UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MATHEUS MIGNONI

Aferindo performance na geração automática de Tesauros com técnicas de BigData

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Claudio Fernando Resin Geyer
Co-orientador: Prof. Dr. Aline Villavicencio

Porto Alegre
2017

RESUMO

Este trabalho contempla, através da utilização de grandes massas de dados (Big Data) e cluster de computadores, os benefícios que as técnicas de computação distribuída podem prover no processamento de linguagens naturais (PLN), mais especificamente, na geração automática de Tesauros. Baseando-se em um pacote de programas existentes para geração de modelos semânticos distribucionais, que suportavam apenas a utilização de múltiplas threads, foi desenvolvido uma implementação, a partir do Framework Apache Flink, capaz de usufruir das vantagens existentes em um ambimente distribuído com múltiplas máquinas trabalhando em paralelo. A intenção da nova implementação é aprimorar a qualidade dos resultados e diminuir o tempo necessário para computação do mesmos, em comparação com simples implementações sequenciais. Os resultados obtidos mostram que alguns algoritmos obtêm ganhos relevantes e outros nem tanto. Foi concluído que a plataforma BigData pode auxiliar na geração de Tesauros, aumentando a capacidade de processameto de tarefas, antes pouco paralelizáveis, sem implicar em custo de codificação muito elevado para o programador.

Palavras-chave: Apache Flink. Tesauros. Performance. Big Data.

Automatic generation of Thesaurus with BigData techniques

ABSTRACT

This document present, through the use of large data masses (Big Data) and cluster of computers, the benefits of distributed computing techniques can provide in the processing of natural languages processing(NLP), more specifically in the automatic generation of Thesaurus. Based on a package of existing programs for generation of distributional semantic models, which only supported the use of multiple threads, an implementation was developed, from the Apache Flink Framework, able to take advantage of existing advantages in a distributed environment with multiple machines Working in parallel. The new implementation intent to improve the quality of the results and decrease the time required to compute them, compared to simple sequential implementations. The results obtained show that some algorithms obtain relevant gains and others not so much. It was concluded that the BigData platform is undoubtedly the future of computing large amounts of data, however, there are spaces for optimizations.

Keywords: Apache Flink. Thesauri. Performance. Big Data.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CPU	Central Processing Unit
HDFS	Hadoop Distributed File System
PLN	Processamento de Linguagem Natural
VM	Virtual Machine

SUMÁRIO

1 INTRODUÇÃO

A sociedade está cada vez mais instrumentada e, como resultado, está produzindo e armazenando informações em grande quantidade. Gerenciar e obter informações sobre os dados produzidos é um desafio, e também a chave, para uma vantagem competitiva. Soluções analíticas que focam os dados estruturados e desestruturados são importantes, pois eles podem nos ajudar a obter informações, não apenas a partir de dados privados, mas também de grandes quantidades de dados disponíveis publicamente na Web (ADAMS; RAUBAL, 2014). Ainda no ano de 1997, o termo "BigData"foi utilizado pela primeira vez, pela NASA, para descrever que Datasets estavam tornando-se tão grandes que não cabiam mais na memória de um único computador (COX; ELLSWORTH, 1997). Desde por volta de 2008 o termo tem sido usado pra descrever um interessante fenômeno na academia, governo, mercado e na mídia, no qual dados não são mais vistos como uma entidade com valor limitado depois de um primeiro uso, ao invés, eles são uma entrada para ser usada continuamente em inovação, serviço de criação, e como meio de coletar informaçãos não disponíveis anteriormente (MAYER-SCHöNBERGER; CUKIER, 2013).

Uma das maneiras de mitigar o custo de lidar com quantidades tão grandes de dados, é através da utilização de processamento paralelo e distribuído. MapReduce é o modelo de programação paralelo mais utilizado atualmente. Programas MapReduce são escritos em um determinado estilo influenciado por construções de programação funcional, e torna apto o processamento distribuído de operações Map e Reduce, facilitando a geração automática de programas distribuídos e eximindo o programador de preocupar-se com problemas provindos da paralelização (DEAN; GHEMAWAT, 2010)

Hadoop[1] é uma engine para processamento distribuído de dados, no qual foi incluído um sistema de arquivos distribuído (HDFS) confiável e escalável (SHVACHKO et al., 2010). Frameworks como Apache Flink[2] e Apache Spark[3] melhoraram o modelo de programação MapReduce e permitiram workflows arbitrários. Estes frameworks orquestram múltiplos nodos computacionais organizados em um cluster. Os nodos comunicam-se um com os outros para distribuir a carga de trabalho (VERBITSKIY; KAO, 2010).

Atualmente, o custo por espaço de armazenamento tem diminuido, permitindo, por um lado, o crescimento e a riqueza da descrição dos dados, mas, por outro lado, o aumento da quantidade de dados a serem processados. Apesar da queda dos custos

[1] Apache Hadoop, http://hadoop.apache.org/ [Acessado Jun 20, 2017]
[2] Apache Flink, https://flink.apache.org/ [Acessado Jun 20, 2017],
[3] Apache Spark, http://spark.apache.org/ [Acessado Jun 20, 2017],

de armazenamento, o processamento deste material é um processo pesado de computação e alguns algoritmos continuam levando semanas até produzirem resultados. Neste panorama, de crescimento dos dados e falta de poder de processamento, o conceito do MapReduce entra como uma possível solução

Processamento de Linguagens Naturais, sub-campo da inteligência artificial e linguística que estuda os problemas provindos do processamento e manipulação de linguagens naturais, necessita amplamente de poder computacional, tanto em termos de espaço como de tempo de processamento, parar lidar com as grandes quantidades de dados provenientes dos Corpus utilizados como entrada de dados. Implementamos através do Framework Apache Flink um algoritmo existente para geração de Tesauros, assim buscaremos apresentar comparações de performance entre as duas implementações distintas. A primeira implementação foi originalmente criada por Calos Ramisch[4] e chama-se Minimantics[5], apesar de ela utilizar estruturas otimizadas e linguagem baixo nível, sua programação tradicional limita à utilização dos recursos apenas da máquina em que o programa estava rodando, podendo obter ganho de paralelização apenas se a CPU da máquina hospedeira possuisse múltiplos *cores* e habilitassemos o uso de múltiplas Threads na aplicação, enquanto a segunda implementação, que programamos em Flink baseando-nos na implementação original do Minimantics, utiliza um Framework para processamento distribuido dos dados, afim de beneficiar-se dos ganhos provenientes do processamento paralelo e distribuição daa tarefaa entre várias máquinas de um Cluster.

1.1 Motivação

Performance é a razão mais comum para utilizarmos programação paralela. Entretanto, nem todos os programas tornar-se-ão mais eficientes pelo uso de paralelização. Em muitos casos, algoritmos consistem de partes que são paralelizáveis e outras partes que são essencialmente sequenciais. Sempre precisamos questionar o ganho potencial de performance com a utilização de programação paralela.

O ganho obtido com o aumento do poder de processamento dos clusters torna-se vantajoso apenas é alcançado o particionamento equivalente da execução de uma tarefa ao longo dos nodos, evitando a subutilização dos recursos e extraindo seu máximo potêncial. A paralelização automática de programas continua a ser um desafio técnico, porém, mo-

[4]http://pageperso.lif.univ-mrs.fr/~carlos.ramisch/ [Acessado Jul 01, 2017]
[5]https://github.com/ceramisch/minimantics [Acessado Jul 01, 2017]

delos de programação têm contribuído efetivamente para aumentar o grau de paralelismo obtido.

A implementação, mais alto nível, herdada no momento que opta-se pela utilização de um framework, impacta em overhead de informações, necessidade de mais recursos e tratamentos especiais, que, se não forem superados pelo benefício da execução paralela, podem tornar o processamento da aplicação em ambiente distribuído menos eficiente que a implementação sequencial em linguagens baixo nível.

1.2 Objetivo

O principal objetivo desde trabalho é desenvolver uma aplicação, utilizando os conceitos e técnicas de BigData, a partir do modelo de implementação de MapReduce, com a finalidade de reduzir, através de paralelismo, o tempo necessário para a geração de Tesauros.

Defrontando os benefícios obtidos no processamento paralelo das informações, com o ônus pelos custos necessários no tratamento de condições de corrida e mecanismos de comunicação e sincronização, este trabalho apresentará conclusões seguras sobre os benefícios de utilizar o processamento paralelo, auxiliado pelos mais atuais frameworks que implementam técnicas de processamento distribuído e modelo de programação MapReduce, visando o problema elucidado: a geração automática de Tesauros. Dessa forma, ao concluir, será aprovado ou refutado a utilização dessas técnicas no problema.

1.3 Organização do trabalho

Este trabalho está organizado da seguinte forma: No capítulo 2 são abordados os conceitos técnicos que dão base para o trabalho, no capítulo 3 são descritas as características da ferramenta (framework) utilizada para obter as comparações almejadas, já no capítulo 4 introduzimos o algoritmo original proposto para obter automaticamente a relação semântica das palavras e como ele foi implementado no contexto de processamento distribuído. Por final, no capítulo 5 evidenciaremos os resultados obtidos, sua escalabilidade e eficiência.

2 CONCEITOS

Este capítulo descreve os conceitos técnicos que fundamentam a execução deste trabalho. Eles dão base para nosso objetivo, a performance da geração automática de Tesauros distribucionais, através de técnicas de programação distribuída. Com o interesse principal orbitando Big Data e PLN, abordamos a ligação entre as áreas contempladas, concepções e desafios individuais.

2.1 Inter-relação Big Data e PLN

A capacidade de usar grandes quantidades de dados para aprender insights importantes e construir modelos preditivos será um gande ganho para as todas as áreas. Softwares tradicionais, planilhas e software analíticos não conseguem nem de perto gerenciar e processar grandes dados. Há muita informação. Uma gama de aplicações voltadas a Big Data, como o Apache Flink, foram criadas para auxiliar no gerenciamento dessas quantidades enormes de dados e obtenção de resultados, podendo entregar respostas até em tempo-real.

Computadores conseguem fazer muitas coisas mais rápido e eficiente que humanos conseguem. Todavia, há uma área em particular que humanos tem vantagem, e esta área é a linguagem. Nossos cérebros são treinados para reconhecer padrões de fala e entender significados em velocidades que embaraçariam o mais rápidos computadores existentes atualmente. De fato, computadores entendem e processam informações estruturadas e desambiguas, sendo linguagem a direção oposta disto. Palavras podem se basear em referências prévias e terem significados exclusivos em cada cultura, fazendo com que palavras da mesma língua tenham significados diferentes.

Há um benefício significativo em juntar PLN com análise de dados. Podendo analisar com precisão grandes quantidades de dados não estruturados, como e-mails, mensagens de texto e ligações por vez, pode-nos levar a insights mais precisão do comportamento humano, especialmente quando combinado com outros dados estruturados. NLP pode ajudar a encontrar um meio de categorizar esta informação em databases estruturados, que podem ser analisados rapidamente, como os elementos numéricos de big data. Se encontrarmos um meio de rastrear e analisar linguagem de um modo similar a como nós rastreamos outras formas de dados (estruturados), estaremos abrindo portas para insights mais detalhados e melhores modelos preditivos (ALLOUCHE, 2014).

2.2 PLN

Processamento de Linguagens Naturais é uma área que estuda os problemas da geração e compreensão automática de línguas humanas naturais. Sistemas de geração de linguagem natural convertem dados em linguagem compreensível ao ser humano e sistemas de compreensão de linguagem natural convertem ocorrências de linguagem humana em representações mais formais, mais facilmente manipuláveis por programas de computador. Alguns desafios do PLN são compreensão de linguagem natural, fazer com que computadores extraiam sentido de linguagem humana ou natural e geração de linguagem natural.

2.2.1 Similaridade Semântica

De acordo com Sébastien Harispe, em Semantic Similarity from Natural Language and Ontology Analysis (HARISPE et al., 2015), o presente estado da arte, para estimar e quantificar a similaridade semântica/relacionalidade de entidades semânticas, contempla duas abordagens distintas: O primeiro fia-se em técnicas de PLN e modelos semânticos pré-definidos, enquanto o segundo é baseado, de maneira mais ou menos formal, em conhecimento aplicável e legível por computadores, como similaridade estatística e similaridade topológica, que englobam, ‘semantic networks’, tesauros e ontologias.

Utilizaremos, neste trabalho, medidas de similaridade estatísticas utilizando abordagem baseada no índice de informações compartilhadas. Essa abordagem compreende todas as técnicas que utilizam o cálculo de similaridade semântica entre dois termos através do grau de informações que eles têm em comum, ou seja, o grau de informações que elas compartilham (RESNIK, 1995). Palavras que co-ocorrem bastante próximas de um outra palavra específica são consideradas como sendo "característica"ou "propriedades"desta palavra. Portanto, um conjunto de classes de palavras pode ser extraído através do mapeamento das classes da taxonomia para descobrir os níveis hierárquicos, classes e subclasses das quais o termo pertence.(VENCESLAU; BEZERRA; LINO, 2013)

O Aprendizado por similaridade pode muitas vezes superar as medidas de similaridade predefinidas. Em termos gerais, essa abordagens constrói um modelo estatístico de documentos e utiliza-a para estimar a similaridade; onde podemos nos beneficiar diretamente com a utilização de Big Data.

2.2.2 Tesauros

"Tesauro é uma linguagem especializada, normalizada, pós-coordenada, usada com fins documentários, onde os elementos linguísticos que a compõem - termos, simples ou compostos - encontram-se relacionados entre si sintática e semanticamente."(CURRáS, 1995)

Tesauros são recursos léxicos que contém informações sobre palavras semanticamente relacionadas. Esses recursos são importantes para diversas aplicações, como a tradução automática de textos (CHO et al., 2014). A construção de um Tesauro pode dar-se ou manualmente, onde o resultado gerado é de boa qualidade, porém com alto custo e baixa cobertura, ou automaticamente, baseando-se em hipóteses distribucionais e cálculos de similaridade semântica. A geração automática de Tesauros foi o objetivo utilizado na aferição de performance no contexto distribuído.

2.3 Big Data

Os dados produzidos e armazenados por computação crescem enormemente a cada ano. De acordo com a IDC's[1], estima-se que no ano de 2020 a humanidade terá 40 Zettabytes (40,000,000 Petabytes) de dados que demandarão processamento de algum tipo. Este volume de dados necessitará que tenhamos uma capacidade de processamento muito além do que a infraestrutura de TI atual consegue fornecer (ANJOS; FEDAK; GEYER, 2016)

Para explicar o fenômeno de Big Data, ele é frequentemente descrito usando **cinco 'V's** (ASSUNçãO et al., 2015): **(i) Volume**: refere-se a vasta quantidade de dados gerada a cada segundo, este aumento faz com que os 'data sets' fiquem muito largos para armazenar e analisar utilizando as técnicas tradicionais de bancos de dados. Como a tecnologia Big Data nós podemos agora armazenar e usar esses datasets com a ajuda de sistemas distribuídos, onde os dados são particionados para poderem ser armazenados em diferentes locais e depois re-ajuntados por software quando necessário. **(ii) Velocidade**: refere-se a velocidade com a qual novos dados são gerados e a velocidade na qual os dados movem-se ao redor da rede, a tecnologia Big Data permite-nos analisar os dados enquanto eles ainda estão sendo gerados, sem sequer precisar armazená-los em bancos de dados. **(iii) Variedade**: refere-se aos diferentes tipos de data que podemos lidar. No passado, focáva-

[1] IDC's Digital Universe Study, sponsored by EMC, December 2012.

mos em estruturas de dados que obrigatoriamente precisam se encaixar perfeitamente em tabelas ou bancos de dados relacionais. Com Big Data podemos utilizar os mais variados tipos de dados (estruturados e não estruturados), incluindo mensagens, conversações em mídias sociais, fotos, dados de sensores, vídeos e voz, trazendo junto, claro, com os tradicionais dados estruturados. **(iv) Veracidade**: refere-se a desordem ou confiabilidade dos dados. Por ter muitas formas e muita quantidade, a qualidade e precisão dos dados são menos controláveis, mas big data e tecnologias analíticas permitem-nos trabalhar com estes tipos de dados. O volume muitas vezes compensa a falta de qualidade ou precisão. **(v) Valor**: O mais importante dos cincos 'v's. É extremamente desejável termos acesso a big data, mas a menos que possamos transformá-la em valor é inútil. É importante que façamos um estudo de casos para qualquer tentativa de alavancar e coletar grandes quantidades de dados, pois, embarcar em iniciativas de big data sem uma clara compreensão dos custos e benefícios é uma armadilha fácil a ser caída.

A computação paralela é a resposta para conseguirmos lidar com essa enorme quantidade de dados e informação, e processá-las em tempo hábil, beneficiando-se do princípio de que grandes problemas podem ser divididos em problemas menores e então resolvidos concorrentemente (em paralelo) (ADVE et al., 2008). Entretanto, a tarefa de programar para um ambiente distribuído e paralelo é mais difícil do que para a tradicional computação sequencial, diversos novos problemas - como condições de corrida, troca de mensagens, sincronizações - surgem e precisam ser tratados pelo programador, tornando muitas vezes inviável a escrita de programas eficientes sem o auxílio de modelos de programação e frameworks. Assim, modelos de programação foram propostos para abstrair do programador final a tarefa de solucionar os problemas provindos da distribuição.

2.3.1 MapReduce

MapReduce é um modelo de programação proposto pelo Google e atualmente adotado por muitas companhias, no qual tem sido empregada como uma boa solução de processamento e análise de dados (DEAN; GHEMAWAT, 2010), inspirado nas primitivas Map e Reduce presentes em linguagens funcionais, ele abstrai a complexidade de aplicações paralelas, dividindo e espalhando os conjuntos de dados em centenas de milhares de máquinas e por reunir juntamente computação e dados (WHITE, 2015).

As fases de Map e Reduce são tratadas pelo programador, enquanto a fase de Shuffle é criada enquanto a tarefa está sendo executada. Os dados de entrada são divididos em

pequenos pedaços chamados chunks, que normalmente tem um tamanho de 64MB. Os dados são serializados e distribuídos para as máquinas que compõem o sistema de arquivos distribuído (Distributed File System - DFS). Quando uma aplicação está rodando, a máquina Master atribui tarefas para os Workers e monitora o progresso da tarefa. A máquina Worker incubida de uma tarefa de Map executa sua função e emite pares key/value como resultado intermediário, no qual são temporariamente armazenados no disco da máquina. O modelo de execução cria uma barreira computacional, na qual permite que tarefas sejam sincronizadas entre os produtores e consumidores. Uma tarefa de Reduce não começa a ser processada até que todas as tarefas de Map tenham sido completadas. Uma função hash é aplicada nos dados dos resultados intermediários para determinar quais keys irão compor a tarefa de Reduce. O grupo de keys selecionadas forma uma partição. Cada partição é transferida para uma máquina única durante a fase de Shuffle, para executar a próxima fase. Depois de uma função Reduce ter sido aplicada aos dados, um novo par key/value resultante é gerado. Seguindo isso, os resultados são armazenados no DFS e são disponibilizados para os usuários.

MapReduce utiliza um sistema de gerenciamento para replicação de dados e controle de execução. Adicionalmente, ele tem uma arquitetura de gerenciamento baseada no modelo Master/Worker, enquanto uma troca de dados slave-to-slave requer um modelo peer-to-peer (P2P) (WHITE, 2015). O worker é um nodo que pode rodar função Map ou Reduce no ambiente MR. Uma máquina é caracterizada como retardatária quando a execução das suas tarefas está abaixo da média de execução para o cluster. Se ela sofrer a classificação de retardatária após a primeira distribuição de tarefas, ela não será receberá novas tarefas nos seus slots livres.

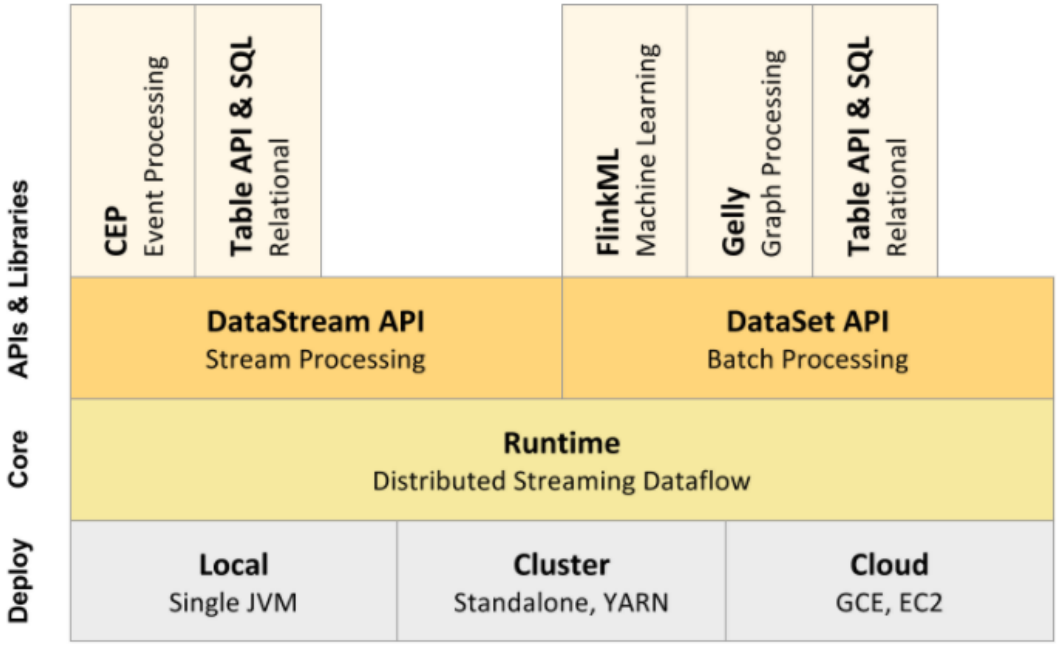
Apesar de trazer todo suporte necessário para escrita de programas paralelos, o MapReduce na sua forma mais crua é simples e carece de algumas funcionalidades, assim, diversos frameworks, baseados no modelo de programação MapReduce, foram desenvolvidos por empresas e comunidades.

2.3.2 Apache Flink

Apache Flink é uma plataforma de processamento de dados distribuídos para uso em Big Data. Suporta combinação de processamentos de dados em memória e armazenados em disco. Flink processa os dados provindos de duas maneiras: batch e stream, com stream sendo a implementação padrão e batch rodando como um caso especial de

stream limitado. Flink foi designado como uma alternativa para MapReduce, o antigo motor de processamento batch-only que emparelhava com HDFS na "encarnação"inicial do Hadoop. O Projeto Apache Flink é open source e adere aos termos de licenciamento do Apache Software Foundation. Seu núcleo provê uma engine para processamento massivo de dados, utilizando fluxos de dados distribuídos e escaláveis via stream ou batch. Sua stack, como visto na figura 2.1, conta com: APIs em Scala, Java e Python, como por exemplo, DataSet API (processamento de batchs), Table API (Queries relacionais), DataStream API (análise em tempo real de streams), entre outros; Bibliotecas de domínio específico, como por exemplo, FlinkML (Biblioteca Machine Learning para Flink), Gelly (Biblioteca de Grafos para Flink), entre outros; Shell para análise interativa. É considerado um Framework da 4a geração para Big Data Analytics.

Figura 2.1: Representação bottom-up do framework Apache Flink



Fonte: https://flink.apache.org/

2.3.3 Flink Conceitos

Os programas Flink são programas normais que implementam transformações (ex: filtering, mapping, joining, grouping, aggregating) em Collections distribuídas. Collections são criadas inicialmente a partir de fontes (ex: pela leitura de arquivos, kafka topics, ou por outras collections locais ou em memória). Os resultados gerados são retornados

via sinks, no qual podemos, por exemplo, escrever os dados em arquivos distribuídos, ou retornar pela saída padrão (ex: terminal de linha de comando). Os programas Flink rodam em uma variedade de contextos, standalone, ou incorporados em outros programas. A execução pode acontecer em uma máquina JVM local, ou em cluster de máquinas.

Nesta sub-seção é apresentado, com breve detalhamento, os conceitos que embasam o núcleo do Framework Flink, o fluxo dos dados durante a execução, como o paralelismo se dá internamente e detalhes adicionais sobre a API Dataset utilizada.

2.3.3.1 Programas e Fluxos de Dados

Os blocos básicos que constituem um programa em Flink são *streams* e *transformations* (Note que a estrutura DataSet, utilizada na API de batch do Flink, é internamente um *stream* limitado). Conceitualmente, um *stream* é um fluxo de dados (potencialmente) infinito, e uma *transformation* é uma operação que recebe um ou mais *streams* como entrada (*source*) e produz um ou mais *streams* como saída (*sink*).

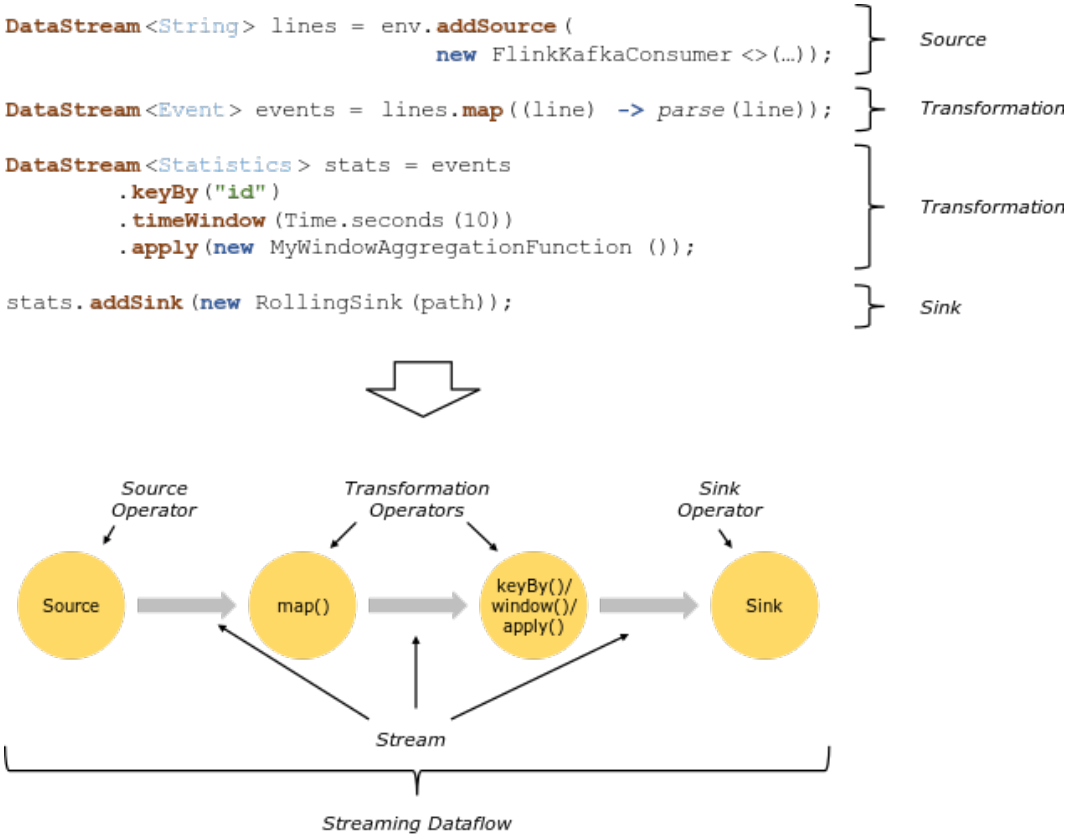
Quando executados, programas Flink são mapeados para streaming de fluxos de dados, consistindo de streams e operadores de transformations. Cada fluxo de dados começa em uma ou mais fontes e termina em uma ou mais saídas. Os fluxos de dados assemelham-se a Grafos Acíclicos Direcionados (directed acyclic graphs - DAGs) - Embora formas especiais de cíclos sejam permitidas via construções de iterações (The Apache Software Foundation Official Website).

2.3.3.2 Flink Paralelismo

Programas em Flink são inerentemente paralelos e distribuídos. Um programa em Flink consiste de múltiplas tarefas, uma tarefa é dividida em várias instâncias paralelas para execução, chamada de sub-tarefa, e cada sub-tarefa processa um subset dos dados de entrada da tarefa. As sub-tarefas são independentes uma das outras, e executam em diferentes threads e possivelmente em diferentes máquinas.

O número de instâncias paralelas (sub-tarefas) de uma tarefa é chamado de paralelismo, ele que definirá o paralelismo de um operador em particular. O paralelismo de um stream é sempre o do seu operador produtor. Diferentes operadores no mesmo programa podem ter diferentes níveis de paralelismo.

Figura 2.2: Fluxo de dados de um programa em Flink

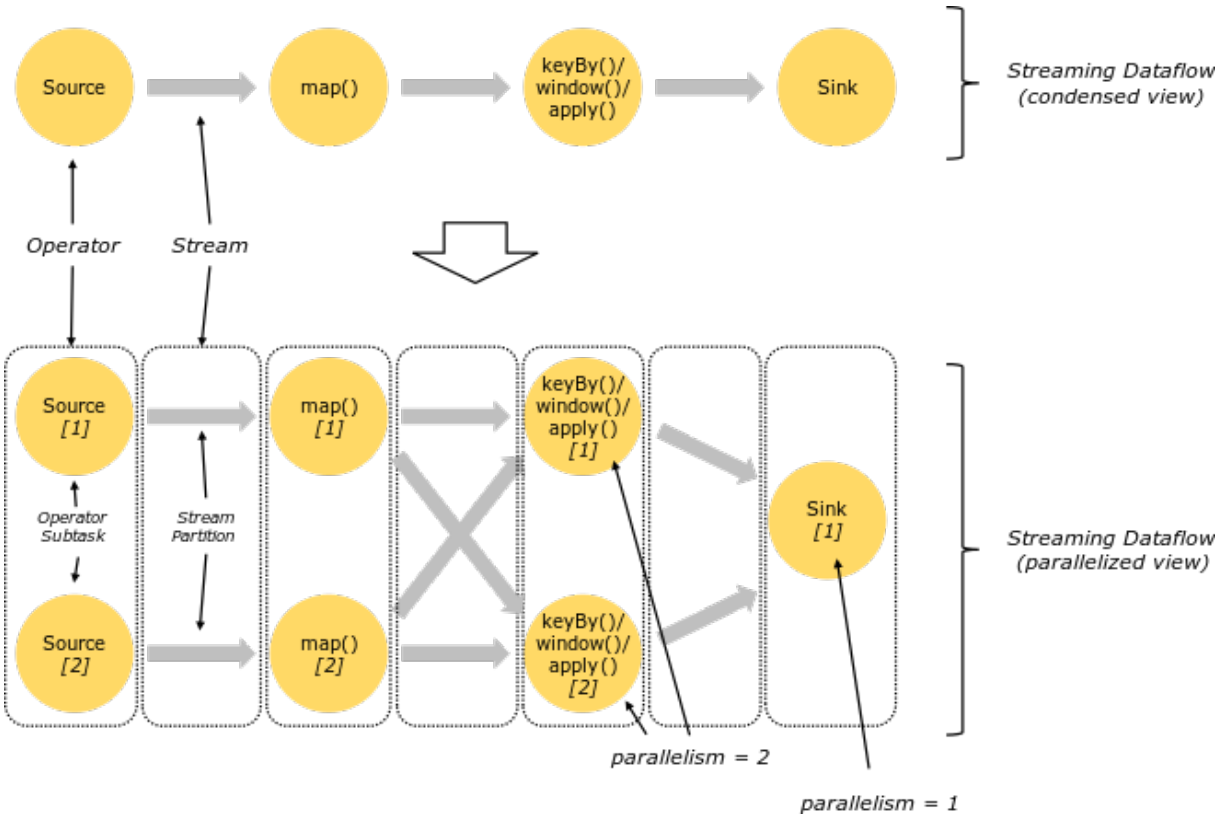


Fonte: https://flink.apache.org/

Como visto na figura 2.3, *Streams* podem transportar dados entre operadores através de dois padrões:

- Um-pra-um: Preserva o particionamento e ordenameno dos elementos. Isto significa que a subtarefa irá ver os mesmos elementos na mesma ordem que eles foram produzidos pela subtarefa anterior. Este transporte pode ser visto, por exemplo, na figura 2.3, a *stream* entre o Source[1] e o operador map()[1].
- Redistribuínte: Cada subtarefa do operador envia dados para diferentes subtarefas alvo, dependendo da *transformation* selecionada. Exemplos de *transformations* em que ocorre isso são: **i) keyBy()** (que reparticiona pela key), **ii) broadcast()** ou **iii) rebalance**() (que reparticiona randomicamente). Em uma permutação redistribuinte o odenamento dos elementos é somente preservado dentro de cada subtarefa, recebida e enviada. A ordenação dentro de cada chave é preservada, mas o paralelismo introduz não-determinismo dependendo da ordem no qual o resultado agregado de cada keys diferente chega no Sink. Este transporte pode ser visto, por exemplo, na figura 2.3, com a relação entre o map() e keyBy(), assim também como

Figura 2.3: Paralelismo em um programa em Flink



Fonte: https://flink.apache.org/

entre keyBy() e *Sink*.

2.3.3.3 Dataset API

Dependendo do tipo da fonte de dados, isto é, fontes limitadas ou ilimitadas, deve ser escrito um programa para batch ou um programa para streaming, onde a API Dataset é utilizada para manipular para arquivos batch.

Listaremos as funções da API mais utilizadas na nossa implementação:

- *readTextFile*: Única função para entrada de dados utilizada pela nossa implementação, isso significa que todos os dados manipulados durante a execução foram lidos a partir de arquivos de textos, armazenados no HDFS, e foram manuseados como Strings dentro do programa.
- *writeAsText*: Única função para saída de dados utilizada pela nossa implementação, isso significa que todos os resultados gerados tiveram como saída a escrita em arquivo de texto, sendo armazenados no HDFS.
- *flatMap*: A transformação mais utilizada pela nossa aplicação, ela permite aplicar

uma função especificada pelo usuário para cada elemento presenet no Dataset de entrada. É uma variante da transformação *map*, que pode retornar uma número arbitrário (nenhum, um ou vários) de resultados para cada elemento do Dataset percorrido.

- *map*: Muito similar a transformação *flatMap* listada acima, porém o mapeamento é feito um-para-um, isto é, a função definida pelo usuário deve retornar um elemento pra cada elemento percorrido no Dataset.
- *reduceGroup*: Uma transformação de reduce que é aplicada em um dataset agrupado, reduz cada grupo para um único elemento usando uma função de redução definida pelo usuário. Para cada grupo, a funçõ de redução sucessivamente combina pares de elementos em um elemento até que reste apenas um único elemento para cada grupo. Note que os grupos são definidos através de uma key, especificada através da função groupBy listada a seguir.
- *groupBy*: Agrupa os dados de um dataset através da key especificada, é utilizada em conjunto com transformações que operam sobre datasets agrupados.
- *filter*: Aplica uma função de filtro, definida pelo usuário, para cada elemento do Dataset, preservando apenas os elementos que retornaram *true* na saída da função.
- *join*: Esta função é grande responsável por aumentar a complexidade de execução do programa. Ela junta os elementos de dois Datasets em um. Os elementos de ambos Datasets são agrupados em keys especificadas e então são juntados os que possuírem igualdade das keys. Existem algumas maneiras diferentes de realizar uma transformação de junção, como: Join normal, Join com função especificada por usuário que é aplicada ao final de cada junção, Join com indicação de tamanho dos Datasets envolvidos na junção, OuterJoin, LeftJoin, RightJoins, entre outros.
- *cross*: Função mais cara existente, desafia qualquer cluster de dados existente! Combina dois Datasets inteiros em um. Ela constrói todas as combinações possíveis de pares de elementos de ambos Datasets de entrada, isto é, constrói um produto cartesiano. Pode aplicar ao final de cada combinação uma função definida pelo usuário.

3 PROBLEMA

4 PROTÓTIPO

Este capítulo descreve as principais características da implementação utilizada, como os programas que auxiliaram e o código do algoritmo Minimantics. Afim de abstrair a necessidade de resolver os problemas provindos da paralelização, o trabalho foi desenvolvido utilizando os mais modernos recursos baseados no paradigma MapReduce. Os resultados finais foram alcançados através do Framework Apache Flink e seu suporte nativo à linguagem de programação Scala, entretanto, até chegar na implementação final e em resultados satisfatórios, outros Frameworks e linguagens foram testados, porém, algumas dificuldades, que serão descritas mais adiante, foram encontradas.

4.1 Corpus

Optamos por um Corpus acessível e grande, UKWaC: corpus of British English, a fim de validar os dados gerados e poder levar o framework Apache Flink à níveis compatíveis com sua proposta de lidar com grandes quantidades de dados.

O ukWaC é um corpus Inglês muito grande (mais de 2 bilhões de palavras), ele foi construído a partir de “web crawling” e sua proposta é servir como um corpus de propósito geral, seu conteúdo foi arrecadado em domínios “.uk” , por tanto, é plausível dizer que este corpus é principalmente britânico, embora não é descartado que outros idiomas não possam aparecer, em menor número, ao longo que páginas não inglesas também podem existir em domínio “.uk”. Ele foi criado em 2007 como parte do WaCky project, um consórcio informal de pesquisadores interessados em explorar a web como fonte de dados linguísticos.

Trabalhamos com a versão "Non-POS-tagged"do Corpus, que contém os textos puros, sem pré-rotulação dos seus elementos. Obtivemos acesso ao Corpus no seu tamanho pleno e em mais três subsets de tamanhos variados..

Na tabela **??** é possível ver os diferentes tamanhos de corpus utilizados. Ao referenciarmos a utilização de um dos corpus neste trabalho, empregaremos a descrição contida no campo “Nome” para nos referenciarmos ao corpus .

Tabela 4.1: Diferentes Corpus utilizados

Nome	Nº Palavras	Tamanho	Idioma
ukWaC Subset 1M	1 Milhão	2.3MB	Inglês Britânico
ukWaC Subset 10M	10 Milhões	24MB	Inglês Britânico
ukWaC Subset 100M	100 Milhões	230MB	Inglês Britânico
ukWaC	2 Bilhões	2.8GB	Inglês Britânico

5 RESULTADOS

Este capítulo apresenta os resultados do trabalho, com base nas metodologias de avaliação descritas na próxima seção 5.1, os testes de performance foram executados em máquinas individuais e cluster com a configuração descrita na seção 4.2.

5.1 Metodologia de Avaliação

Para as etapas do Minimantics, utilizamos como medida de avaliação o tempo necessário para execução das etapas do algoritmo, comparando speedups e ganhos de eficiência ao paralelizar. Já na etapa de avaliação dos Tesauros, criamos um método de avaliação que é baseado no experimento com humanos, onde apresentamos as frases geradas ao usuário e perguntamos uma nota de avaliação. Os resultados gerados foram baseados em uma média aritmética de 10 rodadas de execuções, a fim de barrar resultados excepcionais que poderiam mascarar resultados dentro da normalidade.

5.2 Especificação do Ambiente

5.2.1 Hardware

O ambiente utilizado para execução do algoritmo e obtenção dos resultados é um cluster formado por máquinas virtuais instanciadas na plataforma **Microsoft Azure**. As **VMs** (Virtual Machines) possuem a configuração denominada "Standard A3" na plataforma. Uma VM possuia a seguinte especificação de hardware:

- Processamento: vCPU 4 Core (2.10Ghz)
- Memória: 7GB RAM
- Disco Rígido: 256GB (com velocidade máxima de 500 IOPS[1])

Afim de manter a homogeneidade do ambiente, todas as VMs foram instanciadas a partir de uma mesma imagem de máquina virtual e possuem exatamente a mesma configuração uma das outras. O **Cluster**, formado por 4 instâncias VMs, tinha poder equivalente a:

[1] IOPS = Input/Output per second

- Processamento: vCPU 16 Core (2.10GHz)
- Memória: 28GB RAM
- Disco Rígido: 1TB (com velocidade máxima de 500 IOPS)

5.2.2 Software

O sistema operacional instalado nas máquinas era o **Ubuntu 16.04.1 LTS** (GNU/Linux 4.4.0-79-generic x86-64). O Framework utilizado para dar suporte a computação paralela foi o **Apache Flink versão 1.2**, com todo o código do algoritmo escrito na linguagem **Scala versão 2.12**, nativamente suportada pelo Flink e suas APIs. Em conjunto com o Flink, foi utilizado **Hadoop Distributed File System (HDFS) versão 2.7.3** para dar suporte ao sistema de arquivos distribuído.

O Framework Flink conta com duas estruturas principais: JobManager e TaskManager. É necessário que se defina manualmente a alocação de recursos para os JobManagers e TaskManagers. O **JobManager**, que coordena todo deploy que é feito, é responsável por agendar execuções e gerir os recursos do ambiente. Por padrão há apenas uma instância de JobManager por Cluster, desconsiderando casos de redundância, a instância do JobManager, configurada no nosso cluster, possuía 2048MB de memória alocado para si. Já os **TaskManangers**, também chamados de “Workers”, são os responsáveis diretos pela execução das tarefas. Foi alocado um TaskManager por máquina, e cada TaskManager instanciava 4 "slots"(que somavam 16 em todo o Cluster).

5.3 Tempos obtidos

O algoritmo original foi executado com uma paralelização máxima de 4, dado que ele não tem suporte a programação distribuída em multimáquinas e o máximo de cores disponíveis nas vCPUs utilizadas era 4; Na plataforma Flink, o paralelismo máximo executado foi 16. Os tempos aqui expostos foram organizados em tabelas distintas que referem-se ao tamanho do Corpus de entrada.

Tabela 5.1: Tempos* obtidos a partir do Corpus com 1 milhão de palavras

		uKWaC Subset 1M		
		Original	Flink Scala	Flink Python
1 Core	FilterRaw	13.319	12.900	24.499
	BuildProfiles	108	4.300	102.748
	CalculateSimilarity	35.122	705.500	10.235.420
4 Cores	FilterRaw	9.189	7.800	104.996
	BuildProfiles	-	4.125	42.645
	CalculateSimilarity	14.477	42.645	2.625.549
16 Cores	FilterRaw	-	5.750	45.586
	BuildProfiles	-	5.143	26.760
	CalculateSimilarity	-	62.200	557.755

*em milisegundos

Tabela 5.2: Tempos* obtidos a partir do Corpus com 10 milhãos de palavras

		uKWaC Subset 10M		
		Original	Flink Scala	Flink Python
1 Core	FilterRaw	170.368	181.111	-
	BuildProfiles	544	26.571	-
	CalculateSimilarity	809.598	19.620.000	-
4 Cores	FilterRaw	106.570	58.600	1.205.897
	BuildProfiles	-	13.143	274.471
	CalculateSimilarity	300.882	5.490.032	-
16 Cores	FilterRaw	-	35.250	346.337
	BuildProfiles	-	10.500	114.310
	CalculateSimilarity	-	1.589.750	14.561.226

*em milisegundos

Tabela 5.3: Tempos* obtidos a partir do Corpus com 100 milhãos de palavras

		uKWaC Subset 100M	
		Original	Flink Scala
1 Core	FilterRaw	1.870.259	2.061.750
	BuildProfiles	2.662	187.889
	CalculateSimilarity	12.528.762	424.800.000
4 Cores	FilterRaw	1.248.592	619.500
	BuildProfiles	-	73.286
	CalculateSimilarity	3.946.393	-
16 Cores	FilterRaw	-	35.250
	BuildProfiles	-	10.500
	CalculateSimilarity	-	23.766.393

*em milisegundos

Tabela 5.4: Tempos* obtidos a partir do Corpus com 2 bilhãos de palavras

		uKWaC Full	
		Original	Flink Scala
1 Core	FilterRaw	59.819.400	64.565.773
	BuildProfiles	25.769	3.533.333
	CalculateSimilarity	524.281.052	-
4 Cores	FilterRaw	36.661.766	14.413.333
	BuildProfiles	-	1.491.000
	CalculateSimilarity	158.645.297	-
16 Cores	FilterRaw	-	8.751.429
	BuildProfiles	-	616.000
	CalculateSimilarity	-	-

*em milisegundos

5.4 Speedups

A seguir, os gráficos de speedup para os algoritmos executados

REFERÊNCIAS

ADAMS, B.; RAUBAL, M. Bridging data in the clouds: An environment-aware system for geographically distributed data transfers, cluster, cloud and grid computing (ccgrid). In: IEEE/ACM INTERNATIONAL SYMPOSIUM, 2014. [S.l.]: IEEE, 2014. p. 92–101.

ADVE, S. V. et al. **Parallel Computing Research at Illinois: The UPCRC Agenda**. [S.l.], 2008. Available from Internet: <https://web.archive.org/web/20081209154000/ http://www.upcrc.illinois.edu/documents/UPCRC_Whitepaper.pdf>.

ALLOUCHE, G. Natural language processing and big data: A powerful combination. 2014. Available from Internet: <"http://www.dataversity.net/ natural-language-processing-big-data-powerful-combination/">.

ANJOS, J. C. S.; FEDAK, G.; GEYER, C. F. R. Bighybrid: a simulator for mapreduce applications in hybrid distributed infrastructures validated with the grid5000 experimental platforms. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, p. 2317–2563, jun 2016.

ASSUNçãO, M. D. et al. Big data computing and clouds: Trends and future directions. **Journal of Parallel and Distributed Computing**, Elsevier, v. 79-80, p. 3–15, may 2015.

CHO, K. et al. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In: CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING (EMNLP). Doha, Qatar: Association for Computational Linguistics, 2014. p. 1724–1734.

COX, M.; ELLSWORTH, D. Application-controlled demand paging for out-of-core visualization. In: IEEE/ACM INTERNATIONAL SYMPOSIUM, 1997. [S.l.]: IEEE, 1997. p. 235–ff. 8th Conference on Visualization.

CURRáS, E. **Tesauros, linguagens terminológicas**. Instituto Brasileiro de Informação em Ciência e Tecnologia (IBICT), 1995. Available from Internet: <http:/livroaberto.ibict.br/handle/1/454>.

DEAN, J.; GHEMAWAT, S. Mapreduce: A flexible data processing tool. **Commun. ACM**, ACM, New York, NY, USA, v. 53, n. 1, p. 72–77, jan. 2010. ISSN 0001-0782. Available from Internet: <http://doi.acm.org/10.1145/1629175.1629198>.

HARISPE, S. et al. **Semantic Similarity from Natural Language and Ontology Analysis**. [S.l.]: Morgan & Claypool, 2015.

MAYER-SCHöNBERGER, V.; CUKIER, K. **A Revolution that Will Transform how We Live, Work, and Think**. New York: Houghton Mifflin Harcourt, 2013.

RESNIK, P. Using information content to evaluate semantic similarity in a taxonomy. In: **Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995. (IJCAI'95), p. 448–453. ISBN 1-55860-363-8, 978-1-558-60363-9. Available from Internet: <http://dl.acm.org/citation.cfm?id=1625855.1625914>.

SHVACHKO, K. et al. The hadoop distributed file system. In: IEEE/ACM INTERNATIONAL SYMPOSIUM, 2010. [S.l.]: IEEE, 2010. 26th Symposium on Mass Storage Systems and Technologies (MSST).

The Apache Software Foundation. **Apache Flink: Scalable Stream and Batch Data Processing**. <https://flink.apache.org/>. Accessedo em Jul 02 2017.

VENCESLAU, A. D. P.; BEZERRA, E. P.; LINO, N. C. Q. **Uma abordagem para identificação de domínios de aplicação em ambiente de convergência digital**. 2013.

VERBITSKIY, L. T. I.; KAO, O. When to use a distributed dataflow engine: Evaluating the performance of apache flink. In: SYMPOSIUM ON MASS STORAGE SYSTEMS AND TECHNOLOGIES (MSST), 2016. [S.l.]: IEEE, 2010. "Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)".

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