ANDRÉ CAMILO BOLINA

AVALIAÇÃO DO FRAMEWORK MAPREDUCE PARA PARALELIZAÇÃO DO ALGORITMO APRIORI

LAVRAS – MG

2013

ANDRÉ CAMILO BOLINA

AVALIAÇÃO DO FRAMEWORK MAPREDUCE PARA PARALELIZAÇÃO DO ALGORITMO APRIORI

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso para a obtenção do título de Bacharel em Ciência da Computação.

Orientador
Profa. Dra. Marluce Pereira Rodrigues

Co-Orientador
Prof. Dr. Ahmed Ali Abdalla Esmin

LAVRAS – MG

2013

ANDRÉ CAMILO BOLINA

AVALIAÇÃO DO FRAMEWORK MAPREDUCE PARA PARALELIZAÇÃO DO ALGORITMO APRIORI

Monografia apresentada ao Colegiado do Curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

APROVADA em 15 de Abril de 2013.

Prof. Dr. Denilson Alves Pereira	UFLA
Prof. Dr. Cristiano Leite de Castro	UFLA

Profa. Dra. Marluce Pereira Rodrigues
(Orientador)

Prof. Dr. Ahmed Ali Abdalla Esmin
(Co-Orientador)

LAVRAS – MG
2013

Dedico este trabalho e a conclusão de meu curso aos meus pais Nélio e Cibele.

AGRADECIMENTOS

Agradeço aos meus pais Nélio e Cibele por tanta dedicação e preocupação. Obrigado por me colocarem a frente de vocês mesmos!
Aos meus irmãos César e Lucas, minha cunhada Karina e meus sobrinhos Tiago e Henrique pela inspiração e exemplos.
A minha namorada Heloísa, pela paciência, amor e força durante toda realização deste trabalho.
Aos amigos de Lavras: Francisco, Luiz, Igor, Pig, Ronan, Pablo, Bruno, Matheus, Luara, Flávia, Taísa, Dona Nena e Seu Marco.
A quem torceu ou me ajudou de alguma forma a realizar este trabalho, muito obrigado a todos vocês!

Um pouco de ciência nos afasta de Deus.

Muito, nos aproxima.

(Louis Pasteur)

RESUMO

A mineração de padrões frequentes é uma área da computação de ampla utilização. Seu objetivo é encontrar padrões de informações relevantes em grandes quantidades de dados. Porém, os principais algoritmos para mineração de padrões frequentes possuem alto tempo de execução, visto o grande volume de dados com que trabalham. Sendo assim, a programação paralela e distribuída e os frameworks de paralelização de algoritmos são uma boa alternativa para reduzir o tempo de execução necessário para processar as aplicações. Este trabalho propõe a implementação paralela e distribuída do algoritmo Apriori, bastante conhecido na área da mineração de padrões frequentes, utilizando para isso o Framework MapReduce. Os resultados são comparados com o algoritmo DMTA (Distributed Multithread Apriori), que também executa o algoritmo Apriori de forma paralela e distribuída, mas utilizando as bibliotecas MPI e OpenMP para criar e gerenciar processos e threads.

Palavras-Chave: Processamento Paralelo e Distribuido; Mineração de Dados; Apriori; MapReduce;.

ABSTRACT

The frequent-patterns mining is an area of extensive use in computing, its your objective is to find information about relevant patterns in large amounts of data. But the main algorithms for frequent-patterns mining have a high execution time, due to the large volume of data they work with. Therefore, parallel programming and frameworks that use this concept seem a good solution to reduce the execution time and level of computing required by these algorithms. This work proposes the parallel and distributed implementation of the Apriori algorithm, well known in the research area of frequent-patterns mining, using MapReduce Framework. The results were compared with the DMTA algorithm (Distributed Multithread Apriori), which also implements the Apriori algorithm in distributed and parallel, but using MPI and OpenMP libraries to create and manage processes and threads.

Keywords: Parallel Programing; Data Mining; Apriori; MapReduce;.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

1.1 Contextualização e Motivação

Com a disseminação e popularização do uso de sistemas computacionais, o aumento das ferramentas de coleta automática de dados e o amadurecimento das tecnologias de banco de dados, houve um grande crescimento na quantidade de dados nesses bancos. Com isso, a capacidade humana de interpretar e examinar em tão grande volume é superada, o que gera, segundo (FAYYAD *et al.*, 1996), a necessidade de novas ferramentas e técnicas para a análise automática e inteligente de banco de dados.

A partir dessa necessidade surgiu a Mineração de Dados (Data Mining), núcleo do processo amplo de KDD (Descoberta de Conhecimento em Bases de Dados, do inglês Knowledge Discovery in Databases), definida como a aplicação de algoritmos para extração de conhecimento em bases de dados. A aplicação desses algoritmos a grandes bases de dados torna-se uma tarefa computacionalmente dispendiosa, surgindo a necessidade de ferramentas que tornem a execução cada vez mais rápida.

Além disso, o campo da análise de dados é requerido em praticamente todas as aplicações computacionais, seja na fase de desenvolvimento de ferramentas, de solução de problemas ou na aplicação das ferramentas desenvolvidas. Dependendo da disponibilidade de modelos apropriados para os fenômenos responsáveis pela produção dos dados, a análise de dados pode ser exploratória (formulação de hipóteses e tomada de decisão) ou confirmatória (validação de modelos).

A Figura 1.1 busca ilustrar o conhecimento existente nos bancos de dados e que pode subsidiar a tomada de decisão. A partir dessa visão fez-se necessário criar uma nova geração de métodos e técnicas capazes de auxiliar o ser humano a buscar

conhecimento útil nas bases de dados. A Mineração de Dados ou Data Mining surgiu como uma das principais soluções para auxiliar o processo de Descoberta de Conhecimento em Bases de Dados ou KDD.

Figura 1.1: Piramide de evolução dos dados (NAVEGA, 2002)

Na base do triângulo estão os dados, os quais tomam o maior volume da memória do computador, e oferecem pouca utilidade estratégica na hora de se tomar decisões. A partir dos dados é possível obter muita informação através de aplicativos desenvolvidos para fins específicos ou através de ferramentas dos Sistemas Gerenciadores de Banco de Dados (SGBD) que exigem conhecimento por parte do analista para se obter o máximo proveito do grande volume de dados disponíveis e crescentes.

A partir das informações provenientes do tratamento dos dados é possível extrair e tratar um tipo de informação mais completa, o conhecimento, normalmente em menor quantidade que os dados e as informações, mas de maior inteligibilidade para auxiliar o processo de tomada de decisão. Finalmente, no topo do triângulo

da Figura 1.1, aparecem as decisões tomadas pelo homem com base no conhecimento obtido pelas ferramentas de Mineração de Dados.

Na Mineração de Dados, está presente a área de indução de regras de associação e padrões frequentes. Este processo teve origem na análise de cestas de mercadorias compradas pelos clientes (AGRAWAL *et al.*, 1993). Cada cesta é representada por uma transação e rotulada por um identificador, juntamente com os itens, que são os produtos comprados por um dado cliente. Pela análise desses dados, propõe-se encontrar tendências ou regularidades no comportamento das compras de clientes. Portanto, a prospecção de regras de associação torna-se útil para descobrir relacionamentos interessantes, que são muitas vezes desconhecidos por especialistas, em grandes quantidades de dados (TAN *et al.*, 2006).

Assim como em outras áreas da mineração de dados, na indução de regras de associação e padrões frequentes, mesmo que existam algoritmos que possam eficientemente encontrar padrões frequentes, o tempo de execução aumenta exponencialmente com o aumento do número de transações no banco de dados, isso devido a todo o banco de dados ser percorrido para verificar o suporte de cada itemset candidato. Ou seja, para calcular o suporte de cada itemset a base deve ser percorrida verificando a existência do mesmo em cada transação.

Para evitar isto alguns algoritmos propõe o armazenamento de dados em estruturas secundárias que auxiliem estes cálculos, porém ambos os casos geram novas consultas para cada itemset. Portanto a aplicação de computação paralela e distribuída é uma solução comumente utilizada, e o uso de novas tecnologias nesta área algo que deve ser estudado e aplicado, como o framework de paralelização de algoritmos, MapReduce, que está sendo amplamente utilizado para paralelizar algoritmos para executar grandes volumes de dados.

1.2 Objetivos

O objetivo principal deste trabalho é o estudo e implementação de um algoritmo paralelo baseado no Apriori (AGRAWAL; SRIKANT, 1994), que realize apenas a geração de conjuntos de itemsets, visto que esta é a parte de maior computação e processamento dentre as etapas algoritmo, utilizando o framework MapReduce, para comparação com outra implementação do Apriori que utilize de bibliotecas para paralelização.

O presente trabalho tem como objetivos específicos:

- Estudar os algoritmos de mineração de padrões frequentes baseados no Apriori existentes na literatura, que utilizam o framework MapReduce.
- Implementar o algoritmo que mais se destacar na etapa anterior.
- Avaliar o desempenho do algoritmo implementado comparando com um algoritmo paralelo baseado no Apriori que utilize outra técnica de paralelização.

1.3 Justificativas

As implementações de algoritmos para o processo de mineração de padrões, e mesmo demais processos da mineração de dados, que utilizam o processamento paralelo e/ou distribuído obtêm resultados visivelmente melhores que as implementações para o mesmo processo que utilizem do processamento sequencial. Isso ocorre porque o processamento que era realizado por apenas um núcleo de processamento na versão sequencial pode ser dividido entre vários núcleos utilizando a versão paralela e/ou distribuída.

As pesquisas que aplicam o processamento paralelo utilizam de diversas ferramentas para fazê-lo. Há na literatura um grande número de implementações que utilizam as bibliotecas MPI (MPI, 2012) e OpenMP (BARNEY, 2012) e que obtiveram êxito em acelerar o processo de mineração de padrões. Entretanto, recentemente várias pesquisas que se utilizam do framework MapReduce para o mesmo objetivo vem sendo realizadas.

Desta forma, uma análise dos algoritmos propostos na literatura que utilizaram o framework MapReduce permitirá identificar se estas implementações apresentaram bom desempenho, além de permitir identificar também as vantagens e desvantagem obtidas com relação ao uso desta ferramenta em comparação ao uso das bibliotecas MPI e OpenMP.

1.4 Organização

O restante deste trabalho está organizado da seguinte forma: na Seção 2 é apresentado o referencial teórico. A seção 3 apresenta detalhes do algoritmo DMTA, utilizado para comparação com o algoritmo proposto. A Seção 4 apresenta a metodologia do projeto com detalhes do ambiente, execuções e implementação. As Seções 6 e 7 apresentam os resultados encontrados e a conclusão, respectivamente.

2 REFERENCIAL TEÓRICO

2.1 Mineração de Dados

A Mineração de Dados, ou Data Mining, tem estado em ascensão nos últimos anos graças à larga disponibilidade de enormes quantidades de dados e à necessidade eminente de transformar esses dados em informação utilizável (HAN; KAMBER, 2001).

Segundo (SOUSA, 2010), minerar dados é vasculhar uma grande quantidade de dados à procura de padrões, regras de associação e grupos com similaridade, com o objetivo de obter conhecimento a cerca dos dados em questão.

Segundo (BRAGA, 2005), Mineração de Dados é o conjunto de métodos que provê, automaticamente, padrões em uma base de dados livre da tendenciosidade e limitação de uma análise baseada meramente na observação humana.

De acordo com (HAN; KAMBER, 2001), Knowledge Discovery in Databases, ou KDD, consiste numa sequência de passos que contém mineração de dados. Os passos de KDD são os seguintes:

- Limpeza de dados - Consiste na retirada de ruídos e dados irrelevantes. Essa fase é muito importante, pois esses dados retirados da base, além de tornar a mineração mais lenta, poderiam atrapalhar na aquisição de conhecimento (SILVA, 2000).

- Integração de dados - Algumas vezes, os dados, os quais se quer analisar, estão separados em diversas bases de dados. Assim, essa fase visa integrar todos os bancos de dados envolvidos em apenas um, a fim de facilitar o uso das ferramentas de mineração de dados.

- Seleção dos dados - Onde os dados relevantes à tarefa de análise são obtidos a partir do banco de dados.

- Mineração dos dados - Uma fase essencial, onde se seleciona os métodos a serem utilizados para localizar os padrões nos dados. Em seguida, ocorre a efetiva busca pelos padrões de interesse numa forma particular de representação ou conjunto de representações.

- Avaliação dos padrões encontrados - Nessa fase, é feita uma avaliação dos resultados da mineração dos dados, a fim de constatar que a mesma reconheceu padrões que são interessantes e/ou úteis.

- Apresentação do conhecimento - Visualização dos padrões encontrados na mineração para o usuário final.

Muitos autores divergem quanto às funcionalidades da mineração de dados, mas algumas funcionalidades sempre presentes são classificação, predição, agrupamento e associação. O foco deste trabalho é o estudo de algoritmos de regras de associação, portanto, as demais funcionalidades não serão abordadas.

2.2 Padrões Frequentes e Regras de Associação

Segundo (KUN-MING *et al.*, 2010), mineração de padrões frequentes é definida como segue. Suponha que $DB = \{T_1, T_2, ..., T_m\}$ seja um banco de dados transacional, no qual cada "T_i" (transação) consista em um subconjunto do conjunto de itens $I = \{i_1, i_2, ..., i_m\}$. Um *itemset* "$x$" é um subconjunto de itens de I. O suporte de um itemset "x" em um banco de dados DB, é denotado por $supDP(x)$, e consiste no número de transações no banco de dados que contêm "x". Formalmente, o suporte $supDP(x) = |(t|t \in DB \; e \; x \in t)|$. O problema de mineração de padrões

frequentes consiste em encontrar todos os itemsets "x" com $supDP(x) \geq s$, para um determinado limite "s", onde $|DB| \geq s \geq 1$.

As regras de associação, por sua vez, foram definidas em (AGRAWAL *et al.*, 1993) da seguinte forma. Sejam $I = \{i_1, i_2, ..., i_m\}$ um conjunto de m itens distintos e D uma base de dados formada por um conjunto de transações, onde cada transação T é composta por um conjunto de itens (itemset), tal que $T \subseteq I$. Uma regra de associação é uma expressão na forma $A \rightarrow B$, onde $A \subset I$, $B \subset I$, $A \neq \emptyset$, $B \neq \emptyset$ e $A \cap B = \emptyset$. O suporte de uma regra de associação $A \rightarrow B$, $SupDP(A \rightarrow B)$, é dado por $SupDP(A \cup B)$. Já a confiança desta regra, $Conf(A \rightarrow B)$, representa, dentre as transações que contêm A, a porcentagem de transações que também contêm B, ou seja, $Conf(A \rightarrow B) = SupDP(A \rightarrow B) \div SupDP(A)$. O problema da mineração de regras de associação consiste em encontrar todas as regras envolvendo os itemsets frequentes, que possuam confiança maior que um determinado valor.

Como exemplo da mineração de regras de associação, imagine que um programa de obtenção de conhecimento, depois de examinar milhares de candidatos ao vestibular de uma universidade particular, forneceu a seguinte regra: se o candidato é do sexo feminino, trabalha e teve aprovação com boas notas no vestibular, então não efetiva a matrícula. Uma reflexão justifica a regra oferecida pelo programa: de acordo com os costumes, uma mulher em idade de vestibular, se trabalha é porque precisa, e neste caso deve ter feito inscrição também em universidade pública gratuita. Se a candidata teve boas notas, provavelmente foi aprovada na universidade pública onde efetivará matrícula. Claro que há exceções, mas a grande maioria poderia obedecer à regra encontrada.

Dentre os principais algoritmos para mineração de padrões frequentes e regras de associação se encontra o Apriori (AGRAWAL; SRIKANT, 1994).

2.3 Apriori

O algoritmo Apriori (AGRAWAL; SRIKANT, 1994) é um dos algoritmos mais representativos na mineração de padrões frequentes e regras de associação. Sua ideia principal é baseada na observação de que se conjuntos de itens (itemset) são frequentes, a combinações destes pode gerar novos conjuntos que também sejam frequentes.

Na primeira etapa do algoritmo, conta-se o suporte dos itens individuais e determina-se quais deles são frequentes, ou seja, estão acima do suporte mínimo. Em cada passagem subsequente, utilizam-se conjuntos de sementes, sendo estes os conjuntos de itens frequentes encontrados na passagem anterior.

Usa-se as sementes para a geração de novos conjuntos de itens potencialmente frequentes, chamados de itens candidatos, e conta-se o suporte real para esses conjuntos de itens candidatos durante a passagem sobre a base de dados.

No final da passagem, pode-se determinar quais dos conjuntos de itens candidatos são realmente frequentes, e torna-se a semente para a próxima passagem. Este processo continua até que nenhum conjunto de itens frequentes novos sejam encontrados. Em seguida são verificadas as regras de associação e definidas as que estão acima da confiança minima desejada.

O fluxograma apresentado na Figura 2.1 busca ilustrar os passos anteriores, que descrevem a execução do algoritmo. Para descobrir conjuntos de itens frequentes, o algoritmo faz várias passagens sobre a base de dados.

Ainda para implementações sequenciais, o uso de técnicas diversas e aplicações de outras áreas da computação, como lógica Fuzzy, foram utilizadas para gerar otimizações ao algoritmo Apriori, como em (LI *et al.*, 2005), (HONGA *et al.*, 2004) e (PATEL *et al.*, 2011). Mesmo em (AGRAWAL; SRIKANT, 1994),

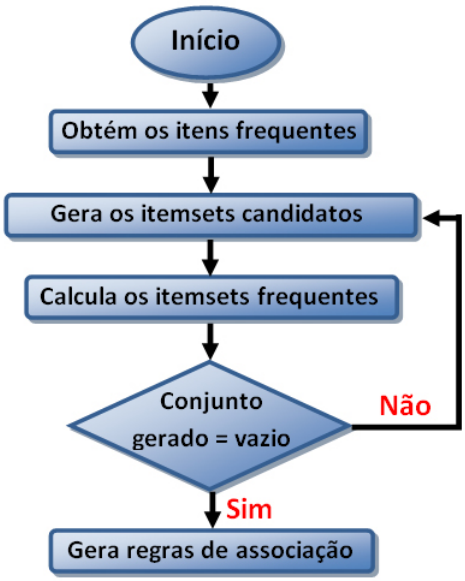

Figura 2.1: Fluxograma de execução do Apriori

por exemplo, é proposto a utilização de identificadores para transações, invertendo o sentido comum de transações e seus itens para os itens e as transações em que estão presentes.

Porém, com o avanço e popularização das plataformas *multicores*, o uso da computação paralela começou a ser utilizado para fins de otimização do algoritmo Apriori, como em (YE; CHIANG, 2006a), (YU; ZHOU, 2008) e (KUN-MING *et al.*, 2010). A implementação do DPA (KUN-MING *et al.*, 2010), ou Distributed Parallel Apriori, além da otimização através da paralelização, buscou otimizar também a busca de candidatos, evitando comparações com a base de dados a cada passagem. Esta implementação se destaca por alcançar bons resultados tanto no tempo de execução quanto no balanceamento de cargas.

2.4 Processamento Paralelo e Distribuído

O volume de dados existente hoje em dia é imenso, e tende a se tornar ainda maior devido ao fato da digitalização estar sendo utilizada em diversas áreas de conhecimento. Qualquer aplicação produzida armazena informações de seus usuários, de seu ambiente e de sua execução, na maioria das vezes sem se preocupar com o gasto de memória e processamento necessário para um futuro processamento ou manuseio destas informações.

Sendo assim, para analisar estes dados que, além de possuírem grande volume estão totalmente dispersos, sendo necessários meios que suportem grande quantidade de dados. Para aumentar sua capacidade e poder de processamento é necessário um ambiente, composto por diversas máquinas interligadas, que permita realizar computação paralela e distribuída. Onde computação paralela é a divisão de processamento dentro de uma mesma máquina e computação paralela é a divisão de processamento entre máquinas diferentes.

O uso de ambientes distribuídos está se tornando essencial para atender às necessidades dos usuários. Uma simples aplicação de grande porte necessita de centenas ou até milhares de servidores para suportar todas as solicitações dos clientes. A criação de sistemas dessa natureza é algo bastante complexo, pois é extremamente importante atender a todos os requisitos necessários de um ambiente distribuído, tal como a transparência, a segurança, a escalabilidade e a concorrência de componentes. Porém o desenvolvimento dessas aplicações é bastante complexo, pois demanda um alto conhecimento do funcionamento desses ambientes (FILHO, 2011).

Para o desenvolvimento de algoritmos paralelos e distribuídos, diversas bibliotecas e plataformas de programação e comunicação foram criadas, como o MPI (Message Passing Interface) (MPI, 2012), biblioteca que realiza a comunicação

entre processos, o OpenMP (BARNEY, 2012) que realiza e gerencia a criação e execução de threads. Existem também frameworks criados para auxiliar o processamento paralelo e distribuído, dentre estes está o MapReduce.

2.5 Framework MapReduce

O MapReduce é um modelo de programação que auxilia o desenvolvedor a implementar sistemas com transparência quanto ao processamento e tarefas realizadas. Trata-se de uma camada introduzida entre a aplicação desenvolvida e o ambiente computacional, de forma que o desenvolvedor não precise conhecer detalhes do ambiente para desenvolver suas aplicações.

Sua implementação utiliza o conceito de cluster de computadores, fornecendo um sistema escalar e com alto poder de processamento. Com este modelo é possível processar uma grande quantidade de dados em vários computadores. Todo o tratamento deve ser realizado pelo MapReduce, que irá manipular e gerenciar os dados em um conjunto de dispositivos.

Sua principal característica é a capacidade de separar os dados e distribuí-los junto ao código que deverá ser executado entre os diversos nós do sistema. Esse modelo possui raízes na programação funcional, utilizando o conceito de mapear (map) e reduzir (reduce), característica comum do paradigma funcional (FILHO, 2011).

Um exemplo típico onde os desenvolvedores começam suas atividades com MapReduce é o WordCount, que se destina a contar o número de ocorrências de cada palavra nos arquivos de entrada fornecidos. O seu funcionamento se dá através da operação de contagem de palavras, que ocorre em duas etapas, sendo uma fase de mapeador e uma fase de redutor.

Na fase mapeadora, primeiro a entrada é indexada em palavras, então forma-se pares de chave-valor com estas palavras, onde a chave é a palavra em si e o valor é igual a 1. Na fase de reduzir, as palavras são agrupadas em conjunto e os valores para as chaves semelhantes são adicionados. Isso daria o número de ocorrência de cada palavra na entrada. Assim, reduzir forma uma fase de agregação para as chaves. A Figura 2.2, de (SHARMA, 2012), ilustra este exemplo para uma entrada fictícia.

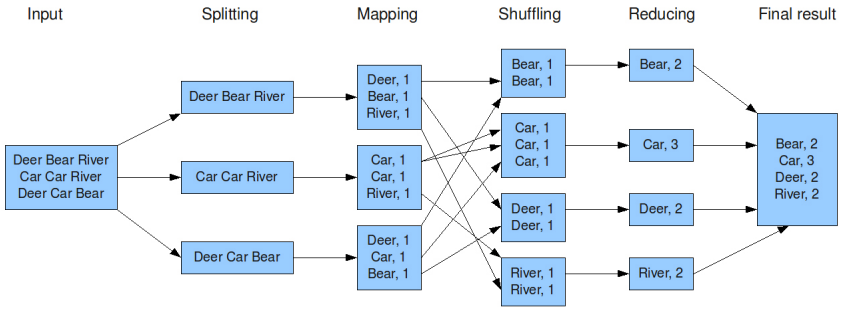

Figura 2.2: Ilustração do exemplo WordCount

Para o exemplo ilustrado pela Figura 2.2 o arquivo de entrada (Input) possui 3 linhas com 3 palavras cada, as linhas sofrem o processo de divisão (Splitting), em seguida cada palavra é mapeada (Mapping) e são criados os pares chave-valor, sendo cada palavra uma chave e o valor 1. Após isto os pares sofrem uma ordenação (Shuffling), onde os pares com chave semelhantes são agrupados juntamente e sofrem a redução (Reducing), onde para cada par de chaves iguais os valores são somados. Por fim os pares são novamente agrupados e é obtida a saída final (Final result).

Dentre as implementações do framework MapReduce se encontra a implementação do Hadoop MapReduce (APACHE, 2012). Esta implementação será a utilizada neste trabalho, pois ela faz parte do Apache Hadoop Project (APACHE, 2012), que conta também com implementações de diferentes componentes paralelos e distribuídos, como o HDFS e o HBase (APACHE, 2012), respectivamente

um sistema de arquivo e um sistema de banco de dados distribuídos que podem auxiliar as implementações a serem realizadas.

2.6 HDFS e HBase

O Hadoop Distributed File System (HDFS) (APACHE, 2012) é um sistema de arquivos distribuído projetado para rodar em hardware comum, faz parte do projeto Apache Hadoop, juntamente com o framework MapReduce. Utilizado por grandes empresas, como Adobe, IBM, ImageShack, LinkedIN e Facebook, ele possui conceitos como tolerância a falha de hardware, portabilidade, alta latência de acesso.

Um de seus princípios, por exemplo, é de que mover o processamento é mais barato do que mover os dados, isto porque quando um processamento é requisitado por uma aplicação é muito mais eficiente se ele acontece próximo aos dados em que opera, especialmente quando os dados são grandes. Mover o processamento minimiza o congestionamento da rede e aumenta a vazão do sistema.

A estrutura de ambiente gerada pela junção do MapReduce e do HDFS é apresentada abaixo pela Figura 2.3, de (HEDLUND, 2011).

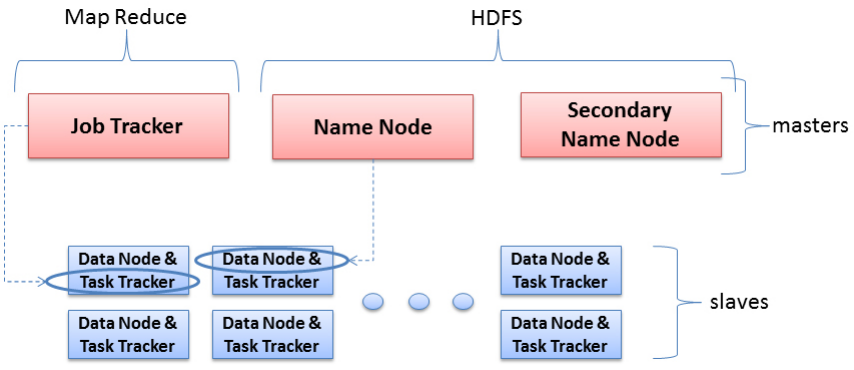

Figura 2.3: Estrutura do ambiente que possui o HDFS e o MapReduce.

A Figura 2.3 ilustra a divisão de funcionalidades entre máquina mestre (*master*) e máquinas escravas (*slaves*). O MapReduce inicia na máquina mestre o processo "Job Tracker", responsável por definir as tarefas de cada máquina do cluster, incluindo a própria máquina mestre. O HDFS por sua vez inicia na máquina mestre os processos "Name Node"e "Secondary Name Node", responsáveis por realizar a divisão e cópia dos arquivos dentre as máquinas integrantes do cluster, também incluindo a própria máquina mestre.

Além destes processos exclusivos da máquina mestre o MapReduce e o HDFS criam, respectivamente, os processos "Task Tracer"e "Data Node". Estes processos são responsáveis por realizar a execução das tarefas Map e Reduce e consultas e acessos a arquivos do sistema distribuído, em todas as máquinas integrantes do ambiente.

O HBase (APACHE, 2012) por sua vez é um banco de dados distribuído de código livre, também implementado pela Apache e que possui fácil integração ao framework MapReduce. Esta integração permite que o volume de dados trabalhado pelos algoritmos que rodam sobre o framework possa ser amplamente aumentado. Dentre as principais características deste banco de dados encontram-se as seguintes:

- Escalabilidade linear e modular;
- Leitura e escrita estritamente coerente;
- Compartilhamento automático e configurável de tabelas;
- Classes convenientes para apoio aos trabalhos que utilizem o Hadoop MapReduce;
- Fácil uso da API Java para acesso de clientes;

2.7 Trabalhos Relacionados

Mesmo que o algoritmo Apriori possa eficientemente encontrar padrões frequentes, o seu tempo de execução aumenta com o aumento do número de transações no banco de dados, isso devido a cada itemset candidato ser testado com todo o banco de dados. Cada itemset candidato com o mesmo tamanho pode ser testado de forma independente, permitindo que o algoritmo possa ser paralelizado.

Algumas pesquisas, portanto, aplicaram técnicas de computação paralela e distribuída para efetivamente acelerar o processo de mineração (AGRAWAL; SHAFER, 1996), (CHEUNG *et al.*, 1996), (CHEUNG *et al.*, 2002), (YE; CHIANG, 2006a). Porém, em um ambiente distribuído a irregularidade e desequilíbrio nas cargas de computação dos processadores podem fazer com que o desempenho global seja fortemente degradado. Logo, o equilíbrio de carga entre os processadores no processo de mineração é muito importante para a mineração paralela e distribuída. Para o algoritmo Apriori também já houve propostas de implementação paralela e distribuída (EINAKIAN; GHANBARI, 2006), (PARTHASARATHY *et al.*, 2001), (KUN-MING *et al.*, 2010), (WU *et al.*, 2011).

O framework MapReduce também vem sendo utilizado em implementações do Apriori, como (MING-YEN *et al.*, 2012), que propõe a implementação no método MapReduce para três diferentes implementações já testadas do algoritmo Apriori. O presente trabalho buscará soluções baseadas em estudos como este, que conseguiu adaptar o algoritmo aos paradigmas Map e Reduce, utilizando de algoritmos paralelos que já alcançaram bons resultados sem a utilização do framework, como a implementação do DMTA, para comparar sua eficiência.

Além de (MING-YEN *et al.*, 2012), trabalhos como (JEONG *et al.*, 2012), (CRYANS *et al.*, 2010) e (YANG *et al.*, 2010) foram estudados e utilizados para auxiliar na definição do tamanho e escolha das bases de dados utilizadas, forma

de implementação, valores de suporte, número de máquinas e outras decisões. No decorrer deste trabalho, informações do algoritmo implementado, do ambiente de execução ou de configurações do framework são acompanhadas das citações dos trabalhos que auxiliaram as respectivas decisões.

3 DMTA

O algoritmo utilizado para comparação com o algoritmo proposto neste trabalho é denominado DMTA (*Distributed Multithread Apriori*) e implementa a geração de conjuntos frequentes do algoritmo Apriori explorando a utilização de processos e threads.

Este algoritmo foi proposto e desenvolvido pelo mesmo autor desta monografia, porém no contexto de projeto de iniciação científica, e o artigo com os resultados obtidos está em fase de revisão para publicação na Revista de Sistemas de Informação da Faculdade Salesiana Maria Auxiliadora.

Com isto, ao se utilizar o algoritmo DMTA para comparação, podemos obter uma boa base para as comparações dos resultados e comportamentos, além de garantir o controle e igualdade do ambiente de execução. Características que não seriam possíveis ao realizar comparação com as demais implementações estudadas, visto que não são fornecidos códigos e que os ambientes de execuções são diferentes e dificilmente podem ser reproduzidos com exatidão.

Diferentemente do algoritmo proposto neste trabalho, que utiliza um framework para paralelização, o DMTA utiliza as bibliotecas MPI, responsável pela criação e gerenciamento de processos, e OpenMP, responsável pela criação e gerenciamento de threads. Além disto, o algoritmo DMTA realizou uma implementação do Apriori utilizando-se da estrutura de lista invertida.

O algoritmo DMTA está descrito na Subseção 3.1, representado na forma de uma entrada, uma saída e uma sequência de instruções divididas em 9 etapas.

3.1 O algoritmo DMTA

Entrada: Um banco de dados $DB = (T_0, T_1, ..., T_{n-1})$, onde $T_i = (i_0, i_1, ..., i_{m-1})$; um conjunto de processos P, em que P_0 é o processo mestre (MP) e os processos escravos (SP) vão de P_1 até P_p; um suporte mínimo S.

Saída: Todos os padrões frequentes presentes no DB.

Etapa 1: Cada processo lê o DB.

Etapa 2: Cada processo define as TIDs das transações e inverte o sentido inicial do DB, de transações com seus itens para itens e suas TIDs, armazenando o novo formato em uma estrutura de dados.

Etapa 3: Cada processo calcula o peso dos 1-itemsets candidatos e, se o valor é maior que o suporte mínimo, define como frequente, descartando os demais.

Etapa 4: Definir k=1, onde k é o número de itens associados por itemsets.

Etapa 5: Cada processo faz o cálculo das linhas (conjuntos) sob sua responsabilidade.

Etapa 6: Cada processo verifica os conjuntos atribuídos a ele, gera suas threads, divide os conjuntos entre elas, gerando os seus (k+1)-itemsets candidatos.

Etapa 7: Cada processo calcula o peso dos seus candidatos, define quais são frequentes, descartando os demais.

Etapa 8: Cada processo escravo envia o seu conjunto de (k+1)-itemsets frequentes ao processo mestre.

Etapa 9: Se não existem, ou existir apenas um (k+1)-itemsets frequentes, encerra-se o algoritmo; senão soma-se 1 a k, o processo mestre envia o conjunto dos itemsets frequentes aos demais processos e repete-se os passos 5-9.

Os resultados experimentais obtidos, disponíveis em http://dl.dropbox.com/u/39916215/DMTA.pdf, mostram que o DMTA obteve excelentes resultados, especialmente no caso de grandes volumes de dados e baixo suporte mínimo. Os resultados também mostram que o DMTA tem um bom equilíbrio de carga entre processos.

O algoritmo pode, assim, proporcionar uma estratégia útil e distribuída para problemas de mineração de dados. Portanto é uma escolha viável de comparação com o algoritmo proposto, que pretende alcançar resultados similares.

3.2 Exemplo de Execução

Suponha um ambiente com 2 máquinas (cada uma com 4 núcleos) e que sejam criados dois processos (cada um em uma máquina diferente). Considere também que há uma base de dados composta por 5 transações e 13 itens e um suporte de 15%.

Nas Etapas 1 e 2 do algoritmo DMTA, cada processo lê o banco de dados no formato horizontal (TID x Itens) da Figura 3.1(a), e o converte para o formato vertical (Itemsets x TIDs) da Figura 3.1(b).

TID	Itens
1	A C D G P T
2	A F L M O
3	B F C M
4	B C K S L T
5	A C O P B

(a)

→

Itemsets	TIDS
A	1 2 5
B	3 4 5
C	1 3 4 5
D	1
F	2 3
G	1
K	4
L	2 4
M	2 3
O	2 5
P	1 5
S	4
T	1 4

(b)

Figura 3.1: Banco de dados e leitura inicial

Na Etapa 3, cada processo calcula o peso de cada itemset (número de transações em que o itemset ocorre), ordena-os decrescentemente e descarta aqueles abaixo do suporte mínimo (15%), como apresentado na Figura 3.2.

Itemset	TID	Peso
C	1 3 4 5	4
A	1 2 5	3
B	3 4 5	3
F	2 3	2
L	2 4	2
M	2 3	2
O	2 5	2
P	1 5	2
T	1 4	2
D	1	1
G	1	1
K	4	1
S	4	1

→

Itemset	TID	Peso
C	1 3 4 5	4
A	1 2 5	3
B	3 4 5	3
F	2 3	2
L	2 4	2
M	2 3	2
O	2 5	2
P	1 5	2
T	1 4	2

Figura 3.2: Definição dos itemsets frequentes

A divisão de carga e tarefas proposta pelo DPA assume que os processos realizam a interseção entre conjuntos aproximadamente o mesmo número de vezes, para encontrar os itemsets frequentes. Porém a estrutura de dados que armazena os itemsets e as transações em que estão presentes, usada tanto para o DPA quanto para o DMTA, é ordenada decrescentemente para fins de divisão de carga, de forma que as primeiras linhas contenham os itens mais frequentes. Assim, o custo para realizar a interseção entre as primeiras linhas, atribuídas aos primeiros processos pelo DPA, é maior do que entre as últimas.

O algoritmo DMTA apresenta uma nova proposta de balanceamento, onde os dados são divididos de forma circular entre os processos, como pode ser visualizado na Figura 3.3 (Etapa 5).

O DMTA utiliza a biblioteca OpenMP para criar as threads em cada processo, dividindo ainda mais as cargas de trabalho e as comparações realizadas pelo algoritmo, reduzindo-se assim o tempo ocioso dos núcleos de processamento e ace-

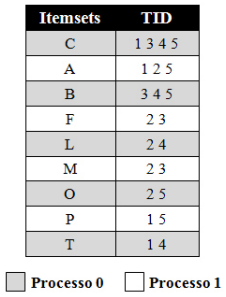

Figura 3.3: Divisão dos itemsets frequentes entre processos

lerando a execução A Figura 3.4 apresenta o escalonamento das threads para o exemplo (Etapa 6).

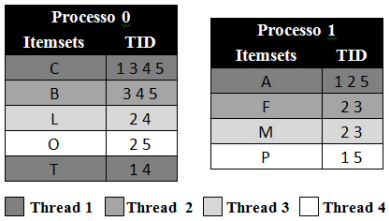

Figura 3.4: Escalonamento das threads

No escalonamento das threads foi utilizado o método “static“ (implementado pelo OpenMP) com “chunk” de tamanho 1, que realiza um balanceamento igual ao utilizado para processos. Foram testados outros tipos de escalonamento do OpenMP, como o “dynamic” e o “guided” com outros valores de “chunk”. Porém o utilizado foi o que apresentou melhor balanceamento de carga.

A definição do número de threads que serão criadas dentro de um processo é realizada em tempo de execução. Desta forma, é possível executar o algoritmo tanto em ambientes homogêneos (todas as máquinas com mesmo número de núcleos de processamento) quanto em ambientes heterogêneos (máquinas com diferentes números de núcleos de processamento).

O trecho que conta com a criação de threads é a parte de maior computação do algoritmo, no qual são realizadas as interseções das listas de TIDs dos itemsets buscando as TIDs comuns a dois conjuntos. A Figura 3.5 apresenta a interseção entre os conjuntos de 1-itemsets frequentes, C e A, para o exemplo.

Itemsets	TID	Peso
C	1 3 4 5	4
A	1 2 5	3

→

Itemsets	TID	Peso
C A	1 5	2

Figura 3.5: Interseção dos itemsets

Esta interseção é de responsabilidade da Thread 1 no Processo 0, conforme Figuras 4 e 5. A interseção também deve ocorrer entre o item C com todos os que estão abaixo dele na Figura 4, ou seja, CA, CB, CF, CL, CM, CO, CP, CT. É verificado o número de TIDs comuns entre eles e se este número gera um peso que atenda ao suporte estabelecido. O mesmo processo deverá ser repetido para os demais itemsets.

A Figura 3.6(a) apresenta o resultado do exemplo proposto para os conjuntos de 2-itemsets e a Figura 3.6(b) o resultado para os conjuntos de 3-itemsets.

Itemset	TID	Peso
CB	3 4 5	3
CA	1 5	2
CP	1 5	2
CT	1 4	2
AO	2 5	2
AP	1 5	2
FM	2 3	2

(a)

Itemset	TID	Peso
CAP	1 5	2

(b)

Figura 3.6: Conjuntos de itemsets frequentes

A estrutura de dados que armazena o resultado dos k-itemsets frequentes é compartilhada entre as threads do processo e o acesso a essa estrutura é realizado de forma atômica pelas threads.

4 METODOLOGIA

Neste trabalho, foi realizada uma pesquisa bibliográfica para obtenção de conhecimento acerca dos temas mineração de padrões frequentes, algoritmos paralelos baseados no Apriori, técnicas de processamento paralelo e distribuído e framework de paralelização MapReduce.

Foram estudados artigos relacionados a paralelização de algoritmos utilizando o framework MapReduce, especialmente os artigos que apresentam uma solução de paralelização do algoritmo Apriori usando este framework. Baseado no conhecimento obtido, foi escolhido implementar um algoritmo Apriori paralelo e distribuído que executasse sobre um ambiente distribuído implementado pelo projeto Apache Hadoop (APACHE, 2012).

Foi implementado o algoritmo Apriori MapReduce, como explicado nas seções seguintes. Para avaliar o desempenho do algoritmo implementado foram realizados experimentos com bases de dados amplamente utilizadas pela comunidade de mineração de dados, considerando bases reais e sintéticas. Os resultados obtidos foram comparados com o algoritmo DMTA, que utiliza outra técnica de paralelização do algoritmo Apriori.

A pesquisa realizada utilizou a metodologia quantitativa, cuja essência em Ciência da Computação é verificar o quão melhor é usar um programa/sistema novo frente à(s) alternativa(s) (WAINER, 2007). O estudo realizado utilizou de métricas para comparação do tempo de execução e *speedup*, além de propor novas melhorias utilizando os novos sistemas e ambientes existentes.

Os experimentos que foram realizados visaram mostrar:

- O tempo de execução que é gasto pelo framework para realizar inicializações do ambiente, mas que não depende da base de dados utilizada;

- Os tempo de execução e speedup para diferentes bases (sintética e real) para o MapReduce Apriori e para o DMTA;
- O balanceamento de carga apresentado pelo MapReduce Apriori;
- Vantagens e desvantagens da utilização do framework em relação a utilização de outras bibliotecas.

5 MRA (MAPREDUCE APRIORI)

O MRA, ou MapReduce Apriori, é o algoritmo proposto por este trabalho. Enquanto as demais implementações, como (AGRAWAL; SRIKANT, 1994) e o DMTA, precisam confrontar a base de dados ou uma estrutura de dados várias vezes para gerar seus resultados finais, o MRA realiza a geração dos itemsets candidatos em cada transação através da combinação dos itens 1 a 1, 2 a 2, 3 a 3 e assim por diante até que não haja mais combinações possíveis e depois realiza a contagem dos itemsets candidatos em todas as transações através do método Reduce. Dessa forma, não é necessário consultar a base de dados ou uma estrutura auxiliar várias vezes.

A forma de funcionamento do MRA foi proposta de maneira similar por (ZHAO; DU, 2012), porém ao invés de utilizar a base de dados distribuída HBase, ele utilizou a estrutura sequencial HashTable. O MRA foi implementado de forma similar ao algoritmo de código fonte livre disponível online em http://sourceforge.net/projects/apriorimapred.

O funcionamento do MRA, assim como os algoritmos apresentados em (LIN *et al.*, 2012) e (JEONG *et al.*, 2012) consiste em 2 etapas, a primeira etapa é responsável por encontrar os 1-itemsets frequentes, ou seja, os itens que individualmente superam o suporte mínimo estabelecido e a segunda etapa responsável por encontrar os k-itemsets, para k maior e igual a 2, ou seja, todas as combinações possíveis entre os itens e conjuntos existentes. As Figuras 5.1 e 5.2 buscam ilustrar o funcionamento destas etapas.

A Figura 5.1 apresenta o funcionamento da primeira etapa do MRA, onde a entrada é dividida em linhas (transações), em seguida a função Map é executada. Nesta função cada token (item) é armazenado como chave e atribuído o inteiro 1 como valor no par de valores <chave, valor> trabalhado pelo Framework. Após

isto o Framework executa uma classificação, posicionando os pares de mesma chave de maneira próxima, para que na função seguinte, Reduce, o algoritmo seja capaz de somar os valores dos pares com mesma chave de forma mais otimizada. O resultado final é armazenado no banco de dados distribuído HBase para uso na etapa seguinte.

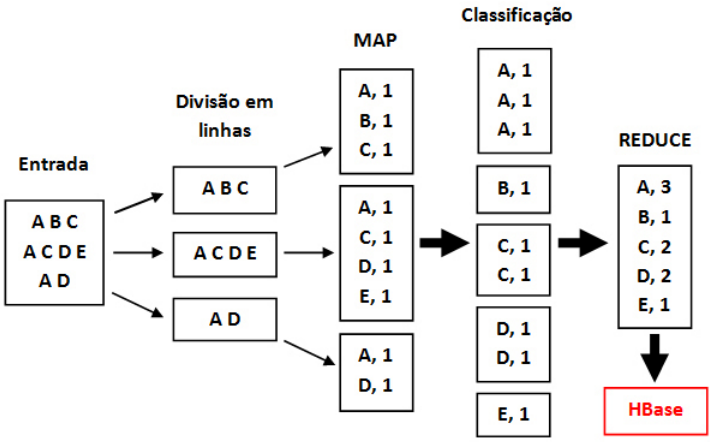

Figura 5.1: Ilustração da primeira etapa do MRA.

A Figura 5.2 apresenta o funcionamento da segunda etapa do MRA, nesta etapa a entrada é novamente dividida em linhas (transações), porém, com os dados da execução da etapa anterior, armazenados no banco de dados distribuído (HBase), é realizado uma poda e eliminado os itens que não possuem suporte acima do suporte mínimo das transações.

Em seguida a função Map é executada, no Map desta segunda etapa os itens de cada transação são utilizados para formarem as combinações possíveis, resultando nos conjuntos de itemsets candidatos. Após isto, os processos da etapa anterior se repetem e o Framework executa uma classificação, posicionando os pares de mesma chave de maneira próxima, para que na função seguinte, Reduce, o algoritmo seja capaz de somar os valores dos pares com mesma chave de forma mais otimizada.

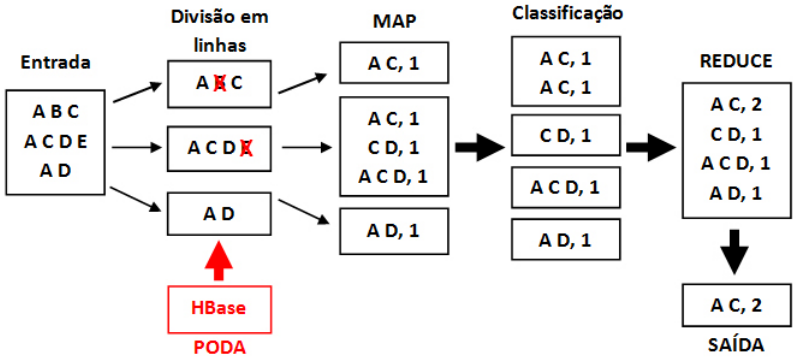

Figura 5.2: Ilustração da segunda etapa do MRA.

A saída desta etapa é verificada para calcular se o suporte dos conjuntos encontrados é superior ou igual ao suporte mínimo, e só então é escrito no arquivo de saída, juntamente com a saída da primeira etapa, formando a saída final do algoritmo com todos os conjuntos de itemsets frequentes.

O framework MapReduce aceita algumas linguagens de programação como Python, C e Java, mas devido principalmente a portabilidade, sintaxe similar a outras linguagens populares e ser a linguagem nativa do framework, a linguagem Java foi escolhida para utilização neste trabalho.

As etapas acima descritas foram implementadas em forma de código, na linguagem Java, respeitando os conceitos e propriedades do Framework. Em relação a outros algoritmos, como o DMTA, o código do MRA é bem menor e de mais simples sintaxe, isto porque não é necessário definir o que cada máquina irá fazer de maneira individual ou mesmo programar trechos de comunicação entre as máquinas.

Entretanto os algoritmos para o Framework devem possuir uma estrutura definida, formada por métodos e tipos de variáveis já estabelecidos e específicos, o que torna difícil a adaptação de algoritmos existentes ao paradigma, mas garante que os algoritmos adaptados alcancem o máximo desempenho.

Para que fique mais clara a programação para o framework, abaixo são apresentados os trechos do código que se referem as funções Map e Reduce da primeira etapa do MRA.

```
1  public static class map1Itemsets extends Mapper<LongWritable,
2                                                  Text, Text, Text> {
3     private Text item = new Text();
4     private Text valorUm = new Text();
5     public void map(LongWritable key,Text value,Context context)
6                                                        throws IOException {
7        String transacao = value.toString();
8        StringTokenizer itens = new StringTokenizer(transacao);
9        while (itens.hasMoreTokens()) {
10          item.set(itens.nextToken());
11          valorUm.set(Integer.toString(1));
12          context.write(item, valorUm);
13       }
14    }
15 }
```

O trecho acima se refere a função Map, nas linhas 3 e 4 encontram-se as declarações das variáveis "item" e "valorUm" que serão armazenadas no par <chave, valor>. Na linha 6 o método Map propriamente dito é declarado. Em seu interior a variável "transacao" recebe a linha do arquivo de entrada, que é automaticamente selecionada pelo framework. Esta variável é transformada numa sequência de tokens, onde cada token é um item. Para cada token, ou item, a variável de saída "context" recebe o par formado pela chave que é o item e a variável "valorUm" que possui armazenada o número inteiro 1.

Abaixo é apresentado o trecho referente a função Reduce da primeira etapa do algoritmo, seu número de linha em relação ao código do Map é maior devido a conexão com o banco de dados distribuído.

```
1  public static class reduce1Itemsets extends Reducer<Text, Text,
2                                                    Text, Text>
3  {
4     static Configuration configuracaoHBase;
5     static HTable tabela;
6     protected void setup(Context context) throws IOException,
7                                              InterruptedException {
8        configuracaoHBase =
9              HBaseConfiguration.create(context.getConfiguration());
10       tabela = new HTable(configuracaoHBase, "Saida");
11    }
12    protected void cleanup(Context context) throws IOException,
13                                             InterruptedException {
14       tabela.close();
15    }
16    public void reduce(Text key,Iterable<Text> values,
17                               Context context) throws IOException
18    {
19       Iterator<Text> valores = values.iterator();
20       int soma = 0;
21       Put linha = new Put(Bytes.toBytes(key.toString()));
22       while (valores.hasNext()) {
23          String valor = valores.next().toString();
24          soma += Integer.parseInt(valor);
25       }
26       linha.add(Bytes.toBytes("Itemset"),
27                 Bytes.toBytes(key.toString()),
28                         Bytes.toBytes(Integer.toString(soma)));
29       tabela.put(linha);
30    }
31 }
```

Nas linhas 3 e 4 são declaradas as varíaveis de tipos "Configuration"e "HTable", próprios do framework e do banco de dados distribuído. Estas varíaveis são

responsáveis pela conexão com o banco de dados e para realizar esta conexão são necessários alguns comandos, envolvendo estas varíaveis, e que devem ser executados em cada máquina.

Se estes comandos estivessem presentes nas funções Map ou Reduce seriam executados a cada transação, o que não é necessário e causaria aumento no tempo de execução. Se estivessem presentes na função Main seriam executados apenas pela máquina Mestre e as demais máquinas não conseguiriam utilizar a conexão.

Portanto a conexão com o HBase é realizada na função Setup e encerrada na função Cleanup, como visto no trecho do código acima. Estes métodos são pré-definidos pelo Framework e são executados por ele ao inicio e ao fim da execução do trecho em cada máquina.

Como a função Reduce da primeira etapa e a função Map da segunda etapa são os momentos em que dados são escritos e lidos no Banco de Dados, os métodos Setup e Cleanup são necessários apenas nestes trechos do algoritmo proposto.

Seguindo adiante no código apresentado, na linha 19 se inicia a declaração do método "reduce". Neste método o algoritmo proposto recebe automaticamente do framework um par com a chave, sendo esta o item, e uma lista de valores, sendo cada valor igual a 1 (como gerado no Map), representando cada transação em que aquela chave está presente.

Em seguida para cada valor desta lista é incrementada a variável "soma"com o próprio valor e ao fim colocado na tabela o Itemset e o valor da soma, representando o suporte mínimo do item.

Abaixo é apresentado também o trecho do código referente a segunda etapa do algoritmo, responsável pela geração de todos os k-itemsets frequentes, para "k" maior ou igual a 2.

```
1  public static class mapKItemsets extends Mapper<LongWritable,
2                                                  Text, Text, Text>
3  {
4     private Text itemset = new Text();
5     private Text supItemset = new Text();
6     static Configuration configuracaoHBase;
7     static HTable tabela;
8     protected void setup(Context context)
9                         throws IOException,InterruptedException {
10       configuracaoHBase =
11            HBaseConfiguration.create(context.getConfiguration());
12       tabela = new HTable(configuracaoHBase, "Saida");
13    }
14    protected void cleanup(Context context)
15                        throws IOException,InterruptedException {
16       tabela.close();
17    }
18    public void loopKItemsets(Vector<String> entrada,
19            Vector<String> saida, int tamanho, int k, int inicio,
20                              Context context) throws IOException {
21       int contador, total;
22       for(contador=inicio;contador<tamanho;contador++) {
23          if(k==0){
24             saida.add(entrada.get(contador));
25          } else {
26             saida.add(entrada.get(contador));
27             int init=1;
28             StringBuffer itemsetParc = new StringBuffer();
29             for(String item:saida)
30             {
31                if(init==1){
32                   itemsetParc.append(item);
33                   init=0;
34                } else {
```

```
35                  itemsetParc.append(" ");
36                  itemsetParc.append(item);
37               }
38            }
39            itemset.set(itemsetParc.toString());
40            supItemset.set(Integer.toString(1));
41            context.write(itemset, supItemset);
42         }
43         if(contador < tamanho-1) {
44            loopKItemsets(entrada, saida, tamanho,
45                                 k+1,contador+1, context);
46         }
47         total = saida.size();
48         if(total>0){
49            saida.remove(total-1);
50         }
51      }
52   }
53   public void map(LongWritable key,Text value,
54                            Context context) throws IOException
55   {
56      String transacao = value.toString();
57      StringTokenizer itens = new StringTokenizer(transacao);
58      Vector<String> itensFrequentes = new Vector<String>();
59      int contador=0;
60      while (itens.hasMoreTokens()) {
61         String item = itens.nextToken();
62         Get linha = new Get(Bytes.toBytes(item));
63         Result linhaSaida = tabela.get(linha);
64         byte [] valorSaida = linhaSaida.getValue(
65                Bytes.toBytes("Itemset"), Bytes.toBytes(item));
66         int supItem = Integer.parseInt(
67                                    Bytes.toString(valorSaida));
68         if(supItem >= suporteMinimo) {
69            itensFrequentes.add(item);
```

```
70                  contador++;
71              }
72          }
73          if(!itensFrequentes.isEmpty()) {
74              Vector<String> combinations = new Vector<String>();
75              loopKItemsets(itensFrequentes,combinations,
76                                              contador,0,0, context);
77          }
78      }
79  }
80
81  public static class reduceKItemsets extends Reducer<Text,
82                                                      Text, Text, Text>
83  {
84      public void reduce(Text key,Iterable<Text> values,
85                                  Context context) throws IOException
86      {
87          Iterator<Text> valores = values.iterator();
88          int soma = 0;
89          while (valores.hasNext()) {
90              String valor = valores.next().toString();
91              soma += Integer.parseInt(valor);
92          }
93          if (soma >= suporteMinimo) {
94              context.write(key, new Text(Integer.toString(soma)));
95          }
96      }
97  }
```

Como é possível visualizar no código a função Reduce da segunda etapa tem funcionamento semelhante a mesma função na primeira etapa. Já a função Map abrange mais cálculos e computação, incluindo dentro do trecho responsável pela mesma o método Map em si e um outro método, recursivo, para geração dos itemsets candidatos.

Para que o algoritmo possa de fato utilizar todos estas funções e métodos, acima relatadas, são necessárias configurações e definições iniciais apresentadas no seguinte trecho, e que devem ser realizadas na função *main* do código.

```
1  Job job1 = new Job(configuracao, "");
2  job1.setJarByClass(aprioriMR1.class);
3  job1.setMapperClass(map1Itemsets.class);
4  job1.setCombinerClass(reduce1Itemsets.class);
5  job1.setReducerClass(reduce1Itemsets.class);
6  job1.setInputFormatClass(TextInputFormat.class);
7  job1.setOutputFormatClass(TextOutputFormat.class);
8  job1.setMapOutputKeyClass(Text.class);
9  job1.setMapOutputValueClass(Text.class);
10 job1.setOutputKeyClass(Text.class);
11 job1.setOutputValueClass(Text.class);
12 FileInputFormat.addInputPath(job1, new Path(args[0]));
13 FileOutputFormat.setOutputPath(job1, new Path(args[1]));
14 Job jobK = new Job(configuracao, "");
15 jobK.setJarByClass(aprioriMR1.class);
16 jobK.setMapperClass(mapKItemsets.class);
17 jobK.setCombinerClass(reduceKItemsets.class);
18 jobK.setReducerClass(reduceKItemsets.class);
19 jobK.setMapOutputKeyClass(Text.class);
20 jobK.setMapOutputValueClass(Text.class);
21 jobK.setOutputKeyClass(Text.class);
22 jobK.setOutputValueClass(Text.class);
23 jobK.setInputFormatClass(TextInputFormat.class);
24 jobK.setOutputFormatClass(TextOutputFormat.class);
25 FileInputFormat.addInputPath(jobK, new Path(args[0]));
26 FileOutputFormat.setOutputPath(jobK, new Path(args[2]));
```

As configurações acima permitem, dentre outras coisas, definir que serão executados 2 Maps e 2 Reduces (1 Map e 1 Reduce para cada etapa descrita anteriormente), definir os formatos da entrada e saida destas funções, definir o nome da

classe responsável por estas funções e definir os arquivos de entrada e saída, neste caso informados via parâmetros na chamada do algoritmo.

6 EXPERIMENTOS E RESULTADOS

Para avaliar o desempenho do algoritmo MRA proposto, foram realizados os seguintes experimentos:

- Execução com diferentes bases de dados com número de itens pequenos, visando mostrar o tempo de execução que é gasto pelo framework para realizar inicializações do ambiente;
- Medida do tempo de execução e do speedup para diferentes bases (sintética e real) para o MapReduce Apriori e para o DMTA;
- Medida do tempo de execução de cada processo durante a execução do MRA, visando verificar o balanceamento de carga dos processos.

Os tempos de execução foram calculados no DMTA subtraindo-se os valores encontrados em uma função para obter o tempo do sistema no fim e no inicio do código. Para o MRA a mesma técnica foi utilizada, porém a interface gráfica do framework também disponibiliza esta informação.

6.1 Bases de dados

As bases de dados utilizadas nas execuções e suas características são apresentadas na Tabela 6.1. As bases estão no formato frequentemente utilizado em outros trabalhos, onde cada linha do arquivo representa uma transação e os itens presentes nesta transação estão dispostos na linha separados por tabulação.

A base MSNBC é uma base de dados real com acessos de usuários a páginas do site msnbc.com em 28 de Setembro de 1999. Esta base compõe um repositório online para aplicações em KDD (Descoberta de Conhecimento em Bases de

Tabela 6.1: Especificações das Bases de Dados

Nomeclatura	*Especificações*
T10I4D100KN100K	Base sintética: 100.000 transações, com média de 10 itens por transação e 100.000 itens possíveis.
MSNBC	Base real: 1.000.000 transações, com média de 2 itens por transação e 17 itens possíveis.

Dados, em português) da UCI (University of California, Irvine). O repositório foi utilizado nos trabalhos de (SONG *et al.*, 2013) e (VALENCIO *et al.*, 2011), e a base foi utilizada por (ZANG, 2010) e (IVANCSY; VAJK, 2006).

A base T10I4D100KN100K é uma base sintética, gerada pelo IBM Quest Syntactic Data Generator (ALMADEN, 1994). Segundo (ZHENG *et al.*, 2001) esta é uma base frequentemente utilizada na comunidade de pesquisa de regras de associação e além deste, (AGRAWAL; SRIKANT, 1994), que propuseram o algoritmo Apriori utilizaram-na, assim como (RAMARAJ; VENKATESAN, 2009), (LAN *et al.*, 2009), (YE; CHIANG, 2006b) que estudaram o Apriori e (LI *et al.*, 2012), (CRYANS *et al.*, 2010) e (YANG *et al.*, 2010), que realizaram a sua implementação utilizando o framework MapReduce.

6.2 Ambiente de execução

O ambiente de execuções e testes utilizado para as implementações é composto de quatro máquinas ligadas em rede com as configurações conforme apresentadas na Tabela 6.2. Esta quantidade de máquinas é suficiente para as execuções e estudos dos resultados, visto que trabalhos que utilizaram do mesmo framework, como

(MING-YEN *et al.*, 2012) e (CRYANS *et al.*, 2010), utilizaram, respectivamente, quatro e cinco máquinas.

Tabela 6.2: Configuração do ambiente de execução

Ambiente de Hardware	
CPU	AMD Athlon (TM)64 Processor 3500+ 2.200 MHz 512Kb Cache
Memória	1,3GB DDR RAM
Disco Rígido	5GB
Rede	10/100 Mbps Fast Ethernet
Ambiente de Software	
S.O.	Ubuntu 11.04 LTS
Apache	Hadoop MapReduce 1.0.4, HBase 0.94.4 OpenMP 4.0, MPICH 3.0.2

A instalação e configuração do Apache Hadoop MapReduce foi realizada conforme (NOLL, 2011) e a instalação e configuração do Apache HBase foram realizadas seguindo os passos disponíveis em (CHUANG, 2011).

Ao inicio das execuções, alguns testes causaram estouro de memória nas máquinas, isto já foi relatado por (CRYANS *et al.*, 2010), que possuía máquinas com maior espaço de armazenamento, mas que trabalhava com bases maiores. Para evitar isso, assim como em (CRYANS *et al.*, 2010), após as execuções os dados de saída e configurações eram apagados do sistema de arquivos distribuído.

Além disto, para que os dados fossem divididos entre as máquinas, as bases foram divididas manualmente em 2,3 ou 4 arquivos, de acordo com o número de máquinas que se desejava utilizar na execução.

Uma estratégia semelhante foi utilizada por (LI *et al.*, 2012) que multiplicou suas bases iniciais para aumentar o tamanho da base final, não afetando o desempenho os resultados do estudo.

Isto foi necessário pois o framework está apto a realizar o processo de divisão de dados automaticamente para bases muito maiores do que as utilizadas, considerando o seu potencial de processamento.

6.3 Bases iniciais

Para ilustrar o tempo inicial gasto pelo *Framework* para iniciar seus processos e realizar as configurações foram realizadas 5 execuções para cada base de dados, que variam de 10 a 500 transações, todas com 100 itens possíveis e média de 10 itens por transação.

Todas as execuções utilizaram 4 máquinas e o suporte mínimo igual a 0, ou seja, foram encontradas todas as combinações possíveis de itens (itemsets). Os tempos médios das execuções estão exibidos através de gráfico na Figura 6.1.

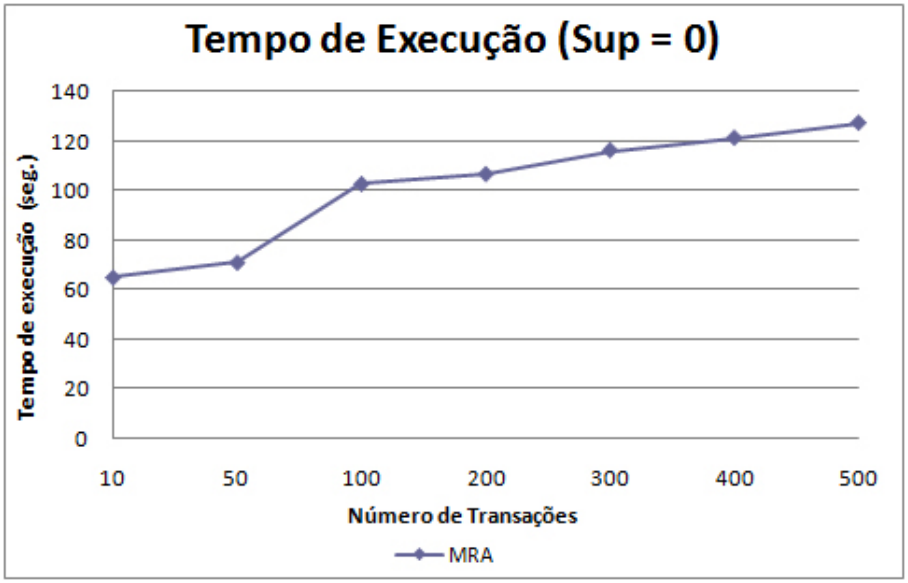

Figura 6.1: Tempo de execução para diferentes bases.

É possível perceber um aumento no tempo de execução com o aumento do número de transações, porém este aumento não é proporcional. Ao utilizar como

entrada uma base com 10 transações o tempo de execução é de cerca de 60 segundos, o que pode parecer alto, porém se utilizarmos uma entrada 5 vezes maior este tempo aumenta em apenas 10 segundos, o que indica que na verdade o tempo de processamento das 10 transações é uma pequena parcela do tempo total, que consiste no tempo necessário pelo Framework para identificar as máquinas, definir as tarefas e enviá-las às maquinas.

Utilizando uma base 50 vezes maior que a menor, ou seja, uma base de 500 transações o tempo apenas dobra, ou seja, enquanto a base apresenta um aumento de 50 vezes o tempo apresenta um aumento de 2 vezes, o que indica o alto poder de processamento do framework.

6.4 Base sintética

Em seguida, utilizando 4 máquinas, foram realizadas 5 execuções para cada valor de suporte mínimo utilizando a base sintética padrão de estudo, formada por 100 mil transações. O intervalo de suporte mínimo testado foi de 0,75% a 2%, quando o algoritmo já apresentou tempo de execução estável. A Figura 6.2 apresenta os resultados encontrados.

É possível visualizar uma grande redução no tempo de execução entre o suporte de 0,75% e o de 1%, em seguida existem pequenas quedas até o valor de 1,75%, onde o tempo de execução é bem próximo ao do valor de 2%, o que indica uma estabilidade. Isto indica um grande intervalo de itens que possuem suporte entre 0,75% e 1% e que são podados na primeira etapa do algoritmo ao se utilizar o suporte de 1% ou maior.

Esta sequência de quedas, seguida por uma estabilização do tempo de execução também é encontrada em outros estudos e implementações do Apriori como (LIN *et al.*, 2012), (JEONG *et al.*, 2012) e DMTA.

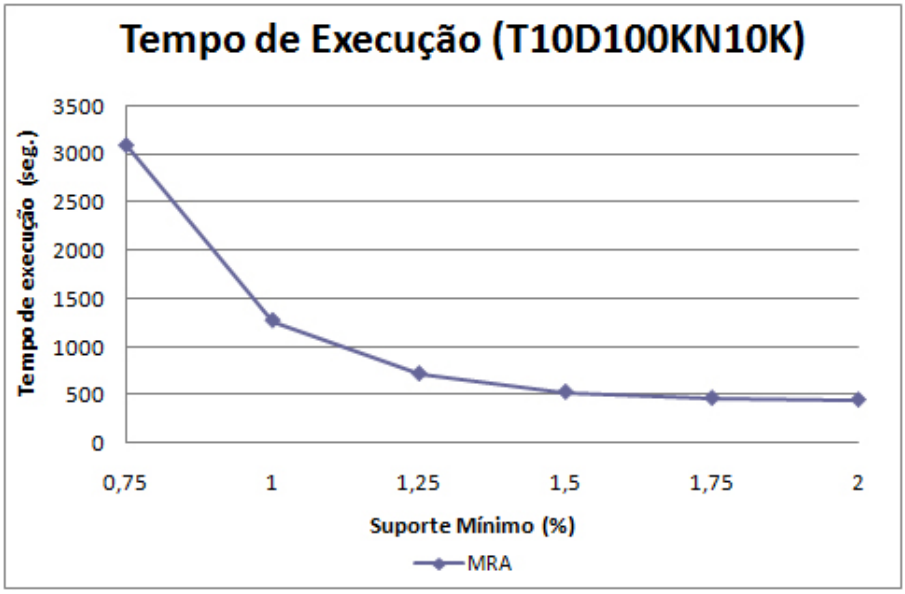

Figura 6.2: Tempos de execução para a base sintética.

Ainda com a base de dados sintética foram realizadas execuções mantendo um suporte de 1%, com o qual o algoritmo apresentou nível de processamento mediano em relação aos demais valores e que também foi utilizado por (YANG *et al.*, 2010) para execuções com base de dados de mesmo tamanho, porém variando o número de máquinas utilizadas pelo Framework para identificar o Speedup alcançado pelo algoritmo.

Para estas execuções, as mesmas condições foram aplicadas ao algoritmo DMTA, também no mesmo ambiente. Os resultados das execuções para os 2 algoritmos podem ser visualizados na Figura 6.3.

As linhas traçadas por ambos os algoritmos são bem próximas, isto indica uma igual capacidade de divisão de carga e processamento entre os mesmos. Ambos os algoritmos apresentam crescimento linear para os valores de Speedup com 2 e 3 máquinas, porém para 4 máquinas o algoritmo MRA diminui o crescimento constante que vinha apresentando. Isto porque para este número de máquinas o

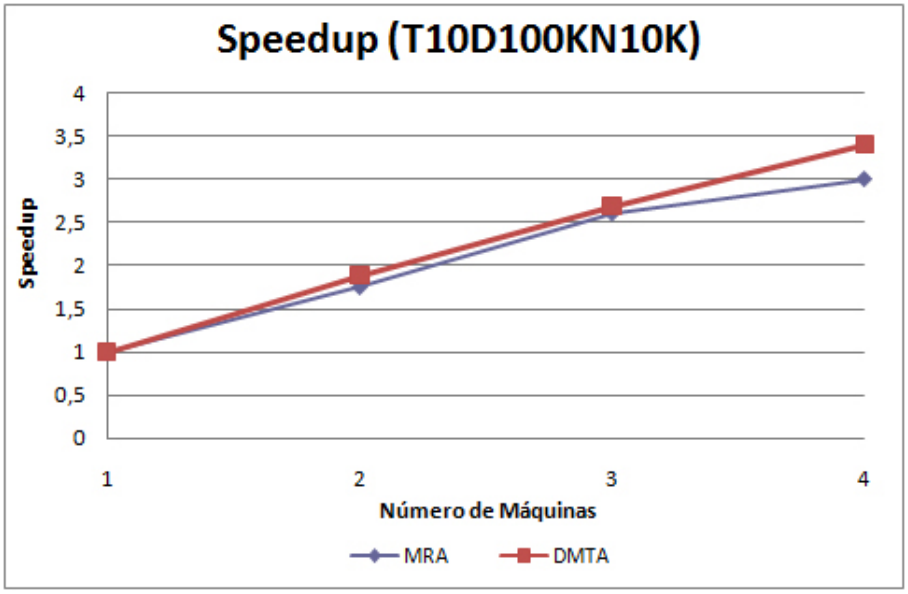

Figura 6.3: Speedup dos algoritmos DMTA e MRA para a base sintética.

tempo gasto pelo algoritmo para inicializar o ambiente, dividir tarefas e acessar o banco de dados supera o tempo ganho pela divisão de carga e processamento entre as máquinas para este volume de dados.

6.5 Base real

Para a base de dados real também foram realizadas 5 execuções para cada valor de suporte mínimo, utilizando 4 máquinas. Foram utilizados valores de suporte de 0% a 100%, não seguindo um intervalo fixo. Os resultados são exibidos na Figura 6.4.

Diferente da base sintética, os resultados não apresentam grande queda no tempo de execução ao se variar o suporte mínimo, isto porque para este caso a variação do suporte não elimina muitos itens, visto que como é pequeno o número de possíveis itens, estes apresentam alto suporte. Sendo assim o aumento do suporte

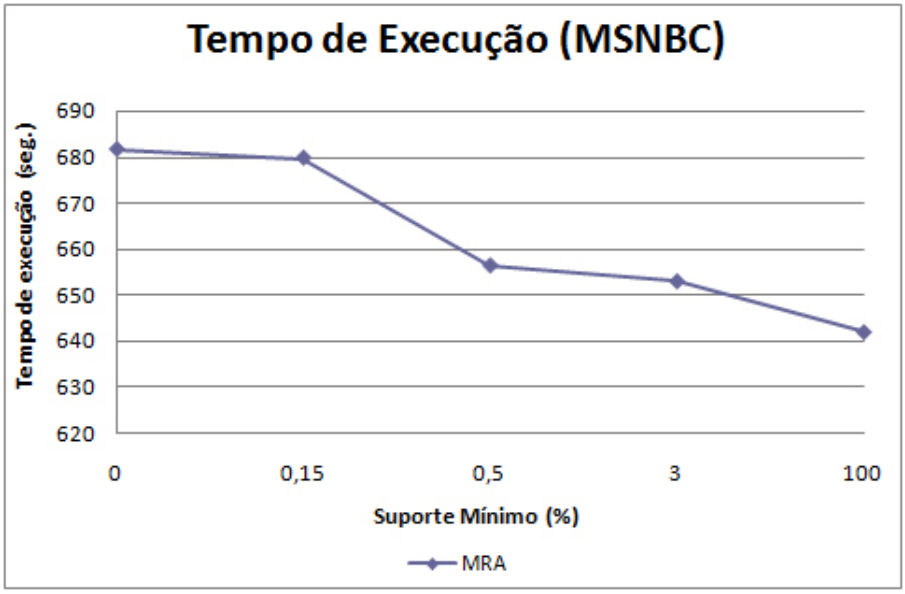

Figura 6.4: Tempos de execução para a base real.

mínimo não os elimina facilmente e portanto não reduz o nível de computação necessária, fazendo com que o tempo de execução se mantenha quase constante. Por exemplo, entre o suporte de 0%, que apresenta todos os itemsets existentes na base de dados, e o suporte de 100%, que apresenta apenas os itemsets presentes em todas as transações, o tempo de execução sofre uma queda de cerca de 40 segundos, representando apenas 6% do tempo total.

Assim como para a base sintética, foram realizados testes de Speedup com a base real, porém o suporte estabelecido para as execuções foi de 0%. A Figura 6.5 apresenta os resultados, também foram feitas execuções nas mesmas configurações e ambiente para o algoritmo DMTA.

Os resultados de speedup do MRA para a base real assim como a para a base sintética apresentaram crescimento até o uso de 3 máquinas, porém apresentou uma queda de Speedup com a utilização de 4 máquinas, algo que não ocorreu na base sintética.

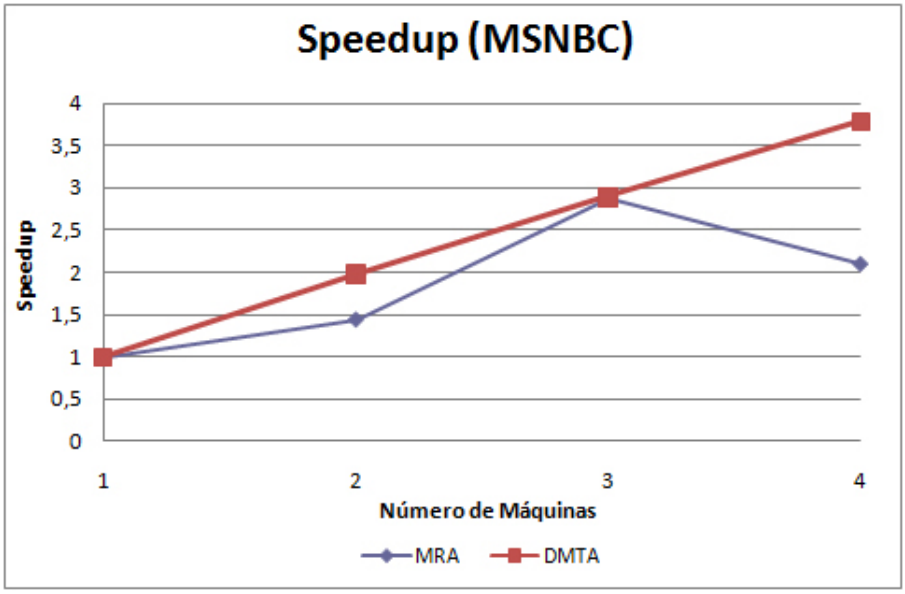

Figura 6.5: Speedup dos algoritmos DMTA e MRA para a base real.

Isto ocorre devido a base real possuir características ideais para o máximo desempenho do MRA, que são o pequeno número de itens possíveis e baixa média de itens por transação, o que permite ao algoritmo proposto alcançar este nível máximo com um menor número de máquinas do que quando utilizadas outras bases.

6.6 Comparação entre tempos de execução

Para ambas as bases, sintética e real, foram comparados também o tempo de execução obtidos pelo DMTA e pelo MRA. Para a base sintética os tempos obtidos pelo DMTA são bem menores que os tempos obtidos pelo MRA, como visto na Figura 6.6.

Isto ocorre porque como o MRA desvincula as transações do contexto geral ele precisa gerar os conjuntos de k-itemsets possíveis, enquanto o DMTA recebe

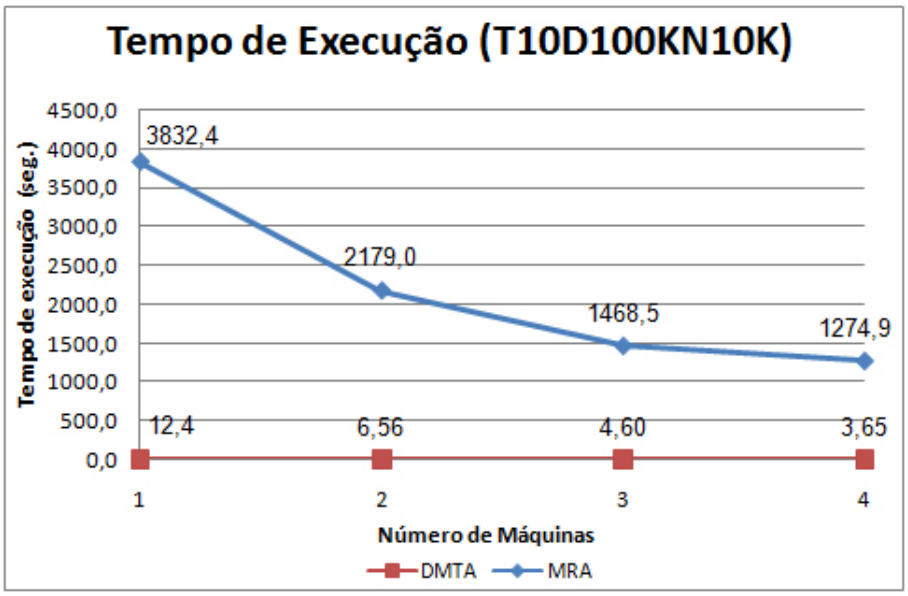

Figura 6.6: Comparação entre tempos de execução dos algoritmos DMTA e MRA para a base sintética.

e analisa os resultados dos conjuntos de k-itemsets possíveis a cada vez que o valor de k aumenta, podendo encerrar a execução se o critério de parada que é não encontrar k-itemsets frequentes for gerado.

Para a base real, os resultados apresentados na Figura 6.7, já são diferentes, os tempos são próximos, sendo melhores para o MRA quando as execuções possuem 1 e 3 máquinas e melhor para o DMTA em execuções com 2 e 4 máquinas, porém em todos os casos a diferença entre o tempo de execução dos algoritmos não passa de 20% do tempo total.

Isto ocorre porque o pequeno número de itens disponíveis e a pequena média de itens por transação são condições ideais para a execução do MRA, que gera todas as combinações possíveis por transação. Além disto, para esta base o critério de parada não é alcançado tão facilmente, e isto faz com que ambos os algoritmos realizem um grande processamento.

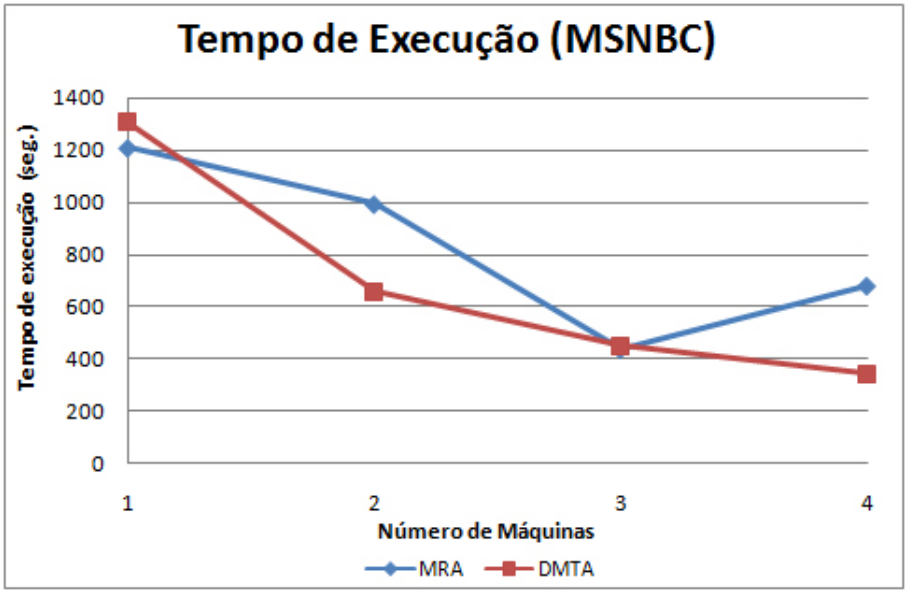

Figura 6.7: Comparação entre tempos de execução dos algoritmos DMTA e MRA para a base real.

6.7 Balanceamento

A interface web proporcionada pelo framework exibe o tempo gasto por cada máquina para executar as tarefas atribuídas a ela, o que permite identificar o balanceamento de carga. Os resultados destes tempos para execuções com 4 máquinas, utilizando as bases sintética e real com valores de suportes de 1% e 0%, respectivamente, estão apresentados na Figura 6.8 abaixo.

É possível identificar um balanceamento maior para a base real do que para a base sintética, isso pode ocorrer devido a maior média de itens por transação na base sintética. Com mais itens por transação, é possível gerar um número maior de combinações entre os mesmos, e com uma média maior é possível casos que exigem grande computação, o que atrasa o tempo de execução em determinadas máquinas.

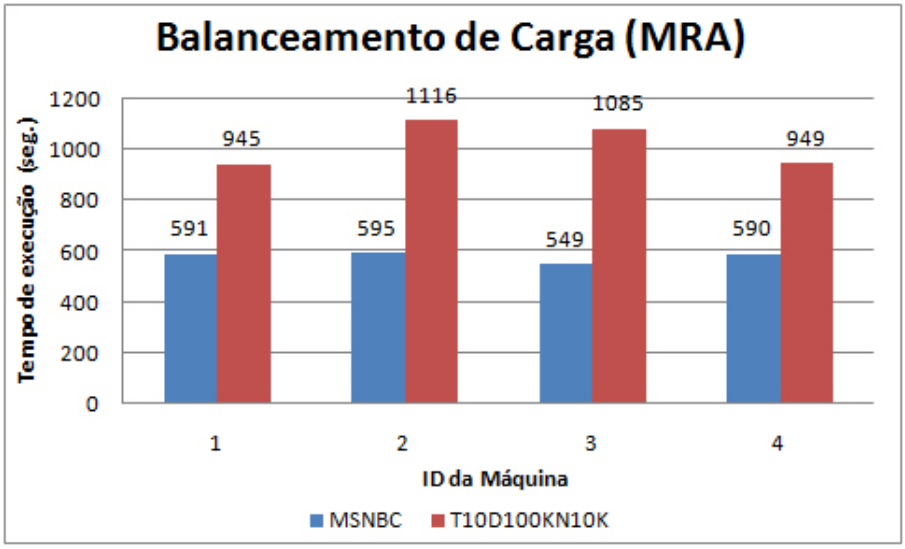

Figura 6.8: Balanceamento de carga para o algoritmo proposto.

6.8 Framework versus Bibliotecas

A Tabela 6.3 busca apresentar as vantagens e desvantagens entre o uso do framework Hadoop MapReduce e das bibliotecas MPI e OpenMP.

As características apresentadas na tabela podem ser constatadas através dos resultados obtidos. De forma geral os resultados obtidos para o MRA e o DMTA utilizando a base sintética e a base real foram aproximados em termos de speedup, mas com alguma diferença para tempos de execução.

Os valores de speedup próximos ocorrem devido a comprovada eficiência do framework para paralelização e distribuição de dados e tarefas entre um ambiente qualquer, porém as diferenças no tempo de execução existem devido as limitações de programações encontradas pelo paradigma apresentado.

As técnicas de paralelização são diferentes e o framework por si só possui um custo computacional, o qual busca ser compensado com uma série de facilidades no momento da programação.

Tabela 6.3: Comparação entre o uso do MapReduce e das bibliotecas MPI e OpenMP

MPI e OpenMP	*Hadoop MapReduce*
Simples instalação e fácil adição de novas máquinas	Configurações iniciais complexas, mas em um ambiente já configurado é fácil adicionar máquinas
Consome memória RAM apenas durante a execução, de acordo com o tamanho das estruturas geradas	Consome alto nível de memória RAM constantemente.
Necessita de espaço e da cópia local dos dados no HD de todas as máquinas integrantes	Armazenamento de dados apenas no HDFS, sistema de dados distribuído
Entradas precisam ser copiadas localmente para cada máquina para acessos sequenciais	Entradas podem ser acessadas de forma paralela no HDFS
Códigos grandes, com maior nível detalhamento das tarefas. É necessário preocupar com a divisão de dados e atribuição de tarefas	Códigos pequenos e mais simples. O Framework cuida de todo o processo de paralelização
Não há acompanhamento da execução a não ser que seja implementado	Interface web para acompanhar a execução com alto nível de detalhes
Em caso de falhas na rede ou em uma máquina o algoritmo é encerrado com erro, sem informações adicionais	Em caso de falhas, o framework cuida de atribuir novamente as tarefas incompletas a outras máquinas. Relatando tudo em LOGs
Limitado pelas estruturas de dados criadas pela linguagem C/C++	Estruturas próprias permitem dados em volume de Petabytes

O algoritmo MRA, por exemplo, foi implementado em cerca de 1 mês, enquanto o algoritmo DMTA vem sendo trabalhado há pelo menos 2 anos, e ainda sim apresentaram valores de speedup e tempo de execução semelhantes para algumas configurações de execução.

Ambas as ferramentas apresentam limitações, as bibliotecas se limitam totalmente ao hardware local e não são capazes de tratar erros ou trabalhar com grande volume de dados, mas dão liberdade de programação ao usuário.

Já o framework possui as características como tratamento de erros e ambiente virtual de execução, mas exige um maior processamento das máquinas e apresenta uma série de definições que não podem ser alteradas no momento da programação.

Embora ambos os algoritmos estejam aptos a processar quaisquer bases de dados, o ideal é que haja uma identificação das características da base para verificar quais dos algoritmos obteriam os melhores resultados.

Em geral bases com pequena média de itens por transação, ainda que com grande número de itens possíveis ou grande número de transações, são ideais para aplicação do MRA, já que este algoritmo oferece uma série de vantagens devido a utilização do framework e ainda sim tempo de execução similar ao DMTA.

Para bases com alta média de itens por transação, entretanto, é aconselhável a aplicação do DMTA, visto que este algoritmo possui controle da geração de itemsets em etapas, o que permite a existência do critério de parada e que pode assim acelerar o tempo de execução, evitando cálculos de conjuntos desnecessários.

7 CONCLUSÃO E TRABALHOS FUTUROS

Este presente trabalho buscou realizar o estudo e a implementação de uma versão paralela do algoritmo para mineração de padrões frequentes, Apriori, utilizando o framework MapReduce. A motivação foi o crescente aumento no volume de dados gerados dia-a-dia, consequentemente gerando um aumento do tempo de execução gasto por algoritmos existentes, e o grande número de estudos que aplicaram o framework a problemas e algoritmos já existentes da computação.

Os resultados foram analisados e comparados com o algoritmo DMTA, que é outra implementação paralela do Apriori. Pode-se concluir que o algoritmo proposto, ou MRA, apresentou comportamento similar ao algoritmo de comparação tanto em relação ao tempo de execução obtido, alterando o suporte, quanto ao speedup obtido, alterando o número de máquinas.

O MRA apresentou mesmo comportamento, porém tempo de execução maior, em relação ao DMTA quando utilizadas bases com grande média de transações. Já para bases com pequena média de itens por transação, o algoritmo proposto alcançou tempo de execução similar, superando o DMTA para determinados números de máquinas.

Avaliando os resultados, percebe-se uma melhor aplicação do algoritmo proposto a bases de dados que envolvam grande número de transações com pequena média de itens, normalmente encontrado em logs de acesso a páginas web. Além disto, foi possível concluir que o uso do framework conseguiu simplificar e acelerar o processo de implementação, também oferecendo segurança e mais suporte a falhas durante os processos de execução.

Como trabalhos futuros pode-se alterar o fluxo da segunda etapa do algoritmo MRA, acrescentando um controle do número de itens por conjunto gerado, o que possibilitaria a implementação de um critério de parada e mais podas durante as

demais execuções desta etapa. É possível também buscar uma forma global de conexão e acesso ao banco de dados distribuído, visto o custo de computação e tempo de execução utilizado para esta função na versão atual.

Outra otimização seria a implementação de um critério de parada, que permitiria acelerar o tempo de execução, uma vez que candidatos não frequentes não seriam nem mesmo calculados, diminuindo assim as combinações geradas pelo algoritmo. Para isto seria necessária a utilização das podas em etapas, ou seja, a cada novo valor de *k* seria necessário uma verificação dos (k-1)-itemsets para descobrir quais conjuntos podem ser podados. Combinando-se esta estratégia com o uso do HBase pode-se encontrar uma boa solução para implementação do Apriori utilizando as ferramentas do projeto Apache Hadoop.

REFERÊNCIAS BIBLIOGRÁFICAS

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Mining association rules between sets of items in large databases. Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, p. 207–216, 1993.

AGRAWAL, R.; SHAFER, J. C. Parallel mining of association rules. IEEE Transactions on Knowledge and Data Engineering, v. 8, n. 6, p. 962–969, 1996.

AGRAWAL, R.; SRIKANT, R. Fast algorithms for mining association rules. In Proceedings of the 20th international conference on very large databases, p. 487–499, 1994.

ALMADEN, I. Quest synthetic data generation code. <http://almaden.ibm.com/cs/quest/syndata.html>, 1994. Acessado em: 11/03/2013.

APACHE. Apache hadoop project. <http://hadoop.apache.org/index.html>, 2012. Acessado em: 11/03/2013.

BARNEY, B. Openmp. <https://computing.llnl.gov/tutorials/openMP/>, 2012. Acessado em: 11/03/2013.

BRAGA, L. P. V. *Introdução à Mineração de Dados*. 2ª edição revisada e ampliada. ed. Rio de Janeiro: E-Papers Serviços Editoriais, 2005.

CHEUNG, D. W.; LEE, S. D.; XIAO, Y. Effect of data skewness and workload balance in parallel data mining. IEEE Transactions on Knowledge and Data Engineering, v. 14, n. 3, p. 498–514, 2002.

CHEUNG, D. W.; NG, V. T.; FU, A. W. Efficient mining of association rules in distributed databases. IEEE Transactions on Knowledge and Data Engineering, v. 8, n. 6, p. 911–922, 1996.

CHUANG, T.-C. How to install hadoop cluster(2 node cluster) and hbase on vmware workstation. <chu-angtc.info/ParallelComputing/SetUpHadoopClusterOnVmwareWorkstation.htm>, 2011. Acessado em: 11/03/2013.

CRYANS, J.-D.; RATTE, S.; CHAMPAGNE, R. Adaptation of apriori to mapreduce to build a warehouse of relations between named entities across the web. p. 185–189, 2010.

EINAKIAN, S.; GHANBARI, M. Parallel implementation of association rules in data mining. In Proceedings of the 38th southeastern symposium on system theory, p. 21–26, 2006.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From data mining to knowledge discovery in databases. *AI magazine*, v. 17, n. 3, p. 33, 1996.

FILHO, C. L. A. S. Processamento de dados em larga escala na computação distribuída. 2011.

HAN, J.; KAMBER, M. *Data Mining: Concepts and Techniques*. San Francisco: Morgan Kaufmann, 2001.

HEDLUND, B. Understanding hadoop clusters and the network. <http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/>, 2011. Acessado em: 11/03/2013.

HONGA, T.-P.; KUOB, C.-S.; WANGO, S.-L. A fuzzy aprioritid mining algorithm with reduced computational time. Applied Soft Computing, v. 5, n. 1, p. 1–10, 2004.

IVANCSY, R.; VAJK, I. Frequent pattern mining in web log data. *Acta Polytechnica Hungarica*, v. 3, n. 1, p. 70–77, 2006.

JEONG, B.-S.; CHOI, H.-J.; HOSSAIN, A.; RASHID, M.; KARIM, R. A mapreduce framework for mining maximal contiguous frequent patterns in large dna sequence datasets. *IETE Technical Review*, v. 29, n. 2, p. 162–168, 2012.

KUN-MING, Y.; JIAYI, Z.; TZUNG-PEI, H.; JIA-LING, Z. A load-balanced distributed parallel mining algorithm. Expert Systems with Applications, v. 37, p. 2459–2464, 2010.

LAN, Q.; ZHANG, D.; WU, B. A new algorithm for frequent itemsets mining based on apriori and fp-tree. v. 2, p. 360–364, 2009.

LI, N.; ZENG, L.; HE, Q.; SHI, Z. Parallel implementation of apriori algorithm based on mapreduce. p. 236–241, 2012.

LI, Z.-C.; HE, P.-L.; LEI, M. A high efficient aprioritid algorithm for mining association rule. Machine Learning and Cybernetics, v. 3, p. 1812–1815, 2005.

LIN, M.-Y.; LEE, P.-Y.; HSUEH, S.-C. Apriori-based frequent itemset mining algorithms on mapreduce. p. 76:1–76:8, 2012.

MING-YEN, L.; PEI-YU, L.; SUE-CHEN, H. Apriori-based frequent itemset mining algorithms on mapreduce. In Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, 2012.

MPI. The message passing interface (mpi) standard. <http://www.mcs.anl.gov/research/projects/mpi/>, 2012. Acessado em: 11/03/2013.

NAVEGA, S. Princípios essenciais do data mining. Intelliwise Research and Training, <http://www.intelliwise.com/reports/i2002.htm>, 2002. Acessado em: 11/03/2013.

NOLL, M. G. Running hadoop on ubuntu linux (multi-node cluster. <http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/>, 2011. Acessado em: 11/03/2013.

PARTHASARATHY, S.; ZAKI, M. J.; OGIHARAAND, M.; LI, W. Parallel implementation of association rules in data mining. Knowledge and Information Systems, v. 3, n. 1, p. 1–29, 2001.

PATEL, B.; CHAUDHARI, V. K.; KARAN, R. K.; RANA, Y. Optimization of association rule mining apriori algorithm using aco. International Journal of Soft Computing and Engineering, v. 1, n. 1, 2011.

RAMARAJ, E.; VENKATESAN, N. Bit stream mask-search algorithm in frequent itemset mining. 2009.

SHARMA, Y. A word count example – apache hadoop. <http://www.confusedcoders.com/tutorials/big-data/apache-hadoop/a-word-count-example-apache-hadoop/>, 2012. Acessado em: 11/03/2013.

SILVA, M. P. S. Mineração de dados - conceitos, aplicações e experimentos com weka. UERN e INPE, 2000.

SONG, Q.; NI, J.; WANG, G. A fast clustering-based feature subset selection algorithm for high-dimensional data. *Knowledge and Data Engineering, IEEE Transactions on*, v. 25, n. 1, p. 1–14, 2013.

SOUSA, G. H. A. Estudo de algoritmos híbridos para clusterização de dados usando pso. Universidade Federal de Lavras, 2010.

TAN, P.; STEINBACH, M.; KUMAR, V. e. a. Introduction to data mining. Pearson Addison Wesley Boston, 2006.

VALENCIO, C.; OYAMA, F.; ICHIBA, F.; SOUZA, R. de. Multi-relational algorithm for mining association rules in large databases. p. 269–274, 2011.

WAINER, J. Métodos de pesquisa quantitativa e qualitativa para a ciência computação. Sociedade Brasileira de Computação e Editora PUC-Rio, p. 221–262, 2007.

WU, C.; LAI, L. F.; HUANG, L. T.; JHANAND, S. S.; LU, C. A fine-grained scheduling strategy for improving the performance of parallel frequent itemsets mining. International Journal of Computational Science and Engineering, v. 6, n. 4, p. 264–274, 2011.

YANG, X. Y.; LIU, Z.; FU, Y. Mapreduce as a programming model for association rules algorithm on hadoop. p. 99–102, 2010.

YE, Y.; CHIANG, C. C. Efficient mining of association rules in distributed databases. In Proceedings of the fourth international conference on software engineering research, management and applications, p. 87–94, 2006.

YE, Y.; CHIANG, C.-C. A parallel apriori algorithm for frequent itemsets mining. p. 87–94, 2006.

YU, K.-M.; ZHOU, J.-L. A weighted load-balancing parallel apriori algorithm for association rule mining. IEEE International Conference Granular Computing, p. 756–761, 2008.

ZANG, H. Non-redundant sequential association rule mining based on closed sequential patterns. 2010.

ZHAO, S.; DU, R. Distributed apriori in hadoop mapreduce framework. <http://www.chinacloud.cn/upload/2012-10/12100114454882.pdf>, 2012. Acessado em: 11/03/2013.

ZHENG, Z.; KOHAVI, R.; MASON, L. Real world performance of association rule algorithms. p. 401–406, 2001.