Minicurso 1

Análise em Big Data e um Estudo de Caso utilizando Ambientes de Computação em Nuvem

Ticiana L. C. da Silva, Antonio Cavalcante Araújo Neto, Flávio R. C. Sousa, José Antônio F. de Macêdo e Javam C. Machado

Resumo

As novas aplicações de mineração de dados no contexto de Big Data ou análise em Big Data necessitam gerenciar grandes volumes de dados de diferentes tipos e estruturas. As técnicas de mineração aplicadas sob tais dados possibilitam a extração de novos conhecimentos ou a realização de predições, por exemplo. Entretanto, o processamento intensivo de dados está além da capacidade de qualquer máquina individual e requer que a computação seja realizada em clusters. Dessa forma, a utilização de técnicas de mineração para grandes volumes de dados exige adaptações em seus algoritmos de forma a paralelizar a execução e utilizar ambientes de larga escala para o armazenamento e processamento, atualmente disponíveis por meio de infraestruturas de computação em nuvem. Este minicurso tem como objetivo apresentar os principais algoritmos de mineração para Big Data considerando a utilização de ambientes de computação em nuvem, além de apresentar boas práticas para o desenvolvimento de soluções neste contexto. Também é apresentado um estudo de caso considerando o domínio de dados de monitoramento de tráfego, enfatizando o uso das técnicas de mineração e da infraestrutura de nuvem discutidas. Por fim, são apresentadas as considerações finais sobre o tema, destacando oportunidades e desafios encontrados.

1.1. Introdução

Nas duas últimas décadas, o aumento contínuo do poder computacional tem produzido um fluxo enorme de dados [QU et al. 2012]. A cada dia, 2.5 quintilhões de bytes de dados são criados e 90% dos dados no mundo hoje foram produzidos nos últimos dois anos [Wu et al. 2013]. Como exemplos deste cenário, pode-se citar modernos centros de pesquisa em física, tais como DZero, que normalmente geram mais de um terabyte de dados por dia. A rede social Facebook possui mais de um bilhão de usuário e cem horas de novos vídeos são armazenados a cada minuto no Youtube. Estes exemplos exigem arma-

zenamento eficiente, análise de grande volume de dados e tomada de decisão instantânea. A este contexto de gestão e análise de dados tem-se denominado "Big Data".

De acordo com [Begoli and Horey 2012], Big Data é a prática de coleta e processamento de grandes conjuntos de dados e sistemas e algoritmos utilizados para analisar estes conjuntos de dados massivos. Já [Wu et al. 2013] argumenta que Big Data refere-se a grandes volumes heterogêneos, fontes autônomas com controle distribuído e descentralizado, procurando explorar as relações complexas e em evolução entre os dados.

Apesar das enormes possibilidades do uso da informação aferidas ao fenômeno Big Data, não é trivial entender quais técnicas e métodos de gerenciamento de dados serão adequados para lidar com os desafios trazidos por este novo cenário. Apesar do termo Big Data ser explicado através dos famosos 3 V's: Volume, Velocidade e Variabilidade, não são claros os desafios trazidos por tais caracteristicas às tecnologias de banco de dados desenvolvidas ao longo dos últimos 40 anos. De fato, banco de dados tem como premissa lidar com grande volumes de dados por meio de inúmeras técnicas de processamento distribuído. A velocidade é tratada pelos bancos de dados que manipulam *stream* de dados, e a variabilidade foi amplamente pesquisada em métodos para integração de dados estruturados e semiestruturados, os quais visam lidar com a heterogeneidade de modelos e dados. Consequentemente, segundo Dan Suciu [Suciu 2013], o que Big Data traz de novo ao contexto de gerenciamento de dados são três aspectos: dados são armazenados em memória principal, necessidade de inúmeras iterações sobre os dados, a ocorrência de falhas é uma regra.

Com o atual uso de infraestruturas massivas de servidores virtualizados para processar um grande volume de dados distribuídos, podemos supor que a maior parte dos dados já processados estarão armazenados em memória, em suas respectivas máquinas virtuais. Desta forma, a nova métrica de complexidade para processamento de consultas deixa de ser a número de acessos a disco para se tornar quantidade de comunicação entre os servidores que processam as informações. Claramente, os métodos e técnicas de banco de dados deverão levar em consideração esta mudança de complexidade.

Diversos tipos de análise de dados no contexto Big Data exigem computação iterativa, incluindo clusterização usando K-Means, PageRank, consultas recursivas relacionais, análise de redes sociais, entre outros. Essas técnicas visam processar os dados iterativamente até que a computação satisfaça uma determinada condição de parada ou convergência. Este tipo de computação iterativa não é provido pelas tecnologias de banco de dados atuais. De fato, serão necessárias técnicas mais sofisticadas para lidar com computação iterativa, levando em conta os custos de comunicação.

Outro ponto importante a destacar é que banco de dados paralelos lidam com a falha dos nós de maneira excepcional. Em um cenário Big Data, onde uma grande quantidade de máquinas estão envolvidas, claramente as falhas deixam de ser uma exceção para se tornarem uma regra. Desta maneira, é necessário que o gerenciamento de dados em Big Data lide com a falha dos nós como um evento frequente, e não como uma exceção ao processamento.

Um outro ponto importante a destacar no cenário Big Data é a infraestrutura computacional necessária para lidar com grandes volumes de dados heterogêneos e distribuí-

dos. Considerando o grande volume a ser analisado e que o processamento intensivo dos dados está além da capacidade de qualquer máquina individual, o cenário Big Data demanda a disponibilização de novos modelos computacionalmente eficientes
[Lin and Dyer 2010]. Neste contexto, a computação baseada em *clusters* permite aproveitar grande número de processadores em paralelo para resolver um problema de computação [Pavlo et al. 2009]. Claramente, a disponibilização de uma infraestrutura de processamento em larga escala exige uma infraestrutura de software compatível, a qual possa tirar vantagem da grande quantidade de máquinas e mitigar o problema de comunicação entre essas máquinas. Com o interesse em *clusters*, aumentou a quantidade de ferramentas para utilizá-los, dentre as quais se destaca o framework MapReduce [Dean and Ghemawat 2008] e sua implementação de código aberto *Hadoop*, utilizado para gerenciar grandes quantidades de dados em *clusters* de servidores. Este framework é atraente porque oferece um modelo simples por meio do qual os usuários podem expressar programas distribuídos relativamente sofisticados.

A implantação e o gerenciamento de sistemas de *clusters* de larga escala envolvem grandes investimentos na aquisição e manutenção de equipamentos. Para melhorar o gerenciamento e reduzir os custos, as aplicações de Big Data têm utilizado ambientes de *Cloud Computing* ou Computação em Nuvem [Agrawal et al. 2011]. Estes ambientes permitem que empresas e indivíduos aluguem capacidade de computação e armazenamento sob demanda e com pagamento baseado no uso, em vez de fazerem grandes investimentos de capital necessários para a construção e instalação de equipamentos de computação em larga escala [Sousa et al. 2010]. Além disso, a Computação em Nuvem fornece ambientes com grande capacidade de armazenamento, escaláveis, elásticos, com elevado desempenho e alta disponibilidade. Assim sendo, a nuvem fornece uma alternativa viável para a construção de aplicações de gestão e análise de grandes volumes de dados [Agrawal et al. 2011].

As aplicações de Big Data estão se expandindo em todos os domínios de ciência e engenharia, incluindo física, biologia e medicina. Estas aplicações demostram que o gerenciamento de grandes volumes de dados está além da capacidade das ferramentas de software para armazenar e processar estes dados dentro de um intervalo de tempo aceitável. O desafio fundamental para as aplicações de Big Data é explorar os grandes volumes de dados e extrair informações úteis ou conhecimento para futuras ações [Rajaraman and Ullman 2012].

Em muitas situações, o processo de análise deve ser eficiente e quase em tempo real, pois o armazenamento de todos os dados observados é quase inviável [Wu et al. 2013]. Como resultado, um volume de dados sem precedentes necessita de análise eficaz. Para tanto, técnicas de mineração de dados podem ser utilizadas para analisar e entender os dados a serem manipulados. A análise é baseada em modelos capazes de sumarizar dados, extrair novos conhecimentos ou realizar predições. Estes modelos podem ser utilizados para construir um software que possibilite identificar o perfil de clientes para conceder empréstimos bancários, aplicações de recomendação de busca de amigos em redes sociais, que envolvem grafos com milhões de nós e arestas ou, ainda, sistemas de software que identifiquem possíveis ameaças terroristas [Rajaraman and Ullman 2012].

Os três principais componentes para sucesso de um projeto de mineração de dados

no contexto de Big Data são: (i) presença de um cenário de aplicação em que seja possível identificar a demanda por descoberta de conhecimento; (ii) um modelo que realize a análise desejada; (iii) uma implementação eficiente que seja capaz de gerenciar um grande volume de dados [Rajaraman and Ullman 2012]. Outros fatores devem ser considerados para resolver um problema com um grande volume de dados, tais como o tamanho e a complexidade dos dados, a facilidade com a qual os dados podem ser eficientemente transportados e principalmente se o algoritmo pode ser paralelizado de forma eficiente [Schadt et al. 2010].

Entretanto, a análise dos dados não é uma tarefa trivial. É preciso utilizar técnicas automatizadas que garantam que os dados serão explorados corretamente. Utilizar técnicas tradicionais permitindo, por exemplo, que o conhecimento seja obtido apenas por intervenções de especialistas humanos não é uma decisão viável, devido ao grande volume de dados. As técnicas de mineração podem ser utilizadas para extrair conhecimento tanto de tipos de dados convencionais, também chamados alfanuméricos (i.e. letras, números e datas), quanto de dados não convencionais, a exemplo de tipos de dados espaciais, espaço temporais, biológicos, assim como dados estruturados, bem como não estruturados, por exemplo, vídeos, fotos e som. Dessa forma, outro desafio relevante da análise de dados em Big Data é gerenciar dados de diferentes tipos e estruturas [Rajaraman and Ullman 2012].

Este minicurso apresenta e discute as principais questões relacionadas à análise para Big Data, destacando os pontos chaves e as boas práticas para a construção de novos algoritmos para Big Data. Além disso, como o ambiente de Computação em Nuvem pode auxiliar na gestão e no processamento destes dados. Este trabalho está organizado da seguinte forma: a Seção 1.2 apresenta as tecnologias e infraestruturas utilizadas para a análise de Big Data. A Seção 1.3 apresenta os principais algoritmos de mineração de dados em Big Data. A Seção 1.4 apresenta um estudo de caso baseado em dados de trânsito. Por fim, a Seção 1.6 apresenta as considerações finais sobre o tema, lições aprendidas e destaca tendências direcionamentos futuros, como as oportunidades e desafios encontrados.

1.2. Infraestrutura para Análise em Big Data

A necessidade em gerenciar e analisar uma grande quantidade de dados e fornecer alto desempenho tem sido requisitos comuns em muitas empresas. Para tratar estes problemas, diferentes soluções têm sido propostas, entre elas a migração/construção de aplicações para ambientes de computação em nuvem, além de sistemas baseados em *Distributed Hash Table* (DHT) ou estrutura de *arrays* multidimensionais [Sousa et al. 2010]. Dentre estas soluções, destaca-se o paradigma MapReduce [Dean and Ghemawat 2008], concebido para apoiar o processamento distribuído de grandes conjuntos de dados em *clusters* de servidores e sua implementação de código livre *Hadoop* [White 2012].

Computação em Nuvem provê soluções baratas e eficientes para armazenamento e análise de grandes volumes de dados [Li and Zhang 2011]. Existem diferentes definições e conceitos de computação em nuvem. Segundo [Mell and Grance 2009], computação em nuvem pode ser definido como um modelo que permite acesso sob demanda a um agrupamento de recursos computacionais que podem ser configuráveis, como CPU, armazenamento, memória, entre outros. Eles podem ser rapidamente fornecidos e liberados

com o mínimo esforço de gerenciamento ou assistência do provedor da nuvem.

Algumas propriedades fundamentais distinguem computação em nuvem dos sistemas distribuídos tradicionais (e.g. sistemas em grade, clusters, P2P, etc) e estão relacionadas ao seu caráter atrativo: (i) Auto-Serviço sob demanda, (ii) Elasticidade rápida, (iii) Pagamento à medida que o serviço é utilizado (*Pay-as-you-go*) (iv) Nível de qualidade de serviço (SLA), (v) Agrupamento ou *Pooling* de Recursos. A seguir, falaremos mais de cada uma dessas características.

Auto-serviço sob demanda diz respeito à possibilidade do consumidor utilizar os recursos computacionais (tempo de servidor, por exemplo) quando necessário sem qualquer intervenção humana com o provedor do serviço. Outra caracterítica é a elasticidade que permite escalar mais recurso com o aumento da demanda, e liberar recurso, na retração dessa demanda. Para o consumidor, os recursos disponíveis para provisionamento muitas vezes parecem ser ilimitados e podem ser alocados em qualquer quantidade e a qualquer momento. Os recursos em sistemas na nuvem são automaticamente controlados e o seu uso é medido em um nível de abstração apropriado para o tipo de serviço (como armazenamento, processamento, por exemplo). No modelo *Pay-as-you-go*, o consumidor só paga pelo que utiliza e pelo tempo de utilização. A utilização dos recursos pode ser monitorada, controlada e informada, gerando transparência tanto para o provedor como para o consumidor do serviço utilizado.

O SLA é a garantia mínima de qualidade de serviço. Os recursos de computação do provedor são agrupados para atender a múltiplos consumidores em modelo multi-inquilinos. Os diferentes recursos físicos e virtuais são dinamicamente atribuídos e reatribuídos conforme a demanda dos consumidores. Há uma certa independência de localização geográfica, uma vez que o consumidor em geral não controla ou conhece a localização exata dos recursos fornecidos (como armazenamento, processamento, memória e comunicação de rede), mas pode ser capaz de especificar a localização em um nível de abstração mais alto (como país, estado ou *datacenter*). Consequentemente, qualquer serviço computacional que seja executado em um ambiente em nuvem precisará estar em conformidade com tais propriedades [Coelho da Silva 2013].

O Hadoop implementa o modelo de programação MapReduce, juntamente com um sistema de arquivo distribuído chamado *Hadoop Distributed File System* (HDFS) para permitir o processamento de grandes volumes de dados em ambientes distribuídos. O HDFS foi projetado e otimizado para ser executado em centros de dados e fornecer elevada vazão, baixa latência e tolerância a falhas individuais de servidores.

Em clusters ou ambientes de computação em nuvem, é comum haver falhas nos nós ou na rede. Dessa forma, é necessário o uso de estratégias para tratar estas falhas, principalmente considerando que o processamento pode demandar horas para seu término. A cada falha pode ser necessário abortar e reiniciar todo o processamento. Um cenário possível é a execução nunca ser concluída com êxito. Algumas medidas poderiam ser adotadas como, por exemplo: (i) armazenar arquivos redundantes e (ii) dividir o processamento entre os nós computacionais. Assim sendo, quando uma tarefa é interrompida por uma falha, ela não prejudica o processamento de outras tarefas, basta que somente ela seja reiniciada. O HDFS implementa estratégias para o tratamento de falhas, como o uso de replicação e distribuição dos dados.

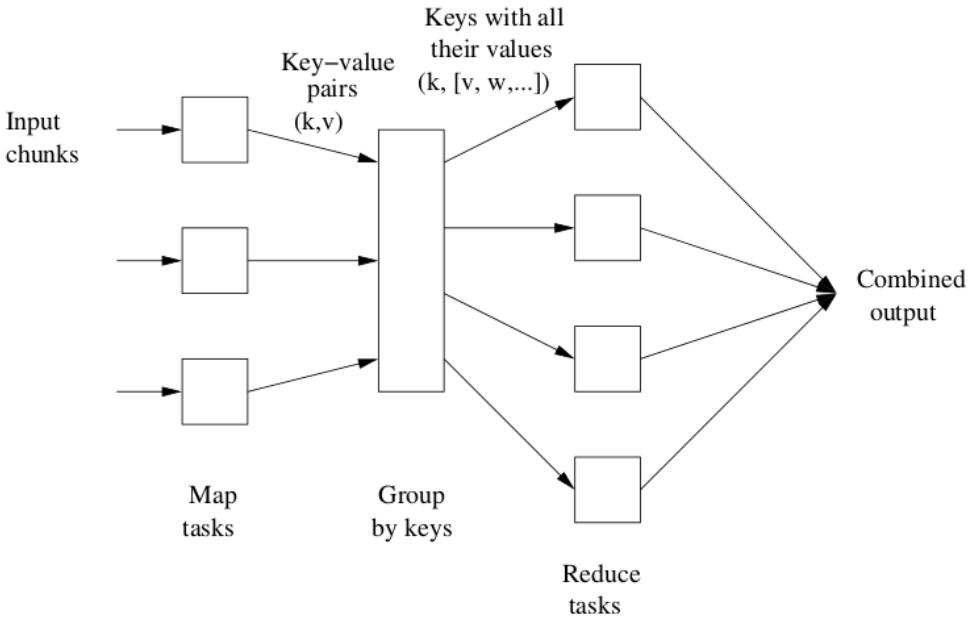

Figura 1.1. Funcionamento das funções Map e Reduce retirada do livro [Rajaraman and Ullman 2012]

O Hadoop utiliza o modelo de programação MapReduce para a execução de programas. Este modelo é baseado em duas primitivas da programação funcional: Map e Reduce. A execução MapReduce é realizada da seguinte forma: (i) A função Map recebe como entrada uma lista de pares chave-valor $\langle K_1, V_1 \rangle$ e sua saída também é uma lista de pares chave-valor intermediários $\langle K_2, V_2 \rangle$; (ii) Os pares chave-valor $\langle K_2, V_2 \rangle$ são definidos de acordo com a implementação da função Map fornecida pelo usuário, e ao final de cada tarefa Map, são coletados por um nó mestre. Em seguida, ordenados pela chave. As chaves são divididas entre todas as tarefas Reduce. Os pares chave-valor que possuem à mesma chave são designados a mesma tarefa Reduce; (iii) A função Reduce recebe como entrada todos os valores V_2 de uma mesma chave K_2 e produz como saída pares chave-valor $\langle K_3, V_3 \rangle$ que formam o resultado do processo MapReduce. As tarefas Reduce executam sobre uma chave por vez, e a forma de combinação dos valores é determinada pelo código da função Reduce dada pelo usuário. O funcionamento das funções Map e Reduce pode ser acompanhado como apresentado na Figura 1.1.

A arquitetura MapReduce é Mestre-Escravo. O nó Mestre possui várias responsabilidades, entre elas criar certo número de tarefas Map e tarefas Reduce (esses números podem ser dados de entrada pelo usuário). Além disso, o Mestre aloca as tarefas aos nós Escravos. No entanto, é desejável limitar o número de tarefas Reduce, pois cada tarefa Map cria um arquivo intermediário para cada tarefa Reduce, dessa forma se existem muitas tarefas Reduce, o número de arquivos intermediários pode ser muito grande. O nó Mestre guarda o estado de cada tarefa Map e Reduce (em espera, em execução ou finalizada). O nó Escravo reporta ao Mestre quando termina uma tarefa, e só então uma nova tarefa é escalonada pelo Mestre ao Escravo.

Os arquivos intermediários criados por uma tarefa Map (para cada tarefa Reduce)

são salvos no disco local do nó Escravo. O Mestre é informado da localização e do tamanho desses arquivos, e escalona uma ou mais tarefas Reduce que os recebem de entrada. Cada tarefa Reduce executa o código escrito pelo usuário e grava a saída do sistema de arquivo distribuído HDFS. O Hadoop gerencia a execução paralela, coordena a execução das tarefas Map e Reduce e trata as falhas durante a execução.

Um exemplo de computação MapReduce bastante difundido é a contagem do número de ocorrências que cada palavra aparece em uma coleção de documentos. Nesse exemplo, a entrada é um repositório de documentos. A função Map desse exemplo usa chaves que são do tipo Strings (as palavras) e os valores são inteiro. A tarefa Map lê o documento e o quebra em uma sequência de palavras w_1, w_2,..., w_n. Em seguida, emite uma sequência de pares chave-valor onde o valor é igual a 1. Dessa forma, a saída do Map para determinado documento do repositório é uma sequência de pares chave-valor: $\langle w_1, 1\rangle$, $\langle w_2, 1\rangle$,...,$\langle w_n, 1\rangle$.

Uma única tarefa Map pode processar muitos documentos. Então, a saída irá conter mais de uma sequência. Outro ponto importante é se uma palavra w aparece m vezes entre todos os documentos a serem lidos, então a saída conterá m pares chave-valor $\langle w, 1\rangle$. A função Reduce combina esses m pares em um único par $\langle w, m\rangle$.

1.3. Mineração de Dados em Big Data

1.3.1. Boas práticas para Algoritmos em Big Data

O processo de minerar dados é formado por um conjunto de técnicas para descoberta de conhecimento a partir de grandes bases de dados. Tais técnicas baseiam-se em modelos capazes de sumarizar dados, extrair novos conhecimentos ou realizar predições. Classificação é uma técnica de mineração de dados que está na categoria de aprendizagem supervisionada, ou seja, é fornecida uma classe a qual cada amostra do conjunto de dados de treinamento pertence. Os algoritmos que implementam essa técnica são preditivos, pois suas tarefas de mineração desempenham inferências nos dados com o intuito de fornecer previsões ou tendências, obtendo informações não disponíveis a partir dos dados disponíveis.

Outras duas técnicas também mais conhecidas, porém não supervisionadas, são Clusterização e Associação. Nestes algoritmos, o rótulo da classe de cada amostra do treinamento não é conhecido, e o número ou conjunto de classes a ser treinado pode não ser conhecido a priori, daí o fato de ser uma aprendizagem não-supervisionada. Clusterização é uma técnica de aprendizagem não supervisionada muito utilizada em mineração de dados. Assume papel importante em aplicações como reconhecimento de padrões e recuperação de informação. Os algoritmos de clusterização particionam os dados em diferentes grupos, tal que a similaridade entre elementos que pertencem ao mesmo grupo seja minimizada e a similaridade entre elementos que pertencem a grupos diferentes seja maximizada. A Associação é capaz de estabelecer regras entre os atributos. O próprio algoritmo elege atributos determinantes (lado esquerdo da regra) e os atributos resultantes (lado direito) na tarefa de mineração revelando associações entre valores dos atributos, tendo o algoritmo sua ênfase no compromisso entre precisão e cobertura [Silva 2004].

Conforme discutido anteriormente, os algoritmos de mineração de dados em sua

versão centralizada não são adequados para processar dados Big Data. Assim, podemos elencar um conjunto de boas práticas para que esses algoritmos sejam modificados e satisfatórios para o contexto de Big Data. Os trabalhos apresentados na próxima seção seguem esses princípios, o que os torna adequados para grandes volumes de dados.

Do nosso conhecimento, não há nenhuma publicação ou estudo que apresente quais os requisitos que um algoritmo deve ter para se tornar adequado ao contexto Big Data. O que conseguimos perceber foi um conjunto de boas práticas que estão presentes em muitos algoritmos que trabalham com grandes volumes de dados. Um dos nossos trabalhos futuros é caracterizar e descobrir quais outras características se enquadrariam também nesse cenário.

O primeiro passo para se obter um bom algoritmo que manipule dados Big Data é paralelizá-lo e utilizar o paradigma de divisão e conquista [Ji et al. 2012], isso pode ser alcançado utilizando MapReduce, que é capaz de manipular grandes volumes de dados e eficientemente executar o algoritmo em paralelo. Além disso, os dados devem estar distribuídos entre os nós de processamento. Essa distribuição ou particionamento dos dados deve ser balanceada entre os nós do cluster, a fim de evitar sobrecarga em um nó em particular. É necessário nesse caso observar o desvio da distribuição do dado. Além disso, é comum a utilização de replicação para paralelizar os algoritmos. No contexto de Big Data, é importante evitar replicação de dados, já que possivelmente pode resultar em crescimento exponencial da quantidade de dados a se processar. Outro ponto relevante é que os sistemas sobre os quais esses algoritmos executam também devem ser satisfatórios para a análise de dados Big Data.

Em [Cohen et al. 2009, Herodotou et al. 2011], são elencados 6 princípios que tornam um sistema adequado para Big Data, conhecido como MADDER: *Magnetic*, *Agile*, *Deep*, *Data-lifecycle-awareness*, *Elasticity* e *Robustness*. A seguir, explicamos cada um deles.

- *Magnetic*: O sistema é capaz de manipular qualquer fonte de dados independente da presença de possíveis *outliers*, de não haver esquema ou estrutura, além da ausência de valores para alguns atributos.
- *Agile*: O sistema se adapta à rápida evolução dos dados.
- *Deep*: O sistema suporta análises mais complexas, utilizando modelos estastísticos e técnicas de aprendizagem de máquina, por exemplo.
- *Data-lifecycle-awareness*: O sistema otimiza a movimentação, o armazenamento e o processamento de dados Big Data durante todo seu ciclo de vida.
- *Elasticity*: O sistema é capaz de ajustar o uso dos recursos e custos operacionais aos requisitos dos usuários e do processamento da carga de trabalho.
- *Robustness*: O sistema continua a prover serviço mesmo com possíveis adversidades como, por exemplo, falha de hardware e dados corrompidos.

Em [Begoli and Horey 2012], é possível encontrar alguns princípios para projetos em Big Data.

1.3.2. Exemplos de Algoritmos de Mineração de Dados modificados para tratar grandes volumes de dados

A seguir, reunimos os trabalhos dos 2 últimos anos das principais conferências e periódicos em banco de dados e mineração de dados. Os principais trabalhos que seguem as boas práticas apresentadas anteriormente são apresentados nas próximas subseções.

1.3.2.1. Itens Frequentes e Regras de Associação

A descoberta dos itens/palavras mais frequentes e Regras de Associação são técnicas bastante conhecidas em mineração e aplicações de banco de dados. Porém algumas dificuldades surgem quando a técnica é aplicada a um grande volume de dados, no caso do algoritmo Apriori [Agrawal et al. 1994], o tempo gasto pode ser exponencial [Yang et al. 2010]. Os dados possuem um conjunto de itens como, por exemplo, uma lista de compras de um consumidor, uma sequência de páginas visitadas por meio de um browser, uma coleção de vídeos avaliados por um espectador, um sequência de mutações nos genes de um amostra de DNA. Dado esse conjunto de dados, é interessante identificar padrões e regras de associação entre os itens que nele se encontram, ou seja, descobrir quais itens aparecem juntos frequentemente nesse conjunto de dados.

Os trabalhos [Agrawal et al. 1994] e [Yang et al. 2010] propõem uma implementação do algoritmo Apriori paralelizada via MapReduce. Porém, múltiplas iterações de MapReduce são necessárias durante a computação. Uma iteração produz um item frequente, e deve-se continuar iterando até que não existam mais itens novos a serem adicionados. Essa estratégia pode ser muito custosa para um grande volume de dados.

O trabalho [Riondato et al. 2012] também apresenta um algoritmo paralelizado usando MapReduce para mineração de itens frequentes e regras de associação, além disso não utiliza replicação de dados. Para isso, são criadas várias amostras randômicas e pequenas do conjunto de dados. O algoritmo de mineração é executado nessas amostras independentes e em paralelo. Os resultados gerados são coletados, agregados e filtrados para prover uma única saída. Como [Riondato et al. 2012] minera subconjuntos randômicos do conjunto de dados, o resultado final é uma aproximação da solução real. O artigo fez uma análise propabilística para mostrar que realmente provê garantias quanto a aproximação com a solução real. O usuário final deve fornecer a acurácia e a confiança como parâmetros e a estratégia computa uma aproximação da coleção de interesse que satisfaz esses parâmetros.

Embora [Riondato et al. 2012] não seja a primeira proposta (como vimos anteriormente) para solucionar tais problemas em mineração de dados utilizando MapReduce, o que o difere dos outros [Cryans et al. 2010, Ghoting et al. 2011, Hammoud 2011, Li et al. 2008, Li and Zhang 2011, Yang et al. 2010] são dois aspectos fundamentais. O primeiro, os outros trabalhos usam replicação de dados para paralelizar a computação, possivelmente resultando em crescimento exponencial da quantidade de dados. Ao contrário do [Riondato et al. 2012] que somente minera um número de amostras randômicas do conjunto de dados inicial. Como o MapReduce apresenta custo alto em relação à movimentação dos dados entre diferentes máquinas (denotado como “shuffling”), [Riondato et al. 2012] garante melhor desempenho que os anteriores. Outra diferença é

que os trabalhos anteriores criam as regras de associação sequencialmente após a coleta via MapReduce dos itens frequentes, ao contrário de [Riondato et al. 2012] que tanto a extração dos itens frequentes quanto a criação das regras são feitas via MapReduce. Considerando as boas práticas apresentadas anteriormente, tais diferenças mostram que [Riondato et al. 2012] é mais adequado para ser utilizado no contexto de Big Data do que os trabalhos [Cryans et al. 2010, Ghoting et al. 2011, Hammoud 2011, Li et al. 2008, Li and Zhang 2011, Yang et al. 2010].

1.3.2.2. Classificação

Os algoritmos de Classificação são capazes de realizar predição de categorias. Os mais conhecidos são árvore de decisão [Quinlan 1986], redes bayesianas [Michie et al. 1994] e os vizinhos mais próximos [Cover and Hart 1967]. Da mesma forma que para os outros algoritmos de mineração de dados apresentados anteriormente, as pesquisas também focaram em oferecer técnicas de classificação de processamento paralelo e distribuído para grandes volumes de dados.

O trabalho [He et al. 2010] propõe uma versão paralelizada dos algoritmos de árvore de decisão, redes bayesianas e vizinhos mais próximos utilizando MapReduce. Para o algoritmo dos k-vizinhos mais próximos, a computação para encontrar a similaridade entre os dados do conjunto de treino e teste é feita independentemente. Assim, o conjunto de dados fornecido como entrada pode ser particionado em n blocos que podem ser processados em diferentes nós. Além do conjunto de dados de entrada, também é fornecido o número k de vizinhos mais próximos. A função de Map calcula a similaridade das amostras dadas de entrada e a função Reduce ordena as similaridades e seleciona os k-vizinhos mais próximos.

Para o classificador utilizando redes bayesianas, o cálculo intensivo a ser realizado é o das probabilidades condicionais. Inicialmente, ocorre a fase de treino utilizando MapReduce. Na função de Map, os atributos e seus valores são identificados com valor de chave igual a 1. Na função de Reduce é contabilizada a frequência de cada atributo e seu valor. Na fase de teste também se utiliza MapReduce, a função de Map lê as probabilidades geradas pela fase de treino e calcula a probabilidade de amostras de testes pertencerem a cada categoria da classificação, bem como se a predição da categoria foi realizada corretamente ou não. Na função Reduce, calcula-se a quantidade de amostras categorizadas corretamente e erroneamente.

A ideia básica do algoritmo de árvore de decisão é recursivamente escolher o melhor atributo para dividir os nós da árvore. Após selecionar um atributo, o conjunto de treino é dividido em várias partições de acordo com o valor do atributo escolhido. Na fase de treino, o processo de MapReduce é iniciado para cada partição que recursivamente computa o melhor atributo para dividir os dados no nó corrente da árvore. As regras de decisão são armazenadas e novas regras são geradas. Na fase de teste, a função Map verifica para cada amostra, as regras de decisão e prevê uma categoria. Somente a função Map é necessária.

Alguns trabalhos como [Kim and Shim 2012], [Lu et al. 2012], [Ghoting et al. 2011], [Zhou et al. 2012], [Panda et al. 2009] apresentam a aplicação e implementação dos al-

goritmos de classificação discutidos acima utilizando MapReduce, ou ainda, aplicando algumas boas práticas para Big Data.

1.3.2.3. Clusterização

O k-means continua a ser o mais popular método de clusterização, além de ser identificado como um dos mais importantes dentre os algoritmos de mineração de dados [Wu et al. 2008]. Entretanto, em termos de qualidade e eficiência, a ordem de execução do k-means pode ser exponencial no pior caso. Além disso, a solução encontrada pode ser de um ótimo local, e possivelmente longe de se obter um ótimo global. O algoritmo k-means++ [Arthur and Vassilvitskii 2007] foi proposto como envolução do k-means e oferece melhor estratégia de inicialização dos centróides.

No k-means, são selecionados k centróides randomicamente em uma única iteração de acordo com uma distribuição específica, já em [Arthur and Vassilvitskii 2007] são k iterações e seleciona-se um ponto em cada iteração considerando que qualquer ponto não tem a mesma probabilidade de ser selecionado que os outros (pois esta probabilidade é constantemente alterada quando cada novo centróide é selecionado). Este é o ganho do algoritmo k-means++, porém devido a sua natureza sequencial, a aplicabilidade é limitada para grandes volumes de dados: deve-se iterar k vezes sobre os dados para encontrar um bom conjunto de centróides inicialmente. Idealmente, gostar-se-ia de ter um algoritmo que precise de um menor número de iterações, e que selecione mais de um ponto em cada iteração da mesma maneira que o K-means++. Em [Bahmani et al. 2012], é proposta uma estratégia paralela para reduzir o número de iterações necessárias, a fim de obter uma boa inicialização utilizando MapReduce. [Bahmani et al. 2012] segue essa intuição e encontra o ponto ideal de centróide (ou o melhor ponto de trade-off), definindo cuidadosamente o número de iterações e considerando que a probabilidade de selecionar cada ponto é diferente de qualquer outro ponto. Com esse cenário, a aplicabilidade do [Bahmani et al. 2012] é possível para grandes volumes de dados.

Um dos mais importantes algoritmos de clusterização é o DBScan(*Density-based Spatial Clustering of Application with Noise*) [Ester et al. 1996]. Suas vantagens em relação às outras técnicas de clusterização são agrupar dados em clusters de formato arbitrário, não requerer o número de clusters a priori, além de lidar com *outliers* no conjunto de dados. Em [He et al. 2011], é proposta uma implementação do DBScan em 4 estágios de MapReduce, utilizando particionamento baseado em grid. Como o conjunto de dados é particionado, o artigo também apresenta uma estratégia para junção de clusters que estão em partições diferentes e que contêm os mesmos pontos em suas fronteiras. Tais pontos estão replicados nas partições e a descoberta de quais clusters podem ser reunidos em um só cluster é averiguada a partir deles. Note que o número de pontos de fronteira replicados pode afetar a eficiência da clusterização, pois tais pontos não somente aumentam a carga de cada nó computacional, mas aumentam ainda o tempo de reunir os resultados dos diferentes nós computacionais.

Semelhante ao trabalho anterior, em [Dai and Lin 2012] também é proposta a implementação do DBScan utilizando MapReduce. O conjunto de dados é particionado de modo a manter o balanceamento de carga entre os nós computacionais, e ainda minimi-

zar os pontos de fronteiras entre as partições. Dessa forma, aumenta-se a eficiência da clusterização e do processo de junção dos clusters. O DBScan é executado em cada nó computacional sobre cada partição utilizando um índice Kd-tree. A junção de clusters que estão em partições distintas é feita quando um mesmo ponto estiver em tais partições e for designado como um *core point* em algum dos clusters. Se é detectado que dois clusters devem sofrer processo de junção, os clusters são renomeados. Tal processo também ocorre em [He et al. 2011].

Nosso estudo de caso se assemelha a ambos [He et al. 2011], [Dai and Lin 2012], pois considera o desafio de manipular grandes volumes de dados e ambas soluções utilizam MapReduce para paralelizar o algoritmo DBScan. Porém, nosso estudo de caso utiliza outra técnica de particionamento, propõe outra estratégia de junção de clusters que não necessita de replicação. Por isso, se torna mais adequado às boas práticas listadas anteriormente para Big Data.

1.4. Estudo de Caso

Para esse curso foi preparado um estudo de caso que aborda a aplicação do algoritmo de DBScan [Ester et al. 1996] em um grande volume de dados, utilizando uma plataforma de nuvem e o paralelismo obtido por meio do MapReduce. O enfoque é mostrar como a técnica de mineração foi modificada para ser aplicada em *Big Data*.

1.4.1. Descrição do Estudo de Caso

A divulgação de informações sobre o trânsito nas grandes cidades pode ser obtido de *tweets*, coletadas por meio de GPS nos veículos ou foto-sensores. A informação obtida pode ser usada tanto de forma complementar àquela gerada por câmeras e sensores físicos, orientando as ações dos agentes públicos a curto e longo prazo, quanto diretamente pelos motoristas, em tempo real ou quase-real, apoiando suas decisões quanto ao deslocamento pela cidade [Ribeiro Jr et al. 2012].

Por meio desses dados é possível analisar o comportamento do trânsito nas cidades e descobrir, por exemplo, padrões como: em quais trechos da cidade ocorrem mais congestionamentos? Tal descoberta pode auxiliar a busca por soluções eficazes de reengenharia de trânsito no contexto de cidades inteligentes. Dessa forma, a crescente popularidade desse tipo de canal de informação indica que, em pouco tempo, informações sensoriadas e transmitidas pelos próprios cidadãos podem se tornar a principal fonte de avaliação da situação do trânsito em tempo real [Ribeiro Jr et al. 2012]. Dessa forma, a obtenção de conhecimento a partir desses dados é uma tarefa importante.

O estudo de caso desse trabalho consiste em, a partir de dados de trânsito, detectar em que trechos ocorrem congestionamento em uma cidade com frequência. Para isso, o algoritmo de DBScan visto na Seção 1.3 é utilizado como solução, porém dado o grande volume de dados deve-se utilizar uma versão paralelizada. Dessa forma, outro problema consiste em como paralelizar tal algoritmo no nosso contexto de dados de trânsito. Para isso, o MapReduce é utilizado como plataforma para a execução do DBScan paralelizado. Outros trabalhos, tais como [He et al. 2011], [Dai and Lin 2012], também utilizam MapReduce para paralelizar o algoritmo de DBScan, porém apresentam estratégias e cenário diferentes do apresentado neste texto.

1.4.2. Descrição dos Dados de trânsito

Os dados utilizados nesse estudo de caso são da cidade de Fortaleza e foram coletados de sites de trânsito e de GPS. Tais dados são sobre o tráfego na cidade, informando a velocidade média em que os carros trafegam em determinado local (latitude e longitude) e o horário da coleta.

A manipulação desses dados nesse trabalho é considerada Big Data, (i) pela presença do grande volume de dados a ser gerenciado, (ii) pela variedade dos dados (dados obtidos a partir de diferentes fontes, os quais não estruturam a informação da mesma forma) e (iii) pela velocidade com que essas informações chegam aos conjuntos de dados, dado que a todo instante de tempo, os sensores coletam a posição espaço temporal do veículo, bem como a sua velocidade.

1.4.3. Infraestrutura de Computação em Nuvem

A infraestrutura utilizada nesse estudo de caso é o ambiente de nuvem privada da Universidade Federal do Ceará (UFC) que tem como plataforma o OpenNebula. O número de máquinas virtuais utilizadas foram 11 com sistema operacional Ubuntu, 8GB de memória RAM e 4 unidades de CPU. A versão do Hadoop instalada em cada máquina foi a 1.1.2 e as variáveis de ambiente foram setadas com valores como mostrados na Tabela 1.1. Em cada máquina também deve ser instalado o PostgreSQL 9.1, e ainda a biblioteca *Spatial and Geographic Objects for PostgreSQL* (PostGIS) para manipular objetos espaciais.

Tabela 1.1. Variáveis de configuração do Hadoop setadas.

Variável de Configuração	Valor
hadoop.tmp.dir	/tmp/hadoop
fs.default.name	hdfs://master:54310
mapred.job.tracker	master:54311
mapreduce.task.timeout	36000000
mapred.child.java.opts	-Xmx8192m
mapred.reduce.tasks	11
dfs.replication	5

1.4.4. Algoritmo DBScan para dados Big Data

O estudo de caso envolve diretamente a velocidade do fluxo de trânsito, uma vez que os congestionamentos se caracterizam principalmente por baixas velocidades. Assim, a primeira etapa é a partir dos dados coletados, inicialmente um filtro é aplicado para que apenas registros com velocidades abaixo de 20km/h (consideradas como congestionamento) sejam considerados no conjunto de dados e os outros sejam descartados.

A segunda etapa é considerar a dimensão espacial: latitude e longitude. Nessa fase ocorre a clusterização por proximidade geográfica dos pontos que foram selecionados de baixa velocidade. Dessa forma, será possível visualizar onde as velocidades baixas se concentram considerando a dimensão espacial, que representarão potencialmente os congestionamentos identificados. Porém, uma última etapa ainda é necessária. Além do processamento já realizado, a clusterização por tempo é fundamental para se identificar

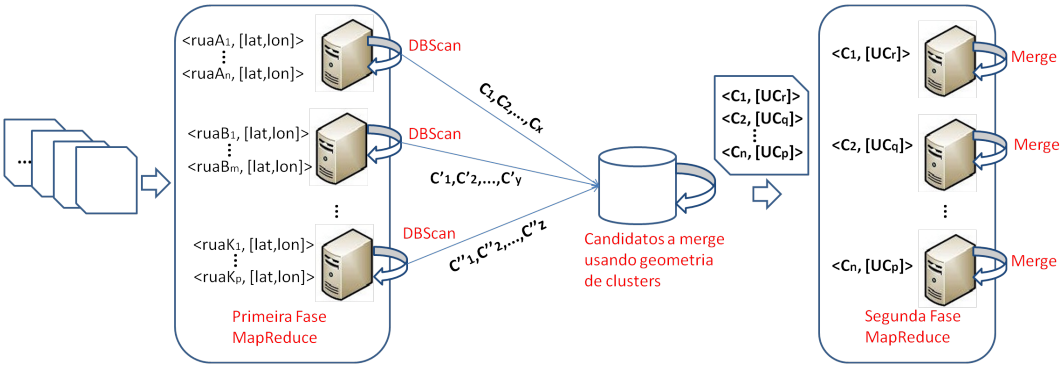

Figura 1.2. Arquitetura da nossa solução

em quais horários os congestionamentos ocorrem. No entanto, para caracterizar melhor a solução proposta, essa fase não é abordada nesse estudo de caso.

Dessa forma, a partir de um grande volume de dados de trânsito de uma cidade, o objetivo é encontrar os trechos da cidade onde há mais congestionamento. Assim, é necessário identificar grupos que possuem alta densidade de pontos nesse conjunto de dados, e ainda que apresentem baixas velocidades. A seguir, os passos para paralelizar o DBScan utilizando o modelo de programação MapReduce que constituem a implementação da solução proposta para o problema, também apresentado na Figura 1.2

- Na primeira fase de Map, cada registro do conjunto de dados será descrito como um par $\langle chave, valor \rangle$, tal que a chave diz respeito à rua em que a informação se refere, e o valor à posição geográfica (latitude e longitude) da coleta.

- Em seguida, ocorre o Reduce que recebe uma lista de valores de mesma chave, ou seja, as posições geográficas de uma mesma rua. Nessa fase, o algoritmo DBScan é aplicado. O resultado é armazenado em uma base de dados, são guardados o id de cada cluster e as informações dos seus pontos (latitude, longitude, se é *core point*, se é *noise*, por exemplo). Esses termos são definidos pela técnica do DBScan.

- Como as ruas de uma cidade se cruzam e o particionamento dos dados é feito por rua, nessa fase é necessário descobrir quais clusters de ruas diferentes se interceptam ou podem ser reunidos em apenas um cluster. Ou seja, dois clusters podem ter pontos a uma distância menor que *eps*, de tal sorte que se os dados fossem processados em um mesmo Reduce ou fossem de uma mesma partição, ter-ia-se apenas um único cluster. Dessa forma, o cluster é também armazenado como um objeto geométrico no banco de dados, e apenas objetos que estão a uma distância de no máximo *eps* poderão passar pela fase de *merge* que é a fase seguinte. Tuplas com pares de clusters candidatos a *merge* são passadas com mesma chave para o próximo MapReduce.

- Essa fase pode ser chamada de *merge* de clusters, ela também é descrita por um processo MapReduce. A função Map é a identidade. A função Reduce recebe como chave o menor id de clusters que devem ser reunidos em um só, e como valor os outros candidatos a *merge* com a chave. Nessa fase, se dois clusters devem

ser reunidos em um só, as informações dos pontos pertencentes aos clusters são atualizadas. A abordagem desse trabalho considera a possibilidade de um ponto que é *noise* em um cluster, com o merge de clusters, pode ter se tornado um *border* ou *core point*.

A seguir os algoritmos implementados da solução proposta.

1.4.5. Implementação dos algoritmos propostos

Na programação MapReduce é necessário definir duas classes que estendam de *MapReduceBase* e implementem uma das seguintes interfaces: *Mapper* e *Reducer*. Os tipos de entrada e saída das funções de Map e Reduce são parametrizadas, sendo que esses parâmetros devem implementar a interface Writable (o Hadoop já fornece alguns tipos básicos que seguem essa interface).

```
1  public static class  ClusterMapper extends MapReduceBase implements Mapper<LongWritable,
       Text, Text, Text>
2      {
3            //registro é formado por uma tripla <Rua, lat, long>
4            public void map(LongWritable key, Text value, OutputCollector<Text, Text>
                 output, Reporter reporter) throws IOException
5            {
6
7                Text street = new Text();
8                String line = value.toString();
9                String[] tokenizer = line.split(",");
10               street.set(tokenizer[0]);
11               String values=tokenizer[1]+" "+tokenizer[2];
12               value.set(values);
13           output.collect(street,value);
14           }
15     }
```

Algoritmo 1.1. Primeiro MapReduce-Fase de particionamento

O procedimento *map* no Algoritmo 1.1 *ClusterMapper* tem como objetivo converter os dados recebidos em arquivos, tal que cada registro é formado por uma tripla $\langle nomedarua, latitude, longitude\rangle$ em pares chave-valor $\langle K, V\rangle$, de tal sorte que K é o nome da rua e V é uma tupla $\langle latitude, longitude\rangle$. Tais pares são passados como entrada para a fase de Reduce a seguir. Isso é possível, pois o Hadoop por padrão particiona os arquivos a serem processados por linha do arquivo, assim cada linha é passada por vez ao procedimento *map*.

Ainda no primeiro *MapReduce* na fase de *Reduce*, o Algoritmo 1.2 recebe para uma mesma chave um conjunto de valores, sendo a chave o nome da rua e cada valor uma posição geográfica pertencente à rua que é uma tupla $\langle latitude, longitude\rangle$. O algoritmo de DBScan utilizando índice KD-tree [Bentley 1975] é aplicado aos pontos de uma mesma rua, sendo essa fase paralelizada para ruas distintas que pertencem ao conjunto de dados de entrada. Os resultados de cada cluster criado nessa fase são armazenados em uma base de dados, nesse estudo de caso utiliza-se o PostgreSQL.

A classe contendo o método principal é apresentada no Algoritmo 1.3 em que o objeto JobConf é construído. Por meio deste é possível determinar todas as configurações iniciais, além dos diretórios de entrada e saída de dados e dos parâmetros de *eps*

e *minPoints* necessários aos algoritmos anteriormente apresentados. Perceba que o Algoritmo 1.2 tem um método para obter o que foi passado de parâmetro para JobConf no método principal.

```
1  public static class ClusterReducer extends MapReduceBase implements Reducer<Text, Text,
       Text, Text> {
2          private static double eps;
3          private static int minPoints;
4          public void configure(JobConf job) {
5              eps = Double.parseDouble(job.get("eps"));
6              minPoints = Integer.parseInt(job.get("minPoints"));
7              super.configure(job);
8          }
9          public void reduce(Text key, Iterator<Text> values, OutputCollector<Text, Text>
               output, Reporter reporter) throws IOException {
10
11         String path = key.toString().replaceAll(" ", "");
12         BufferedWriter out = new BufferedWriter(new FileWriter(file));
13         String[] line;
14         while (values.hasNext()) {
15             line = values.next().toString().trim();
16             out.write(line[0]+","+line[1]+"\n");
17         }
18         out.close();
19         dbscan.loadPoints(file);
20         dbscan.createIndex();
21         dbscan.dbscan(key.toString(), eps, minPoints);
22         output.collect(key, new Text("points clustered"));
23        }
24   }
```

Algoritmo 1.2. Primeiro MapReduce-Aplicação do DBScan nos dados de uma mesma partição

```
1  public class FirstMapReduceKdTree {
2        public static void main(String[] args) throws IOException {
3            JobConf conf = new JobConf(FirstMapReduceKdTree.class);
4            conf.set("eps", args[2]);
5            conf.set("minPoints", args[3]);
6            conf.setJobName("ClusteringKdTree");
7            conf.setOutputKeyClass(Text.class);
8            conf.setOutputValueClass(Text.class);
9            conf.setMapperClass(ClusterMapper.class);
10           conf.setReducerClass(ClusterReducer.class);
11           conf.setInputFormat(TextInputFormat.class);
12           conf.setOutputFormat(TextOutputFormat.class);
13           FileInputFormat.setInputPaths(conf, new Path(args[0]));
14           FileOutputFormat.setOutputPath(conf, new Path(args[1]));
15           conf.setJarByClass(FirstMapReduceKdTree.class);
16           JobClient.runJob(conf);
17     }
18 }
```

Algoritmo 1.3. Classe que prepara os dados e que invoca as funções Map e Reduce

A fase seguinte é feita por meio de dois procedimentos armazenados no PostgreSQL. O SGBD escolhido poderia ter sido outro que tivesse suporte a dados espaciais como, por exemplo, Oracle. O primeiro procedimento quando invocado cria um objeto geométrico de cada cluster obtido da fase anterior. O segundo tem como objetivo encontrar quais clusters deveriam ser apenas um se estivessem em uma mesma partição durante a aplicação do DBScan. Assim, para descobrir quais clusters devem sofrer *merge*, é verificado quais desses objetos estão a uma distância menor ou igual que *eps*. A função utilizada para criar objetos geométricos é st-convexhull e para comparar se a distância

entre dois objetos geométricos é menor que *eps*, utiliza-se ST-Distance. Ambas fazem parte da biblioteca PostGIS.

Para descobrir se dois clusters devem sofrer merge, o Algoritmo 1 a seguir recebe dois clusters dados de entrada C_1 e C_2. Em seguida, para cada ponto $p_i \in C_1$ é verificado quais pontos $p_j \in C_2$ estão a uma distância menor ou igual a *eps* de p_i. Dessa forma, a vizinhança de cada ponto dos dois clusters é atualizada e, consequentemente, o número de vizinhos. Perceba que com isso, é possível que um ponto $p_i \in C_1$ que antes era *border*, pois em C_1 havia menos que *minPoints* vizinhos, com a junção de $C_1 \cup C_2$ pode virar *core* por ter aumentado sua vizinhança. Quando dois pontos quaisquer $p_i \in C_1$ e $p_j \in C_2$ são *core* e estão a uma distância menor que *eps*, isso é suficiente e necessário para que haja *merge* dos clusters. O Algoritmo 1 mostra como é verificado se dois clusters devem se unir, bem como atualiza o número de vizinhos e os *labels ehCore, ehBorder ou ehNoise*.

Se o retorno do Algoritmo 1 indica que deve ocorrer *merge*, os pontos pertencentes a cada cluster são renomeados para que tenham o mesmo identificador de cluster. Anteriormente a essa verificação de *merge*, o conjunto de clusters candidatos é dado de entrada para a segunda fase de MapReduce. Como toda entrada de uma função Map ou Reduce é dada por um par constituído por uma chave e a ela associada um conjunto de valores, a chave, nesse caso, é o identificador de um cluster que é chamado de representante. Os valores são todos os pares de clusters candidatos a *merge* com o representante.

```
1   public static class MergeReducer extends MapReduceBase implements Reducer<Text, Text
        , Text, Text>{
2
3       private static double eps;
4       private static int minPoints;
5       public void configure(JobConf job) {
6       eps = Double.parseDouble(job.get("eps"));
7       minPoints = Integer.parseInt(job.get("minPoints"));
8       super.configure(job);
9       }
10      public void reduce(Text key, Iterator<Text> value, OutputCollector<Text, Text>
            arg2, Reporter arg3) throws IOException {
11      String[] line;
12      String cluster;
13      ArrayList<String> clustersToMerge = new ArrayList<String>();
14      while(value.hasNext()){
15          clustersToMerge.add(value.next().toString());
16      }
17      for(int i = 0; i < clustersToMerge.size(); i++){
18          line = clustersToMerge.get(i).split(" ");
19          if(dbscan.mergeDBScan(line[0], line[1], eps, minPoints)){
20              for (int j = i+1; j < clustersToMerge.size(); j++) {
21                  if(clustersToMerge.get(j).startsWith(line[1]+" ")){
22                      cluster = clustersToMerge.get(j);
23                      cluster = cluster.replaceAll(line[1]+" ",line[0]+" ");
24                      clustersToMerge.set(j, cluster);
25                  }
26                  else if(clustersToMerge.get(j).endsWith(" "+line[1])){
27                      cluster = clustersToMerge.get(j);
28                      cluster = cluster.replaceAll(" "+line[1]," "+line[0]);
29                      clustersToMerge.set(j, cluster);
30                  }
31              }
32          }
33      }
34  }
35  }
```

Algoritmo 1.4. Segundo MapReduce-Verificação de Merge dos clusters candidatos

A segunda fase de MapReduce tem a função Map como identidade, porém a função Reduce apresentada no Algoritmo 1.4 é paralelizada de acordo com o número de representantes distintos de canditados a merge recebidos como chave. Tal algoritmo invoca o Algoritmo 1 para pares de clusters candidatos a merge e verifica se realmente ocorre merge. A atualização dos identificadores de clusters onde ocorre *merge* também é feita na lista de candidatos a *merge* do Algoritmo 1.4.

Algoritmo 1: MergeDBScan

Entrada: $C_i, C_j, eps, minPoints$
Saída: $ocorreMerge$

```
1  adj[][] ⟵ falso;
2  enquanto k ≤ C_i.tamanho faça
3      p_i ⟵ C_i[k];
4      enquanto w ≤ C_j.tamanho faça
5          p_j ⟵ C_j[w];
6          se distancia(p_i, p_j) ≤ eps então
7              p_i.vizinhanca ⟵ p_i.vizinhanca + 1;
8              p_j.vizinhanca ⟵ p_j.vizinhanca + 1;
9              adj[i][j] ⟵ true;
10         fim se
11         w ⟵ w + 1;
12     fim enqto
13     se p_i.vizinhanca ≥ minPoints ∧ ¬p_i.ehCore então
14         p_i.ehCore ⟵ true;
15     fim se
16     k ⟵ k + 1;
17 fim enqto
18 enquanto w ≤ C_j.tamanho faça
19     p_j ⟵ C_j[w];
20     se p_j.vizinhanca ≥ minPoints ∧ ¬p_j.ehCore então
21         p_j.ehCore ⟵ verdadeiro;
22     fim se
23     w ⟵ w + 1;
24 fim enqto
25 enquanto k ≤ C_i.tamanho faça
26     p_i ⟵ C_i[k];
27     enquanto w ≤ C_j.tamanho faça
28         p_j ⟵ C_j[w];
29         se p_j.ehCore ∧ p_i.ehCore ∧ adj[i][j] então
30             ocorreMerge ⟵ verdadeiro;
31         fim se
32         updatePontosNoise();
33         w ⟵ w + 1;
34     fim enqto
35     k ⟵ k + 1;
36 fim enqto
37 renomeiaClusters();
38 retornaocorreMerge;
```

Da mesma forma que apresentado no Algoritmo 1.3, o objeto JobConf é construído no método principal para invocar as funções Map e Reduce presentes nessa segunda fase de MapReduce. Por ser semelhante, omitimos tal método.

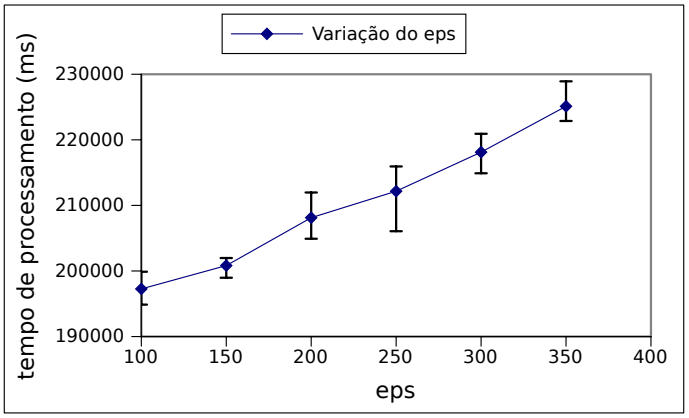

Figura 1.3. Variação do eps

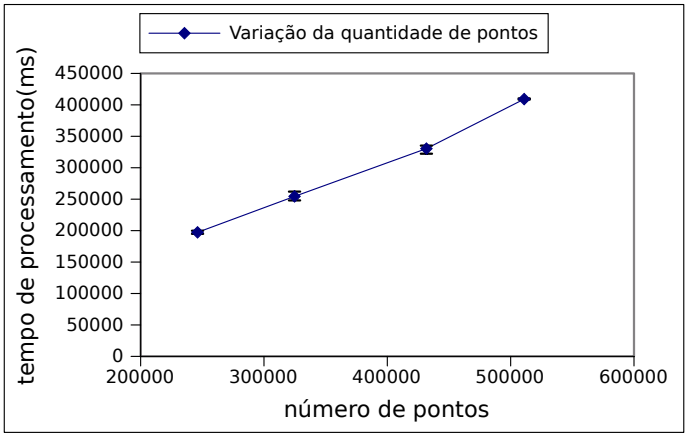

Figura 1.4. Variação da cardinalidade do conjunto de dados

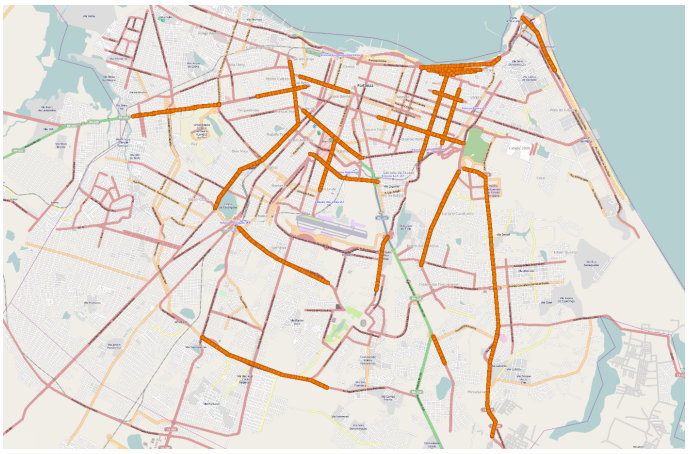

Figura 1.5. Um dos conjuntos de dados utilizados nos experimentos com 246142 pontos plotados

Figura 1.6. Clusters encontrados depois da fase de merge para 246142 pontos (eps=100 e minPoints=50)

1.4.6. Resultados da análise

O primeiro teste variou o valor de *eps* e manteve a mesma quantidade de pontos em cada rua utilizada, isto é, 246142 pontos. A variação de valores de *eps* foi 100,150,200,250,300 e 350, mantendo o valor de *MinPoints* como sendo 50. Como esperado, a Figura 1.3 mostra que com o aumento do *eps*, o tempo de processamento também aumenta, pois mais pontos na vizinhança de um *core point* podem ser expandidos.

A Figura 1.4 ilustra a variação do número de pontos do conjunto de dados de entrada versus o tempo de processamento. A quantidade de pontos usada durante os experimentos foram 246.142, 324.778, 431.698 e 510.952 pontos. Como esperado, quando a quantidade de pontos a serem processados pelo DBScan aumenta, o tempo de processamento também aumenta, mostrando que a nossa solução é escalável.

As Figuras 1.5 e 1.6 apresentam os 246.142 pontos plotados que pertencem ao conjunto de dados de entrada e os clusters encontrados após a aplicação do nosso método para *eps*=100 e *minPoints*=50, respectivamente. Na Figura 1.6, os clusters que sofreram o processo de merge estão com cor diferente de verde, note que o merge só ocorreu para os clusters ou ruas que se cruzam.

1.5. Desafios para Big Data

A utilização de Big Data apresenta diversas vantagens, mas também possui uma série de desafios que devem ser superados para sua ampla utilização. Diante do estudo realizado, alguns desafios para Big Data foram identificados. Estes desafios são oportunidades para trabalhos a serem desenvolvidos nesse contexto [Bertino et al. 2011] [QU et al. 2012].

Gartner afirmou que o principal desafio em Big Data está no volume dos dados gerados, seguido pelo desempenho do sistema e a escalabilidade necessária, além de lidar com o congestionamento na rede. Em [QU et al. 2012], são citados 3 aspectos desafiadores para o processamento em Big Data. Em primeiro lugar, Gerenciamento e Armazenamento para Big Data: as tecnologias atuais não são adequadas para satisfazer as necessidades de Big Data, um dos fatores é que a capacidade de armazenamento dos dis-

positivos cresce mais lentamente que o crescimento da quantidade de dados. Além disso, os algoritmos de gerenciamento não são eficazes para trabalhar com heterogeneidade de dados. Os processos de análise tradicionais esperam, em geral, dados homogêneos. Dessa forma, os dados devem ser estruturados como primeiro passo da análise. O processo de representação, acesso e análise eficiente de dados semiestruturados, que fazem parte do contexto de Big Data, tornam este processo mais complexo. É necessário lidar com erros e incompletude nos dados [Bertino et al. 2011].

Em segundo lugar, Computação e Análise para Big Data: no processamento de consultas em Big Data, a velocidade é fator significante, visto que não é possível percorrer todos os dados de uma base de dados Big Data em pequeno intervalo de tempo. Por exemplo, para descobrir fraude em uma transação de um cartão de crédito, é necessário detectá-la antes que a transação seja completada. Adicionalmente, a utilização de índices não é uma escolha apropriada para contornar o problema, já que estes são mais adequados para dados de tipo simples, ao passo que, em geral, Big Data apresenta estruturas de dados complexas. Assim sendo, a estratégia de paralelizar o processamento com o objetivo de diminuir o tempo de processamento é uma solução viável tratar este problema em Big Data. Se a aplicação é paralelizada, é possível aproveitar a infraestrutura dos provedores de nuvem composta por milhares de computares utilizados sob demanda e com baixo custo.

O terceiro desafio destacado em [QU et al. 2012] é a Segurança em Big Data: o volume dos dados cresce exponencialmente e acarretam grandes desafios no monitoramento e na segurança destes dados. Os métodos de segurança tradicionais devem ser repensados, pois deve ser possível realizar o processo de mineração dos dados para Big Data sem expor informações confidenciais de usuários. As tecnologias atuais para proteção da privacidade são baseadas principalmente no conjunto de dados estáticos, enquanto os dados em Big Data são alterados dinamicamente, incluindo padrão de dados, variação do atributo e adição de novos dados. Muitos dos serviços online atualmente requerem compartilhamento de informações privadas (como em aplicações como facebook), porém é importante estar atento a quem controla essa informação compartilhada e como ela pode ser usada para o casamento com outras informações. Assim, é um desafio alcançar de maneira eficaz a proteção da privacidade dos usuários neste contexto.

Um desafio citado por [Bertino et al. 2011] é a questão da escalabilidade. No passado, o objetivo era ter processadores cada vez mais rápidos, e prover os recursos necessários para lidar com o crescimento do volume de dados. Contudo, atualmente o volume de dados está aumentando em uma velocidade maior do que os recursos computacionais. Outra questão é a colaboração humana, alcançada atualmente por meio de *crowd-sourcing*, sendo a Wikipédia o melhor exemplo disso. É necessário, nesse contexto, lidar também com informações erradas por terem sido postadas por usuários mal intencionados.

Em [Fan et al. 2013], são citados os desafios apenas para análise em Big Data. Eles são decorrentes da alta dimensionalidade dos dados e pelo seu volume. O fato de haver várias dimensões pode ocasionar problemas, tais como o acúmulo de *outliers* e correlações incorretas dos dados. Alta dimensionalidade combinada com o grande volume de dados ocasiona outros problemas como, por exemplo, alto custo computacional e al-

goritmos complexos. O grande volume de dados é obtido a partir de múltiplas fontes, em diferentes pontos de tempo, utilizando diferentes tecnologias. Dessa forma, os problemas de heterogeneidade, de variações experimentais e estatísticas, nos obrigam a desenvolver procedimentos robustos e mais adaptáveis.

Em relação ao gerenciamento de grandes infraestruturas para armazenar e processar os dados em nuvem, faz-se necessário utilizar soluções eficientes para a gestão destas infraestruturas. Apesar da abordagem Hadoop apresentar bons resultados, a sua utilização é complexa e não existe uma relação entre as tarefas realizadas e o nível gerencial, por exemplo, garantir que o processamento seja realizado dentro de determinado intervalo. Assim sendo, um desafio é construir sistemas que facilitem a gestão de grandes infraestruturas e que permitam utilizar informações gerenciais. Nesta direção, a versão do Hadoop YARN aumenta o poder dos clusters Hadoop, melhorando a escalabilidade e a utilização de recursos baseados em SLAs [Hortonworks 2013].

1.6. Conclusão

A cada dia, um grande volume de dados é gerado por diferentes sistemas e dispositivos. Estes dados, chamados de Big Data, precisam ser armazenados e processados e podem ser analisados para obter tendências, por exemplo. Contudo, realizar estas tarefas não é simples, principalmente devido ao volume, velocidade e variedade. No tocante a análise para Big Data, os sistemas de gerenciamento de dados tradicionais não são adequados para lidar com uma escala tão grande. Algumas propostas exploram o paralelismo por meio de um grande número de máquinas para explorar o poder de armazenamento e computação. Neste contexto, a computação em nuvem fornece um ambiente escalável, elástico e com pagamento baseado no uso, o que permite construir soluções mais adequadas à gestão e análise de grandes volumes de dados.

Este trabalho apresentou os principais aspectos de Big Data, destacando a análise de dados. Foi apresentada a infraestrutura para análise em Big Data, com foco no Hadoop. Também foram apresentados algoritmos de mineração adaptados para Big Data e boas práticas para a concepção de novos algoritmos, assim como um estudo de caso em ambientes de computação em nuvem.

Por fim, foram discutidos alguns desafios importantes para Big Data, tais como armazenamento, processamento, análise, segurança, gerenciamento de grandes infraestruturas. Apesar de algumas soluções atenuarem estes desafios, ainda existe muito a fazer, principalmente porque o volume de dados e a velocidade com que estes dados são gerados cresce de forma exponencial. Estes desafios geram oportunidades de pesquisa, de forma que a análise para Big Data possa ser utilizada pelas empresas, melhorando seus produtos, serviços e a qualidade de vida de todos.

Referências

[Agrawal et al. 2011] Agrawal, D., Das, S., and El Abbadi, A. (2011). Big data and cloud computing: current state and future opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 530–533, New York, NY, USA. ACM.

[Agrawal et al. 1994] Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining asso-

ciation rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499.

[Arthur and Vassilvitskii 2007] Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics*, pages 1027–1035.

[Bahmani et al. 2012] Bahmani, B., Moseley, B., Vattani, A., Kumar, R., and Vassilvitskii, S. (2012). Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7):622–633.

[Begoli and Horey 2012] Begoli, E. and Horey, J. (2012). Design principles for effective knowledge discovery from big data. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, pages 215–218. IEEE.

[Bentley 1975] Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. In *Communications of the ACM*, volume 18, pages 509–517. ACM.

[Bertino et al. 2011] Bertino, E., Bernstein, P., Agrawal, D., Davidson, S., Dayal, U., Franklin, M., Gehrke, J., Haas, L., Halevy, A., Han, J., et al. (2011). Challenges and opportunities with big data.

[Coelho da Silva 2013] Coelho da Silva, T. L. (2013). Processamento elástico e não-intrusivo de consultas em ambientes de nuvem considerando o sla. *Dissertação de Mestrado - Universidade Federal do Ceará*, page 55.

[Cohen et al. 2009] Cohen, J., Dolan, B., Dunlap, M., Hellerstein, J. M., and Welton, C. (2009). Mad skills: new analysis practices for big data. *Proceedings of the VLDB Endowment*, 2(2):1481–1492.

[Cover and Hart 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27.

[Cryans et al. 2010] Cryans, J.-D., Ratte, S., and Champagne, R. (2010). Adaptation of apriori to mapreduce to build a warehouse of relations between named entities across the web. In *Advances in Databases Knowledge and Data Applications (DBKDA), 2010 Second International Conference on*, pages 185–189. IEEE.

[Dai and Lin 2012] Dai, B.-R. and Lin, I.-C. (2012). Efficient map/reduce-based dbscan algorithm with optimized data partition. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 59–66. IEEE.

[Dean and Ghemawat 2008] Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[Ester et al. 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231.

[Fan et al. 2013] Fan, J., Han, F., and Liu, H. (2013). Challenges of big data analysis. *arXiv preprint arXiv:1308.1479*.

[Ghoting et al. 2011] Ghoting, A., Kambadur, P., Pednault, E., and Kannan, R. (2011). Nimble: a toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 334–342. ACM.

[Hammoud 2011] Hammoud, S. (2011). Mapreduce network enabled algorithms for classification based on association rules.

[He et al. 2010] He, Q., Zhuang, F., Li, J., and Shi, Z. (2010). Parallel implementation of classification algorithms based on mapreduce. In *Rough Set and Knowledge Technology*, pages 655–662. Springer.

[He et al. 2011] He, Y., Tan, H., Luo, W., Mao, H., Ma, D., Feng, S., and Fan, J. (2011). Mr-dbscan: An efficient parallel density-based clustering algorithm using mapreduce. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 473–480. IEEE.

[Herodotou et al. 2011] Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B., and Babu, S. (2011). Starfish: A self-tuning system for big data analytics. In *CIDR*, volume 11, pages 261–272.

[Hortonworks 2013] Hortonworks (2013). *Hadoop Yarn*. `http://hortonworks.com/hadoop/yarn/`.

[Ji et al. 2012] Ji, C., Li, Y., Qiu, W., Jin, Y., Xu, Y., Awada, U., Li, K., and Qu, W. (2012). Big data processing: Big challenges and opportunities. *Journal of Interconnection Networks*, 13(03n04).

[Kim and Shim 2012] Kim, Y. and Shim, K. (2012). Parallel top-k similarity join algorithms using mapreduce. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 510–521. IEEE.

[Li et al. 2008] Li, H., Wang, Y., Zhang, D., Zhang, M., and Chang, E. Y. (2008). Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 107–114. ACM.

[Li and Zhang 2011] Li, L. and Zhang, M. (2011). The strategy of mining association rule based on cloud computing. In *Business Computing and Global Informatization (BCGIN), 2011 International Conference on*, pages 475–478. IEEE.

[Lin and Dyer 2010] Lin, J. and Dyer, C. (2010). Data-intensive text processing with mapreduce. *Synthesis Lectures on Human Language Technologies*, 3(1):1–177.

[Lu et al. 2012] Lu, W., Shen, Y., Chen, S., and Ooi, B. C. (2012). Efficient processing of k nearest neighbor joins using mapreduce. *Proceedings of the VLDB Endowment*, 5(10):1016–1027.

[Mell and Grance 2009] Mell, P. and Grance, T. (2009). The nist definition of cloud computing. *NIST*, page 50.

[Michie et al. 1994] Michie, D., Spiegelhalter, D. J., and Taylor, C. C. (1994). Machine learning, neural and statistical classification.

[Panda et al. 2009] Panda, B., Herbach, J. S., Basu, S., and Bayardo, R. J. (2009). Planet: massively parallel learning of tree ensembles with mapreduce. *Proceedings of the VLDB Endowment*, 2(2):1426–1437.

[Pavlo et al. 2009] Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., and Stonebraker, M. (2009). A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 165–178, New York, NY, USA. ACM.

[QU et al. 2012] QU, W., LI, K., XU, Y., AWADA, U., JIN, Y., QIU, W., LI, Y., and JI, C. (2012). Big data processing: Big challenges and opportunities. *Journal of Interconnection Networks*, 13(03n04):1250009.

[Quinlan 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

[Rajaraman and Ullman 2012] Rajaraman, A. and Ullman, J. D. (2012). *Mining of massive datasets*. Cambridge University Press.

[Ribeiro Jr et al. 2012] Ribeiro Jr, S. S., Rennó, D., Gonçalves, T. S., Davis Jr, C. A., Meira Jr, W., and Pappa, G. L. (2012). Observatório do trânsito: sistema para detecção e localização de eventos de trânsito no twitter. *SBBD*, pages 81–88.

[Riondato et al. 2012] Riondato, M., DeBrabant, J. A., Fonseca, R., and Upfal, E. (2012). Parma: a parallel randomized algorithm for approximate association rules mining in mapreduce. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 85–94. ACM.

[Schadt et al. 2010] Schadt, E. E., Linderman, M. D., Sorenson, J., Lee, L., and Nolan, G. P. (2010). Computational solutions to large-scale data management and analysis. *Nature reviews. Genetics*, 11(9):647–657.

[Silva 2004] Silva, M. (2004). Mineração de dados-conceitos, aplicações e experimentos com weka. *Livro da Escola Regional de Informática Rio de Janeiro-Espírito Santo. Rio Janeiro: SBC*.

[Sousa et al. 2010] Sousa, F. R. C., Moreira, L. O., Macêdo, J. A. F., and Machado, J. C. (2010). Gerenciamento de dados em nuvem: Conceitos, sistemas e desafios. In *SBBD*, pages 101–130.

[Suciu 2013] Suciu, D. (2013). Big data begets big database theory. In *BNCOD*, pages 1–5.

[White 2012] White, T. (2012). *Hadoop: the definitive guide*. O'Reilly.

[Wu et al. 2008] Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Philip, S. Y., et al. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37.

[Wu et al. 2013] Wu, X., Zhu, X., Wu, G.-Q., and Ding, W. (2013). Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints):1.

[Yang et al. 2010] Yang, X. Y., Liu, Z., and Fu, Y. (2010). Mapreduce as a programming model for association rules algorithm on hadoop. In *Information Sciences and Interaction Sciences (ICIS), 2010 3rd International Conference on*, pages 99–102. IEEE.

[Zhou et al. 2012] Zhou, L., Wang, H., and Wang, W. (2012). Parallel implementation of classification algorithms based on cloud computing environment. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 10(5):1087–1092.