.

Ravel

O objetivo da ferramenta Ravel (ISAACS, 2014) também é a visualização de rastros. Suas entradas são em formato OTF (*Open Trace Format*) e sua diferença em relação aos demais é que ele estrutura melhor as operações com a utilização de linhas de tempo lógicas.

FrameSoc e Ocelotl

FrameSoc (Pagano et al., 2013) é uma ferramenta de análise de desempenho, capaz de lidar com grandes volumes de dados. Como entrada, ela suporta diversos formatos como Pajé, CTF, Paraver e OTF2. Além dos rastros, é possível armazenar informações como metadados e anotações. A ferramenta converte tudo para um modelo de dados genérico e armazena em uma base de dados relacional.

Para visualizar grandes volumes de dados, a ferramenta baseia-se no Ocelotl (Dosimont et al., 2014). Esse módulo gerencia uma agregação espaçotemporal dados via um arquivo de configuração parametrizável pelo usuário.

2.1.2 Ferramentas de visualização orientadas a tarefas

O modelo de computação orientado a tarefas segue o conceito de um conjunto de porções de trabalho independentes, de tamanhos variados, que podem ser executadas em paralelo. Este insere uma camada intermediária entre o hardware e o software desenvolvido que facilita a implementação, pois provê uma certa independência de arquitetura. Além disso, quem analisa o desempenho de aplicações não é mais o seu desenvolvedor, mas sim aqueles responsáveis pelo ambiente de execução.

As ferramentas de visualização para esse modelo de computação se inspiram em características daquelas desenvolvidas para o modelo BSP, pois este dominou o cenário de HPC por muito tempo. Todavia, como diversas premissas são quebradas nesse novo paradigma, as funcionalidades outrora utilizadas com eficiência para a análise de desempenho não são mais suficientes.

DAGViz

DAGViz (HUYNH et al., 2015) é composto por dois passos:

1. extração do DAG dos arquivos de uma execução paralela;
2. visualização hierárquica do DAG.

Essa ferramenta traz uma visualização diferente do modelo BSP, exibindo as tarefas como um grafo hierárquico. Nele, o analista pode colapsar e expandir os grupos de tarefas. Dados de tempo de execução não são tratados pela ferramenta.

Rastros de execução com dependências de tarefas

A ferramenta desenvolvida por Haugen et al. (2015) traz um gráfico no estilo espaço-tempo (Gantt). Há algumas outras funcionalidades como a identificação de dependências de tarefas (apenas o primeiro nível) a medida que o usuário passa o mouse sobre as caixas que representam as tarefas.

Como entradas, são utilizados a representação do DAG e os *traces* de execução. Essa ferramenta é desenvolvida especificamente para o ambiente de execução PaRSEC.

Temanejo

Temanejo (KELLER et al., 2012) é um *debugger* para o modelo de programação baseado em tarefas, onde o analista visualiza um DAG. Ele suporta grande parte dos ambientes de execução de aplicações baseadas em tarefas, como OmpSs, StarPU e PaRSEC. Suas funcionalidades são focadas em depuração, permitindo que o usuário possa identificar e consertar parâmetros e dependências de tarefas.

Delay Spotter

Delay Spotter (Huynh; Taura, 2017) é uma ferramenta, construída sobre o DAGViz, que possibilita a identificação de atrasos em ambientes de execução. Ela divide os estados dos *workers* em três categorias, permitindo a identificação de delays decorrentes de problemas de escalonamento. Como em ambientes heterogêneos com tarefas variadas, a presença de *delays* faz parte da execução das aplicações, essa ferramenta é pouco efetiva.

TaskInsight

TaskInsight (CEBALLOS et al., 2018) é uma ferramenta que objetiva identificar o comportamento de memória e seu impacto na execução de tarefas. Apesar de prover algumas estatísticas e possibilitar algumas identificações de anomalias, apenas essa análise não é o suficiente para identificar a maioria dos pontos de otimização de aplicações baseadas em tarefas.

StarVZ

Arcabouço que é objeto deste trabalho, o StarVZ (GARCIA PINTO et al., 2018) possui a visualização de dados mais avançada dentre as ferramentas citadas. Construído com uma abordagem de *script*, ele possui um grande poder de customização. Ele será detalhado no Capítulo 3.

2.2 Ferramental para Big Data

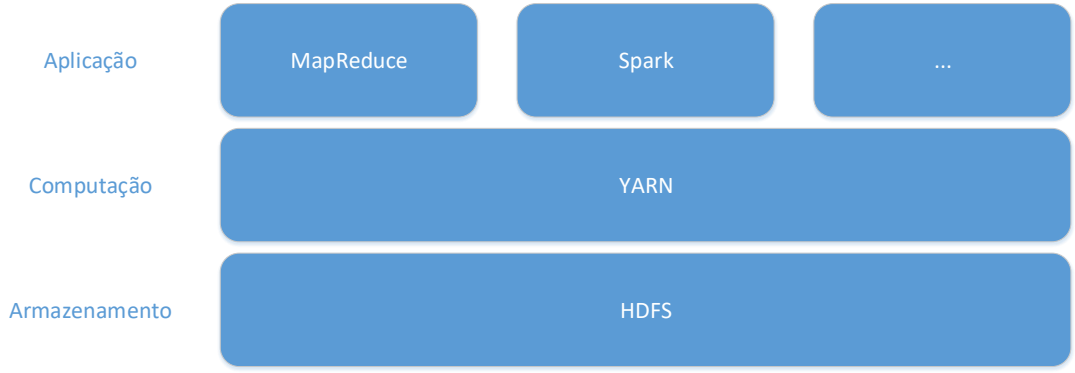
Esta Seção contextualiza as ferramentas para tratamento de grandes volumes de dados que serão utilizadas neste trabalho. Na Seção 2.2.1 contextualizamos a plataforma Hadoop e na Seção 2.2.2 falamos sobre Spark.

2.2.1 Hadoop

O Hadoop é um arcabouço que permite o processamento distribuído de grandes volumes de dados. A principal motivação que implicou em seu desenvolvimento foi o fato de que as velocidades de leitura e escrita dos discos rígidos não evoluíram da mesma forma que o seu tamanho. Ele resolve este problema lendo grandes volumes de dados de muitos discos, paralelizando a leitura.

O Hadoop pode ser separado em camadas, como mostra a Figura 2.1. A primeira camada, de aplicação, é a de mais alto nível, a qual o usuário usualmente interage. Ele suporta o modelo de programação MapReduce, originalmente desenvolvido pelo Google com o objetivo de processar grandes volumes de dados. Tal modelo baseia-se em duas primitivas presentes em linguagens funcionais: *map* e *reduce*. Esta abordagem foi adotada pois constantemente era necessário mapear pedaços de dados de uma entrada à uma chave identificadora e, em seguida, realizar algum tipo de computação sobre os dados mapeados uma mesma chave (DEAN; GHEMAWAT, 2008).

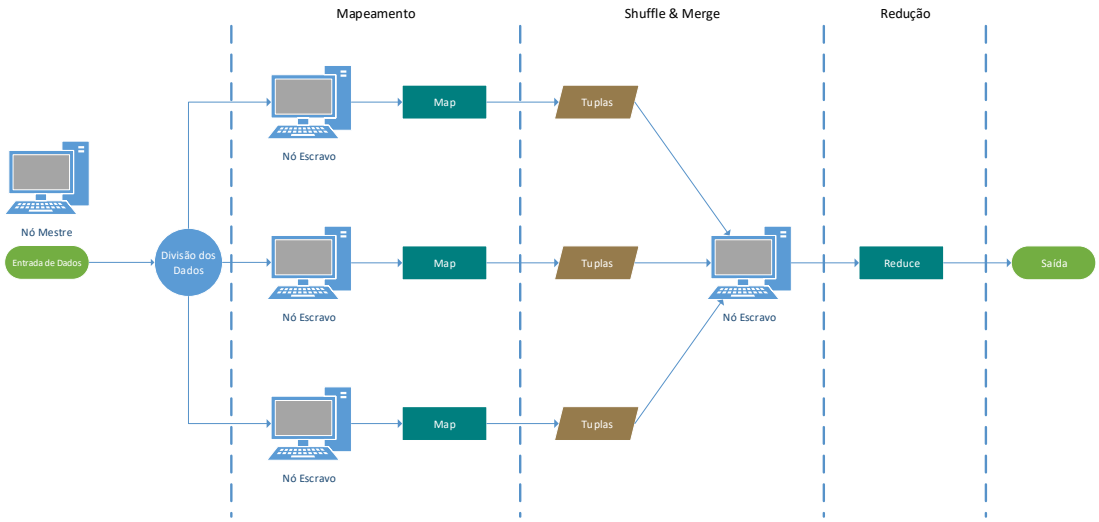
Figura 2.1: Organização do Hadoop em camadas.



A Figura 2.2 mostra o fluxo de execução básico do MapReduce. A Fase de *Shuffle & Merge* não será detalhada no trabalho, mas é importante salientar que

existe um processamento entre as fases de mapeamento e redução.

Figura 2.2: Fluxo de execução do MapReduce.



Primeiramente temos os dados sendo inseridos no Sistema de Arquivos Distribuído (*Distributed File System*) da plataforma. Os arquivos são divididos em pedaços de tamanho fixo e distribuídos sobre a plataforma. Isso permite que os dados sejam tratados de forma rápida pois há paralelização em sua leitura.

Com os dados armazenados, pode se iniciar a fase de mapeamento, que resulta em conjuntos de tuplas intermediárias. Estas sofrem um pré-processamento na fase de *Shuffle* e são disponibilizados para os nós que irão executar a fase de redução. Finalmente, os executores da redução obtêm os dados e desempenham suas tarefas, gerando a saída da execução.

Voltando a Figura 2.1, na segunda camada encontra-se o gerenciador de recursos do arcabouço. O YARN (*Yet Another Resource Negotiator*) foi introduzido no Hadoop 2.0 e além de melhorar a implementação do modelo MapReduce, suporta outros paradigmas de computação. Ele fornece APIs para requisitar e manipular recursos, não sendo comum o usuário interagir diretamente com essa camada.

Como o Hadoop trabalha sobre dados muito grandes, é comum que tais cargas de trabalho ultrapassem a capacidade de uma única máquina física. Isso implica na necessidade de particionar os dados sobre múltiplas máquinas. Os sistemas de arquivos que lidam com tal tipo de armazenamento são denominados sistemas de arquivos distribuídos (*distributed filesystems* ou *DFS*). Hadoop possui sua própria implementação de um *DFS*, inspirado no *Google File System*, denomi-

nado *Hadoop Distributed Filesystem*, que é representado como a terceira camada da Figura 2.1.

De forma similar a sistemas de arquivos convencionais, é utilizada a abstração de blocos, que consistem no volume mínimo para leitura e escrita. Enquanto nestes, eles têm tipicamente poucos bytes, o tamanho de blocos no HDFS é na ordem de megabytes, sendo por padrão, 128 (configurável pelo usuário, por arquivo). Tal abstração permite ao Sistema: armazenar arquivos maiores do que qualquer disco no ambiente; simplifica o gerenciamento do armazenamento de dados; e facilita replicação para garantir alta disponibilidade.

Um HDFS possui dois tipos de nós: *namenode* e *datanodes*. Eles seguem uma política mestre-escravo, onde o nó do primeiro tipo atua como mestre e gerencia o sistema de arquivos e nós do segundo tipo recebem requisições (de clientes ou do namenode) e armazenam ou recuperam blocos. Também é possível executar um *secondary namenode* para evitar pontos únicos de falha na arquitetura. O principal papel deste nó é manter uma cópia dos dados de gerência do *namenode* principal, em caso de falhas, ele pode assumir o lugar deste.

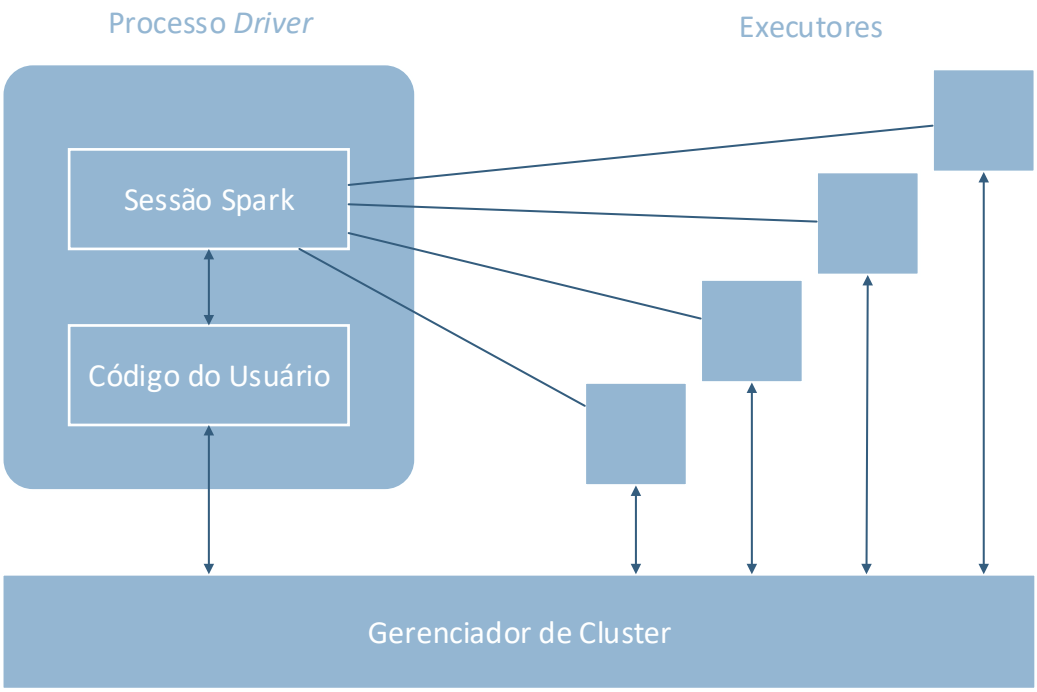
2.2.2 Spark

Spark é um conjunto de uma engine unificada e um conjunto de bibliotecas para processamento de dados distribuídos (CHAMBERS; ZAHARIA, 2018). A motivação para sua criação foi a necessidade de muitos tipos de processamento e a tendência do ferramental ser cada vez mais específico no cenário de Big Data.

A arquitetura do Spark pode ser visualizada na Figura 2.3. A engine usualmente é executada sobre um cluster de máquinas gerenciado por um `Gerenciador de Cluster`, que mantém o controle dos recursos disponíveis. Ele oferece suporte a três gerenciadores: um do próprio Spark, Hadoop YARN ou Mesos (HINDMAN et al., 2011).

O `Processo Driver` é responsável por manter o ciclo de vida da aplicação. Dentre suas responsabilidades estão: responder às entradas do programa do usuário; analizar, distribuir e agendar trabalho sobre os `Executores`; e manter informação relevante sobre a execução. Os `Executores` realizam duas funções: executam tarefas que lhes são atribuídas e reportam o estado das computações para o `Processo Driver`.

Figura 2.3: Arquitetura de uma aplicação Spark.



O Spark é atrelado a um modelo de programação similar ao MapReduce, todavia, possui uma abstração para compartilhamento de dados, chamada *Resilient Distributed Datasets* (RDDs). Estas, consistem em coleções de objetos tolerantes a falhas e que podem ser manipulados em paralelo e são particionados sobre a infraestrutura que executa a engine.

Spark manipula RDDs através de uma API unificada disponibilizada para diversas linguagens, que facilita o desenvolvimento de aplicações. Nela, o usuário é capaz de passar funções locais para serem executadas de forma distribuída. A avaliação dos RDDs é realizada de forma *Lazy*, o que significa que as manipulações são efetivadas apenas quando é executada uma ação nos dados, que se trata de uma instrução para computar o resultado de uma série de transformações. Isso permite que ele encontre um plano de execução eficiente pois pode agregar operações, otimizando o tempo de execução.

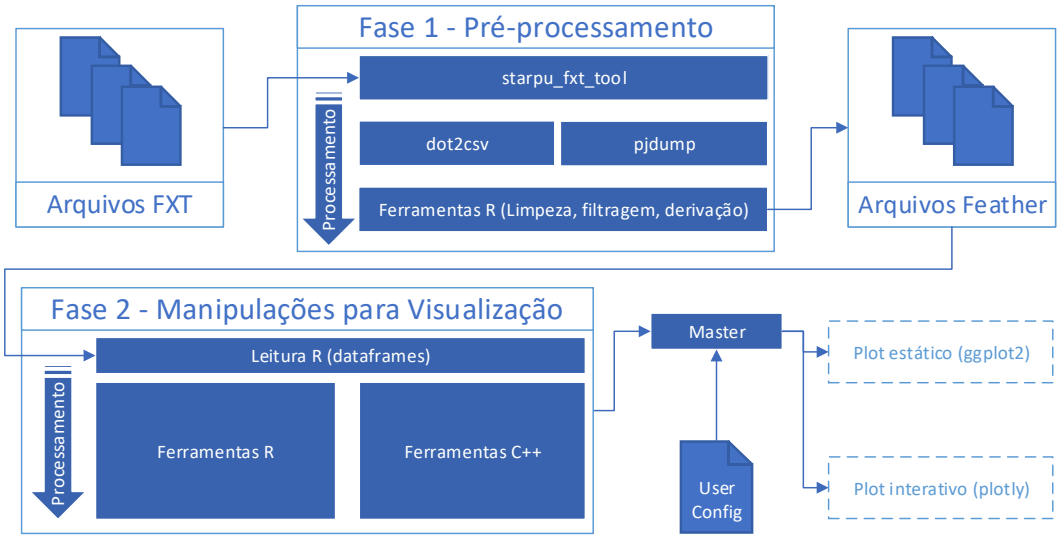
3 O ARCABOUÇO STARVZ

Este Capítulo descreve em detalhes o arcabouço StarVZ. Este é apresentado genericamente na Seção 3.1, a Seção 3.2 apresenta de forma detalhada as fases de sua execução, a Seção 3.3 disserta sobre otimizações já propostas e testadas e a Seção 3.4 fala sobre a motivação e a abordagem adotadas neste trabalho.

3.1 Visão Geral

O StarVZ (GARCIA PINTO et al., 2018) é um fluxo de processamento de análise de desempenho cujo objetivo é auxiliar na avaliação e na verificação de hipóteses sobre a execução de aplicações baseadas em tarefas em ambientes heterogêneos, executados sobre o ambiente de execução StarPU (AUGONNET et al., 2011). Dentre as ferramentas de visualização de rastros desse tipo de aplicação, na pesquisa realizada foi identificado que o StarVZ se destaca pela utilização de ferramentas de Ciência de Dados para análise de desempenho.

Figura 3.1: Fluxo de processamento do StarVZ.



Construído com uma abordagem de *script*, ele possui grande poder de customização. No trabalho de Garcia Pinto et al. (2018), pode-se visualizar diversos gráficos de execução da aplicação: gráfico com comportamento de tarefas; gráfico com a quantidade de tarefas submetidas; o comportamento do ambiente de execução, com os estados dos *workers* StarPU; quantidade de tarefas prontas; taxa de

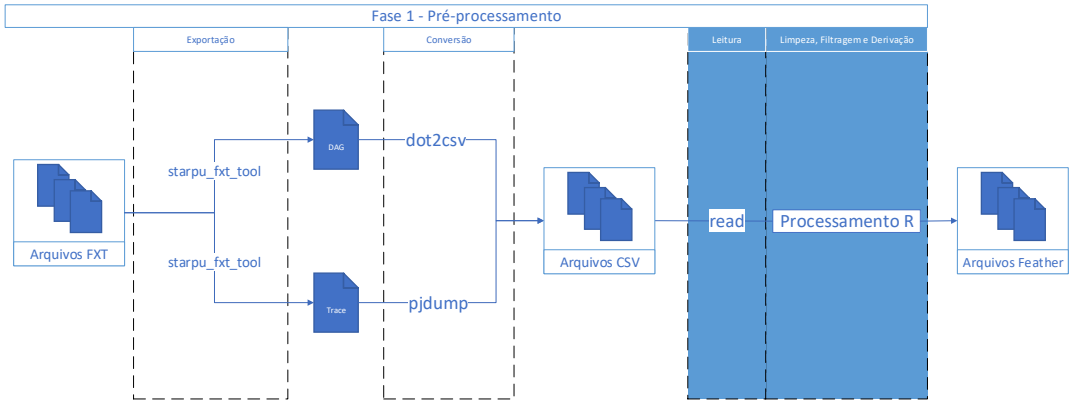
GFlops do ambiente; tráfego de dados entre a memória das GPUs; transferências de rede MPI; e o número de operações MPI concorrentes.

Ele é composto de duas fases, que podem ser visualizadas de forma simplificada na Figura 3.1. Cada uma delas é formada por uma combinação de diversas ferramentas, resultando em um arcabouço rápido, consistente, flexível e versátil.

3.2 Fases

Na primeira fase, que pode ser visualizada na Figura 3.2, os rastros de execução do StarPU, que são arquivos no formato binário FXT, são transformados e exportados através da ferramenta `starpu_fxt_tool` para dois arquivos: DAG no formato DOT e Trace no formato PAJÉ. Essa etapa gera eventos datados, que descrevem o comportamento da aplicação para todos os recursos envolvidos. Também são gerados dados sobre o ambiente de execução do StarPU, como número de tarefas submetidas, número de tarefas prontas, arquitetura da plataforma, etc.

Figura 3.2: Fluxo de pré-processamento do StarVZ.



Em seguida, é realizada uma verificação de integridade estrutural e temporal dos arquivos `trace`. Ela é executada via `pj_dump`, gerando cinco arquivos: o arquivo *states* possui informações sobre as tarefas executadas e seu comportamento no ambiente de execução; *variables* consiste em métricas de desempenho da plataforma e ambiente de execução; *events* possui informações sobre a utilização de recursos; *links* contém dados sobre comunicação MPI; e as informações de plataforma são registradas no arquivo *entities*. O DAG também passa por uma conversão e por fim, todos os arquivos são gravados no formato de Valores Sepa-

rados por Vírgula (*Comma-Separated Value*, comumente abreviado para CSV).

Finalmente, os dados escritos em CSV são lidos, filtrados, agregados e combinados em uma ferramenta implementada na linguagem R, utilizando-se bibliotecas oriundas do pacote `tidyverse` para a manipulação de dados. As saídas são escritas no formato *FEATHER* (WICKHAM, 2019).

A segunda fase do fluxo de processamento inicia pela leitura das saídas da anterior. Cada arquivo torna-se uma tabela, e os dados são unificados em uma lista. É possível ter múltiplos rastros de aplicações sendo analisados em paralelo para comparação, basta multiplicar a leitura com entradas diferentes e elas podem posteriormente ser combinadas em uma única visualização. A criação dos gráficos ainda possui processamento de dados, o que traz certa flexibilidade para as visualizações. Como última etapa dessa fase, o usuário parametriza o sistema com um arquivo de configuração no formato YAML, para que a montagem da visualização seja configurável. Finalmente, o usuário pode analisar os gráficos gerados pela ferramenta.

Nos experimentos realizados em Garcia Pinto et al. (2018), em uma máquina equipada com um Intel(R) Xeon(R) CPU E3-1225 v3 @3.20GHz e 32GB de memória principal e com uma entrada de aproximadamente 18GB, a primeira fase do fluxo de processamento levou cerca de 32 minutos para executar. A segunda fase, para executar a leitura dos arquivos *FEATHER* e gerar a visualização levou em torno de 2 minutos.

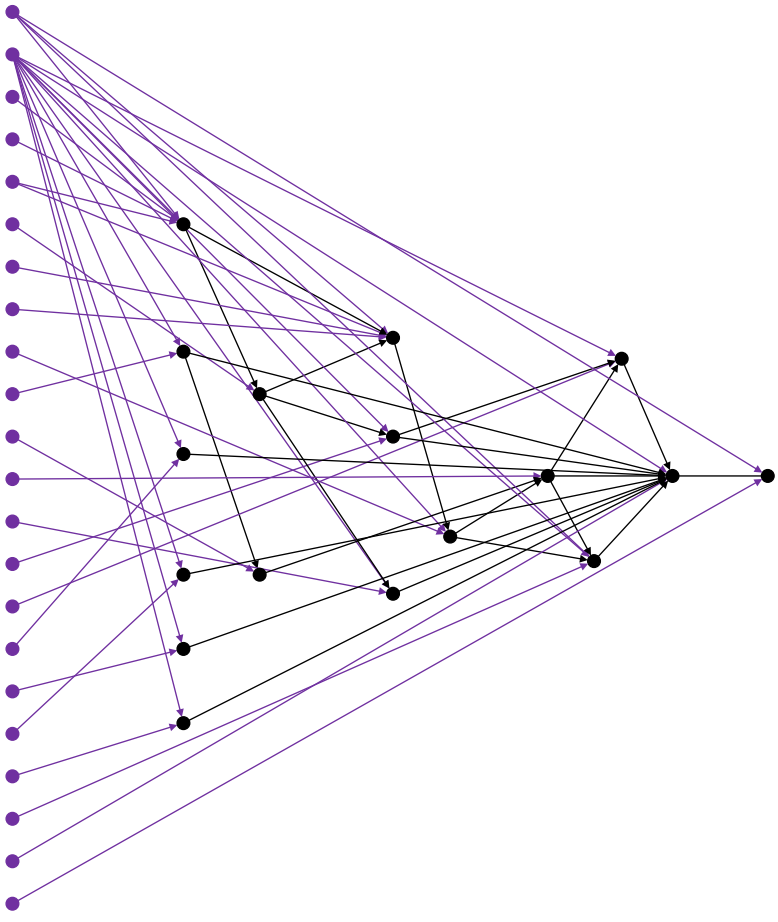
3.3 Trabalhos Relacionados

O trabalho de Alles and Schnorr (2018) foi o primeiro a tentar otimizar o fluxo do StarVZ. Sua principal motivação foi a melhoria de desempenho da etapa de manipulação de dados, a mais custosa de acordo com os experimentos observados, com o objetivo de permitir o processamento de entradas maiores em um tempo aceitável. Nele utilizou-se `Drake` (LANDAU et al., 2019), uma biblioteca para a linguagem de programação R, cujo foco é executar apenas as partes necessárias de um fluxo de processamento de análise de dados, evitando os passos desnecessários que não mudarão suas saídas. Isso é realizado modelando as computações como um Grafo Acíclico Direcionado (*directed acyclic graph* ou DAG) de tarefas e armazenando em cache os resultados daquelas já executadas. Além

disso, `Drake` também possui suporte a paralelismo (*Implicit parallelism*) para a execução de tarefas independentes.

Para modelar o StarVZ dessa forma, foram necessárias mudanças cujo objetivo era explicitar dependências e postergar junções de tabelas. Depois da identificação dos fluxos independentes no fluxo de processamento, foi utilizado `Drake` para criar um plano de execução, que consiste na declaração das tarefas onde cada uma é representada por uma função R. Na criação do plano, a biblioteca analisa cada uma das tarefas, suas entradas e saídas, para determinar dependências e gerar o DAG. O resultado da modelagem do StarVZ nessa abordagem pode ser visualizado na Figura 3.3.

Figura 3.3: DAG de execução do StarVZ.



Fonte: Inspirada na Figura de contida no trabalho de Alles and Schnorr (2018)

Com esta modelagem, não foram observadas melhorias de desempenho no StarVZ. De acordo com Alles and Schnorr (2018), a cache de resultados intermediários da biblioteca acaba tendo que armazenar muitos dados em disco devido ao tamanho das tabelas geradas pelas entradas, prejudicando o tempo to-

tal de processamento. Em relação ao suporte a paralelismo, ele é limitado pela falta de suporte nativo a *multithread* da linguagem R. Por isso, `Drake` oferece essa funcionalidade instanciando múltiplas sessões R, que são processos separados. A comunicação entre esses processos é realizada pelo mesmo sistema de cache citado anteriormente, consequentemente, resultando nos mesmos problemas.

Outra abordagem que poderia ser adotada para melhorar o desempenho do StarVZ seria distribuir os fluxos independentes do fluxo de processamento pois elas podem trazer um ganho de desempenho em um ambiente viável. Ao simplesmente paralelizar, é possível que os mesmo problemas com o tamanho das tabelas sejam enfrentados, exigindo máquinas com muita memória para obter-se os resultados desejados.

A primeira opção seria distribuir os fluxos independentes utilizando o pacote `Snow`, que significa *Simple Network of Workstations* (TIERNEY, 2018). Este é desenvolvido para linguagem R e permite a utilização de *Explicit parallelism*, oferecendo integração com três interfaces de baixo nível: PVM (*Parallel Virtual Machine*), através do pacote rpvm; MPI, através do pacote Rmpi (YU, 2013); e uma integração com sockets, no caso de não ter nenhuma das outras opções disponíveis no ambiente. A utilização de `Snow` pode ser feita através da biblioteca `Snowfall` (KNAUS, 2015), que é uma camada cujo objetivo é melhorar a usabilidade, facilitando ainda mais a utilização.

A segunda alternativa seria a utilização da biblioteca `future` (BENGTSSON, 2019). Esta provê uma forma simples e uniforme de avaliar expressões de forma assíncrona, utilizando recursos diversos. Ela funciona utilizando o conceito de abstração *future*, que basicamente define um valor que poderá estar disponível no futuro. Tal valor pode ter seu estado como resolvido ou não resolvido, estado em que se ele for utilizado bloqueará o processo até sua resolução. Os *futures* podem ser resolvidos de diversas formas: sequential, onde são resolvidos sequencialmente no processo R corrente; multiprocess, que resolve utilizando *multicore*; cluster, que é o mais interessante para adoção pois resolve com sessões R na máquina local e/ou em máquinas remotas; e etc.

3.4 Motivação e Abordagem

Há algum tempo, vivemos na era dos dados. O volume armazenado em meios eletrônicos é dificil de medir, mas existem alguns estudos que podem nos auxiliar a ter uma noção disso. A IDC (*International Data Corporation*) (ZWOLENSKI; WEATHERILL, 2014) estima que, de acordo com o histórico de crescimento do mundo digital, este deve crescer entre 2013 e 2020 de 4.4 para 44 zettabytes [1].

Conforme explicitado na Seção 3.2, o tempo total de execução da primeira Fase do StarVZ, para uma carga de trabalho de 18 gigabytes levou cerca de 32 minutos. Para viabilizar o processamento de maiores volumes de dados em um tempo aceitável, trazendo as vantagens da utilização da ferramenta para esses cenários, é necessário otimizar esta Fase do fluxo de processamento.

Decompondo esse tempo entre todas as ferramentas utilizadas, a Tabela 3.1 exibe o tempo de cada ferramenta no fluxo de processamento para a execução mencionada no trabalho de Garcia Pinto et al. (2018). Como o volume de dados utilizado foram apenas 18 gigabytes, essas aplicações não possuem um tempo de execução aceitável para tratamento de grandes volumes de dados.

Tabela 3.1: Arcabouço StarVZ - Tempos de execução da primeira Fase.

Ferramenta	**Tempo de Execução**	**Tempo Relativo**
starpu_fxt_tool	10 minutos	31,25%
pj_dump	9 minutos	28,12%
Ferramenta R	13 minutos	40,62%
Total	32 minutos	-

Como na segunda fase, não é evidente um ponto de otimização urgente, tendo em vista que nos experimentos foram gerados resultados em poucos minutos, a motivação deste trabalho é a otimização do ponto mais crítico do tempo de execução da primeira fase do fluxo de processamento. Como as ferramentas desenvolvidas em R manipulam tabelas, fazendo operações comuns a Ciência de Dados, este trabalho migrará as tabelas normais para tabelas Spark, com o intuito de distribuir o trabalho em diversas máquinas. Além disso, para esta adaptação suportar arquivos muito grandes, essa instância do Spark irá executar sobre o

[1] Um zettabyte significa um trilhão de gigabytes.

Hadoop, garantindo que os dados terão uma manipulação otimizada.

4 CONTRIBUIÇÃO: STARVZ SOBRE SPARK

Já pode ser escrito, finalizou-se a implementação a ponto de poder realizar experimentos.

4.1 Arquitetura Proposta

4.2 Implementação

5 RESULTADOS E AVALIAÇÃO

Serão iniciados na sequência (provavelmente 20/07).

6 CONCLUSÃO E TRABALHOS FUTUROS

Aguardando dois últimos capítulos.

REFERÊNCIAS

ALLES, G. R.; SCHNORR, L. M. Parallel workflow support for starvz using drake. **WSPPD Proceedings**, p. 23 – 25, 2018. Available from Internet: <http://inf.ufrgs.br/gppd/wsppd/2018/proceedings.php>.

AUGONNET, C. et al. Starpu: A unified platform for task scheduling on heterogeneous multicore architectures. **Concurr. Comput. : Pract. Exper.**, John Wiley and Sons Ltd., Chichester, UK, v. 23, n. 2, p. 187–198, feb. 2011. ISSN 1532-0626. Available from Internet: <http://dx.doi.org/10.1002/cpe.1631>.

BENGTSSON, H. **future Github Repository**. 2019. [Acessed: 2019-06-25]. Available from Internet: <https://github.com/HenrikBengtsson/future>.

CEBALLOS, G. et al. Analyzing performance variation of task schedulers with taskinsight. **Parallel Computing**, v. 75, p. 11 – 27, 2018. ISSN 0167-8191. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0167819118300346>.

CHAMBERS, B.; ZAHARIA, M. **Spark: The Definitive Guide : Big Data Processing Made Simple**. O'Reilly Media, 2018. ISBN 9781491912218. Available from Internet: <https://books.google.com.br/books?id=urjpAQAACAAJ>.

COULOMB, K. et al. **Visual Trace Explorer**. 2009. [Acessed: 2019-05-05]. Available from Internet: <http://vite.gforge.inria.fr/index.php>.

DEAN, J.; GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. **Commun. ACM**, ACM, New York, NY, USA, v. 51, n. 1, p. 107–113, jan. 2008. ISSN 0001-0782. Available from Internet: <http://doi.acm.org/10.1145/1327452.1327492>.

Dosimont, D. et al. A spatiotemporal data aggregation technique for performance analysis of large-scale execution traces. In: **2014 IEEE International Conference on Cluster Computing (CLUSTER)**. [S.l.: s.n.], 2014. p. 149–157. ISSN 1552-5244.

ESCHWEILER, D. et al. Open trace format 2: The next generation of scalable trace formats and support libraries. In: . [S.l.: s.n.], 2012. v. 22, p. 481 – 490. ISBN 9781614990406.

GARCIA PINTO, V. et al. A visual performance analysis framework for task-based parallel applications running on hybrid clusters. **Concurrency and Computation: Practice and Experience**, v. 30, n. 18, p. e4472, 2018. E4472 cpe.4472. Available from Internet: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4472>.

HAUGEN, B. et al. Visualizing execution traces with task dependencies. In: **Proceedings of the 2Nd Workshop on Visual Performance Analysis**. New York, NY, USA: ACM, 2015. (VPA '15), p. 2:1–2:8. ISBN 978-1-4503-4013-7. Available from Internet: <http://doi.acm.org/10.1145/2835238.2835240>.

HINDMAN, B. et al. Mesos: A platform for fine-grained resource sharing in the data center. In: **Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2011. (NSDI'11), p. 295–308. Available from Internet: <http://dl.acm.org/citation.cfm?id=1972457.1972488>.

Huynh, A.; Taura, K. Delay spotter: A tool for spotting scheduler-caused delays in task parallel runtime systems. In: **2017 IEEE International Conference on Cluster Computing (CLUSTER)**. [S.l.: s.n.], 2017. p. 114–125. ISSN 2168-9253.

HUYNH, A. et al. Dagviz: A dag visualization tool for analyzing task-parallel program traces. In: **Proceedings of the 2Nd Workshop on Visual Performance Analysis**. New York, NY, USA: ACM, 2015. (VPA '15), p. 3:1–3:8. ISBN 978-1-4503-4013-7. Available from Internet: <http://doi.acm.org/10.1145/2835238.2835241>.

ISAACS, K. **Ravel MPI trace visualization tool**. 2014. [Acessed: 2019-05-05]. Available from Internet: <https://github.com/LLNL/ravel>.

KELLER, R. et al. Temanejo: Debugging of thread-based task-parallel programs in starss. In: BRUNST, H. et al. (Ed.). **Tools for High Performance Computing 2011**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 131–137.

KNAUS, J. **Snowfall Cran Page**. 2015. [Acessed: 2019-06-25]. Available from Internet: <https://CRAN.R-project.org/package=snowfall>.

KNÜPFER, A. et al. The vampir performance analysis tool-set. In: RESCH, M. et al. (Ed.). **Tools for High Performance Computing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 139–155. ISBN 978-3-540-68564-7.

LANDAU, W. M. et al. **Drake Github Repository**. 2019. [Acessed: 2019-06-27]. Available from Internet: <https://github.com/ropensci/drake>.

Pagano, G. et al. Trace management and analysis for embedded systems. In: **2013 IEEE 7th International Symposium on Embedded Multicore Socs**. [S.l.: s.n.], 2013. p. 119–122.

PILLET, V. et al. Paraver: A tool to visualize and analyze parallel code. **WoTUG-18**, v. 44, 03 1995.

STEIN, B. d. O. et al. **Pajé - trace file format**. 2015. [Acessed: 2019-05-07]. Available from Internet: <https://github.com/schnorr/pajeng/blob/master/doc/lang-paje/lang-paje.pdf>.

TIERNEY, L. **Snow Home Page**. 2018. [Acessed: 2019-06-25]. Available from Internet: <http://homepage.divms.uiowa.edu/~luke/R/cluster/cluster.html>.

WICKHAM, H. **feather: R Bindings to the Feather 'API'**. 2019. [Acessed: 2019-06-14]. Available from Internet: <https://cran.r-project.org/web/packages/feather/index.html>.

YU, H. **RMPI Home Page**. 2013. [Acessed: 2019-06-24]. Available from Internet: <http://fisher.stats.uwo.ca/faculty/yu/Rmpi/>.

ZWOLENSKI, M.; WEATHERILL, L. The digital universe rich data and the increasing value of the internet of things. **Australian Journal of Telecommunications and the Digital Economy**, v. 2, 10 2014.