Utilização de XML numa plataforma de Data Mining distribuído

Ruy Ramos, Carlos Adriano Gonçalves and Rui Camacho

LIACC, Rua de Ceuta 118 - 6^o 4050-190 Porto, Portugal
FEUP, Rua Dr Roberto Frias, 4200-465 Porto, Portugal

{ruyramos,rcamacho}@fe.up.pt, cadriano.goncalves@gmail.com

Resumo O processo de Extracção de Conhecimento em Bases de Dados (*Knowledge Discovery in Databases* - KDD) envolve a análise de extensas bases de dados e recurso a complexos algoritmos de análise de dados (*Data Mining*). Este processo requer, geralmente, recursos computacionais dedicados e de elevado custo o que reduz significativamente o número de utilizadores capazes de efectuar tais análises. Neste artigo apresentamos uma arquitectura baseada em computadores pessoais distribuídos numa rede de computadores de uma organização e que permite a realização de tarefas de KDD sem recursos computacionais dedicados e sem perturbar o funcionamento da organização. A arquitectura é denominada *Harvard - **HARV**esting **A**rchitecture of idle machines fo**R** **D**ata mining*. O Harvard utiliza uma linguagem de especificação e controlo de tarefas baseada em XML. A linguagem XML no caso do Harvard é imprescindível para a interoperabilidade entre os diferentes componentes do ambiente descrevendo claramente todos os aspectos da tarefa de *KDD* a ser executada de forma distribuída. Os resultados alcançados por diferentes nós do sistema são transcritos em XML, de modo a facilitar a apresentação ao utilizador do ambiente *Harvard* e ainda permitir integração com outros sistemas de extracção de conhecimento.

1 Introdução

O crescente uso das tecnologias da informação associado ao crescimento e desenvolvimento económico de vários sectores (empresarial, governo, comunidade científica e académica, entre outros) têm permitido que as organizações e a sociedade em geral reúnam uma enorme quantidade de dados e informações sistematicamente em Bases de Dados (BD) [4].

Do ponto de vista humano, torna-se praticamente impossível interpretar, analisar e obter resultados a partir de grandes quantidades de dados sem o auxílio de computadores. Torna-se muito difícil diferenciar informações verdadeiramente importantes e úteis em tão grande quantidade de dados [4,8]. Neste caso, a aplicação de técnicas automáticas de análise capazes de "extraír" informação útil torna-se inevitável dada a complexidade e a extensão das BD.

A aplicação de técnicas de *Data Mining* para auxiliar a análise de grandes quantidades de dados torne-se uma alternativa viável e aplicável, usando para isso os algoritmos de *Machine Learning* como *Inductive Logic Programming* (ILP) [12] e *Association Rules Discovery* (ARD) [1]. Uma desvantagem destes algoritmos é que são computacionalmente muito exigentes e este aspecto tem limitado o seu uso em aplicações de extracção de conhecimentos no mundo real.

Além disso, os algoritmos de *Machine Learning* foram originalmente construídos para abordagens sequenciais monolíticas pressupondo sempre um ambiente centralizado. É necessário que os dados estejam todos num único local para que as técnicas de *Data Mining* possam ser aplicadas. A quantidade de informação actualmente disponível é tão grande que centralizá-la se torna oneroso e extremamente complexo. Há casos mesmo em que os dados estão dispersos e não podem ser centralizados devido a limitações de rede, falta de espaço de armazenamento, questões de segurança, confidencialidade ou privacidade. Assim, há necessidade de adoptar novas abordagens para análise de dados nos seus respectivos locais de armazenamento.

Desenvolver técnicas de *Data Mining* para arquitecturas distribuídas tem sido um grande desafio da comunidade científica de *Knowledge Discovery in Databases - KDD*. O desenvolvimento de um novo estágio do KDD denominado de DPKDD (*Distributed and Parallel Knowledge Discovery in Databases*) está ligado principalmente pelo avanço em áreas como *storage, access e analysis* [10] e ao uso de tecnologias como *Grid Computing* [6].

A nossa proposta contempla uma arquitectura computacional dedicada ao processo de *Data Mining Distribuído* denominada de *Harvard - **HARV** esting **A**rchitecture of idle machines fo**R** **D**ata mining,* com os seguintes objectivos: fornecer ao analista de dados (utilizador) uma linguagem simples mas poderosa para especificar as tarefas de análise de dados (*KDD*); viabilizar a utilização optimizada de computadores pessoais quando ociosos em prol de um processamento cooperativo e útil; construir uma plataforma que possa ser usada independentemente dos sistemas operativos instalados (*Linux* ou *Windows*) na organização; permitir análise de dados de forma distribuída (computação distribuída e acesso a dados fisicamente distribuídos); permitir o uso de diferentes algoritmos de *Machine Learning* e suas implementações (ferramentas) sem qualquer alteração da plataforma, sendo portanto independente da ferramenta usada pelo analista; e permitir recuperação de fallhas do próprio sistema (tolerância a falhas).

Além dos objectivos apresentados, o *Harvard* apresenta as seguintes vantagens: o utilizador pode facilmente definir o processo de análise de dados mais adequado para suas necessidades; com este ambiente pode-se expandir o uso da análise de dados a um vasto leque de organizações já que os custos envolvidos são bastante baixos, as máquinas usadas são as mesmas que servem para as tarefas rotineiras. E além disso, o processamento computacional *Harvard* não perturba o trabalho normal da organização uma vez que só usa recursos computacionais desocupados ou ociosos.

O *Harvard* permite ao utilizador descrever cada tarefa do processo de KDD através da linguagem de anotação XML (*Extensible Markup Language*)[3] e especificar o fluxo de sua execução (*workflow*) numa linguagem descritiva de sintaxe simples mas no entanto poderosa. O sistema corre num *Servidor* com uma colecção ilimitada de estações *Cliente*. Tanto o *Servidor* como os *Clientes* são programados em *Java*. Os *Clientes* podem aceder aos dados directamente de uma base de dados (usando JDBC) e todo *software* necessário de análise de dados, via protocolo HTTP.

Em termos de trabalhos correlatos destacamos o Condor [11] da Universidade de Wisconsin-Madison e o BOINC (*Berkeley Open Infrastructure for Networking Computing*) [2]. O ambiente Condor caracteriza-se por gerir um ambiente composto de vários servidores e/ou *cluster*, como o "Beowulf". É basicamente um ambiente *Grid Computing* para gerir a capacidade de processamento de recursos computacionais de um "campus" ou de uma organização. Tem como características principais a gestão de *jobs*, definição de políticas de uso do ambiente distribuído e gestão e monitoramento dos recursos computacionais. Há uma variante denominada Condor-G [7] que facilita a integração com ambientes *Grid Computing* baseados no *Globus Toolkit* [5]. Por outro lado, o BOINC é uma arquitectura computacional distribuída que utiliza recursos computacionais registados voluntariamente a partir da Internet. É uma plataforma que permite aos cientistas gerir recursos computacionais públicos disponíveis na Internet em benefício de projectos científicos de grande dimensão. Alguns projectos como o *SETI@home*, *Folding@home* utilizam a plataforma BOINC. A arquitectura é composta por servidores de projectos que operam suas próprias aplicações e bases de dados, e partilham processamento voluntário disponível a partir de estações ligadas à Internet que "doam"ciclos de processamento por intermédio de aplicações instaladas para esta finalidade.

O artigo está organizado da seguinte forma: na Secção 2 descrevemos como as tarefas do processo de KDD podem ser especificadas por meio de uma linguagem simples mas poderosa usando como base XML; na Secção 3 apresentamos a arquitectura do *Harvard* e seu funcionamento; e por fim apresentamos as conclusões e trabalhos futuros na última Secção.

2 O processo de Extracção de conhecimento

Segundo Fayyad [4], *Data Mining* é o núcleo principal de um processo mais amplo denominado de *Knowledge Discovey in DataBases* (KDD), ou Descoberta de Conhecimento em Bases de Dados, conforme apresentado na Figura 1.

Knowledge Discovey in DataBase (KDD) é um processo não trivial de identificar, validar e reconhecer padrões de dados que possam prover informação válida gerando conhecimento sobre uma determinada Base de Dados. Os dados referem-se a representação de factos e os padrões são definidos por uma expressão que descreve um subconjunto desses mesmos dados [4].

De acordo com *Fayyad* [4] o processo de KDD é composto pelas fases:

- preparação dos dados;

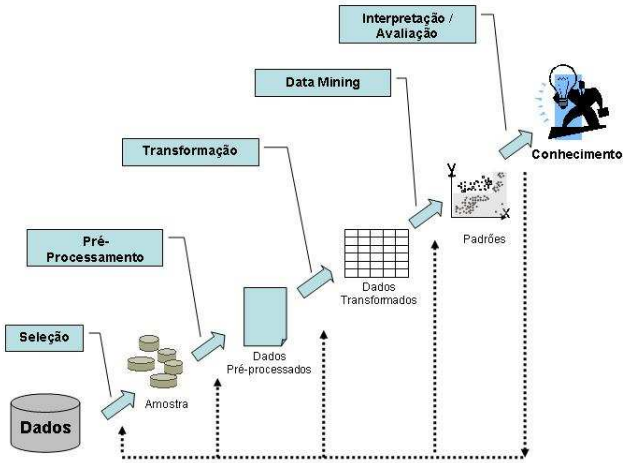


Figura 1. Processo de Extracção do Conhecimento

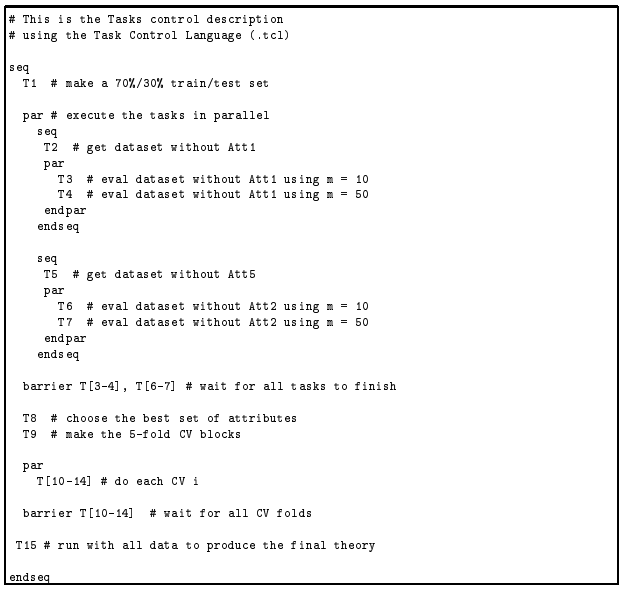
- selecção de dados;
- pré-processamento de dados;
- transformação de dados;
- *data mining*;
- interpretação e avaliação do conhecimento.

2.1 A fluxo do processo de análise de dados

O *Harvard* inicia o processo de análise de dados lendo, de um ficheiro, a especificação do fluxo do processo (*workflow*) juntamente com a descrição de cada uma das tarefas do processo KDD. O fluxo do processo é representado por um grafo com dois tipos de nós: sequenciais e paralelos. Cada nó regista o conjunto de tarefas a serem executadas, representadas e designadas como UT (Unidades de Trabalho). Na Secção 3, que descreve a arquitectura do *Harvard*, as UT são descritas com maior detalhe.

No caso do nó sequencial, as tarefas deverão necessariamente ser executadas em ordem de precedência devido às suas interdependências no processo de análise de dados. Por outro lado, os nós paralelos especificam tarefas que podem ser executadas em simultâneo.

Na Figura 2, apresentamos um exemplo de descrição de um processo KDD. A figura mostra ainda em detalhe um conjunto de tarefas definidas $T1$ ... $T15$ que abrangem todo o processo, desde o pré-processamento, seleccionando os atributos mais relevantes, até à consolidação dos resultados ($T15$). A descrição de cada tarefa Tn está codificada em XML em ficheiros distintos que serão processados pelo módulo Gestor de Tarefas (GT) do *Harvard*, conforme detalhado na Secção 3.



```
# This is the Tasks control description
# using the Task Control Language (.tcl)

seq
  T1  # make a 70%/30% train/test set

  par # execute the tasks in parallel
    seq
     T2  # get dataset without Att1
     par
       T3  # eval dataset without Att1 using m = 10
       T4  # eval dataset without Att1 using m = 50
     endpar
    endseq

    seq
     T5  # get dataset without Att5
     par
       T6  # eval dataset without Att2 using m = 10
       T7  # eval dataset without Att2 using m = 50
     endpar
    endseq

  barrier T[3-4], T[6-7] # wait for all tasks to finish

  T8  # choose the best set of attributes
  T9  # make the 5-fold CV blocks

  par
    T[10-14] # do each CV i

  barrier T[10-14]  # wait for all CV folds

 T15 # run with all data to produce the final theory

endseq
```

Figura 2. Descrição do processo KDD em tarefas sequencias e paralelas.

2.2 A especificação das tarefas

A linguagem de especificação das tarefas, conforme ilustrado na Figura 3, parte inicialmente da sintaxe básica da XML. A partir deste contexto são definidas variáveis com seus respectivos parâmetros que posteriormente serão interpretadas pelo *Havard* no seu módulo que trata da especificação da tarefa - o Gestor de Tarefas (GT).

Na especificação de cada tarefa o utilizador (analista de dados) deverá fornecer em detalhe toda a informação relevante. Isso inclui descrever entre outros ítens:

- local e nome do ficheiro que contém os dados para análise;
- algoritmo(s) a ser(em) usado(s) com respectivos parâmetros de análise;
- nome da aplicação que implementa o algoritmo escolhido (*e.g.*: C4.5)
- ferramentas de pré-processamento a ser usada (*e.g.*: aplicação de *script perl* para eliminar ou consolidar atributos);
- local e formato de representação dos resultados.

2.3 O uso da linguagem XML no *Harvard*

A linguagem XML usada no *Harvard* permite que o utilizador possa especificar claramente o processo de KDD pois "provê um conceito para descrever, armazenar, permutar e manipular dados estruturados" [9,3].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE tasks SYSTEM ''tasks.dtd'' >
#  Run C4.5 on a data set stored in a Data Base and return the
#  in a "results table" of the same DB
#####      Task Description Language (.tdl) file         #####
<workunit>
 <id> T1 </id>
                        ### data ###
   <fetch-data>
      <method> jdbc     </method>
      <server> dbserver </server>
      <user>   dmuser   </user>
      <db>     kdd99    </db>
      <password> _______</password>
      <db-access>
       <source-data>
          <query> select * FROM data LIMIT 5000 OFFSET 100 </query>
          <file>  kdd99.data </file>
      </source-data>
      ...
     </db-access>
   </fetch-data>
                      ### source code ###
  <fetch-code>
   <source-code>  # C4.5 code to construct the Decision Tree
     <getMethod> http </getMethod>
     <url> http://www.fe.up.pt/~rcamacho/c4.5 </url>
   </source-code>
   ...
  </fetch-code>
                    ### sub-tasks execution ###
 <execution>
    <results-storage>
      <method> jdbc </method>
      <server> dbserver </server>
      <user>   dmuser   </user>
      <db>     kdd99    </db>
      <password> _______ </password>
    </results-storage>
### sub-task 1 ###
 <subtask>
   <exec-mode>   noninteractive           </exec-mode>
   <exec-command> c4.5 -f kdd99 -m 100 -u </exec-command>
   <results-file> c45.output              </results-file>
   <exec-time>  30 </exec-time>
 </subtask>
### sub-task 2 ###
   ...
 </execution>
                    ### required resources ###
   <resources>
     <hd> 10 </hd>
     <ram> 0.5 </ram>
     <opsystem> linux </opsystem>
   </resources>
</workunit>
```

Figura 3. Descrição em detalhe de uma tarefa.

Decorre que a linguagem XML se caracteriza por permitir estruturar a informação de forma hierárquica, facilitar a edição devido à sintaxe simples (qualquer editor de texto pode ser usado), e permitir a fácil compreensão dos dados estruturados, já que não requer nenhuma ferramenta sofisticada para visualização. No caso do *Harvard* os dados estruturados em XML permitem a fácil interpretação pelos diferentes módulos da plataforma, e também a geração dos resultados para posterior integração, seja com outras ferramentas de análise de dados, ou para o armazenamento em bases de dados.

Considerando que o *Harvard* é desenvolvido em *Java*, o processamento de ficheiros em XML é facilitado devido as bibliotecas (*classes*) disponíves. E considerando as características da linguagem XML, alterações em ficheiros XML não significam necessariamente alterações em código de programação Java. Pode-se, por exemplo, acrescentar novas *tags* como requisito de um novo módulo do *Harvard*, ou simplesmente para melhor apresentar os resultados.

O utilizador pode através de um simples editor ou de qualquer outra ferramenta de edição/visualização para XML especificar o processo de análise de dados bastando para isso carregar um *template* de tarefa do *Harvard* que define todos os atributos passíveis de especificação com seus respectivos parâmetros. Uma vez escrito o ficheiro de especificação de tarefas, este é inserido como parâmetro de activação do *Harvard.*

Cabe destacar que os resultados do *Harvard*, por serem representados em XML, não necessitam de tratamento especial para apresentação ao utilizador, sendo visualizados em qualquer navegador compatível.

3 O ambiente Harvard

A infra-estrutura básica do *Harvard* é composta por dois tipos de *nós:* um nó *Servidor* e vários nós *Clientes*. Para além destes "componentes principais", a arquitectura pode aceder a um servidor *Web* para obter programas de análise de dados e um sistema gestor de BD para obter dados, armazenar resultados e efectuar um *back-up* actualizado de informação dos nós *Clientes*. Uma visão global da arquitectura da infra-estrutura básica pode ser vista na Figura 4.

A arquitectura inclui as seguintes características:

- A plataforma é baseada numa arquitectura *Servidor/Cliente* (mestre/escravo)
- Os nós *Clientes* estão sujeitos a uma política de utilização que permite a sua activação/desactivação sempre que essa política o determinar
- A linguagem de programação da infra-estrutura básica é o *Java*
- A arquitectura é modular
- A infra-estrutura básica é independente do(s) algortimo(s) de análise de dados
- O *cliente* pode ser executado tanto em sistemas operativos *Linux* como *Windows*
- Em caso de falha de um qualquer nó a tarefa em execução nesse nó é reinicializada num outro nó (tolerância à falhas dos clientes)

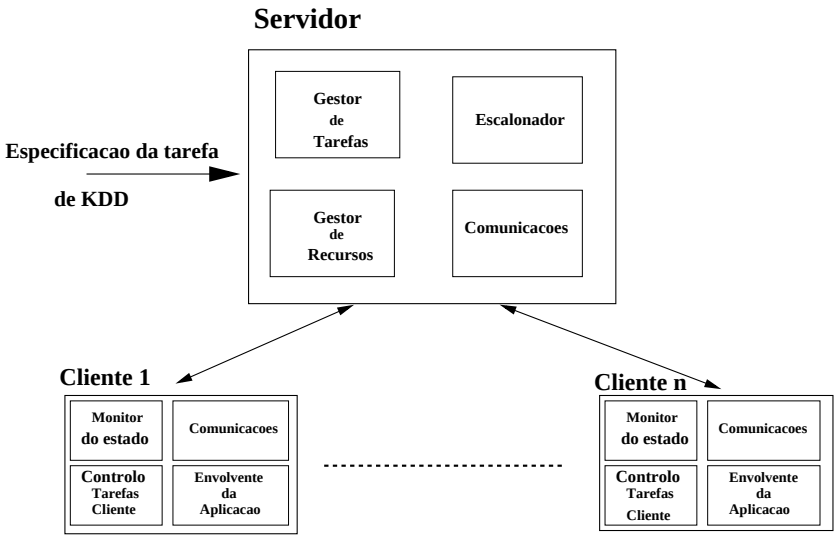


Figura 4. Arquitectura *Harvard*

- Em caso de falha do Servidor um dos clientes (previamente designado) assume o papel de servidor (tolerância a falhas do Servidor)
- Permite acesso directo a Bases de Dados e a repositórios Web para transferência de dados ou de programas

3.1 Servidor

O nó *Servidor* quando inicia "lê" informações estáticas sobre todos os recursos computacionais disponíveis. A parte dinâmica da informação dos recursos computacionais é actualizada regularmente durante a execução. O nó servidor "lê", ainda, o conjunto de tarefas a realizar, descritas como um conjunto de Unidades de Trabalho (UT). Para cada UT é seleccionada "a melhor" máquina onde vai ser executada. O servidor atribui essa UT à máquina indicada. Quando uma UT termina associa o resultado recebido à UT e devolve-o ao utilizador.

O *Servidor* é constituído por quatro módulos:

- o gestor de tarefas (GT),
- o escalonador (ESC),
- o gestor de recursos (GR) e
- o módulo de comunicação (COM).

Módulo Gestor de Tarefas (GT): Este módulo GT basicamente controla todo o processo KDD representado através de um grafo de *workflow* com todas as tarefas pertinentes ao processo de análise de dados. As tarefas por sua vez são convertidas em uma ou mais UT (unidades de trabalho) que serão processadas

pelo ambiente de forma independente. À medida que cada uma das tarefas do processo termina, os resultados são associados e armazenados no respectivo nó do grafo e um *back-up* feito num BD residente numa máquina diferente do *Servidor*. Isso permite o acompanhamento do processo de análise de dados passo-a-passo pelo utilizador e facilita a consolidação final dos resultados para apresentação. O módulo GT interage com o módulo ESC provendo todas as especificações necessárias para a distribuição e execução das tarefas.

Módulo Escalonador (ESC): O módulo ESC recebe do módulo GT as tarefas para serem executadas pelos clientes. As tarefas são especificadas em uma ou mais unidades de trabalho (UT) de acordo com o formato mostrado na Figura 5. Para cada UT, o escalonador solicita ao módulo de gestão de recursos (GR) uma máquina adequada à execução da tarefa. O módulo GR devolve a especificação de uma máquina disponível e adequada para a tarefa ou a indicação de que não há máquinas disponíveis para essa tarefa. O escalonador compõe a mensagem de requisição da tarefa e entrega-a ao módulo de comunicação para ser enviada. O escalonador recebe ainda mensagens que indicam a conclusão de tarefas. Deve recolher os resultados e devolvê-los ao utilizador.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<workunit>
  <identification> T1 </identification>
  <apllication>
    <urlapl>www.fe.up.pt/ilp/IndLog/indlog.tgz</urlapl>
    <script>www.fe.up.pt/ilp/IndLog/script-ilp.scp</script>
    <parameters>www.fe.up.pt/ilp/IndLog/parameters.txt
    </parameters>
  </apllication>
  <data>
    <dataset>kdd99</dataset>
    <DBserver>www.fe.up.pt/mysql</DBserver>
    <DB>kdd99</DB>
    <translationscript>toilp.scp</translationscript>
  </data>
  <requirements>
    <memory unit="Mbyte">1000</memory>
    <processor>Pentium</processor>
    <harddisc unit="Mbyte">1000</harddisc>
  </requirements>
  <estimatedtime unit="s"> 30 </estimatedtime>
  <results>
   <filename>kdd99.out.gz</filename>
   <DBserver>www.fe.up.pt/mysql</DBserver>
   <DB>kdd99</DB>
  </results>
</workunit>
```

Figura 5. Exemplo de especificação de uma tarefa em UT usando a linguagem XML.

Módulo Gestor de Recursos (GR): Este módulo mantém informação sobre as máquinas de acordo com a especificação indicada na Figura 6, e que inclui: informação estática e informação dinâmica que é regularmente actualizada. Entre a informação de cada máquina está o estado dela: indisponível, disponível ou em utilização. O conjunto de máquinas é mantido em 3 filas (uma para cada estado possível). Este módulo recebe ainda pedidos do módulo escalonador no formato

de um subconjunto da especificação de uma máquina e devolve uma máquina adequada ao pedido ou a indicação de que não há máquinas disponíveis para esse pedido. O módulo GR recebe periodicamente mensagens de cada cliente indicando a carga de trabalho no cliente e quantos utilizadores estão a usar a máquina.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<machine>
 <identifier>
   <ip>0.0.0.0</ip>
   <hostname>tau4</hostname>
 </identifier>
 <hardware>
   <cpu>Intel</cpu>
   <clockspeed unit="Gh">1.6</clockspeed>
   <ram unit="Mbyte">50</ram>
   <hd unit="Mbyte">250</hd>
 </hardware>
 <opsystem>Linux</opsystem>
<availability>
  <static> </static>
  <workload> </workload>
  <login> </login>
</availability>
</machine>
```

Figura 6. Descrição dos recursos disponíveis.

Módulo de Comunicações (COM): O módulo de comunicação é o único ponto de interacção directa entre *Servidor* e *Clientes*. Implementa processos de comunicação por RMI, *socket* e processa utilizando o protocolo HTTP. Realiza ainda os acessos a BD com JDBC. Este módulo é igual para o *Servidor* e *Clientes* e está detalhado na secção a seguir onde são descritos os módulos do *Cliente*.

3.2 Cliente

Um *Cliente* recebe uma UT, realiza as operações especificadas nessa UT e devolve o resultado ao servidor. O *Cliente* realiza em simultâneo a monitorização do estado da máquina em que executa. É da responsabilidade do *Cliente* activar e monitorizar o programa aplicação. Toda a comunicação entre o *Servidor* e *Cliente* é feita pelo módulo de comunicação. Um *Cliente* quando inicia regista-se no *Servidor*.

Um *Cliente* é constituído por quatro módulos:

- um módulo de monitorização do estado da máquina (MON),
- um módulo de execução de tarefas (TRAB),
- um módulo envolvente do programa aplicação (WRAP) e
- um módulo de comunicação (COM).

Módulo Controlo de Tarefas ou Trabalhador (TRAB): Este módulo interpreta as mensagens vindas do servidor e que são de três tipos: *execução de uma Unidade de Trabalho, terminação da tarefa actual e terminação de toda a actividade do Cliente*. No caso de uma mensagem para execução de uma tarefa este módulo interpreta os campos da especificação da UT. Vai buscar o programa aplicação, lança a aplicação, vai buscar os dados e guarda-os localmente no directório onde colocou a aplicação. A aplicação é carregada através de HTTP do servidor *Web* de aplicações. Os dados são carregados de uma BD usando JDBC. Na especificação da UT estão as *queries* SQL para acesso à BD e o *script* de execução da aplicação. O ficheiro resultado da execução da tarefa é colocado numa tabela da BD e comunicado ao servidor o fim da tarefa.

Módulo Envolvente da Aplicação (WRAP): Este módulo lê, interpreta linha a linha, o *script* de controlo da execução da tarefa, e coloca cada linha no *stdin* da aplicação e recolhe o seu *stdout* e *stderr*. Coloca o *stdout* num ficheiro como resultados da execução da tarefa.

Módulo de Monitorização de Estado dos Recursos (MON): Este módulo verifica o número de utilizadores e carga da máquina e comunica regularmente essa informação ao Servidor.

Módulo de Comunicações (COM): A comunicação é feita de acordo com o destinatário e a natureza da mensagem. O tipo de protocolo está indicado num campo XML da mensagem. As mensagens trocadas entre clientes e servidor utilizam RMI ou *sockets*. Existe mais um tipo de mensagem que requer o uso do HTTP e é utilizada para trazer a aplicação do servidor *Web* de aplicações para o local do cliente. Um último tipo de mensagens indica a utilização de JDBC para trazer os dados a serem processados pela aplicação. O módulo de comunicações é constituído por quatro sub-módulos e duas filas de mensagens: uma de mensagens a enviar e outra de mensagens recebidas. Cada sub-módulo é implementado por *threads* independentes que consultam filas de mensagens, identificam, e processam as mensagens que lhes competem. Cada um destes sub-módulos implementa um protocolo: sub-módulos RMI; sub-módulos *sockets*; sub-módulo HTTP e; sub-módulo JDBC. Os sub-módulos HTTP e JDBC respondem a mensagens dirigidas especificamente ao módulo de comunicações e são utilizados para buscar aplicações e dados, respectivamente.

4 Conclusões e trabalhos futuros

Neste artigo descrevemos a arquitectura e funcionamento do *Harvard* como plataforma para análise de grandes quantidades de dados. O uso da linguagem XML neste caso particular é fundamental não só pela facilidade em se codificar de forma estruturada um processo de análise de dados, mas também por proporcionar a interoperabilidade e integração com ambientes similares de análise

de dados. Também possibilida a criação de ferramentas de fácil interacção com o utilizador (analista de dados) e apresentação dos resultados. Destacamos que não utilizamos todo o potencial da linguagem nesta primeira versão do *Harvard*, e sim o que de essencial ela proporciona - a versatilidade. A versatilidade da linguagem XML foi um dos factores que permitiram seu uso no *Harvard*, seja ao nível de troca de mensagens, entrada de dados ou ainda na consolidação e apresentação de resultados. Dada a evolução da linguagem XML e das tecnologias agregadas, pretendemos expandir o *Harvard* para uma versão baseada em tecnologias *Web service*, inclusive com integração ao um ambiente *Grid Computing*, notadamente o *Globus Toolkit* (GT4). Pretende-se também efectuar a definição adequada de um *subset* da linguagem XML dedicado ao processo de KDD.

Referências

1. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.
2. D.P. Anderson. Boinc: a system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10, 8 Nov. 2004.
3. Tim Bray. Extensible markup language (xml) 1.0 (fourth edition). Technical report, W3C, 2006.
4. Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.
5. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2)(2):115–128, 1997. Provides an overview of the Globus project and toolkit.
6. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*, chapter 11, pages 259–278. MORGAN-KAUFMANN, 1999. The whole contents of the book is also useful for a complete understanding of Globus.
7. J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: a computation management agent for multi-institutional grids. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 55–63, 7-9 Aug. 2001.
8. J. Han and M. Kamber, editors. *Data Mining: Concepts and Techniques*. Morgan-Kaufmann Publishers, 2001.
9. Paulo Heitlinger. *O guia pratico da XML*. Centro Atlantico Ltda, Lisboa, 10 2001.
10. H. Kargupta and P. Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery*. AAAI/MIT Press, 2000.
11. M.J. Litzkow, M. Livny, and M.W. Mutka. Condor-a hunter of idle workstations. In *Distributed Computing Systems, 1988., 8th International Conference on*, pages 104–111, 13-17 June 1988.
12. Stephen Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.