UNIVERSIDADE FEEVALE

FELIPE ALFREDO KUNZLER

Avaliação de desempenho entre algoritmos sequenciais e suas implementações distribuídas no Apache Spark

(Título Provisório)

Anteprojeto de Trabalho de Conclusão

Novo Hamburgo

2018

FELIPE ALFREDO KUNZLER

Avaliação de desempenho entre algoritmos sequenciais e suas implementações distribuídas no Apache Spark

(Título Provisório)

Anteprojeto de Trabalho de Conclusão de Curso, apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação pela Universidade Feevale

Orientador: Juliano Varella de Carvalho

Novo Hamburgo

2018

RESUMO

O constante aumento no volume de dados produzido todos os dias por novas tecnologias de comunicação, dispositivos móveis e redes sociais, trouxe diversos desafios que se referem ao processamento destes dados em tempos aceitáveis. O modelo de programação Map Reduce proposto pela Google, permite que programas sejam expressos através de duas funções: *map* e *reduce*, que podem ser executadas paralelamente por centenas ou milhares de computadores, viabilizando o processamento de altos volumes de dados. Apache Hadoop foi um dos primeiros e mais populares *frameworks* a prover uma implementação para o modelo de programação Map Reduce, onde vários módulos como o Hadoop Distributed File System (HDFS) e Hadoop YARN foram desenvolvidos de forma a auxiliar o processamento de dados pelo Map Reduce, provendo alta estabilidade e facilitando o gerenciamento dos recursos do cluster. Entretanto, a arquitetura proposta pelo Hadoop Map Reduce só possibilita o reuso de dados entre diferentes tarefas através da escrita e leitura de dados pelo disco. Assim, diversos algoritmos de propriedade iterativa, que precisam aplicar funções repetitivamente sobre os mesmos dados, acabam por não obter tempos de execução ótimos quando implementados com Map Reduce. Uma nova abordagem para a computação de alto volumes de dados é proposta pelo Apache Spark. Focando no processamento distribuído em memória, o Spark possibilita o reuso de dados entre diferentes tarefas através da memória primária ao invés do disco rígido, trazendo melhor desempenho na execução em várias classes de algoritmos. Desta maneira, neste trabalho se propõe o estudo e a investigação da plataforma Apache Spark, assim como o desenvolvimento de algoritmos sequenciais convencionais e de forma paralela, através das APIs estruturadas disponibilizadas pela plataforma. Além disso, os resultados obtidos em ambas abordagens serão analisados e comparados, de forma a variar o volume de dados e outros parâmetros como o número de nodos do cluster.

Palavras-chave: Apache Spark. Computação distribuída. Algoritmos. Big Data.

SUMÁRIO

MOTIVAÇÃO

Nunca antes tanta informação foi gerada e disponibilizada para consulta e processamento. A cada segundo, grandes quantidades de dados surgem e são transferidas para todos os cantos do mundo. A digitalização em massa que vem ocorrendo nos últimos anos, somado a grande disponibilidade de dispositivos conectados a internet e ligados a sensores, já são estimados por ultrapassar a população humana em quantidade. Assim, disponibilizando altos volumes de dados que podem ser levados em consideração para análise e criação de valor (DE MAURO; GRECO; GRIMALDI, 2014).

Esse conceito se tornou popular como *Big Data*. Cenário, onde o volume de dados, o número de transações e as diferentes fontes de informação são tão grandes e complexas, que requerem ferramentas, tecnologias e processos especializados para permitir a viabilização do seu processamento em tempos aceitáveis (SU; PATTNAIK, 2016).

Big Data é comumente associado aos três V's, sendo estes: Volume, Velocidade e Variedade. O Volume se refere ao montante de dados, indo de *gigabytes*, *terabytes* até *zettabytes*. A Velocidade é a alta taxa de transferência e atualização dos dados, criando uma corrente constante de novos dados, que acaba por deixar uma estreita janela para o processamento em tempo real. Por último, a Variedade engloba as diferentes fontes de onde os dados são gerados, assim como os diferentes formatos, podendo ser estruturados como em um banco de dados relacional; semiestruturados como arquivos XML; e não estruturados, como textos, imagens e vídeos (SU; PATTNAIK, 2016).

Diversos desafios nasceram deste contexto, simplesmente obter e armazenar essas grandes quantidades de dados não é mais suficiente, e seu processamento não pode ocorrer usando a mesma infraestrutura e ferramentas tradicionais. Surgiu a necessidade de criar rápidas e eficientes maneiras para processar e analisar tais volumes de dados (ELGENDY; ELRAGAL, 2014).

Map Reduce é um modelo de programação paralelo proposto pela Google em 2004. Esse modelo permite que programas sejam expressos a partir de duas funções principais: o *map*, onde os dados são inicialmente transformados em uma lista de pares contendo chaves e valores; e o *reduce*, que é responsável por processar o conjunto de valores de uma determinada chave. Essa abstração foi inspirada em linguagens de programação funcionais como Lisp, que primam

o uso de funções sem efeitos colaterais, assim como a imutabilidade, facilitando a paralelização de programas (DEAN; GHEMAWAT, 2004).

O modelo funcional do Map Reduce permite que cada função *map* e *reduce* seja executada isoladamente por dezenas ou centenas de computadores. Assim proporcionando a paralelização e distribuição do processamento em larga escala de forma automática. O *framework* de implementação do Map Reduce fica então responsável pelos detalhes técnicos da distribuição, como o particionamento dos dados de entrada, a divisão das tarefas entre os nodos, a tolerância a falhas, e a comunicação pela rede (DEAN; GHEMAWAT, 2004).

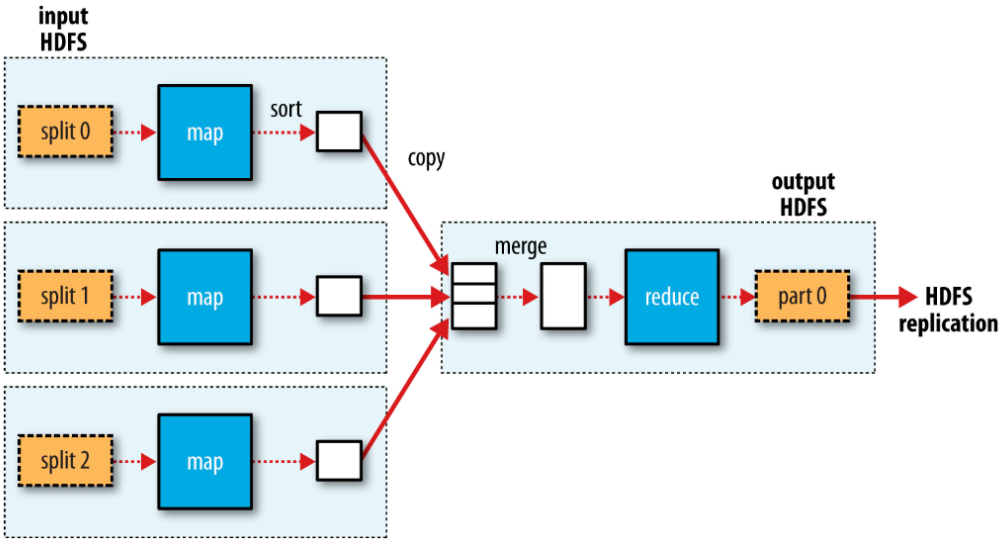

Figura 1 - Modelo de execução simples de um programa Map Reduce.
Fonte - (WHITE, 2015)

A Figura 1 ilustra o fluxo de dados de um programa Map Reduce simples, que contém apenas uma tarefa de *reduce*. Cada bloco pontilhado azul representa um nodo no cluster, que é responsável por processar uma parte dos dados (*splits*) através da função *map.* Após este processamento, os resultados de cada nodo são ordenados e enviados para o nodo *reducer*, onde o conjunto de chaves e valores serão processados pela função *reduce* especificada (WHITE, 2015).

Apache Hadoop foi um dos primeiros e mais populares *frameworks* a prover uma implementação para o modelo de programação Map Reduce. São fornecidos módulos que permitem o processamento distribuído de grandes conjuntos de dados em *clusters* de computadores, que podem escalar de poucas dezenas até milhares de máquinas. Os principais projetos são: Hadoop Map Reduce, a implementação do modelo para processamento paralelo proposto pela Google; Hadoop YARN, responsável pelo gerenciamento de recursos do *cluster*;

Hadoop Distributed File System (HDFS), um sistema de arquivos distribuído de alta performance (APACHE SOFTWARE FOUNDATION, 2018).

Fundamentalmente, Hadoop Map Reduce é um sistema de processamentos em *batches*, com foco na alta taxa de transferência de dados (WHITE, 2015). Porém, essa arquitetura só possibilita o reuso de dados entre diferentes *jobs* através de escritas e leituras em disco, que pode acabar por gerar uma sobrecarga no sistema de arquivos durante a execução de algoritmos que dependem desta propriedade (SAMADI; ZBAKH; TADONKI, 2017).

Algoritmos de computação iterativa, como muitos em aprendizado de máquina e análise de grafos, necessitam aplicar funções repetitivamente sobre os mesmos dados para alcançar seus objetivos. Dessa forma, cada iteração é geralmente representada por um *job* Map Reduce, que precisam recarregar os dados processados pela iteração anterior por disco. Assim, a execução desta classe de algoritmos usando Hadoop Map Reduce acaba por não obter tempos de execução ótimos (SAMADI; ZBAKH; TADONKI, 2017).

Uma nova abordagem para o processamento deste tipo de algoritmo é proposta pelo Apache Spark. Este *framework* se baseia em duas principais abstrações: RDDs (*Resilient Distributed Dataset*), uma estrutura de dados que representa uma coleção de dados distribuída e imutável; e *parallel operations*, um conjunto de operações paralelas que podem ser aplicadas a um *dataset*, como por exemplo, *map*, *flatmap*, *reduce*, *filter* e *foreach*. Adicionalmente, operações de *caching* podem ser usadas para manter o estado da computação atual em memória, permitindo o compartilhamento de dados intermediários entre diferentes nodos (ZAHARIA et al., 2010).

O Apache Spark foi inicialmente desenvolvido na University of California, Berkeley em 2009, provendo *APIs* de baixo nível, que possibilitaram o processamento distribuído de algoritmos iterativos em memória. Conforme o crescimento e adoção do Apache Spark, novos componentes e bibliotecas foram sendo adicionados à plataforma, como *APIs* estruturadas que facilitam o desenvolvimento (*SQL, Dataframes e Datasets*); suporte para *stream processing*; componentes para *machine learning* (MLib) e grafos (GraphX) (CHAMBERS; ZAHARIA, 2018).

Desta forma, neste trabalho se propõe o estudo e a investigação da plataforma Apache Spark, assim como o desenvolvimento de algoritmos sequenciais convencionais e de forma paralela, através das *APIs* estruturadas disponibilizadas pela plataforma. Analisando e comparando os resultados obtidos em ambas abordagens, de forma a variar o volume de dados e outros parâmetros como o número de nodos do *cluster*.

OBJETIVOS

Objetivo geral

Desenvolver algoritmos sequenciais convencionais e de forma paralela usando a plataforma de processamento distribuído Apache Spark, avaliando e comparando o desempenho de ambas abordagens.

Objetivos específicos

- Pesquisar sobre a plataforma de processamento distribuído Apache Spark;
- Investigar a abordagem e arquitetura do Apache Spark para solucionar problemas distribuídos;
- Analisar e definir algoritmos para implementação, assim como o *dataset* a ser utilizado;
- Desenvolver os algoritmos de modo sequencial;
- Desenvolver os algoritmos de modo paralelo, usando o Apache Spark;
- Avaliar e comparar o desempenho das implementações sequenciais convencionais e paralelas em Spark, variando volume dos dados e outros parâmetros;
- Avaliar desempenho da implementação paralela em relação ao número de nodos.

METODOLOGIA

A metodologia a ser seguida neste trabalho tem caráter de pesquisa aplicada, onde se fará uso de conhecimentos e métodos existentes com o intuito de alcançar os objetivos propostos. Inicialmente, será feita a busca de material bibliográfico, trazendo um embasamento teórico e entendimento do contexto referente à computação distribuída e ao *framework* Apache Spark.

Dado o entendimento das técnicas e possibilidades trazidas pelo *framework* a ser utilizado, será feita uma investigação a fim de definir quais os algoritmos convencionais serão analisados neste trabalho. Também serão buscados um ou mais *datasets* para possibilitar a validação dos algoritmos estudados.

Uma vez que os algoritmos forem definidos, será realizado um trabalho de implementação destes de maneira sequencial, inicialmente fora do *framework* Apache Spark, de forma a aprofundar o conhecimento no algoritmo em questão. Em seguida, será realizado o desenvolvimento destes mesmos algoritmos de forma paralela, usando o *framework* Apache Spark e suas principais abstrações que visam facilitar a computação distribuída.

Experimentos serão conduzidos para ambas abordagens em questão, utilizando o mesmo *dataset*, volume de dados e parâmetros dos algoritmos, a fim de obter uma análise comparativa do desempenho de cada abordagem. Os algoritmos sequenciais serão executados em um ambiente convencional, enquanto os algoritmos paralelos desenvolvidos no Spark serão executados em um ambiente distribuído, onde experimentos adicionais serão conduzidos variando as configurações do *cluster*, como o número de nodos.

Por fim, será realizada a análise comparativa dos resultados encontrados, buscando o entendimento do desempenho de ambas abordagens em relação às diferentes configurações e parâmetros utilizados.

CRONOGRAMA

Trabalho de Conclusão I

Etapa	Meses			
	Ago	Set	Out	Nov
Análise do contexto da computação distribuída				
Construção do anteprojeto				
Estudo da plataforma Apache Spark				
Pesquisa sobre os trabalhos relacionados				
Definir algoritmos a serem implementados				
Entrega do TC I				

Trabalho de Conclusão II

Etapa	Meses			
	Mar	Abr	Mai	Jun
Implementar os algoritmos de forma sequencial				
Implementar os algoritmos de forma paralela usando Apache Spark				
Análise das abordagens para execução dos experimentos (On premise e Cloud)				
Execução dos experimentos				
Análise e comparação dos resultados				
Entrega do TC II				

BIBLIOGRAFIA

APACHE SOFTWARE FOUNDATION. **Apache Hadoop Project**. Disponível em: <http://hadoop.apache.org/>. Acesso em: 18 ago. 2018.

CHAMBERS, B.; ZAHARIA, M. **Spark : The Definitive Guide**. [s.l.] O'Reilly Media, 2018.

DE MAURO, A.; GRECO, M.; GRIMALDI, M. What is Big Data? A Consensual Definition and a Review of Key Research Topics. **AIP Procedings**, v. 1644, n. SEPTEMBER, p. 97–104, 2014.

DEAN, J.; GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. **Proceedings of 6th Symposium on Operating Systems Design and Implementation**, p. 137–149, 2004.

ELGENDY, N.; ELRAGAL, A. Big Data Analytics: A Literature Review Paper. **Advances in Data Mining. Applications and Theoretical Aspects**, v. 8557, p. 214–227, 2014.

SAMADI, Y.; ZBAKH, M.; TADONKI, C. Comparative study between Hadoop and Spark based on Hibench benchmarks. **Proceedings of 2016 International Conference on Cloud Computing Technologies and Applications, CloudTech 2016**, p. 267–275, 2017.

SU, X.; PATTNAIK. Introduction to Big Data Analysis. **Ntnu**, p. 1–20, 2016.

WHITE, T. **Hadoop: The Definitive Guide**. 4th. ed. [s.l.] O'Reilly Media, 2015.

ZAHARIA, M. et al. Spark : Cluster Computing with Working Sets. **HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing**, p. 10, 2010.