ALESSANDRO REIS DE ALCÂNTARA

ESTUDO DE FERRAMENTAS BASEADAS NO WEKA PARA MINERAÇÃO DE DADOS DISTRIBUÍDA EM AMBIENTE DE *GRID*

LAVRAS - MG

2012

ALESSANDRO REIS DE ALCÂNTARA

ESTUDO DE FERRAMENTAS BASEADAS NO WEKA PARA MINERAÇÃO DE DADOS DISTRIBUÍDA EM AMBIENTE DE *GRID*

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

Orientador

Profa. Dra. Marluce Rodrigues Pereira

LAVRAS - MG

2012

ALESSANDRO REIS DE ALCÂNTARA

ESTUDO DE FERRAMENTAS BASEADAS NO WEKA PARA MINERAÇÃO DE DADOS DISTRIBUÍDA EM AMBIENTE DE *GRID*

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

Aprovada em *22 de Outubro de 2012*

Prof. Dr. Denilson Alves Pereira	Universidade Federal de Lavras	Denilson Alves Pereira
Prof. Msc. Tiago Amador Coelho	Universidade Federal de Lavras	Tiago Amador Coelho

Marluce Rodrigues Pereira

Profa. Dra. Marluce Rodrigues Pereira

Orientador

LAVRAS - MG

2012

Dedico este trabalho aos meus pais, Ary e Sandra, por estarem sempre ao meu lado, dando todo apoio necessário para que eu alcançasse esta vitória.

AGRADECIMENTOS

Agradeço aos meus pais, Ary e Sandra, aos meus irmãos, Bruno e Lara, e a todos da família. Sem vocês essa conquista não seria possível. Um agradecimento especial para minha namorada Marina que foi minha companheira durante toda minha graduação, sempre me dando o apoio, a força necessária e me auxiliando em todas as decisões. À Professora Dra. Marluce Rodrigues Pereira por todos os ensinamentos e dedicação que guiaram este trabalho e foram de fundamental importância para o resultado dele. Obrigado também aos demais professores por me transmitirem seus conhecimentos. À Gerêncianet pela confiança em meu trabalho e pela primeira oportunidade profissional, que me proporciona um ambiente de trabalho excepcional me incentivando a dar o máximo de meu empenho e dedicação. Aos amigos de Lavras, especialmente Fejão, Túlio, Balão, Markin, Michel, Fontinelle, Berdan e Galeano pela amizade construída neste período, e por todo o companheirismo e ensinamento que levarei comigo para a vida, e também aos demais colegas da turma Comp 2008/2 da UFLA. Aos amigos de Varginha, principalmente Igor, Flavinho, Ruivo e Fernando pela longa amizade.

A todos que contribuiram de alguma forma para meu sucesso, muito obrigado !

RESUMO

A necessidade de processar grandes quantidades de dados está aumentando cada vez mais. A mineração de dados é uma área computacional em que se busca a extração de conhecimento de bases de dados, e muitas vezes estas bases de dados são muito grandes, demorando bastante tempo para serem processadas. Este trabalho apresenta um estudo sobre mineração de dados distribuída em ambiente de *Grid* com o objetivo de aumentar o desempenho computacional da tarefa de mineração de dados. Para isto foram usadas duas ferramentas baseadas no Weka, o Weka4WS e o Weka *Parallel*. É feita uma comparação destas ferramentas entre si e com o Weka original para determinar o quão vantajoso é usar o ambiente de *Grid*.

Palavras-chave: Computação em Grid, Mineração de Dados, Weka, Weka4WS, Weka Parallel

ABSTRACT

The need to process large amounts of datas is increasing more and more. Data mining is an computational area that seeks to extract knowledge of databases, and sometimes these databases are too big, taking a lot of time to be processed. This work presents a study about distributed data mining in Grid environments with the goal of increasing the computational performance of the data mining task. For this, will be used two Weka based tools, the Weka4WS and the Weka Parallel. A comparison is made between those tools themselves and with the original Weka to determine how profitable is using the Grid environment.

Keywords: Grid Computing, Data Mining, Weka, Weka4WS, Weka Parallel

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A mineração de dados é uma área computacional que busca a extração de conhecimento de bases de dados. O principal benefício da mineração de dados é a filtragem dos dados, permitindo que somente os dados de interesse para determinada tarefa sejam enviados ao analista humano que os interpretará. Tarefas de mineração de dados são complexas e muitas vezes demoram para serem executadas, exigindo um grande poder de processamento. A distribuição das tarefas entre vários processadores é uma forma utilizada para acelerar o processamento.

As causas para a alta complexidade da tarefa de mineração de dados são variadas. Pode-se citar os vários passos necessários para a realização da tarefa, como pré-processamento e apresentação de resultados. Também pode-se citar o grande volume de algumas bases de dados, que exigem um alto poder de processamento e de armazenamento de dados, e por fim, a complexidade dos dados, os quais podem estar localizados em servidores distribuídos geograficamente devido ao seu grande volume. Cada um desses itens levam ao aumento da complexidade da tarefa de mineração de dados [STANKOVSKI *et al.*, 2004].

As causas citadas acima geram a necessidade de um grande poder de processamento, desafiando assim, organizações e empresas a encontrarem uma solução para este problema. Com essa necessidade, vários modelos para computação paralela e distribuída foram surgindo. Neste trabalho foi usada uma infraestrutura de *Grid* para a realização de mineração de dados distribuída com a finalidade de aumentar o poder computacional dessa tarefa. Uma infraestrutura de *Grid* busca compartilhar recursos computacionais distribuídos geograficamente com o objetivo de implantar aplicações em larga escala e geralmente, é formada por vários componentes [AOUAD *et al.*, 2005]. Por isso, vários *middlewares* foram desen-

volvidos com o objetivo de facilitar a configuração e posteriormente o uso deste tipo de ambiente. Os *middlewares* para *Grid* são *softwares* que buscam abstrair a complexidade do ambiente de *Grid* facilitando seu gerenciamento pelos usuários.

O objetivo principal deste trabalho é fazer um estudo sobre ferramentas para mineração de dados que utilizem ambiente de *Grid* para acelerar a computação de uma tarefa de mineração de dados. Como objetivos específicos foram realizados: (a) o estudo do funcionamento de um ambiente de *Grid*; (b) uma comparação entre duas ferramentas baseadas no Weka [HALL *et al.*, 2009] para execução em ambientes de *Grid*: Weka4WS [TALIA *et al.*, 2008] e Weka *Parallel* [CELIS; MUSICANT, 2002]; (c) a comparação do desempenho entre as duas ferramentas, e destas com o Weka, utilizando bases de dados de variados tamanhos.

O presente trabalho utiliza o Weka4WS, e este necessita da implantação do *middleware* Globus [FOSTER, 2005]. O Globus é um *middleware* para infraestruturas de *Grid* e possui várias funcionalidades. O presente trabalho foi realizado em parceria com o trabalho de conclusão de curso do aluno Caio César Roma Menten [MENTEN, 2012] que apresenta informações mais detalhadas sobre o Globus, visto que o presente trabalho foca mais na comparação das ferramentas baseadas no Weka para mineração de dados distribuída em ambiente de *Grid*.

O restante deste texto está organizado da seguinte forma. O Capítulo 2 aborda computação em *Grid*, apresentando alguns *middlewares* para implementação de um ambiente de *Grid*. O Capítulo 3 apresenta conceitos relacionados a mineração de dados. No Capítulo 4 é apresentada a ferramenta Weka para realizar mineração de dados, além de ferramentas para mineração de dados distribuída. O Capítulo 5 expõe os métodos usados para a realização dos testes no trabalho presente. O Capítulo 6 mostra os resultados obtidos com os testes. E, por fim, o Capítulo 7

apresenta a conclusão obtida com a elaboração deste trabalho, além de sugestões para trabalhos futuros.

2 COMPUTAÇÃO EM *GRID*

De acordo com [XHAFA *et al.*, 2010], a computação em *Grid* surgiu como um paradigma para computação de alto desempenho e processamento paralelo massivo. *Grids* computacionais são compostos por vários nós computacionais (computadores) que comunicam-se, buscando a todo momento aproveitar a capacidade ociosa de cada nó presente no *Grid*. Infraestruturas em *Grid* fornecem a capacidade de distribuir processamento e armazenamento de dados entre vários nós computacionais, mas estes não necessariamente estão no mesmo local físico, ou seja, *Grids* podem ser distribuídos geograficamente e cada nó pode pertencer a um domínio administrativo diferente dos outros [KOKKINOS *et al.*, 2011]. O termo "*Grid* computacional" surgiu na década de 90 devido a uma analogia feita deste tipo de sistema com os sistemas de energia elétrica chamados de *Power Grids*, em que o usuário utiliza a energia elétrica sem saber onde ela foi produzida.

Grids são muitas vezes confundidos com *clusters* e com a *cloud computing* pois os três são paradigmas de computação remota e distribuída. Um *Grid*, ao contrário de um *cluster*, não tem um processamento centralizado. Um *Grid* é totalmente distribuído sendo que cada nó do *Grid* é totalmente independente dos outros, i.e., se um nó parar de funcionar ele não irá prejudicar a computação como um todo. Já em um *cluster*, se ocorre o mesmo imprevisto, só que com o nó central, todo o processamento será prejudicado. Já a *cloud computing* tem muitos pontos em comum com *Grids*, por isso os dois paradigmas são bastante confundidos. Ambos adotam grandes *datacenters*, ambos oferecem recursos para usuários e ambos visam oferecer um ambiente comum para recursos distribuídos [MANCINI *et al.*, 2009]. A principal diferença da *cloud computing* para a computação em *Grids* é que a primeira não necessita a instalação de nenhum tipo de *software* e o acesso

aos arquivos é feito sempre remotamente, através da Internet. A Tabela 1 mostra essas diferenças entre os três paradigmas.

Paradigma	***Grid***	***Cluster***	***Cloud***
Característica			
Processamento Centralizado	Não	Sim	Não
Distribuído Geograficamente	Sim	Não	Sim
Execução a partir de software instalado na máquina cliente	Sim	Sim	Não
Acesso aos arquivos sempre através da Internet	Não	Não	Sim

Tabela 1: Comparação entre paradigmas utilizados em computação remota e distribuída

O conceito de *Grid* é manifestado em forma de organizações virtuais (OVs) onde a comunidade de ciência eletrônica pode compartilhar recursos e colaborar [KACSUK, 2011]. Organizações virtuais são formadas por um conjunto de usuários e organizações reais que coletivamente oferecem recursos para serem explorados, buscando atingir um objetivo em comum [COPPOLA *et al.*, 2008]. Os usuários de uma infraestrutura de *Grid* são agrupados em OVs que permitem a comunicação de entidadades abstratas, incluindo usuários, instituições e recursos no mesmo domínio administrativo. Quando se é membro de uma OV, um usuário tem garantido privilégios específicos. Por exemplo, um usuário pertencente a uma dada OV "X" terá permissão para ler os arquivos existentes em "X" ou explorar recursos reservados para a colaboração com a OV "X" [BURKE *et al.*, 2011]. A segurança em um ambiente de *Grid* passa também pelo conceito de OVs, sendo que um usuário só tem acesso a esse ambiente mediante a autenticação em uma OV.

Um ambiente de *Grid* é um ambiente altamente complexo e, por isso, é necessário que seja feito um bom gerenciamento dos recursos a serem usados. Cada

recurso, também chamado de *computing element* (CE)[1], tem que ser controlado para após iniciado, parar na hora correta e também se necessário, ser cancelado ou reiniciado. As tarefas realizadas pelos nós[2] dos *Grids* são chamadas de *jobs* e cada *job* possui seus CEs. As informações geradas por sistemas de *Grid* ficam armazenadas em arquivos. Cada arquivo pertence a um *storage element* (SE). Os SEs são os locais em que ficam armazenados todos os dados gerados pelo processamento das tarefas em um *Grid*, sendo que um SE pode armazenar múltiplos arquivos e cada SE pode comunicar com outros para trocar informações presentes nos arquivos.

Um serviço importante aos ambientes de *Grid* são os *web services* (WS), os quais são sistemas de software para computação distribuída desenvolvidos de forma a suportar interações entre máquinas sobre uma rede. WS realizam algum tipo de processamento em um servidor remoto e retornam o resultado para o usuário usando a Internet. Eles possuem uma interface descrita em um formato processável pelas máquinas - WSDL (*Web Service Definition Language*). Este serviço é uma forma padrão de comunicação entre diferentes aplicações [BOOTH *et al.*, 2004], possibilitando a integração de componentes distribuídos através da *web* [TOMA *et al.*, 2006]. Os WS permitem que um usuário autenticado em uma OV submeta *jobs* remotamente para vários nós, viabilizando a computação em *Grid*.

Abaixo são apresentados alguns *middlewares* para computação em *Grid*, inclusive o Globus que é utilizado pelo Weka4WS.

[1] Cada CE em um *Grid* é uma tarefa a ser realizada.
[2] Cada nó de um *Grid* é um processador disponível para execução das tarefas.

2.1 gLite

O gLite surgiu de esforços despendidos pelo projeto EGEE (*Enabling Grids for E-SciencE*) [EGEE, 2011], que tinha como objetivo maior a construção de um ambiente de Grid computacional que ficasse disponível para cientistas 24 horas por dia. Este é um *middleware* para Grid que roda em cima do *Scientific Linux*, e é composto de vários módulos, assim como a maioria dos *middlewares* para *Grid*, o que torna sua instalação algo complexa. Existem módulos que têm uma mesma finalidade e, se instalados juntos, podem gerar problemas no Grid. O projeto EGEE foi descontinuado em 30 de Abril de 2010 e toda a infraestrutura construída durante o projeto agora é mantida pela EGI (*European Grid Infrastructure*).

A EGI compartilha boa parte de sua estrutura com o projeto WLCG. O WLCG (*Worldwide LHC Computing Grid*) é um projeto que visa manter uma infraestrutura de *Grid* para armazenar e processar dados gerados pelo *Large Hadron Collider* (LHC), o maior e mais poderoso acelerador de partículas do mundo. Pelo compartilhamento da estrutura entre estes dois projetos, a infraestrutura utilizada pelo gLite é chamada de WLCG/EGI.

2.2 Legion

O Legion surgiu em 1993 na Universidade de Virginia. O objetivo do grupo de pesquisadores do Legion é fornecer um sistema com alta usabilidade, eficiência e escalabilidade, baseados em princípios sólidos. O sistema é desenvolvido para suportar altos graus de paralelismo em aplicações e gerenciar a complexidade do sistema físico para o usuário [LEGION, 2011].

2.3 Globus *Toolkit*

O Globus *toolkit* 4.0 (GT4) foi utilizado para servir como plataforma de execução deste trabalho pois o Weka4WS executa sobre o GT4. Nesta seção será dada uma visão geral do GT4. O Weka4WS será apresentado no capítulo 4.1.1. Todas as informações apresentadas neste capítulo foram obtidas no manual do GT4 [ALLIANCE, 2011].

O GT4 possui um conjunto de ferramentas, serviços e bibliotecas para viabilizar o uso da tecnologia de *Grid*, permitindo que as pessoas compartilhem com segurança poderes computacionais, bases de dados e outras ferramentas através de corporações, instituições e fronteiras geográficas. Esse *middleware* inclui serviços de *softwares* e bibliotecas para monitorar, descobrir e gerenciar recursos, além de oferecer suporte a segurança e gerenciamento de arquivos.

Existem muitos componentes providos pelo GT4. Esses vários componentes existentes podem ser instalados e usados independentemente. Por exemplo, existem implementações de núcleos de WS feitos para C, Java e Python, assim o usuário pode escolher qual deles instalar, caso só queira criar serviços usando uma das linguagens. Porém, se o usuário preferir ele pode instalar todos os componentes presentes no GT4 em vez de selecionar componente a componente.

A arquitetura do GT4 é dividida em 5 componentes. Abaixo serão apresentadas informações sobre cada um destes componentes que estão representados na Figura 1.

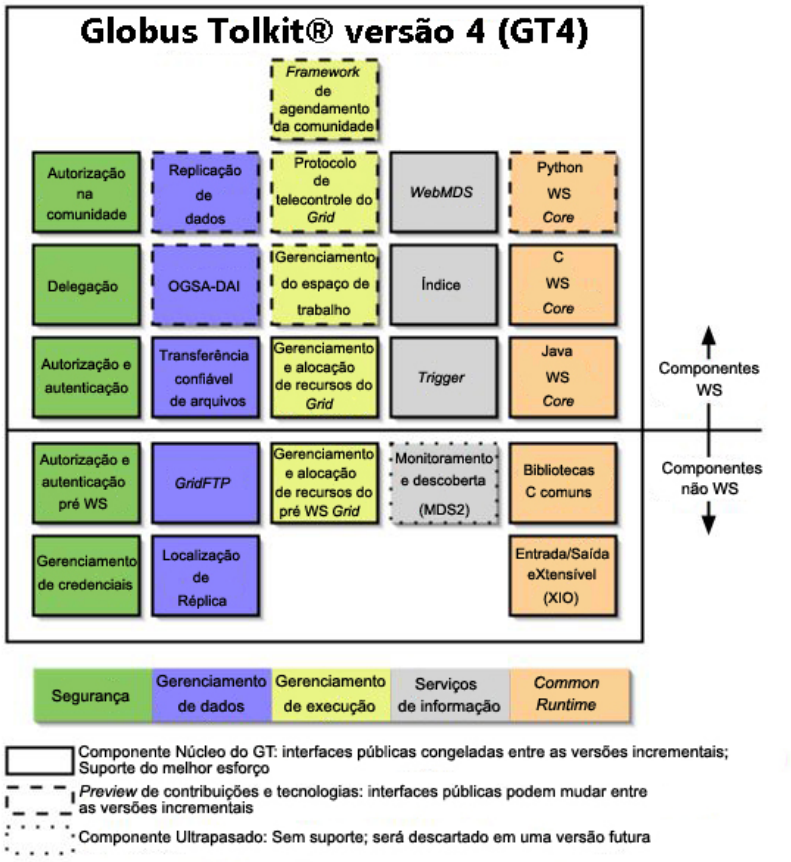

Figura 1: Arquitetura do Globus Toolkit 4 [SOTOMAYOR; CHILDERS, 2006]

Segurança: Um ponto muito importante em um *middleware* para *Grid* é a segurança. No GT4 a segurança é feita utilizando certificados de criptografia de chave pública X.509. Os certificados X.509 usam um modelo hierárquico para distribuir e autenticar certificados. Neste modelo os certificados são assinados por autoridades certificadoras. Essa estrutura monta uma árvore hierárquica da autoridade certificadora raiz, passando por autoridades certificadoras intermediárias até chegar no certificado da entidade final, que

pode ser cliente ou servidor [STORY *et al.*, 2009]. Existem outros métodos de segurança implementados no GT4, como por exemplo, credenciais de segurança utilizando padrões SSH, e autenticação direta fornecendo nome de usuário e senha, o qual não é muito seguro porém fácil de usar. Apesar de haver estes outros métodos de segurança, para utilizar os recursos de WS é necessário fazer o uso de certificados X.509.

Gerenciamento de Dados: Para o gerenciamento de dados existem alguns componentes implementados no GT4. Estes componentes são separados em duas categorias: movimentação de dados e replicação de dados. Existem dois componentes para movimentação de dados, o *GridFTP* e o *Globus Reliable File Transfer* (RFT). Já para a replicação de dados o componente usado é o *Replica Location Service* (RLS). Existe ainda um serviço de gerenciamento de dados de alto nível que combina os serviços RFT e RLS. Este serviço é chamado de *Data Replication Service* (DRS) e ele serve para garantir que um conjunto de arquivos esteja armazenado em um dado local de armazenamento. O DRS é implementado na forma de WS.

Gerenciamento de Execução: O componente do gerenciamento de execução é responsável pelas operações básicas da computação remota, como por exemplo, submissão, monitoramento e cancelamento. O mecanismo básico do GT4 para realizar estas operações é o GRAM - *Grid Resource Allocation and Management*. O GRAM controla os *jobs* submetidos ao *Grid*.

Serviços de Informação: No GT4 existe um sistema para prover serviços de informação, o MDS - *Monitoring and Discovery System*. Este monitora e localiza os recursos e serviços presentes nos *Grids*. Este sistema permite

que o usuário descubra quais recursos são considerados parte de uma OV e permite monitorá-los.

***Common Runtime*:** *Common runtimes* são conjuntos de bibliotecas que auxiliam na execução e no desenvolvimento de um programa. Os componentes de *common runtime* do GT4 fornecem uma série de bibliotecas e ferramentas de forma a permitir que os serviços WS e pré WS sejam independentes de plataforma. No GT4 existem implementações de *common runtime* para as linguagens C, Java e Python, sendo que o Weka4WS utiliza somente a implementação para Java.

Existem várias áreas da computação com potencial para aproveitar as vantagens oferecidas em um ambiente de *Grid*. Dentre essas áreas está a mineração de dados. A mineração de dados e alguns de seus conceitos serão apresentados no Capítulo 3.

Outras informações sobre o Globus *Toolkit* podem ser encontradas no trabalho de conclusão de curso do aluno Caio César Roma Menten, que aprofunda mais no estudo, configuração e instalação da ferramenta [MENTEN, 2012].

3 MINERAÇÃO DE DADOS

A mineração de dados é formada por um conjunto de ferramentas e técnicas que podem ser usadas em diversas áreas para extrair padrões consistentes de bases de dados e auxiliar na descoberta do conhecimento. Primeiramente é realizada a seleção dos métodos a serem utilizados para localizar tais padrões, seguida da efetiva busca pelos padrões de interesse numa forma particular de representação, juntamente com a busca pelo melhor ajuste dos parâmetros do algoritmo para a tarefa em questão [SILVA, 2004]. Essa tarefa envolve vários tipos de algoritmos e pode gerar resultados variados sobre uma mesma base de dados. Uma boa análise dos dados a serem minerados é importante para a escolha do melhor algoritmo a ser utilizado na tarefa de mineração.

O Weka [HALL *et al.*, 2009] é uma ferramenta que será abordada no Capítulo 4 e possui uma biblioteca de algoritmos para mineração de dados. As classes de algoritmos de mineração de dados disponíveis no Weka são: associação, classificação, clusterização e regressão. Estas classes de algoritmos estão definidas a seguir.

Associação: Algoritmos de associação de dados buscam encontrar uma relação entre dois ou mais dados pela associação existente entre eles. Por exemplo, suponha que todas as pessoas que assistam um determinado filme "A" assistam também os filmes "B" e "C". Isto geraria uma regra de associação do tipo "A" $\rightarrow$ "B","C" (se acontece A, também acontecem B e C).

Classificação: Algoritmos de classificação separam os dados minerados em classes diferentes para que possam ser analisados de forma mais adequada utilizando aprendizado supervisionado, i.e., existem dados que servem de mo-

delos para a classificação de todos os outros. Um exemplo é a geração de uma árvore de decisão. Na árvore formada, as folhas são as classes possíveis para a classificação de um dado. Quando o dado é enviado para a classificação, este vai percorrendo a árvore de acordo com as regras pré-definidas até alcançar uma das folhas, que será a classificação final deste dado.

***Clusterização*:** Algoritmos de *clusterização* também separam os dados minerados em classes diferentes, porém o aprendizado em um processo de *clusterização* é não supervisionado, i.e., o algoritmo vai encontrando um padrão entre os dados à medida que vai analisando-os e assim consegue separá-los em classes distintas. Por exemplo, existe um conjunto de dados em que nenhum deles está classificado. É feita a seleção de um dos dados e a avaliação dos dados que estão perto deste dado selecionado. Os dados próximos que possuírem características semelhantes a este dado serão classificados como sendo da mesma classe. Isto é feito até que não haja nenhum dado a ser classificado.

Regressão: A regressão é uma técnica similar a classificação, porém tem como objetivo encontrar um valor numérico ao invés de uma classe discreta. Isto é feito analisando previamente outros atributos numéricos disponíveis na base de dados [SANTOS, 2009].

Com o elevado número de dados computacionais, muitas bases de dados se tornaram bastante complexas. Nem todo dado de uma certa base é sempre útil, então aplicar o processo de mineração de dados na base é importante para se obter somente os padrões que sejam significativos para se realizar um determinado trabalho.

Existem inúmeras aplicações para a área de mineração de dados. A extração de padrões pode ser aplicada desde pequenas empresas até multinacionais através da exploração e análise, de forma automática ou semi-automática, de pequenas a grandes bases de dados, permitindo que essas empresas obtenham informações que as possibilitem desenvolver melhores estratégias de *marketing*, vendas, suporte, aumentando assim, a sua produtividade. Um exemplo clássico de mineração de dados em empresas é a extração de regras de associação em supermercados. Estas regras vão indicar qual produto deve ficar perto de qual, facilitando a procura do cliente. A extração de padrões pode ainda ser utilizada por toda a Internet, a qual possui um número gigantesco de dados para serem minerados, como por exemplo, artigos, *sites*, jogos *online*, documentos, vídeos, entre outras inesgotáveis aplicações que a *web* oferece [HAN *et al.*, 2011].

A mineração de dados é o núcleo de um processo chamado de *"Knowledge Discovery in Databases"*, mais conhecido como KDD.

3.1 KDD

Os conceitos apresentados a seguir seguem o modelo apresentado por [FAYYAD *et al.*, 1996].

O KDD define uma série de passos que podem, se necessário, ser repetidos para a extração do conhecimento em uma base de dados. Este processo utiliza algoritmos de mineração de dados para a extração de padrões e assim, para a obtenção de conhecimento.

O problema básico atribuido ao processo de KDD é o mapeamento de dados de baixo nível em outras formas de representação que podem ser mais compactas, mais abstratas ou mais úteis. KDD se refere a todo processo de descobrimento de

conhecimento útil a partir desses dados, e mineração de dados se refere a um passo particular deste processo.

A Figura 2 mostra os passos básicos que envolvem o processo de KDD.

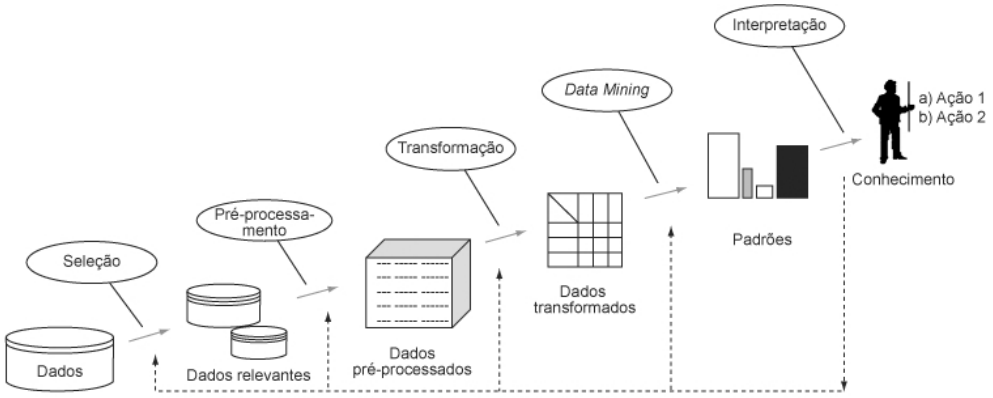

Figura 2: Etapas do Processo de KDD [STEINER *et al.*, 2006]

Seleção de dados: A fase de seleção de dados consiste em selecionar os dados de interesse para a tarefa a ser realizada.

Pré-processamento: Após selecionar os dados que serão usados para a extração de conhecimento deve-se remover dados incompletos e inconsistentes.

Transformação dos dados: É realizado um trabalho em cima dos dados extraídos para facilitar a tarefa de mineração.

Mineração de dados: São aplicados algoritmos para se obter informações úteis dos dados que já passaram pelos três processos anteriores. Vários tipos de informações podem ser extraídas dos dados, sendo que para cada tipo de informação existe uma série de algoritmos possíveis de serem aplicados.

Apresentação e interpretação dos resultados: Para facilitar a visualização das informações extraídas deve-se apresentar os resultados obtidos de alguma forma que facilite a interpretação do analista humano que usará estas informações. Uma falha na interpretação dos resultados pode gerar conclusões incorretas. Desta forma, a apresentação dos resultados de maneira adequada é muito importante dentro do processo de KDD.

Após a aplicação de cada um desses passos se tem em mão uma forma de representação do conhecimento extraída da base de dados analisada. É importante que haja uma boa aplicação dos conhecimentos extraídos, pois o processo de KDD é longo e não deve ser desperdiçado. Esta aplicação depende muito da pessoa que tiver estes conhecimentos em mão e a forma com que esta irá aplicá-los.

Existem na literatura ferramentas que facilitam a realização de mineração de dados, como por exemplo, *IBM Intelligent Miner* [CABENA *et al.*, 1999], *DBMiner* [WANG *et al.*, 1996], *MineSet* [BRUNK *et al.*, 1997] e *Weka* [HALL *et al.*, 2009].

Um dos principais problemas na mineração de grande volume de dados é o desempenho computacional. As ferramentas Weka4WS [TALIA *et al.*, 2008] e Weka *Parallel* [CELIS; MUSICANT, 2002] incorporam ao Weka a capacidade de computação distribuída. O Capítulo 4 aborda as ferramentas Weka, Weka4WS e Weka *Parallel*.

4 WEKA

O Weka (*Waikato Environment for Knowledge Analysis*) surgiu na Nova Zelândia na Universidade de Waikato, é implementado em Java, tem código aberto, e possui vários algoritmos já implementados para a realização de mineração de dados [HALL *et al.*, 2009]. Por ser um *framework* de código aberto a integração de outras aplicações com esta ferramenta é simplificada. Várias aplicações e extensões utilizam o Weka como base. Este *framework* possui uma interface gráfica para a interação com o usuário procurando facilitar a aplicação dos algoritmos em cima dos dados, e possui seu próprio formato para arquivos utilizados como entrada, o *Attribute Relation File Format* (ARFF). Este formato foi desenvolvido pela Universidade de Waikato para ser usado pelo Weka [WAIKATO, 2012b]. Lembrando que outros formatos também podem ser utilizados, como por exemplo, DAT, CSV, além de ser possível abrir uma base de dados diretamente utilizando o JDBC.

O JDBC - Java *Database Connectivity* - é uma API - *Application Programming Interface* - que permite a comunicação do Java com qualquer tipo de banco de dados. O JDBC consiste em um conjunto de classes escritas em Java que provêm uma API padrão para desenvolvedores de base de dados tornando possível a escrita de aplicações usando bancos de dados inteiramente em Java. O JDBC suporta todos comandos SQL - *Structured Query Language* - tornando simples o envio de comandos SQL para sistemas de base de dados relacionais [FISHER *et al.*, 2003].

O arquivo ARFF segue o formato especificado na Tabela 2. O nome da base de dados serve exclusivamente para identificação. Na parte de definição de atributos deve-se definir todos os tipos de dados que a base de dados contém, dando um nome a cada um deles e definindo o seu tipo. Por fim deve haver a apresentação

dos dados em si. Eles serão apresentados cada um em uma linha após o uso do comando @*data*, sendo que cada atributo será separado por vírgula.

Informação	**Notação**
Nome	@relation "Nome Base de Dados"
Atributos	@attribute Nome_Atributo1 Tipo_Atributo1
	@attribute Nome_Atributo2 Tipo_Atributo2
	@attribute Nome_UltimoAtributo Tipo_UltimoAtributo
Dados	@data
Dado 1	Atributo1, Atributo2, Atributo_Final
Dado 2	Atributo1, Atributo2, Atributo_Final
Dado Final	Atributo1, Atributo2, Atributo_Final

Tabela 2: Uso de arquivos do tipo ARFF

A Figura 3 mostra a tela inicial do Weka, o *GUI Chooser*. Ela fornece as opções para o usuário utilizar o *Explorer*, o *Experimenter*, o *Knowledge Flow* e a *Simple CLI*. Todas estas opções oferecem interação com o usuário, sendo que a *Simple CLI* não possui interface gráfica. Dentre estas opções a mais utilizada é o *Explorer*, que pode ser visualizado na Figura 4. No *Explorer* o usuário carrega o arquivo com a base de dados a ser processada e depois pode escolher dentre vários algoritmos para aplicar em cima desta base. Após a execução de cada algoritmo é mostrado o resultado no próprio *Explorer*. Assim, o usuário pode salvar os resultados obtidos e visualizar as informações obtidas de variadas maneiras.

Figura 3: Weka *GUI Chooser*

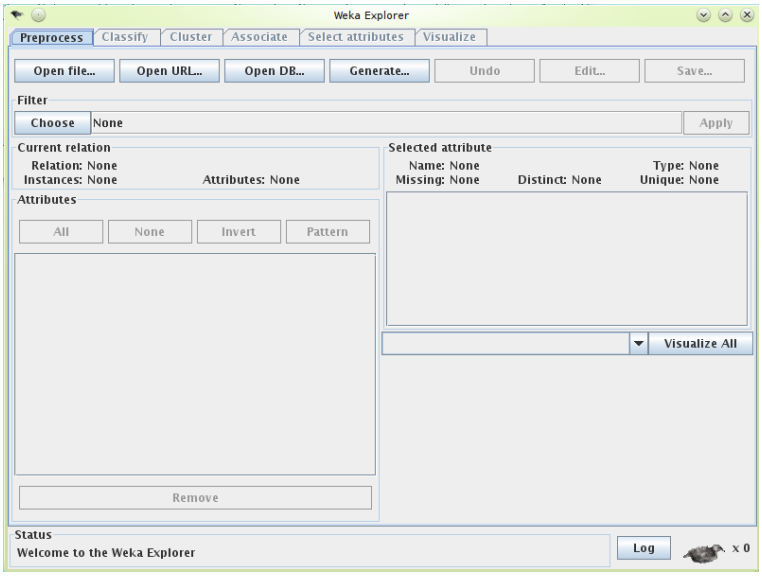

Figura 4: Organização do Weka Explorer

4.1 Ferramentas baseadas no Weka para mineração de dados distribuída

Por ser um *software* de código aberto e escrito em Java, o Weka facilita o desenvolvimento de outras ferramentas a partir dela. O código aberto permite que novos algoritmos sejam implementados e que os algoritmos já existentes no Weka sejam modificados. Já a vantagem de ser escrito em Java é que o Weka é facilmente portado, funcionando em cima de vários sistemas operacionais diferentes e com arquiteturas distintas.

Existem algumas ferramentas para realizar a mineração de dados distribuídas construídas baseando-se no Weka. Pode-se citar o Weka4WS [TALIA *et al.*, 2008], o Weka *Parallel* [CELIS; MUSICANT, 2002] e o *Grid-enabled* Weka [KHOUSSAINOV *et al.*, 2004]. Dentre elas o Weka4WS é a ferramenta mais completa por ser a única que disponibiliza a execução distribuída de todos os algoritmos implementados pelo Weka [PINTO, 2010], além de necessitar do uso do GT4 como plataforma para execução.

4.1.1 Weka4WS

O Weka4WS é uma extensão do Weka para realizar mineração de dados distribuída em ambiente de *Grid* e surgiu na Universidade de Calábria, Itália. As informações presentes neste capítulo são baseadas no trabalho dos desenvolvedores do *framework* [TALIA *et al.*, 2008]. A Figura 5 apresenta o *GUI Chooser* do Weka4WS.

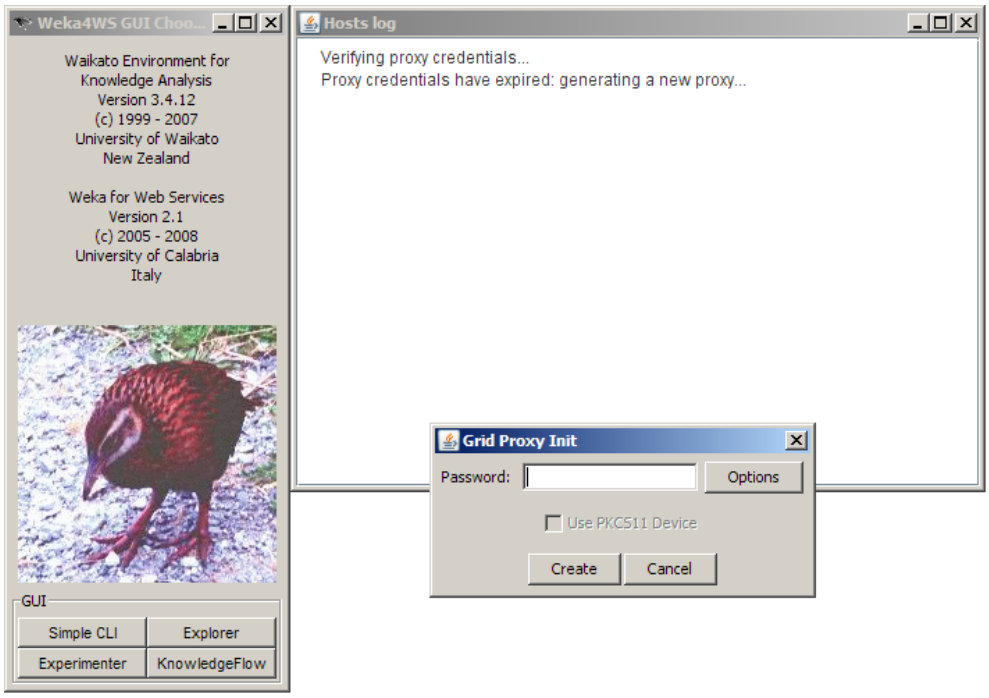

Figura 5: *Weka4WS GUI Chooser*

Com essa ferramenta torna-se real a possibilidade de executar tarefas de mineração de dados remotamente. Várias tarefas podem ser requisitadas de uma mesma máquina para executar em outras máquinas configuradas no ambiente de *Grid*. Os algoritmos do Weka4WS são os mesmos algoritmos sequenciais presentes na biblioteca Weka original, ou seja, os algoritmos não são modificados para realizar execução paralela ou distribuída. A vantagem do Weka4WS é a possibilidade de executar várias tarefas ao mesmo tempo, seja para fazer vários testes com uma mesma base de dados, seja para minerar bases de dados diferentes. Por exemplo, executar dois algoritmos de classificação diferentes para comparar seus resultados em uma base de dados muito grande demoraria muito tempo se a computação

fosse feita localmente, pois primeiro seria executado um algoritmo e em seguida o outro. Pode-se considerar que executar um algoritmo seja um tarefa e neste caso, a execução de dois algoritmos corresponde a duas tarefas que podem ser executadas de forma independente em máquinas diferentes. Desta forma, quando cada tarefa é enviada para uma máquina diferente do *Grid* o tempo de execução total tende a cair pela metade.

O Weka4WS adota a tecnologia *Web Services Resource Framework* (WSRF), utilizando bibliotecas WSRF para a execução de algoritmos de mineração de dados distribuída e para gerenciar a computação distribuída. A tecnologia WSRF define várias especificações técnicas de como deve ser implementado um WS (*Web Service*). Esta tecnologia implementa a filosofia da *Open Grid Services Architecture* (OGSA) que define um modelo arquitetural para sistemas de *Grid* em que recursos distribuídos e aplicações são modelados como WS que interagem entre eles utilizando padrões baseados na Internet.

Este *framework* executa em cima da biblioteca Java WSRF implementada no Globus *toolkit* 4.0.X [3][FOSTER, 2005], então para seu funcionamento é necessário que o GT4.0.X esteja instalado e configurado nas máquinas do *Grid*.

No Weka4WS todos os nós usam os serviços do GT4.0.X para executarem funcionalidades padrões de *Grid*. Esses nós podem ser separados em duas categorias: *user nodes* e *computing nodes*. *User nodes* são as máquinas locais dos usuários, as quais provêem o *software* cliente do Weka4WS. Já os *computing nodes* são as máquinas que provêem os WS e o software de serviço do Weka4WS, permitindo a execução de tarefas de mineração de dados remotamente.

[3] A versão 4.0 do Globus *toolkit* possui nove subversões, de 4.0.0 a 4.0.8. Por GT4.0.X leia-se qualquer versão dentre estas nove.

Os módulos cliente e de serviço do Weka4WS são respectivamente chamados de *Weka4WS-client* e *Weka4WS-service*. Como citado anteriormente o *Weka4WS-client* contém o *software* cliente para execução nos *user nodes*, enquanto o *Weka4WS-service* possui os WS para serem instalados nos *computing nodes*.

A instalação do Weka4WS necessita da instalação completa do GT4.0.X nos *computing nodes* e da instalação do *Java WS Core* nos *user nodes* para ser bem sucedida. As versões 4.2 e 5.0 (a mais atual) do Globus *toolkit* contêm algumas atualizações nas especificações dos WS, o que as tornam incompatíveis com o Weka4WS. O módulo *Weka4WS-service* pode ser instalado somente em plataformas *Unix-like* por ser a plataforma de execução do GT4, enquanto que o *Weka4WS-client* pode ser instalado tanto em *Unix-like* quanto em Windows.

Quanto a segurança, o Weka4WS possui dois pré-requisitos que são indispensáveis para seu funcionamento:

1. Um usuário deve ter um certificado com o proxy válido no formato X.509;
2. O arquivo "/etc/grid-security/grid-mapfile" nos *computing nodes* devem conter uma entrada para mapear o cliente usuário Weka4WS para um usuário local no *computing node*.

A Figura 6 mostra os componentes presentes no Weka4WS e a maneira como eles se comunicam.

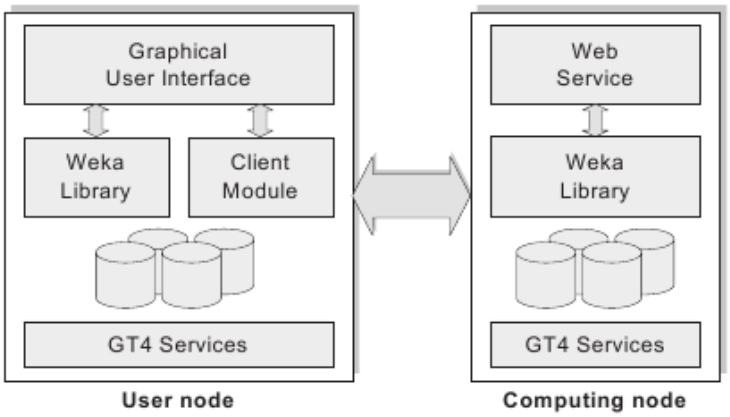

Figura 6: Componentes do Weka4WS [TALIA *et al.*, 2008]

O *computing node* é formado por dois componentes: WS e Weka *Library* (WL). O WS disponibiliza todos os algoritmos de mineração de dados implementados na WL. As chamadas para o WS são executadas chamando os algoritmos presentes na WL.

O *user node* é formado por três componentes: a WL, um módulo cliente e uma interface gráfica de comunicação com o usuário - GUI (*Graphical User Interface*). A GUI é uma versão extendida da GUI do Weka para suportar a execução tanto de tarefas de mineração de dados locais quanto de tarefas distribuídas remotamente. As tarefas locais são realizadas invocando a WL local, enquanto as tarefas remotas são controladas pelo módulo cliente, que opera como um intermediário entre a GUI e o WS em nós computacionais remotos. A Figura 7 mostra uma imagem do Weka4WS *Explorer*. Na caixa azul, são mostrados todos os nós ativos no ambiente de *Grid*, podendo-se selecionar qualquer um destes para a execução de um algoritmo. O botão *reload hosts* recarrega os nós, atualizando a lista de nós ativos no

Grid. Já o botão *proxy* é usado para gerar um novo *proxy*, possibilitando a conexão com o ambiente de *Grid* caso esta seja perdida.

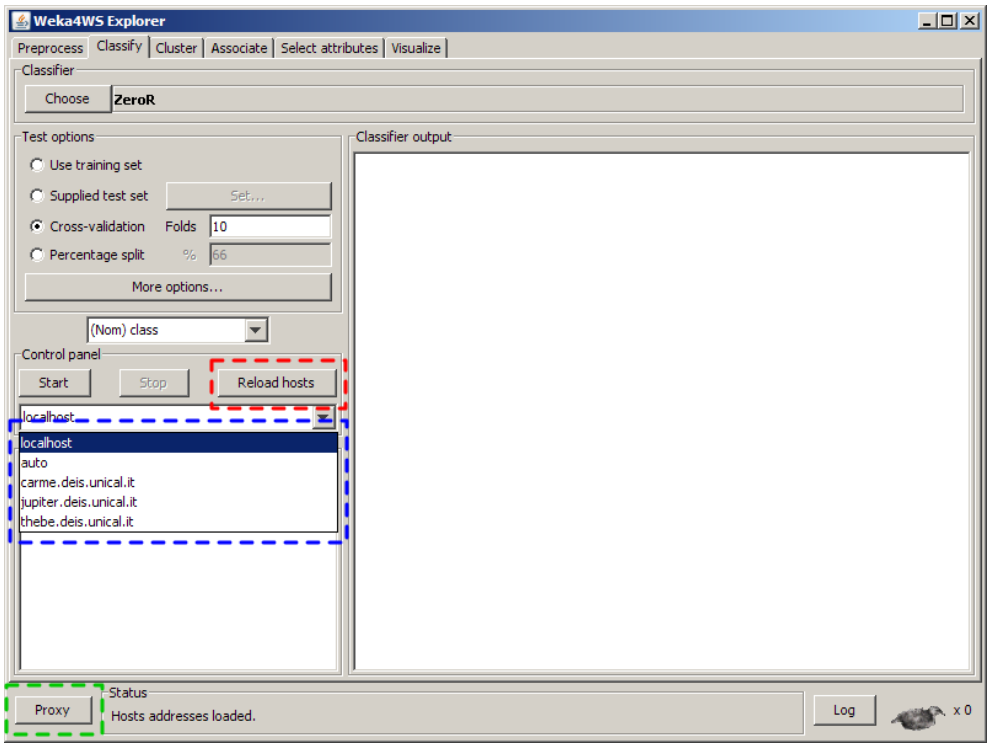

Figura 7: Weka4WS *Explorer* [TALIA *et al.*, 2008]

4.1.2 Weka *Parallel*

O Weka *Parallel* foi desenvolvido para executar algoritmos de mineração de dados possibilitando a paralelização da tarefa de validação cruzada (*cross validation*) em um ambiente distribuído. Somente o algoritmo da tarefa de validação cruzada foi alterado em relação aos algoritmos do Weka. Dessa forma, somente algoritmos que utilizem a validação cruzada vão alcançar um ganho de desem-

penho nesta ferramenta. A forma utilizada para realizar a comunicação entre as máquinas, possiblitando a distribuição da tarefa de validação cruzada, foi realizar uma conexão simples usando a classe *Socket* disponível no pacote java.net [CELIS; MUSICANT, 2002]. Portanto, o Weka *Parallel* não necessita de nenhum tipo de *software* externo para auxiliar na sua execução.

Uma sessão do Weka *Parallel* executa em uma única máquina cliente e vários servidores distribuídos. Cada servidor executa o *software* que permanece escutando a porta de comunicação vasculhando requisições do cliente. Para o cliente saber para qual nó ele enviará cada parte da validação cruzada é utilizado um algoritmo *round-robin*. Com esta técnica o primeiro subconjunto da validação cruzada é enviado para um dos nós disponível, depois o segundo e assim por diante. Quando todos os subconjuntos tiverem sido enviados, o algoritmo é reiniciado, atribuindo cada subconjunto que não tenha sido concluído. Isso continua até que a validação cruzada termine. O algoritmo é executado novamente para garantir que todos os subconjuntos foram executados com sucesso. Isto ocorre pois podem ocorrer problemas durante a execução das tarefas como falhas na rede ou nas máquinas [CELIS; MUSICANT, 2002].

Para o módulo cliente do Weka *Parallel* trabalhar corretamente é necessário que se crie um arquivo de configuração chamado *.weka-parallel*, especificando a porta em que as instâncias dos servidores distribuídos irão escutar e o endereço de cada um dos servidores. Se um servidor tiver mais de um núcleo pode-se repetir o endereço dele no arquivo de configuração, requisitando assim que este servidor execute mais de uma tarefa. A Tabela 3 apresenta um exemplo do arquivo de configuração para o Weka *Parallel*.

PORT = <Número da porta em que todos servidores estarão escutando>
<Endereço IP do computador 1>
<Endereço IP do computador 1>
<Endereço IP do computador 2>
<Endereço IP do computador 3>

Tabela 3: Arquivo de configuração *.weka-parallel* do *Weka Parallel*

Note que o endereço IP do computador 1 aparece duas vezes na tabela acima. Neste caso duas tarefas seriam enviadas para este endereço e apenas uma para os computadores 2 e 3, que estão escutando requisições através da porta definida na primeira linha do arquivo.

Esta ferramenta é útil em ambiente de *Grid* somente para acelerar a mineração de dados quando se usa a técnica de validação cruzada para avaliar a eficiência do algoritmo executado. Esta é uma técnica amplamente utilizada para validação de dados gerados por algoritmos de classificação. É necessário especificar em quantos subconjuntos será dividida a validação cruzada, do total de subconjuntos um deles é usado para testar os dados gerados por todos os outros.

4.1.3 *Grid* Weka

O *Grid* Weka ou *Grid-enabled* Weka foi desenvolvido na Universidade de Dublin, Irlanda e é uma ferramenta derivada do Weka *Parallel* [VALENTEM *et al.*, 2005]. Ela permite a distribuição das tarefas de mineração de dados entre vários computadores utilizando um *grid ad-hoc*. Seus algoritmos são os mesmos do Weka, a grande mudança é a possibilidade da distribuição das tarefas, que assim como no Weka *Parallel* são possíveis através das conexões entre as máquinas uti-

lizando a classe*Socket* disponível no pacote java.net. Dois componentes são muito importantes para o *Grid* Weka, o servidor Weka e o cliente Weka. O servidor Weka é baseado no servidor Weka original, sendo que cada máquina participando do *Grid* executa o componente servidor. O cliente Weka é responsável por aceitar uma tarefa de aprendizado, dados de entrada de um usuário e por distribuir as tarefas no *Grid* [KHOUSSAINOV *et al.*, 2004].

O *Grid* Weka, assim como o Weka *Parallel* não necessita de um *middleware* para execução, porém possui outros recursos. Pode-se citar dois destes recursos: execução remota do treinamento de um modelo de classificação e execução remota e paralela da validação cruzada e do teste de um modelo de classificação [PINTO *et al.*, 2011].

A exemplo do Weka *Parallel* também existe a necessidade de se criar o arquivo de configuração chamado *.weka-parallel*. A configuração do arquivo no *Grid* Weka é um pouco diferente e também é possível que computadores com mais de um núcleo executem mais de uma tarefa ao mesmo tempo. A Tabela 4 mostra como seria a configuração do arquivo *.weka-parallel* no nó cliente do *Grid*.

PORT = <Número da porta em que todos servidores estarão escutando>
<Endereço IP do computador 1> <Número de servidores Weka rodando no computador 1> <Quantidade máxima de memória no computador 1>
<Endereço IP do computador 2> <Número de servidores Weka rodando no computador 2> <Quantidade máxima de memória no computador 2>

Tabela 4: Arquivo de configuração *.weka-parallel* do *Grid Weka*

5 METODOLOGIA

Foram feitos experimentos com o intuito de encontrar a ferramenta mais rápida e eficiente para mineração de dados distribuída. Para isto, foram utilizadas as ferramentas Weka4WS e Weka *Parallel*, além do Weka que serviu como base comparativa para os resultados. O *Grid* Weka não foi utilizado pois não foi possível executar as tarefas distribuídas a partir dele, possivelmente por problemas de incompatibilidade com *hardwares* mais recentes [PINTO, 2010], tendo em vista que ele é um *software* que foi desenvolvido em 2004 [KHOUSSAINOV *et al.*, 2004]. Já o Weka *Parallel* apesar de ser a base do *Grid* Weka e um *software* desenvolvido em 2003, não apresentou nenhum tipo de problema de incompatibilidade.

Foi realizada a instalação do servidor do Weka4WS e do Weka *Parallel* em três máquinas, sendo que uma destas três máquinas também funcionou como nó cliente com a finalidade de enviar as tarefas a serem executadas para os nós servidores. Como o Weka4WS depende do Globus, este *middleware* foi previamente instalado nas três máquinas.

Um passo a passo da instalação do Globus pode ser encontrada no trabalho de conclusão de curso do aluno Caio César Roma Menten [MENTEN, 2012]. Já a instalação do Weka4WS está detalhada no apêndice A e a instalação do Weka *Parallel* no apêndice B.

As configurações dos computadores utilizados estão mostradas na Tabela 5.

Nome	Processador	Velocidade	Memória Física	Memória Cache	Tarefa
node1.ufla.br	Intel Core 2 Quad	2,66GHz	4GB	2MB	Cliente/Servidor
node2.ufla.br	Intel Core 2 Duo	2GHz	2GB	2MB	Servidor
node3.ufla.br	AMD Athlon X2	1GHz	1GB	512KB	Servidor

Tabela 5: Características dos computadores utilizados

Por ser a máquina mais potente do *Grid*, a “node1.ufla.br” foi escolhida para realizar a dupla tarefa de cliente e servidor, além de ter sido a máquina em que foram testadas as bases executadas no Weka original.

As máquinas ficam em uma rede local, interligadas por cabos conforme o esquema mostrado na Figura 8.

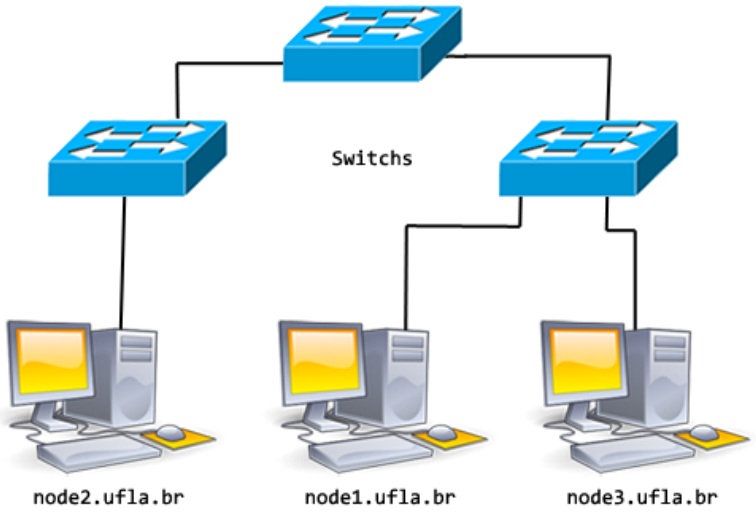

Figura 8: Representação do ambiente utilizado

Note que as máquinas “node1.ufla.br” e “node3.ufla.br” estão conectadas diretamente ao mesmo *switch*, enquanto a máquina “node2.ufla.br” está ligada em outro *switch*. Estes dois *switchs* são ligados a um mesmo *switch* que permite a comunicação entre as três máquinas. Isto pode fazer com que a comunicação do “node1.ufla.br” e do “node3.ufla.br” com o “node2.ufla.br” fique mais lenta pois podemos encontrar interferências na rede durante a transmissão dos dados.

As bases de dados foram escolhidas de forma que tamanhos variados de bases fossem testadas. Testando bases de tamanhos diferentes pode-se obter melhores conclusões sobre quando é realmente útil usar o sistema distribuído em *Grid* para realizar a mineração de dados e quando é melhor usar a mineração de dados localmente. Criou-se cinco conjuntos de testes, cada um com três bases de dados. A comparação entre as ferramentas foi feita com base no tempo total de execução de cada conjunto e também atentando a eficiência do algoritmo quando executado distribuidamente e localmente. Para cada conjunto em cada ferramenta foram feitas cinco execuções. Portanto, o tempo total de execução apresentado no Capítulo 6 é uma média destas execuções. Além da média do tempo de execução foi calculado também o intervalo de confiança.

Abaixo estão informações sobre cada base de dados utilizada. Para selecioná-las foram obtidas várias bases de dados e cada uma foi executada no Weka original. Conforme o tempo de execução e o tamanho de cada base de dados foram formados os cinco conjuntos, sendo que o conjunto 1 ficou com as bases de dados mais demoradas para execução, o conjunto 2 com as bases de dados um pouco menos demoradas até o conjunto 5 que ficou com as bases de dados executadas mais rapidamente. Isto foi feito para verificar a eficiência do ambiente de *Grid* tanto para bases de dados pequenas quanto para bases de dados maiores. Estas bases foram obtidas em [GROUP, 2012] e [WAIKATO, 2012a].

- Conjunto 1

  fbis: Base derivada de conferências da TREC (*Text Retrieval Conference*) [NIST, 2012]. Possui 2463 instâncias e 2001 atributos.

re0: Faz parte da coleção de bases de dados *Reuters-21578*, que possui bases para categorização de textos [GROUP; REUTERS, 2012]. Possui 1504 instâncias e 2887 atributos.

tr41: Base derivada de conferências da TREC (*Text Retrieval Conference*) [NIST, 2012]. Possui 878 instâncias e 7455 atributos.

- Conjunto 2

Sobre: As três bases de dados deste conjunto de teste fazem parte das bases da coleção OHSUMED [HERSH *et al.*, 1994].

oh0: Possui 1003 instâncias e 3183 atributos.

oh10: Possui 1050 instâncias e 3239 atributos.

oh15: Possui 913 instâncias e 3101 atributos.

- Conjunto 3

oh5: É uma base da coleção OHSUMED [HERSH *et al.*, 1994]. Possui 918 instâncias e 3013 atributos.

tr11: Base derivada de conferências da TREC (*Text Retrieval Conference*) [NIST, 2012]. Possui 414 instâncias e 6430 atributos.

tr21: Base derivada de conferências da TREC (*Text Retrieval Conference*) [NIST, 2012]. Possui 336 instâncias e 7903 atributos.

- Conjunto 4

tr12: Base derivada de conferências da TREC (*Text Retrieval Conference*) [NIST, 2012]. Possui 313 instâncias e 5805 atributos.

ECML: Esta base de dados foi preparada depois de se obter e processar várias informações do *site SAGEmap* em Dezembro de 2002. Possui 90 instâncias e 27680 atributos.

tr23: Base derivada de conferências da TREC (*Text Retrieval Conference*) [NIST, 2012]. Possui 204 instâncias e 5833 atributos.

- Conjunto 5

Sobre: As três bases deste conjunto consistem de funcionalidades de dígitos manuscritos extraídos de uma coleção de mapas de utilidades holandeses. Os dígitos são representados em seis arquivos de bases de dados, sendo que três destes arquivos são os utilizados neste conjunto de teste.

mfeat-factors: Possui 2000 instâncias e 217 atributos.

mfeat-fourier: Possui 2000 instâncias e 77 atributos.

mfeat-zernike: Possui 2000 instâncias e 48 atributos.

As Figuras 9 e 10 mostradas a seguir representam a configuração utilizada nos nós clientes do Weka4WS e do Weka *Parallel*, respectivamente.

```
usuario1@node1:~/Alessandro/weka4ws-client-2.1$ cat etc/machines
# Weka4WS configuration file
# http://grid.deis.unical.it/weka4ws/
# Copyright (C) 2008 University of Calabria
# Dept. of Electronics, Computer Science and Systems

# logging = 0 run the service without logging
# logging = 1 run the service with logging
#
# ==================== computing node ==========================
# hostname             | container port | gridFTP port | logging
node1.ufla.br           8443             2811          0
node2.ufla.br           8443             2811          0
node3.ufla.br           8443             2811          0
```

Figura 9: Configuração utilizada no nó cliente do Weka4WS

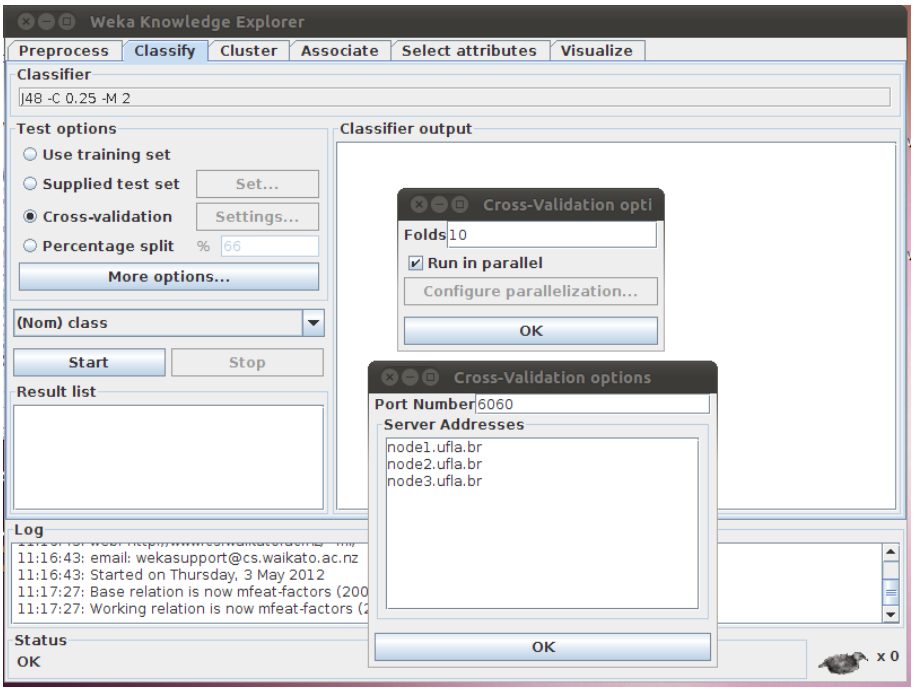

Figura 10: Configuração utilizada no nó cliente do Weka *Parallel*

Observando as Figuras 9 e 10 percebe-se que as configurações a serem feitas no nó cliente das duas ferramentas é relativamente simples. Basicamente deve-se especificar os nós a serem usados para a computação distribuída e a porta em que os respectivos serviços estarão executando.

O algoritmo de mineração de dados utilizado para testar o funcionamento das ferramentas baseadas no Weka para mineração de dados distribuída em ambiente de *Grid* foi o J48, que é a implementação do algoritmo C4.5 *release* 8 na linguagem Java. Ele constrói um modelo de árvore de decisão baseado em um conjunto de dados de treinamento e usa esse modelo para classificar outras instâncias em um conjunto de teste [MACHADO *et al.*, 2006].

Para verificar a eficiência do algoritmo sobre a base de dados testada foi utilizado o *10-fold-cross-validation*, ou seja, a validação cruzada com dez subconjuntos. Este é um método usado para avaliação da eficiência. Quando usado com dez subconjuntos, toda a base de dados é aleatoriamente separada em dez subconjuntos de tamanhos aproximadamente iguais. O modelo de classificação é treinado e testado 10 vezes. Em cada iteração nove subconjuntos são usados para treinamento e o subconjunto restante é utilizado para teste [DELEN *et al.*, 2005]. Existem estudos que mostram que 10 subconjuntos é um número que possibilita um teste bastante eficiente para a validação cruzada [KOHAVI, 1995].

O Weka4WS tem a funcionalidade de executar as 3 tarefas ao mesmo tempo, enviando cada tarefa para um nó do *Grid*, enquanto o Weka *Parallel* não tem tal funcionalidade porém paraleliza a parte de validação cruzada dos algoritmos de classificação. Por isso todas as bases foram testadas aplicando o algoritmo J48 com o método de validação cruzada.

6 RESULTADOS E DISCUSSÃO

As tarefas foram todas executadas conforme proposto no Capítulo 5 sem problemas. As Figuras 11 e 12 mostram a execução de tarefas pelo Weka4WS e Weka *Parallel*, respectivamente.

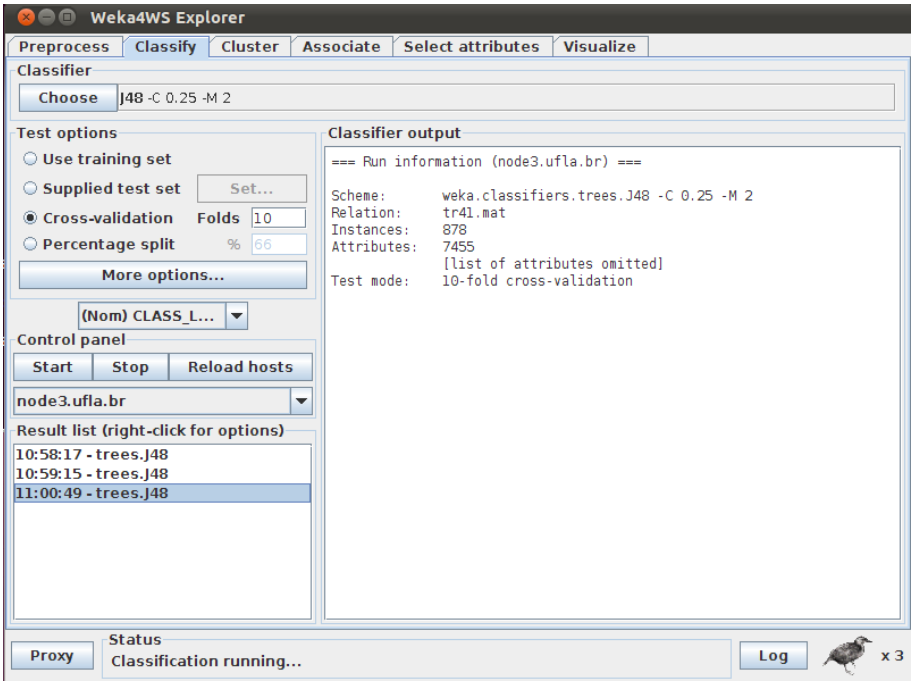

Figura 11: Execução de tarefas no Weka4WS

Repare que na parte inferior direita da Figura 11 aparece o número de tarefas (3) que está sendo executada pelo Weka4WS. No momento cada um dos nós estava executando o algoritmo J48 com o *10-fold-cross-validation* utilizando uma base de dados diferente.

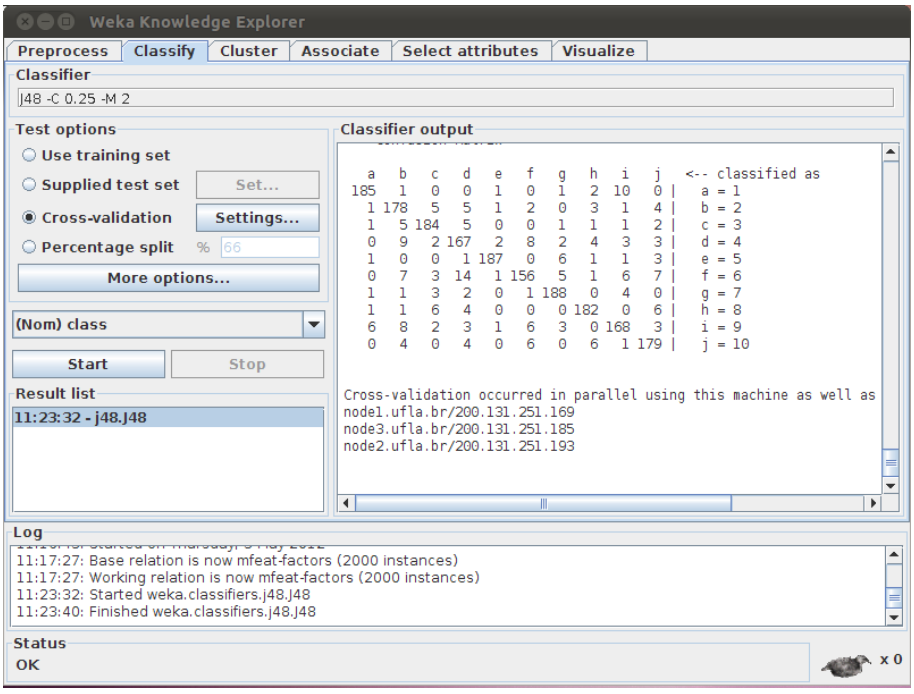

Figura 12: Execução de tarefas no Weka *Parallel*

Na Figura 12 pode-se observar na área do *Classifier Output* o resultado da execução de uma tarefa. Após imprimir todos os resultados, são mostradas na tela as informações sobre quais nós foram utilizados para paralelizar a validação cruzada.

As Tabelas 6, 7, 8, 9 e 10 mostram os resultados para cada base de dados em cada ferramenta testada. O Weka4WS sempre têm entre parentêsis o nome da máquina na qual ele foi executado. Isto ocorre pois a execução de cada conjunto de bases de dados no Weka4WS é feita de uma só vez, disparando-se as três tarefas ao mesmo tempo, enquanto que no caso do Weka *Parallel* e do Weka cada tarefa

deve terminar primeiro para iniciarmos a próxima. A seguir é feita uma explicação de cada coluna presente nessas Tabelas.

Base de Dados: É o nome da base de dados utilizada.

Ferramenta: O nome da ferramenta em que a base de dados foi testada.

Tempo Gasto $\pm$ IC: Mostra o tempo total médio para execução da tarefa proposta mais ou menos a média do tempo calculado para o intervalo de confiança.

Elementos: Mostra o número de instâncias e o número de atributos que cada base de dados possui usando o formato: (número de instâncias / número de atributos).

Acerto: Mostra a porcentagem de instâncias classificadas corretamente nos testes a partir do modelo gerado.

Base de Dados	Ferramenta	Tempo Gasto $\pm$ IC	Elementos	Acerto
fbis	Weka	36min31seg $\pm$ 9min23seg	2463/2001	72,8%
fbis	Weka4WS("node1.ufla.br")	34min53seg $\pm$ 15min25seg	2463/2001	72,8%
fbis	Weka *Parallel*	14min44seg $\pm$ 4min19seg	2463/2001	73,2%
re0	Weka	23min $\pm$ 1min18seg	1504/2887	75,2%
re0	Weka4WS("node2.ufla.br")	30min34seg $\pm$ 14min52seg	1504/2887	75,2%
re0	Weka *Parallel*	9min7seg $\pm$ 2min42seg	1504/2887	75,2%
tr41	Weka	13min10seg $\pm$ 2min10seg	878/7455	90,8%
tr41	Weka4WS("node3.ufla.br")	16min31seg $\pm$ 9min11seg	878/7455	90,8%
tr41	Weka *Parallel*	5min24seg $\pm$ 56seg	878/7455	92,9%

Tabela 6: Execuções do conjunto de teste 1

Base de Dados	Ferramenta	Tempo Gasto ± IC	Elementos	Acerto
oh0	Weka	10min47seg ± 7seg	1003/3183	80,4%
oh0	Weka4WS("node1.ufla.br")	10min22seg ± 1min42seg	1003/3183	80,4%
oh0	Weka *Parallel*	5min17seg ± 8min20seg	1003/3183	80,9%
oh10	Weka	13min44seg ± 20seg	1050/3239	72,9%
oh10	Weka4WS("node2.ufla.br")	19min35seg ± 9min27seg	1050/3239	72,9%
oh10	Weka *Parallel*	6min7seg ± 36seg	1050/3239	73,4%
oh15	Weka	11min19seg ± 28seg	913/3101	74,8%
oh15	Weka4WS("node3.ufla.br")	13min53seg ± 9min12seg	913/3101	74,8%
oh15	Weka *Parallel*	5min ± 52seg	913/3101	75,2%

Tabela 7: Execuções do conjunto de teste 2

Base de Dados	Ferramenta	Tempo Gasto ± IC	Elementos	Acerto
oh5	Weka	7min43seg ± 1min33seg	918/3013	82,2%
oh5	Weka4WS("node1.ufla.br")	7min9seg ± 1min8seg	918/3013	82,2%
oh5	Weka *Parallel*	3min5seg ± 1min13seg	918/3013	80,6%
tr11	Weka	6min13seg ± 1min3seg	414/6430	77,2%
tr11	Weka4WS("node2.ufla.br")	8min52seg ± 5min48seg	414/6430	77,2%
tr11	Weka *Parallel*	2min14seg ± 52seg	414/6430	77,2%
tr21	Weka	7min25seg ± 41seg	336/7903	79,1%
tr21	Weka4WS("node3.ufla.br")	8min27seg ± 5min24seg	336/7903	79,1%
tr21	Weka *Parallel*	2min57seg ± 59seg	336/7903	80,3%

Tabela 8: Execuções do conjunto de teste 3

Base de Dados	Ferramenta	Tempo Gasto ± IC	Elementos	Acerto
ECML	Weka	1min47seg ± 37seg	90/27680	8,8%
ECML	Weka4WS(“node2.ufla.br”)	2min29seg ± 9min12seg	90/27680	8,8%
ECML	Weka *Parallel*	45seg ± 4seg	90/27680	10%
tr12	Weka	3min32seg ± 1min46seg	313/5805	82,7%
tr12	Weka4WS(“node1.ufla.br”)	3min50seg ± 2min54seg	313/5805	82,7%
tr12	Weka *Parallel*	1min14seg ± 7seg	313/5805	80,8%
tr23	Weka	1min18seg ± 40seg	204/5833	91,6%
tr23	Weka4WS(“node3.ufla.br”)	1min21seg ± 53seg	204/5833	91,6%
tr23	Weka *Parallel*	26seg ± 4seg	204/5833	92,1%

Tabela 9: Execuções do conjunto de teste 4

Base de Dados	Ferramenta	Tempo Gasto ± IC	Elementos	Acerto
mfeat-factors	Weka	14seg ± 0	2000/217	88,9%
mfeat-factors	Weka4WS(“node1.ufla.br”)	16seg ± 8seg	2000/217	88,9%
mfeat-factors	Weka *Parallel*	6seg ± 7seg	2000/217	88,7%
mfeat-fourier	Weka	9seg ± 3seg	2000/77	75,2%
mfeat-fourier	Weka4WS(“node2.ufla.br”)	18seg ± 36seg	4000/2000	75,2%
mfeat-fourier	Weka *Parallel*	4seg ± 3seg	4000/2000	75,7%
mfeat-zernike	Weka	6seg ± 3seg	2000/48	71,8%
mfeat-zernike	Weka4WS(“node3.ufla.br”)	14seg ± 7seg	2000/48	71,8%
mfeat-zernike	Weka *Parallel*	2seg ± 3seg	2000/48	71,3%

Tabela 10: Execuções do conjunto de teste 5

Analisando as Tabelas 6, 7, 8, 9 e 10, verificamos que o Weka *Parallel* acelera bem o tempo para execução da tarefa de cada base. Também é importante notar que a porcentagem de acerto do Weka e do Weka4WS foram iguais para todas as bases de dados, comprovando assim que não há modificação nos algoritmos do Weka4WS em relação aos algoritmos do Weka. O Weka *Parallel* tem a porcentagem de acerto sempre diferente do Weka e do Weka4WS, porém a margem de variação é sempre muito pequena. Isto possivelmente ocorre pois a implementa-

ção do algoritmo J48 no Weka *Parallel* tem pequenas diferenças em relação as implementações dos algoritmos J48 do Weka e do Weka4WS já que a versão do Weka utilizado no desenvolvimento do Weka *Parallel* é mais antiga em relação às versões utilizadas do Weka e do Weka4WS. Estas diferenças foram verificadas comparando-se os códigos dos algoritmos J48 de cada ferramenta.

A Tabela 11 mostra o resultado obtido por cada conjunto de base de dados utilizado. A seguir é feita uma breve explicação do significado de cada coluna da Tabela 11.

Conjunto: Representa o número do conjunto de bases de dados executado.

Ferramenta: Apresenta a ferramenta na qual foi executada a tarefa.

Tempo Total $\pm$ IC: Mostra o tempo total médio gasto para execução de todo o conjunto mais ou menos a média do tempo calculado para o intervalo de confiança de todo o conjunto.

Conjunto	Ferramenta	Tempo Total ± IC
1	Weka	72min42seg ± 8min31seg
1	Weka4WS	34min53seg ± 15min25seg
1	Weka *Parallel*	29min16seg ± 7min50seg
2	Weka	35min51seg ± 45seg
2	Weka4WS	19min35seg ± 9min27seg
2	Weka *Parallel*	16min25seg ± 6min8seg
3	Weka	21min21seg ± 3min17seg
3	Weka4WS	8min52seg ± 5min48seg
3	Weka *Parallel*	8min17seg ± 3min1seg
4	Weka	6min38seg ± 3min2seg
4	Weka4WS	3min50seg ± 2min54seg
4	Weka *Parallel*	2min26seg ± 12seg
5	Weka	30seg ± 5seg
5	Weka4WS	18seg ± 36seg
5	Weka *Parallel*	13seg ± 10seg

Tabela 11: Tempo total das execuções de cada conjunto de teste em cada ferramenta

Os resultados da Tabela 11 mostram que geralmente há ganho de tempo quando se utiliza a mineração de dados distribuída em ambiente de *Grid*. No que diz respeito ao intervalo de confiança, em geral o Weka *Parallel* possui os menores intervalos de confiança, enquanto o Weka4WS possui os maiores. Isso mostra que o Weka4WS teve bastante variação no tempo total da execução dos conjuntos, mas ainda sim, o Weka4WS diminui muito o tempo de execução em relação ao Weka. Comparando-se o Weka4WS com o Weka *Parallel* percebe-se que o Weka *Parallel* mesmo não executando as três tarefas ao mesmo tempo foi superior em todos os conjuntos de testes, o que sugere que paralelizar a tarefa de validação cruzada é um método bem interessante para acelerar a computação. Porém, devido ao uso restrito do Weka *Parallel*, o Weka4WS aparece como solução para outros tipos de tarefas de mineração de dados, podendo acelerar bastante a computação também

para os algoritmos de regressão, associação e *clusterização*. Isto seria possível pois todos os algoritmos presentes no Weka também estão no Weka4WS e não há diferença nos algoritmos de uma ferramenta para a outra. Além disso, como o Weka4WS usa como plataforma para execução o *middleware* Globus, ele tem algumas vantagens, como por exemplo um maior controle sobre os usuários que acessam o ambiente de *Grid*, pois todos têm que possuir um certificado assinado digitalmente pela entidade certificadora responsável pelo *Grid*.

Com relação ao tamanho dos arquivos, percebe-se que quanto maior a base de dados a ser enviada para a tarefa de mineração de dados, maior é o ganho em um ambiente distribuído quando comparado ao ambiente sequencial, mesmo computando-se o tempo de transferência da base de dados de uma máquina para outra. Portanto, acaba sendo muito vantajoso realizar mineração de dados distribuída em ambiente de *Grid* quando se tem bases de dados grandes.

Repare que os tempos de cada base executada individualmente no Weka4WS sempre crescem bastante em relação ao Weka quando a tarefa é enviada ao "node2.ufla.br". Isto ocorre pela disposição desta máquina na rede como mostrado no Capítulo 5. Esta organização leva a informação a percorrer um caminho maior na transferência do cliente para o servidor e vice-versa.

7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou um estudo sobre ferramentas baseadas no Weka para mineração de dados distribuída em ambiente de *Grid*. Foram feitos estudos sobre três ferramentas: (a) Weka4WS, (b) Weka *Parallel*, (c) *Grid* Weka, além de estudos sobre ambientes de *Grid* e mineração de dados. Para se fazer os testes deste trabalho foram utilizadas apenas as ferramentas (a) e (b). Comparou-se a eficiência e a rapidez na execução das tarefas nessas duas ferramentas em relação ao Weka original.

Os resultados mostraram que o uso da computação distribuída em ambiente de *Grid* é muito eficiente para tarefas de mineração de dados. Foram obtidos resultados com uma porcentagem de acerto similar a do Weka e com o tempo de execução total dos conjuntos bem menor. A paralelização da tarefa de validação cruzada representa uma estratégia interessante para diminuir o tempo de execução de algoritmos de classificação, como verificado com o uso do Weka *Parallel*. Já o Weka4WS se mostra interessante por também ter um bom ganho de tempo em relação ao Weka, mas também por possibilitar a execução de qualquer algoritmo do Weka, possivelmente permitindo o ganho de tempo para se executar tarefas com qualquer algoritmo do Weka, ao contrário do Weka *Parallel* que representa um ganho computacional somente quando é executado algoritmos de classificação com validação cruzada. O Weka4WS também possui a vantagem de usufruir das funcionalidades do *middleware* Globus, que pode ajudar a complementar funções que faltem nesta ferramenta. Um exemplo disso é o uso de certificados digitais para ter controle de quem pode enviar tarefas para outras máquinas, funcionalidade que com o Weka *Parallel* não é possível, fazendo com que este torne-se bastante inseguro.

Apesar da alta variabilidade dos tempos de execuções no Weka4WS, ele ainda se mostra bem mais rápido que o Weka, fazendo com que seu uso seja viabilizado. Já o Weka Parallel tem uma variabilidade menor nos dados e foi a ferramenta mais rápida nas execuções, mostrando-se uma ótima opção para acelerar tarefas de classificação usando validação cruzada.

Como trabalho futuro sugere-se:

- Implantar um ambiente de *Grid* utilizando mais máquinas, possibilitando um ganho computacional ainda maior em relação ao conseguido neste trabalho;
- Implantar um ambiente de *Grid* com máquinas que não estão conectadas em rede local para verificar o comportamento do ambiente;
- Testar o Weka4WS com outros algoritmos de mineração de dados pra verificar a eficiência do mesmo quando executado com outros algoritmos.

REFERÊNCIAS

ALLIANCE, G. *Globus Toolkit*. Acesso em: Outubro 2011. Disponível em: <http://www.globus.org/toolkit/docs/4.0/>.

AOUAD, L.; PETITON, S.; SATO, M. Grid and cluster matrix computation with persistent storage and out-of-core programming. In: IEEE. *Cluster Computing, 2005. IEEE International*. [S.l.], 2005. p. 1–9.

BOOTH, D.; HAAS, H.; MCCABE, F.; NEWCOMER, E.; CHAMPION, M.; FERRIS, C.; ORCHARD, D. Web services architecture, w3c working group note 11 february 2004. *World Wide Web Consortium, article available from: http://www. w3. org/TR/ws-arch*, p. 13, 2004.

BRUNK, C.; KELLY, J.; KOHAVI, R. Mineset: An integrated system for data mining. In: *Proceedings of the third international conference on Knowledge Discovery and Data Mining*. [S.l.: s.n.], 1997. p. 135–138.

BURKE, S.; CAMPANA, S.; LANCIOTTI, E.; LITMAATH, M.; MÉNDEZ, P.; MICCIO, V.; NATER, C.; SANTINELLI, R.; SCIABÁ, A. *gLite 3.2 User Guide*. 2011.

CABENA, P.; CHOI, H.; KIM, I.; OTSUKA, S.; REINSCHMIDT, J.; SAARENVIRTA, G. Intelligent miner for data applications guide. *IBM RedBook SG24-5252-00*, 1999.

CELIS, S.; MUSICANT, D. Weka-parallel: machine learning in parallel. In: CITESEER. *Carleton College, CS TR*. [S.l.], 2002.

COPPOLA, M.; JÉGOU, Y.; MATTHEWS, B.; MORIN, C.; PRIETO, L.; SÁNCHEZ, O.; YANG, E.; YU, H. Virtual organization support within a

grid-wide operating system. *Internet Computing, IEEE*, IEEE, v. 12, n. 2, p. 20–28, 2008.

DELEN, D.; WALKER, G.; KADAM, A. Predicting breast cancer survivability: a comparison of three data mining methods. *Artificial intelligence in medicine*, Elsevier, v. 34, n. 2, p. 113–127, 2005.

EGEE. *EGEE Portal*. Acesso em: Setembro 2011. Disponível em: <http://www-.eu-egee.org/>.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From data mining to knowledge discovery in databases. *AI magazine*, v. 17, n. 3, p. 37, 1996.

FISHER, M.; ELLIS, J.; BRUCE, J. *JDBC API tutorial and reference*. [S.l.]: Prentice Hall, 2003.

FOSTER, I. Globus toolkit version 4: Software for service-oriented systems. *Network and Parallel Computing*, Springer, p. 2–13, 2005.

GROUP, B. I. R. *Bio Informatics Research Group Dataset Repository*. Acesso em: Abril 2012. Disponível em: <http://www.upo.es/eps/bigs/datasets.html>.

GROUP, C.; REUTERS. *Reuters-21578*. Acesso em: Abril 2012. Disponível em: <http://www.daviddlewis.com/resources/testcollections/reuters21578/>.

HALL, M.; FRANK, E.; HOLMES, G.; PFAHRINGER, B.; REUTEMANN, P.; WITTEN, I. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, ACM, v. 11, n. 1, p. 10–18, 2009.

HAN, J.; KAMBER, M.; PEI, J. *Data mining: concepts and techniques*. [S.l.]: Morgan Kaufmann, 2011.

HERSH, W.; BUCKLEY, C.; LEONE, T.; HICKAM, D. Ohsumed: an interactive retrieval evaluation and new large test collection for research. In: SPRINGER-VERLAG NEW YORK, INC. *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. [S.l.], 1994. p. 192–201.

KACSUK, P. P-grade portal family for grid infrastructures. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, 2011.

KHOUSSAINOV, R.; ZUO, X.; KUSHMERICK, N. Grid-enabled weka: A toolkit for machine learning on the grid. *ERCIM news*, v. 59, p. 47–48, 2004.

KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: LAWRENCE ERLBAUM ASSOCIATES LTD. *International joint Conference on artificial intelligence*. [S.l.], 1995. v. 14, p. 1137–1145.

KOKKINOS, P.; CHRISTODOULOPOULOS, K.; VARVARIGOS, E. Efficient data consolidation in grid networks and performance analysis. *Future Generation Computer Systems*, Elsevier Science Publishers BV, v. 27, n. 2, p. 182–194, 2011.

LEGION. *Legion Project*. Acesso em: Setembro 2011. Disponível em: <http://legion.virginia.edu/>.

MACHADO, M.; MARIETTO, M.; SÁ, C. Web mining aplicado a detecção de perfis de estudantes. *II Congresso Sul Catarinense de Computação*, 2006.

MANCINI, E.; RAK, M.; VILLANO, U. Perfcloud: Grid services for performance-oriented development of cloud computing applications. In: IEEE. *Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE'09. 18th IEEE International Workshops on*. [S.l.], 2009. p. 201–206.

MENTEN, C. C. R. *Estudo e Implantação de Uma Infraestrutura Distribuída em Grid Usando o Middleware Globus*. 2012. Disponível em: <http://www.bcc.ufla.br/monografias/2012/caiocrm.pdf>. Trabalho de Conclusão de Curso - Universidade Federal de Lavras - UFLA.

NIST. *Text Retrieval Conference*. Acesso em: Abril 2012. Disponível em: <http://trec.nist.gov/>.

PINTO, V. G. *Mineração de Dados Paralela e Distribuída Baseada no Ambiente Weka*. 2010. Trabalho de Conclusão de Curso - Universidade Federal de Santa Maria - UFSM.

PINTO, V. G.; CHARÃO, A. S.; ROSE, C. A. F. de. Experimentos de mineração de dados paralela e distribuída com grid weka. In: *Anais da 11ª Escola Regional de Alto Desempenho*. Porto Alegre: Sociedade Brasileira de Computação, 2011. (Anais da Escola Regional de Desempenho, v. 1), p. 193–196.

SANTOS, R. Conceitos de mineraçao de dados na web. *XV Simpósio Brasileiro de Sistemas Multimıdia e Web*, p. 81–124, 2009.

SILVA, M. Mineraçao de dados: Conceitos, aplicaçoes e experimentos com weka. *Relatório técnico, Universidade do Estado do Rio Grande do Norte (UERN) e Instituto Nacional de Pesquisas Espaciais (INPE)*, 2004.

SOTOMAYOR, B.; CHILDERS, L. *Globus Toolkit 4: Programming Java Services*. [S.l.]: Morgan Kaufmann, 2006.

STANKOVSKI, V.; MAY, M.; FRANKE, J.; SCHUSTER, A.; MCCOURT, D.; DUBITZKY, W. A service-centric perspective for data mining in complex problem solving environments. In: *Proceedings of International Conference on*

Parallel and Distributed Processing Techniques and Applications (PDPTA). [S.l.: s.n.], 2004. p. 780–787.

STEINER, M.; SOMA, N.; SHIMIZU, T.; NIEVOLA, J.; NETO, P. S. Abordagem de um problema médico por meio do processo de kdd com ênfase à análise exploratória dos dados. *Gest Prod*, SciELO Brasil, v. 13, n. 2, p. 325–37, 2006.

STORY, H.; HARBULOT, B.; JACOBI, I.; JONES, M. Foaf+ ssl: Restful authentication for the social web. In: CITESEER. *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009)*. [S.l.], 2009.

TALIA, D.; TRUNFIO, P.; VERTA, O. The weka4ws framework for distributed data mining in service-oriented grids. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 20, n. 16, p. 1933–1951, 2008.

TOMA, I.; BURGER, T.; SHAFIQ, O.; DOEGL, D.; BEHRENDT, W.; FENSEL, D. Grisino: Combining semantic web services, intelligent content objects and grid computing. In: IEEE. *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*. [S.l.], 2006. p. 39–39.

VALENTEM, P.; PIMENTA, A.; SANTOS, D. dos; NETO, M. Uma solução software livre para mineração de dados em grades computacionais. *Proceedings of the I Concurso de Trabalhos Acadêmicos em Software Livre*, 2005.

WAIKATO, M. L. G. U. of. *Weka Datasets*. Acesso em: Abril 2012. Disponível em: <http://sourceforge.net/projects/weka/files/datasets/>.

WAIKATO, U. of. *Attribute-Relation File Format (ARFF)*. Acesso em: Janeiro 2012. Disponível em: <http://www.cs.waikato.ac.nz/~ml/weka/arff.html>.

WANG, J.; KOPERSKI, J.; LI, D.; STEFANOVIC, Y.; ZAIANE, B. Dbminer: A system for mining knowledge in large relational databases. In: *Proc. Intl. Conf. on Data Mining and Knowledge Discovery (KDD'96)*. [S.l.: s.n.], 1996. p. 250–255.

XHAFA, F.; PLLANA, S.; BAROLLI, L.; SPAHO, E. Grid and p2p middleware for wide-area parallel processing. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, 2010.

A Instalação do Weka4WS

Este tutorial foi retirado do site oficial do Weka4WS [4].

A.1 Pré-requisitos

Weka4WS necessita que o Globus Toolkit 4.0.x já esteja instalado nos servidores e somente o Java WS Core (um subconjunto do Globus Toolkit) na máquina cliente. Note que isso não é um requisito mínimo, mas um requisito específico: O Globus Toolkit 4.2.x e versões posteriores contém algumas atualizações nas especificações dos Web Services e em alguns outros serviços que os tornam incompatíveis com o Weka4WS.

Como a versão completa do Globus Toolkit 4.0.x roda em plataformas Unix, o pacote Weka4WS-service pode ser instalado somente nesses sistemas, enquanto que o cliente Weka4WS pode ser instalado tanto em Unix quanto em Windows. Para instalar o Globus Toolkit 4.0.x pode-se consultar a monografia do aluno Caio César Roma Menten, onde há uma sessão anexa explicando todo o processo de instalação desta ferramenta.

A.2 Pré-requisitos de segurança

O Weka4WS roda em um contexto de segurança e usa uma autorização via grid: Isto é, somente usuários que estiverem listados no serviço de grid podem usar a ferramenta. Então, para que o Weka4WS execute corretamente, os seguintes pré-requisitos devem ser satisfeitos:

[4] http://grid.deis.unical.it/weka4ws/

1. O usuário Weka4WS deve ter um certificado válido (no formato X.509)
2. O arquivo “/etc/grid-security/grid-mapfile” nos nós computacionais precisa ter uma entrada para mapear o usuário do cliente Weka4WS para um usuário local no servidor Weka4WS.

Segue uma entrada de exemplo para o arquivo “grid-mapfile”:

- "O=KGrid/OU=University of Calabria/CN=John Doe"john

A.3 Servidores

Como usuário root faça o seguinte:

1. Adicione a seguinte linha no arquivo “/etc/sudoers”:

- globus ALL= NOPASSWD: /bin/ls, /bin/cp, /bin/mkdir, /bin/chown, /bin/gzip

Como usuário globus(ou alternativamente como o usuário que executa o Globus), faça o download do pacote Weka4WS-service em um diretório de sua preferência e execute os seguintes passos:

1. Extraia o pacote Weka4WS-service:
   tar -xzvf weka4ws-service-2.1.tgz
2. Entre no diretório que acabou de ser criado:
   cd ./weka4ws-service-2.1
3. Gere o arquivo GAR do Weka4WS executando o comando:
   ./build.sh

4. Habilite o Weka4WS service executando o comando:

   ./deploy.sh

A.4 Máquina Cliente

Faça o download do pacote do cliente do Weka4WS e o extraia em um diretório de sua escolha.

A única configuração necessária para executar o Weka4WS é a configuração do arquivo "machines", localizado na subpasta "etc" do pacote cliente do Weka4WS. Este arquivo contém informações relacionadas aos nós computacionais, formatados com a sintaxe explicada a seguir.

A.4.1 Sintaxe do arquivo 'etc/machines'

As linhas do arquivo devem conter o endereço do host, a porta em que o container do Globus executa, a porta em que o GridFTP executa e a opção de *logging* de um nó computacional. A opção de *logging* pode ser 1 ou 0, que significa habilitado e desabilitado respectivamente. Quando a opção de *logging* é habbilitada, um log detalhado será gerado na tela onde o *container* do Globus estiver executando. Um exemplo do arquivo "etc/machines" está a seguir:

Hostname	Porta do container	Porta do GridFTP	Logging
pluto.deis.unical.it	8443	2811	1
saturn.deis.unical.it	8443	2811	0
cosmos.cs.icar.cnr.it	8443	2811	1

Tabela 12: Arquivo de configuração "etc/machines" do Weka4WS

Cada uma destas entradas deve ser colocada em uma linha do arquivo.

A.5 Execução

A.5.1 Servidores

Como usuário “root” faça o seguinte:

1. Inicie o servidor GridFTP:

- $GLOBUS_LOCATION/sbin/globus-gridftp-server -p <port>
  (onde <port> é a porta desejada; se não especificada, a porta 2811 será usada)

Como usuário “globus” (ou alternativamente como o usuário que roda o container do Globus) faça o seguinte:

1. Inicie o *container* do Globus com:

- $GLOBUS_LOCATION/bin/globus-start-container -p <port>
  (onde <port> é a porta desejada; se não especificada a porta 8443 será usada)

A.5.2 Máquina Cliente

Entre no diretório onde você extraiu o pacote do cliente Weka4WS:

- cd <caminho>/weka4ws-client-2.1
  Em seguida execute o seguinte comando: ./weka4ws.sh

B Instalação do Weka *Parallel*

Este tutorial foi retirado do site oficial do Weka Parallel [5].

Primeiro, instale o Weka Parallel em todas as máquinas que serão usadas no Grid. Para instalar basta baixar o arquivo do site do Weka Parallel e extrair o arquivo jar.

Uma sessão do Weka Parallel é executada em uma única máquina cliente e em vários servidores distribuídos. Cada servidor executa o software em 'background' e fica escutando por novas requisições enviadas pelo cliente. Para cada computador que será usado como um servidor distribuído, entre na pasta onde você extraiu o Weka *Parallel* e execute o programa com o seguinte comando no terminal:

- java -classpath weka.jar weka.core.DistributedServer <número da porta>

Cada computador com o servidor distribuído executando pode atuar como um servidor e vai escutar em uma porta específica por todas as requisições. Cada requisição que os servidores recebem executa em uma *thread* própria, possibilitando que vários computadores executem em um servidor ao mesmo tempo. Se um servidor tem mais de um processador, é possível alterar o arquivo de configuração para aproveitar o multi-processamento.

Este programa servidor pode ser executado manualmente em cada computador ou pode ser incluído em um script de inicialiazação do sistema para iniciar automaticamente quando o computador for ligado. Um log de todas as conexões processadas pelo servidor é automaticamente enviado para a tela, mas pode ser enviado para um arquivo texto se o usuário preferir.

[5] http://weka-parallel.sourceforge.net/

B.1 Arquivo de configuração

Para que o Weka saiba para quais computadores ele pode enviar o trabalho distribuído, um arquivo de configuração precisa existir em /.weka-parallel nos sistemas Unix e em C:
Windows
.weka-parallel nos sistemas Windows. A primeira linha do arquivo de configuração deve ser:

- PORT=XXXX
  Onde XXXX é o número da porta onde cada instância do servidor distribuído está escutando.

O resto do arquivo de configuração consiste nos endereços IP de cada um dos computadores que estão executando o servidor distribuído, um IP por linha.

O arquivo de configuração pode ser criado manualmente ou pode ser criado pela interface gráfica do Weka Parallel nas opções de validação cruzada.

B.2 Executando o Weka em Paralelo

Pode-se executar o Weka *Parallel* por sua interface gráfica.

Após abrir o programa, selecionar a base de dados e selecionar o classificador, pressione o botão próximo a validação cruzada para alterar as configurações de validação cruzada. Neste ponto marque a opção *Run in Parallel* e pressione “OK”. Finalmente, execute o classificador e a validação cruzada irá acontecer em paralelo no ambiente de Grid.