Revista de Sistemas de Informação da FSMA
n. 4 (2009) pp. 2-9

SISTEMAS DE INFORMAÇÃO
http://www.fsma.edu.br/si/sistemas.html

TRILHA ESTUDANTIL

WEKA-G: mineração de dados paralela em grades computacionais

A. Pimenta, P. Valentim, *Faculdade Ruy Barbosa*, D. Santos, *Universidade Estadual de Feira de Santana*, M. Neto, *Universidade Salvador*

***Resumen*—Mineração de dados é uma tecnologia que permite extrair informações úteis de grandes quantidades de dados. Entretanto, minerar um banco de dados, freqüentemente, requer um alto poder computacional. Para resolver tal problema, este trabalho apresenta uma ferramenta (WEKA-G), que executa de forma paralela algoritmos usados no processo de mineração de dados. Como ambiente para tal execução, usamos uma grade computacional por agregar inúmeros recursos dentro de um WAN.**

***Palavras-chave*— mineração de dados, paralelização, grade computacional**

I. INTRODUÇÃO

A cada dia, as bases de dados das organizações vêm assumindo maiores proporções, e extrair informação útil a partir de grande quantidade de dados passou a ser uma tarefa fundamental. Infelizmente, a extração realizada apenas por consultas rotineiras dos usuários, através de SQL, não satisfaz plenamente a esta necessidade; portanto, são precisas outras formas de extração para que informações sejam descobertas a partir desta massa de dados — principalmente aquelas ocultas, imperceptíveis à intuição ou aos olhos humanos.

Neste cenário, a mineração de dados é uma solução que pode ser proveitosa às organizações que querem melhor explorar os dados que possuem em seus repositórios [10]. Segundo Fayyad *et al* [5], mineração de dados "é o processo de descoberta de padrões válidos, novos, potencialmente úteis e compreensíveis embutidos nos dados".

Idade	Escolaridade	Casa própria	Telefone fixo	Pagador
jovem	1º grau	sim	não	mau
jovem	1º grau	sim	sim	mau
maduro	1º grau	sim	não	Bom
idoso	2º grau	sim	não	Bom
idoso	3º grau	não	não	bom
idoso	3º grau	não	sim	mau
maduro	3º grau	não	sim	bom
jovem	2º grau	sim	não	mau
jovem	3º grau	não	não	bom

Tabela 1. Banco de dados de uma empresa de crediário. Fonte: [11]

Para ilustrar, considere a Tabela 1, um caso fictício de uma empresa de crediário, onde se vê informações de nove clientes — cada linha da tabela representa um cliente. De acordo com os valores respectivos dos atributos `Idade`, `Escolaridade`, `Casa própria` e `Telefone fixo`, um cliente pode ser considerado um bom ou mau pagador — `Pagador` é denominado atributo de classificação, enquanto que bom e mau são as classes.

Neste exemplo, o desafio da mineração de dados é, através dos dados apresentados, identificar tendências de comportamento de clientes maus pagadores que ocorrem com uma maior freqüência – neste trabalho o resultado da mineração será denominado conhecimento. Assim, no futuro, a empresa poderá discernir pessoas potencialmente más pagadoras e, conseqüentemente, não aceitar as suas solicitações de crédito, por exemplo. Assim, este tipo de mineração é denominado classificação, no qual uma classe define previamente objetos do negócio.

Entretanto, um problema comum da mineração de dados é a sua aplicação em grandes bases de dados, pois o custo computacional gasto para realizá-la pode ser elevado. Uma possível solução para tal problema é paralelizar algoritmos envolvidos na mineração de dados uma vez que, pelo menos, muitos deles podem ser decompostos em tarefas independentes; conseqüentemente, as tarefas podem ser executadas paralelamente.

Outro ponto que vale ressaltar é que a mineração de dados pode ser um processo iterativo, podendo ser repetido inúmeras vezes até que seja induzido um conhecimento útil de fato. Para ilustrar, na primeira iteração, o conhecimento não foi útil. Então, algo é mudado, os dados podem ser alterados, adicionando ou removendo atributos, discretizando dados, etc. Ou ainda pode trocar algum dos algoritmos usados, pois os resultados gerados por dois

algoritmos distintos são muito instáveis no sentido de que um é melhor em alguns casos e o outro é melhor em outros casos [11]. Enfim, o processo é repetido com variações até que se chegue a um conhecimento desejável. E se cada iteração é custosa, executá-las diversas vezes será mais ainda.

Sendo assim, a mineração de dados de forma paralela pode reduzir o tempo de processamento, caso seja executada, por exemplo, utilizando diversas máquinas, que estarão compartilhando seus processadores, discos rígidos e memória.

Para tanto, é preciso ter um ambiente adequado onde haja máquinas conectadas por uma rede para que as tarefas sejam executadas. Dentre os ambientes existentes, vale destacar as grades computacionais, que é uma plataforma para execução de aplicações paralelas que possui uma alta dispersão geográfica, heterogênea no sentido de hardware e software, compartilhada, sem controle central e com múltiplos domínios administrativos [3].

A proposta deste artigo é apresentar uma ferramenta que viabilize a execução de mineração de dados de forma paralela através de um ambiente em grade computacional.

Na Seção 2, são abordados os dois temas envolvidos neste trabalho: mineração de dados e grades computacionais. A ferramenta WEKA-G é apresentada na Seção 3, bem como detalhes de como foi realizada a paralelização dos algoritmos escolhidos. Em seguida, na Seção 4, são apresentados os resultados obtidos a partir da avaliação experimental realizada na WEKA-G. Os trabalhos correlatos são discutidos na Seção 5. Por fim, na Seção 6, são tecidas as conclusões do trabalho e suas perspectivas.

II. MINERAÇÃO DE DADOS E GRADES COMPUTACIONAIS

Nesta seção é realizado o embasamento teórico necessário para o bom entendimento deste trabalho. Neste sentido, a subseção 2.1 está concentrada na descrição do processo de mineração de dados, enquanto que a subseção 2.2, na caracterização de um ambiente de grade computacional.

Além disso, em cada subseção, apontamos as ferramentas que reutilizamos neste trabalho. Uma vez que já existem algumas ferramentas de mineração de código aberto no mercado, o primeiro passo foi escolher uma para reusar. O mesmo foi feito quanto a tecnologia de grades computacionais.

A. Mineração de Dados

Para minerar um repositório de dados a fim de se extrair conhecimento, é preciso seguir um processo de mineração, composto de etapas, que torna possível buscar informação útil a partir de dados aparentemente desconexos, de diferentes tipos e graus de relevância. Resumidamente, neste trabalho, este processo de mineração foi dividido em três etapas: pré-processamento, mineração e pós-processamento.

Na primeira etapa, os dados são selecionados e formatados, além de acontecer também a remoção de ruídos. Pode ser necessária ainda a extração de amostras. Na segunda etapa, ocorre a mineração propriamente dita, onde o conhecimento é extraído a partir dos dados preparados. Feita a mineração, o conhecimento induzido é avaliado a fim de determinar se o mesmo é útil ou não (etapa de pós-processamento).

A realização de cada etapa não é isolada, ou seja, uma está relacionada à outra. Por exemplo, a indução e avaliação do conhecimento são auxiliadas por uma sub-etapa denominada fragmentação. Nesta, são usados algoritmos que dividem os dados a minerar em dois subconjuntos: treinamento, utilizado para a indução do conhecimento, e teste, utilizado para avaliar a qualidade deste. Cada instância do conjunto de teste é aplicada ao conhecimento induzido pelo algoritmo de mineração propriamente dito usado a partir do conjunto de treinamento corrente. O objetivo é mensurar a qualidade do conhecimento extraído. O foco deste trabalho é a etapa de fragmentação, então a próxima seção é dedicada aos algoritmos de fragmentação.

Algoritmos de fragmentação

Neste trabalho são abordados os seguintes algoritmos de fragmentação: `Cross-Validation` e o `Hold-Out` [9]. A justificativa por esta escolha se deve aos fatos de que ambos já estão implementados, de forma seqüencial, na ferramenta WEKA, que será abordada na seção 2.1.2. Além disso, ambos os algoritmos são compostos de tarefas que podem ser executadas independentemente uma da outra.

Cross-Validation

O pseudocódigo da Figura 1 demonstra a lógica básica do `Cross-Validation` e seu funcionamento convencional.

```
CrossValidation(numeroFolds Inteiro, minerador AlgoritmoDeMineracao)

De i=1 até numeroFolds faça
   Cria subconjunto de treinamento Tr_i
   Classificador C = Aplica minerador em Tr_i
   Cria subconjunto de teste Te_i

   De j=1 até (Numero de instancias de Te_i) faça
         Testa (prediz) a instância j de Te_i em C
   Fim laço.
Fim laço.
Gera dados estatísticos a partir da avaliação realizada.

Fim CrossValidation
```

Figura 1. Algoritmo Cross-Validation em pseudo-código

O algoritmo realiza um laço de repetição de *i* iterações, sendo *i* o número de *folds* (número de pares de subconjuntos treinamento-teste) fornecido como entrada. A cada iteração deste laço é criado um subconjunto de treinamento, e sobre este é aplicado o algoritmo de classificação previamente escolhido. Além disso, é criado também um subconjunto de teste.

Existe ainda outro laço de repetição dentro do anterior que itera *j* vezes, sendo *j* o número de instâncias do subconjunto de teste criado. A cada iteração do segundo laço, é processada uma predição do classificador sobre a instância de teste corrente. Em outras palavras, o conhecimento induzido é testado em cada instância do conjunto de teste, e os resultados de cada um destes testes alimentam uma série de dados estatísticos [12], dentre os quais destaca-se a acurácia (taxa de acerto) do conhecimento induzido, um dos mais usados na mineração de dados. Assim, ao final dos dois laços de repetição o `Cross-Validation` é finalizado, gerando dados estatísticos a respeito do conhecimento induzido.

Hold-Out

O `Hold-Out`, também chamado de `Percent split` [12], recebe como entrada, a informação referente à percentagem na qual os dados serão divididos em subconjuntos de treinamento e teste, além do algoritmo de mineração a ser usado.

```
HoldOut(percentagem Inteiro, minerador AlgoritmoDeMineracao)

Gera subconjunto de treinamento Tr com percentagem% da totalidade da amostra
Gera subconjunto de teste Te com o restante dos dados da amostra, (100-percentagem)%
Classificador C = Aplica minerador em Tr
De j=1 até (Numero de instancias de Te) faça
    Testa (prediz) a instância j de Te em C
Fim laço.
Gera dados estatísticos a partir da avaliação realizada.

Fim HoldOut.
```

Figura 2. Algoritmo Hold-Out em pseudo-código

Cria-se um subconjunto de treinamento com *x*% do tamanho da base de dados fornecida como entrada, sendo *x* a percentagem dada. Com o restante dos dados é criado um subconjunto de teste. O pseudo-código do algoritmo é apresentado na Figura 2.

O algoritmo de mineração é então aplicado sobre o subconjunto de treinamento. A partir daí, acontece um laço de repetição que itera *j* vezes, sendo *j* o número de instâncias do subconjunto de teste. A cada iteração deste laço, é processada uma predição do classificador sobre a instância de teste corrente.

Ao final do laço, o `Hold-Out` é finalizado, gerando dados estatísticos, assim como acontece no `Cross Validation`.

Análise dos algoritmos de fragmentação

Após mensurar cada execução de ambos os algoritmos em foco, chegou-se a conclusão que o gargalo de processamento está no segundo laço de repetição de cada um deles, onde é realizado a avaliação do conhecimento induzido através do conjunto de teste. Portanto, espera-se que ao parelelizar tais partes de cada algoritmo, o tempo de processamento do processo de mineração seja reduzido.

Mineração de dados com WEKA

WEKA é uma ferramenta, de código aberto, para mineração de dados, de interface amigável, que agrega um conjunto de algoritmos de classificação, regras de associação, regressão, pré-processamento e clustering, todos implementados em JAVA [12][2]. Além disso, a aplicação WEKA pode acessar dados oriundos de bancos de dados (via JDBC) ou através da chamada de arquivos de dados próprios.

Todas estas características fazem desta ferramenta, desenvolvida por pesquisadores da Universidade de Waikato na Nova Zelândia, uma das ferramentas mais populares entre a comunidade de mineração de dados.

Por ter sido desenvolvido usando a abordagem de framework, WEKA é extensível, permitindo, portanto, que novos algoritmos ou funcionalidades sejam adicionadas de maneira relativamente confortável.

Finalmente, devido a todas essas características descritas, esta ferramenta foi a escolhida para usar neste trabalho.

B. Grades Computacionais

Grades computacionais surgiram na década de 90 para tornar possível a execução paralela de aplicações em recursos geograficamente distribuídos. A idéia de grades computacionais é bastante simples: executar aplicações de forma paralela em recursos abundantes sem a necessidade de se investir em um supercomputador, muitas vezes inviável.

É comum fazer uma relação entre grade computacional e a rede elétrica, onde os usuários apenas conectam os equipamentos nas tomadas, abstraindo a origem da energia disponibilizada no ponto, qual a estrutura do meio físico, como se dá sua transmissão e distribuição. Algo semelhante acontece nas grades computacionais, pelas quais os usuários conectam-se em uma rede totalmente heterogênea obtendo recursos computacionais, desconsiderando a necessidade de saber de onde vêm esses recursos e como ele é transmitido e distribuído.

Pode-se citar como características de grades computacionais [3]:

- Heterogeneidade: as arquiteturas que formam a grade são extremamente heterogêneas. Elas são máquinas

com hardwares de várias gerações e softwares de diversas versões;

- Compartilhamento: a idéia de compartilhamento se refere ao fato de que a grade não pode ser dedicada a apenas uma aplicação;
- Alta dispersão geográfica: os computadores que compõem as grades computacionais podem estar distribuídos por todo o mundo;
- Múltiplos domínios administrativos: pelo fato de as grades estarem presentes em várias instituições, as mesmas podem estabelecer as políticas de acesso e uso dos serviços que a grade oferece;
- Controle distribuído: O controle é distribuído, pois não existe nenhuma entidade com controle total sobre a grade.

Uma plataforma para execução de aplicações paralelas que apresenta as características acima listadas certamente é uma grade computacional. Entretanto, a ausência de alguma das características não deve desqualificar uma determinada plataforma como grade [3].

Grades Computacionais com `OurGrid`

`OurGrid` é um grade computacional *peer-to-peer* (ponto-a-ponto), criada para ser usada por empresas ou laboratórios de pequeno, médio e até de grande porte, onde cada empresa/laboratório equivale a um *peer* [1]. Ao contrário de outras soluções existentes, como `Globus` [6] que é complexo de instalar e requer uma negociação *off-line* para que o usuário faça parte da grade, o `OurGrid` é bastante simples de instalar, além de não haver necessidade de nenhuma negociação ou intervenção humana. Possui seu foco em aplicações *bag-of-tasks*, ou seja, aplicações cujas tarefas são independentes e não há necessidade de comunicação entre elas para que completem sua computação. Podemos citar como exemplo deste tipo de aplicação, a mineração de dados, que é justamente o objeto de estudo deste artigo.

Sendo assim, por essas características, o `OurGrid` é uma ferramenta adequada para o desenvolvimento da ferramenta WEKA-G. A arquitetura do `OurGrid` possui três componentes principais necessários para sua compreensão: `MyGrid`, `Peer` e `User Agent`. A arquitetura pode ser visualizada na Figura 3.

O `MyGrid` é a interface para o uso da grade computacional, que provê uma série de abstrações que permite o uso conveniente da grade. Uma máquina que possui o `MyGrid`, que também pode ser chamada de `home machine` (`hum`), é responsável por escalonar as tarefas para executar nas máquinas da grade, denominadas `grids machines` (`gums`), que é onde o processamento das tarefas ocorre e que também executa o componente `User Agent` que será descrito mais à frente. A grade possui acesso descentralizado que permite múltiplos usuários, cada um usando uma instância do `MyGrid`, a qual deve estar associada a um *Peer*.

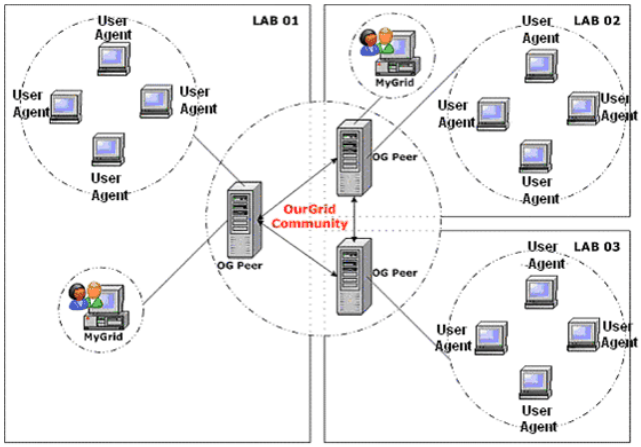

Figura 3. Arquitetura do `OurGrid`

O `Peer` é um componente que executa em uma máquina que tem como função organizar e prover máquinas de um determinado domínio administrativo para que tarefas sejam executadas; porém, se as maquinas do domínio não forem suficientes, as de outros domínios serão usadas, as quais estarão sendo administradas por outros `Peers`. Há duas visões distintas em relação aos `peers`, que são as seguintes: (i) a visão do usuário: um `peer` é um provedor dinâmico de `gums` para execução das tarefas, e (ii) a visão do administrador: um `peer` determina como e quais máquinas serão usadas para executar alguma tarefa.

O `User Agent` é o componente que executa em todas as *gums* e provê processamento para a *hum*, além de fornecer suporte básico para manipulação de falhas. Juntamente com o componente *peer*, permite o uso de computadores em redes privadas, mesmo que estejam em um domínio administrativo diferente ou protegido por um *firewall*.

III. WEKA-G

Tendo em vista o problema apresentado, que há um gargalo de processamento na execução de algoritmos de fragmentação, foi criada a ferramenta WEKA-G.

WEKA-G é uma modificação da ferramenta WEKA original, que suporta uma integração com o `OurGrid`. Mais precisamente, os algoritmos *Cross-Validation* e *Holdout* foram alterados para que pudessem ser executados de forma paralela. Desse modo, a arquitetura do WEKA-G é uma modificação integrando as arquiteturas do WEKA e do `Ourgrid`.

Com isso, o usuário poderá escolher se quer minerar um arquivo localmente ou usando a grade computacional. Para tanto, se optar pela segunda opção, bastará apenas clicar em um *checkbox* na interface do Weka.

É importante salientar que consideramos que o ambiente `OurGrid` já está configurado, pronto para o usuário utilizar. Em outras palavras, a instalação e configuração do

OurGrid é um pré-requisito para usar a ferramenta WEKA-G.

Vale destacar ainda que essa funcionalidade só está disponível para algoritmos de classificação, que usam algoritmos de classificação para induzir e avaliar conhecimento.

Sendo assim, nesta seção, será descrita a arquitetura da WEKA-G e como os algoritmos de fragmentação foram paralelizados.

A. Arquitetura WEKA-G

A arquitetura WEKA-G (Figura 4) é composta pelos componentes do `OurGrid` juntamente com a biblioteca WEKA, modificada para contemplar os algoritmos paralelizados.

Primeiramente, o usuário escolhe os dados a minerar, opta pelo processamento paralelo e dá início ao processo de mineração, tudo isso a partir da interface gráfica de uma máquina local. A partir desse momento, a aplicação WEKA comunica-se com o `OurGrid` através do `MyGrid`, enviando-lhe o *job* criado, que é um conjunto de tarefas a serem executadas na grade. Cada tarefa a ser processada é composta pela linha de comando a ser executada e os arquivos necessários para sua execução. Mais especificamente, no caso do WEKA-G, a linha de comando trata de invocar a classe do algoritmo de fragmentação escolhido, fazendo com que seja executado apenas um dos laços da iteração, aquele responsável pela avaliação do conjunto de teste. Mais detalhes a respeito da paralelização dos algoritmos de fragmentação serão apresetandos na próxima subseção.

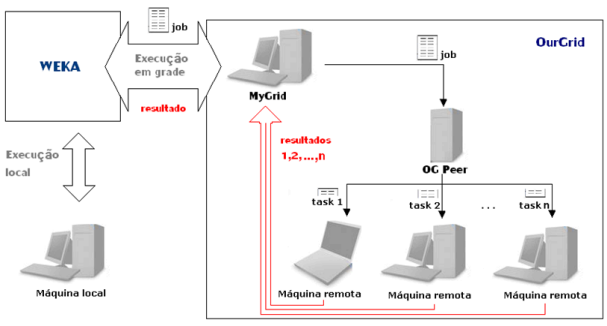

Figura 4. Arquitetura WEKA-G

Definido o *job*, o `MyGrid` envia requisições a um determinado *peer* que, por sua vez, solicita máquinas disponíveis a outros *peers* da grade `OurGrid` para a execução das tarefas.

Uma vez tendo máquinas disponíveis, então as tarefas do *job* enviado são executadas. Após isso, os resultados obtidos (de cada tarefa) são devolvidos ao `MyGrid`, sendo então repassados para a camada WEKA que, por sua vez, reagrupa os resultados. Por fim, estes são mostrados ao usuário através da interface gráfica.

B. Algoritmo Cross Validation

O `Cross Validation` divide o conjunto inicial de dados a minerar em *i* subconjuntos, definido pelo número de *folds*. Destes *i* subconjuntos, um é retido para servir como dados de teste (conjunto de teste) e o restante dos *i-1* subconjuntos são usados como dados de treinamento (conjunto de treinamento). Este processo é repetido *i* vezes, sendo que cada subconjunto será usado pelo menos uma vez como conjunto de teste. O pseudocódigo desta técnica está descrito na Figura 5.

O trecho do algoritmo que é processado na grade computacional, indicado entre chaves na figura, é o teste do conhecimento (classificador) gerado a partir do conjunto de treinamento. Assim, cada instância do conjunto de teste é testada usando o classificador induzido correspondente, em uma máquina da grade computacional.

Cada conjunto de execução desta é dita uma tarefa, contendo uma par classificador-conjunto de teste. As *i* tarefas geradas formam o *job* a ser submetido ao `Mygrid`.

Como algoritmo de mineração para indução de conhecimento, poderá ser usado qualquer um dos disponíveis no `WEKA`, por exemplo, `J48`, `Id3`, `IB1`, `OneR`, `RepTree`, `Prism`, entre outros [12].

Ao final da execução de cada tarefa, as `grid machines` devolvem seus respectivos resultados para o `MyGrid` (`home machine`), responsável pela submissão do *job*. Este componente agrega todos os resultados e agrupa-os, devolvendo para a aplicação WEKA que, por sua vez, gera, em seguida, através da interface, os dados estatísticios a respeito da mineração realizada.

```
CrossValidation(numeroFolds Inteiro, minerador AlgoritmoDeMineracao)

Cria um job J
De i=1 até numeroFolds faça
   Cria subconjunto de treinamento Tr_i
   Classificador C = Aplica minerador em Tr_i
   Cria subconjunto de teste Te_i

   Cria tarefa T_i {
      De j=1 até (Numero de instancias de Te_i) faça
         Testa (prediz) a instância j de Te_i em C
      Fim laço.
   }

   J = J + T_i
Fim laço.

Inicia o componente MyGrid do OurGrid
Submete o job J ao MyGrid

Enquanto (MyGrid não retorna resultados) faça
   Verifica se resultados estão disponíveis
Fim laço
```

Agrupa resultados processados por cada tarefa
Gera dados estatísticos a partir da avaliação realizada

Fim CrossValidation

Figura 5. Algoritmo `Cross Validation` (paralelizado)

De forma similar, o mesmo fluxo apresentado é feito para o algoritmo de fragmentação *Holdout*, onde cada iteração de teste do classificador induzido é enviado para máquinas da grade a fim de executá-las remotamente. Por restrições de espaço, não serão apresentados mais detalhes.

IV. AVALIAÇÃO EXPERIMENTAL

Nesta seção, são descritos os cenários de teste elaborados para avaliar a ferramenta WEKA-G, bem como seus respectivos resultados.

Os testes foram efetuados em três cenários diferentes, descritos abaixo:

Cenário 1

- Uma máquina com sistema operacional Linux executando o WEKA-G. Sua configuração é Intel Celeron M 1.6GHz e 512 MB de memória *RAM*;
- Cinco máquinas com sistema operacional Windows executando o `UserAgent`, conectadas através de uma grade computacional local `OurGrid`. A configuração das máquinas é Intel Celeron 2.26GHz e 512 MB de memória *RAM*.

Cenário 2:

- Uma máquina com sistema operacional Linux executando o WEKA-G. A configuração das máquinas é Intel Celeron M 1.6GHz e 512 MB de memória *RAM*;
- Dez máquinas com sistema operacional `Windows` executando o `UserAgent`, conectadas através de uma grade computacional local `OurGrid`. A configuração das máquinas é Intel Celeron 2.26GHz e 512 MB de memória *RAM*.

Cenário 3:

- Máquina com sistema operacional Linux executando o WEKA-G. A configuração das máquinas é Pentium M 1.7GHz e 1 GB de memória *RAM*;
- Máquinas localizadas em outros domínios, conectadas pela Internet na grade `OurGrid`. Não é possível informar o número de máquinas porque quem faz a distribuição é o escalonador de tarefas (`MyGrid`).

Para os três cenários, foi usado um arquivo (para ser minerado) com 23.000 instâncias, denominado ticdata_categ[1]. O algoritmo de classificação utilizado foi *Prism* e os de fragmentação, `Cross-Validation` e o `Hold-Out`.

Os testes seguiram um roteiro pré-definido onde cada um dos cenários foi executado três vezes. Com os três resultados obtidos destes, foi dada como resposta uma média aritmética, gerando também um desvio padrão. Primeiro foram realizados os testes considerando o cenário 1, depois o cenário 2 e, em seguida, o cenário 3. O roteiro passo a passo é o seguinte:

1. Executar três vezes cada par [algoritmo de classificação-algoritmo de fragmentação] na grade;
2. Calcular a média aritmética dos tempos de resposta obtidos por cada par;
3. Calcular o desvio padrão.

Os testes foram realizados levando em consideração o tempo de execução da mineração utilizando a grade.

Cenário	Tempo de resposta médio	Desvio Padrão
1	25' 46''	1' 47''
2	17' 02''	1' 33''
3	31' 15''	2' 16''

Tabela 2. Avaliação *Cross-Validation*

Vale ressaltar ainda que essa mesma mineração foi executada localmente, para efeitos de comparação com os cenários montados.

Após a execução dos testes, chegou-se aos números apresentados na **Tabela 2**.

Seqüencialmente, obteve-se o tempo médio de 49' 35''. Um gráfico comparativo entre as quatro execuções pode ser visualizado na Figura 6.

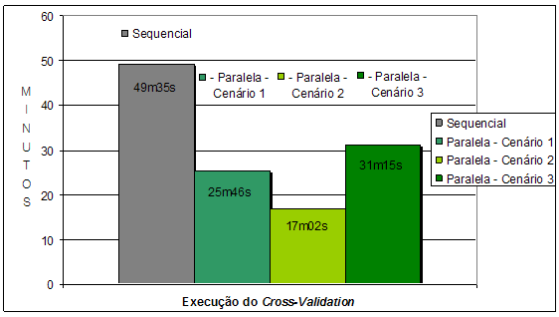

Figura 6. Avaliação *Cross-Validation*

Utilizando o algoritmo de classificação *Prism* e o algoritmo de fragmentação `Hold-Out`, com 50% das instâncias para treinamento (teste), chegou-se aos tempos indicadas na Tabela 3.

Cenário	Tempo de resposta médio	Desvio Padrão
1	7' 58''	1' 55''
2	5' 58''	1' 43''
3	9' 41''	2' 46''

Tabela 3. Avaliação *Hold-Out*

[1] Extraído de http://hakank.org/weka/ticdata_ categ.arff, em setembro/2006.

Ainda considerando a avaliação do algoritmo `Hold-out`, seqüencialmente, obteve-se o tempo médio de 14' 05''. A Figura 7 exibe o gráfico comparativo destes testes.

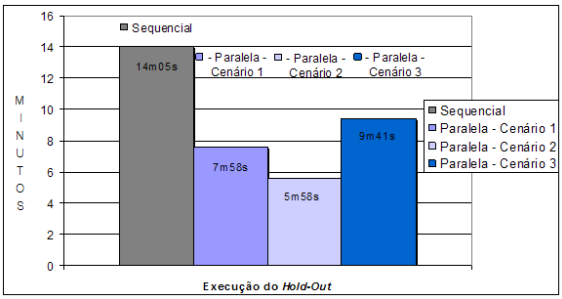

Figura 7. Avaliação *Hold-Out*

Observando as tabelas e gráficos, nota-se que os algoritmos `Cross-Validation` e `Hold-Out`, quando executados em paralelo tiveram um melhor desempenho do que quando executados sequencialmente.

Na situação em que a grade computacional é formada por um maior número de máquinas (disponíveis para execução) que o número de tarefas do *job* – cenários 1 e 2 – , o desempenho do algoritmo é superior, pois se tem recursos suficientes para executar uma tarefa em cada máquina da grade, ao mesmo tempo, sem a necessidade de aguardar os recursos serem liberados.

Foi observada uma limitação referente à execução dos algoritmos utilizando outros domínios. Nesta situação, por se tratar de recursos remotos (conectados via Internet), fora do domínio local, houve um tempo maior comparando com os cenários 1 e 2. É importante lembrar que no cenário 3, os testes realizados poderiam ter obtido resultados diferentes caso a largura da banda fosse diferente, ou houvesse mais, ou menos, *gums* disponíveis nos respectivos domínios utilizados. Por se tratar da Internet e não se saber quais e quantos são os recursos disponíveis, estes fatores devem ser levados em conta. Mesmo nessas circunstâncias, o tempo ainda é menor quando comparado ao da execução local, como pôde ser constatado na Figura 6 e Figura 7.

V. TRABALHOS RELACIONADOS

Como trabalhos correlatos, destacamos dois, *WEKA Parallel* [4] e *Grid WEKA* [8], que têm uma abordagem similar ao WEKA-G. Uma tabela comparativa entre as três abordagens pode ser vista na Tabela 4, onde as colunas indicam cada uma delas e as linhas, as características.

A primeira característica indica quais algoritmos de fragmentação foram paralelizados por cada abordagem. O *WEKA Parallel* e o *GridWEKA* o fizeram apenas para o algoritmo `Cross-Validation`, enquanto que WEKA-G paralelizou tanto este quanto o `Hold-Out` também. É comprovado pela literatura que um mesmo algoritmo de classificação pode conseguir resultados distintos quando associados com diferentes algoritmos de fragmentação [11]. Adicionalmente, não há uma técnica que seja melhor para todos os casos. Sendo assim, dar a opção ao usuário de usar mais de um algoritmo (paralelizado) é importante.

Outro ponto a considerar é o suporte às falhas que eventualmente acontecem na rede. A partir do *WEKA Parallel* foi criado o *Grid WEKA*, onde este dá o suporte às falhas de rede não presente naquele. No caso do WEKA-G, este suporte é delegado ao `OurGrid`, que é o componente da arquitetura responsável por gerenciar todos os recursos relacionados a rede.

	WEKA Parallel	GridWEKA	WEKA-G
Algoritmo(s) utilizado(s)	Cross-Validation	Cross-Validation	Cross-Validation/ Hold-out
Suporte às falhas de rede	Não	Sim	Sim
Não há negociação *off-line*	Não	Não	Sim
Não há necessidade de instalação de software	Não	Não	Sim

Tabela 4. Trabalhos Relacionados

Com o `OurGrid` também é evitado a necessidade de se ter uma negociação *off-line* para poder ter acessos a máquinas para processamento. Além das máquinas que usuários têm acesso, existem também aquelas que estão disponíveis em outros domínios administrativos já que o `OurGrid` é uma grade aberta onde qualquer um pode se juntar. Vale ressaltar que usuários precisam ter acesso às máquinas que deseja executar; comumente, necessitam de mais máquinas do que as que possuem e, para conseguir acesso à novas máquinas é preciso uma negociação com administradores de redes, muitas vezes sem sucesso.

Além disso, uma vez que a grade é aberta, aqueles que aderem, são responsáveis por instalar o software necessário. Então, os usuários do WEKA-G não têm preocupação em ficar instalando softwares, bastando apenas que em cada tarefa criada sejam enviados, através do `OurGrid`, o subconjunto de teste e os códigos necessários para a execução do processo de mineração remotamente. Vale lembrar que é necessário apenas instalar os componentes `OurGrid` em apenas uma máquinas para que o WEKA-G possa ser usado.

Traçando um comparativo dessas duas últimas características (negociação *off-line* e instalação de

software) com os dois trabalhos correlatos, constata-se que ambos contemplam tais critérios, sendo neste caso, uma desvantagem que dificulta o uso por parte dos usuários.

Portanto, conclui-se que WEKA-G é a única abordagem que paraleliza mais de um algoritmo de fragmentação, dando suporte a redes e não necessitando de negociações off-line para obtenção de máquinas nem de instalação de software adicional.

VI. CONSIDERAÇÕES FINAIS

Este trabalho foi elaborado com o intuito de ajudar a minerar de forma mais rápida bancos de dados. É muito comum que usuários executem algoritmos diferentes para um mesmo banco de dados ou ainda que executem o mesmo algoritmo várias vezes, mas com parâmetros diferentes. Daí a necessidade de uma ferramenta como esta.

A concepção da ferramenta foi pensada de forma que facilita a vida do usuário quanto a outros aspectos também:

- Custo financeiro: usando uma grade computacional aberta como o `OurGrid`, permite que empresas e laboratórios que não têm condições de investir em supercomputadores, tenham acesso a recursos computacionais de alto desempenho de forma barata;
- Flexibilidade: a ferramenta é bastante flexível, dando aos usuários muitas opções, das quais destacamos que eles têm acesso a uma biblioteca de algoritmos de mineração e podem optar por fazer suas execuções na grade ou não.

Como trabalhos futuros, pretende-se realizar testes de desempenho no intuito de avaliar melhor os resultados obtidos. Com isto, poder-se-á, por exemplo, avaliar se vale a pena ou não enviar uma execução para a grade computacional — é possível que o tempo de *overhead* faça com que o tempo de execução total acabe sendo maior do que uma execução *standalone*. Estes testes de avaliação ajudarão em outro ponto: até agora estão sendo considerados algoritmos de fragmentação, portanto, também serão investigados os algoritmos de classificação ou *clustering*, por exemplo, para paralelizá-los, caso seja necessário.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Andrade, Nazareno; Brasileiro, Francisco; Cirne, Walfredo; Medeiros, Raissa; Paranhos, Daniel; Santos-Neto, Elizeu; Sauvé, Jacques. Grid Computing for Bag-of-Tasks Apllications. Proceedings of the Third IFIP Conference on E-Commerce, E-Business and E-Government, 2003.

[2] Basu, Sugato; Melville, Prem. Weka Tutorial. University of Texas at Austin. Disponível em: < http://www.cs.utexas.edu/users/ml/tutorials/Weka-tut/sld002.htm >. Acesso em: 8 de outubro 2006.

[3] Cirne, Walfredo**.** Grids Computacionais: Arquiteturas, Tecnologias e Aplicações. Anais do Terceiro Workshop em Sistemas Computacionais de Alto Desempenho, 2002.

[4] Celis, Sebastian; MUSICANT, David R. "Weka Parallel: Machine Learning in Parallel". Relatório Técnico, 2003.

[5] Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P. Data mining and knowledge discovery in databases: an overview. *Comm. of the ACM*, vol. 39, n.11, 1996.

[6] Foster, I. "Globus Toolkit Version 4: Software for Service-Oriented Systems". Proceedings of the IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2006.

[7] Han, J, Micheline, K. "Data Mining: concepts and techniques". Morgan Kaufmann Publishers, 2001, 550 p.

[8] Khoussainov, Rinat, Zuo, Xin & Kushmerick, Nicholas. "Grid-enabled Weka: A Toolkit for Machine Learning on the Grid". *ERCIM News*, No. 59, October 2004.

[9] Kohavi, Ron: "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection". In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1996, pp. 1137 – 1143.

[10] Mendonça, Manoel. Mineração de Dados. Relatório Técnico, 2002.

[11] Santos, David Moises B. "Seleção de modelos de classificação usando heurísticas". Dissertação, Universidade Federal de Campina Grande, 2005.

[12] Witten, Ian H.; Frank, Eibe. "Data Mining: Practical machine learning tools and techniques". 2a edição. Morgan Kaufmann, São Francisco, 2005.