CONSTRUÇÃO DE UM BANCO DE DADOS PARA O LIMA

Jun Kei Silva[1]; Nilton Cézar de Paula[2]

[1]Estudante do Curso de Ciência da Computação; Universidade Estadual de Mato Grosso do Sul, UEMS; 018926@comp.uems.br
[2]Professor do Curso de Ciência da Computação; Universidade Estadual de Mato Grosso do Sul, UEMS; nilton@comp.uems.br

Área Temática: Banco de Dados

Resumo: Este resumo apresenta a construção de um banco de dados através de uma API (*Application Programming Interface*) para auxiliar o sistema LIMA (***L**ight-we**I**ght **M**onitoring **A**rchitecture*) na descoberta de recursos de computação existentes em uma grade computacional. A construção do banco de dados consiste na organização de informações sobre recursos de computação em um formato adequado e no oferecimento de um conjunto de funcionalidades que venham a contribuir para melhorar a manipulação às informações armazenadas. As informações podem vir de diferentes gerenciadores de recursos locais, tais como o Condor, Ganglia ou PBS (*Portable Batch System*).

Palavras-chave: API, Gerenciamento de dados e Gerenciamento de recursos.

Introdução

Computadores interligados em rede e c*lusters* espalhados fisicamente e pertencentes a diferentes domínios administrativos podem ser utilizados de maneira transparente através de uma grade computacional (FOSTER; KESSELMAN; TUECKE, 2001). Para isto, é necessário um sistema de monitoramento de recursos de computação que colete as informações sobre os recursos e as disponibilize a outros subsistemas, tais como de seleção, alocação e gerenciamento de execução de aplicações (FOSTER; KESSELMAN, 1999).

Dentre vários sistemas de monitoramento de recursos propostos nos últimos anos, tais como o Hawkeye (HAWKEYE, 2010), Ganglia (MASSIE; CHUN; CULLER, 2004) e o Monalisa (CIRSTOIU et al., 2007), o sistema LIMA (***L**ight-we**I**ght **M**onitoring **A**rchitecture*) (DE PAULA, 2009) foi concebido com o objetivo de reduzir o uso das redes de comunicação e facilitar a seleção de recursos de uma grade de computação. No LIMA, os recursos são organizados em *pools* e cada *pool* é constituído de um *cluster* ou conjunto de computadores. As informações sobre os recursos são obtidas usando módulos de captura instalados e configurados nos *pools* e elas são distribuídas entre os diversos *pools* da grade.

O LIMA suporta provedores de informações como o Condor e o Ganglia. No entanto, vêm sendo investigadas novas funcionalidades ao LIMA para obter informações de recursos de computação gerenciáveis pelo PBS (*Portable Batch System*) (PBS, 2011). Usando o PBS surge a necessidade de organizar as informações sobre os recursos de computação, coletadas desse tipo sistema, e disponibilizá-las de forma eficiente usando tecnologias de bancos de dados. Assim, foi feito um estudo teórico/prático de bancos de dados para possibilitar a aquisição de conhecimentos para a manutenção dos dados do PBS para serem utilizados pelo LIMA.

Materiais e Métodos

Na primeira etapa foram estudados conceitos relacionados com bancos de dados, principalmente as arquiteturas existentes e as funcionalidades de manipulação de dados, como armazenamento, acesso e atualização. Esse estudo foi realizado através de bibliografia básica do assunto e documentações existentes na Internet (ELMASRI; NAVATHE, 2000). Em seguida, definiu-se as informações que o PBS possa fornecer consultando trabalhos publicados na área. Também, um formato de informações para o banco de dados foi concebido e feita a implementação das funcionalidades de manipulação do banco de dados em forma de API usando a linguagem Java (versão 1.6). O banco de dados utilizado foi o MySQL (versão 5.1) (MySQL, 2011) usando a linguagem SQL (*Structured Query Language*). Também, o ambiente de desenvolvimento utilizado foi o NetBeans IDE (*Integrated Development Environment*) (versão 7.0) e Java Development Kit (JDK) (versão 6). Antes de iniciar a implementação da API foi necessário preparar o ambiente onde foi realizada a instalação e configuração do MySQL, NetBeans IDE e Java . Na seqüência foi desenvolvida a API composta pelas seguintes classes: Main e BancoDados. Por fim, para a geração da documentação da API foi usada a ferramenta Javadoc.

Resultados e discussões

A API desenvolvida é composta por duas classes: Main e BancoDados, apresentadas na Tabela 1. A principal classe, BancoDados, oferece um conjunto de métodos que realiza as funcionalidades para a conexão e manipulação do banco de dados. Para seu desenvolvimento vários exemplos de utilização do comando SQL foram pesquisados, o que ajudou no entendimento do funcionamento de cada comando.

Na Tabela 2 são apresentados os métodos implementados na classe BancoDados.

Tabela 1: Principais classes da API.

Main	Programa principal. É responsável pela criação, manipulação e gerenciamento de informações do banco de dados por meio da criação de um objeto da classe BancoDados e chamada dos métodos desta classe.
BancoDados	Classe com os métodos API que realizam os comandos SQL. São utilizados para acessar e processar os dados contidos no banco de dados.

Tabela 2: Métodos da classe BancoDados

atualizaRegistro	Atualiza um registro existente na tabela nomeada com novos valores.
comandoSQL	Executa um comando SQL, caso não exista implementada uma determinada operação desejada pelo usuário.
criaBD	Cria um banco de dados usando um nome fornecido, caso o banco de dados ainda não exista.
criaTabela	Cria uma tabela com um nome fornecido para um banco de dados em uso.
destroiDatabase	Apaga um banco de dados com todas suas tabelas do ambiente.
destroiTabela	Remove uma ou mais tabelas de um banco de dados em uso.
fechaBD	Fecha a conexão ativa com o MySQL.
insereRegistro	Insere novos registros em uma tabela existente.
selecionaRegistro	Devolve registros selecionados de uma ou mais tabelas.
setaConexaoBD	Estabelece uma conexão com o MySQL.
usaBD	Coloca em uso um banco de dados.

Um exemplo de uso é o método de conexão com o MySQL (Figura 1): "*setaConexaoBD(String Host, String Porta, String nomeBD, String Usuario, String Senha)*", onde os campos apresentados são preenchidos para configurar a conexão e é retornado uma variável do tipo *Connection*. Antes disso, é necessário carregar e linkar a classe do driver JDBC. Na Figura 1, o "*localhost*" é a máquina onde o banco de dados está configurado, "*3306*" é a porta de conexão com o banco, "*ic*" é o nome do banco de dados (ou vazio caso não exista um banco de dados), "*jun*" é o nome do usuário e "*1234*" é a senha do usuário para acesso ao banco de dados. Já a variável *con* do tipo *Connection* irá conter um acesso válido ao banco de dados o que permitirá realizar operações sobre ele, caso o valor seja nulo nenhuma manipulação no banco de dados poderá ser feita.

```
con = bd.setaConexaoBD ("localhost", "3306", "ic", "jun", "1234");
```

Figura 1: Exemplo de conexão com o MySQL.

Após a conexão, cria-se uma tabela com os campos que o PBS disponibiliza tais como quantidade de processadores ou memória RAM disponível, espaço em disco, plataforma de execução de aplicações, disponibilidade do recurso, entre outras informações. O método utilizado para criar a tabela foi "*criaTabela(String nomeTabela, ArrayList valores)*", onde "*nomeTabela*" é inserido o nome da tabela e "*valores*" é uma lista contendo os elementos da coleção especificada, seguindo o padrão do comando SQL CREATE TABLE, como pode ser observado na Figura 2.

```
lista.add("id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT");
lista.add("np LONG");
lista.add("availmem LONG");
lista.add("physmem LONG");
lista.add("netload LONG");
lista.add("state VARCHAR (25)");
lista.add("PRIMARY KEY (id)");
bd.criaTabela("pbsnodes", lista);
```

Figura 2: Exemplo de criação de uma tabela.

Através dos métodos implementados *insereRegistro* e *atualizaRegistro*, a tabela criada, chamada "*pbsnodes*", pode ser preenchida e modificada com informações do usuário, como demonstra a Figura 3. A seqüência dos dados na lista deve seguir a ordem dos campos da tabela.

```
lista.add("4");
lista.add("11734580");
lista.add("3977796");
lista.add("55769945");
lista.add("'free'");
bd.insereRegistro("pbsnodes", lista);
```

Figura 3: Exemplo de modificação de um banco de dados.

Para consultar os registro de uma tabela no banco de dados (Figura 4), chama-se o método "*selecionaRegistro(String nomeRG)*", inserindo no "*nomeRG*" o nome da tabela. As informações são mostradas para o usuário.

```
bd.selecionaRegistro("pbsnodes");
```

Figura 4: Exemplo de consulta no banco de dados.

Caso o usuário não encontrar um método na API que manipula uma determinada operação no banco de dados, é possível enviar um comando SQL por meio do método *comandoSQL(String comando)*, onde o campo a ser preenchido deve conter a operação desejada. A Figura 5 ilustra um exemplo desse método.

```
bd.comandoSQL("SELECT * FROM pbsnodes;");
```

Figura 5: Exemplo de um comando genérico para acesso ao banco de dados.

Todos os métodos da classe BancoDados contém uma saída de mensagem de erro, caso as informações inseridas estejam incorretas ou as operações sejam inválidas.

Conclusão

O presente trabalho apresentou um estudo teórico e prático de banco de dados assim como o trabalho foi proposto, o que possibilitou a aquisição de conhecimentos para a manutenção dos dados do PBS para serem utilizados pelo sistema LIMA.

Usando o PBS é necessário organizar as informações sobre os recursos de computação em um formato adequado, coletadas desse tipo de sistema, e disponibilizá-las de forma eficiente usando tecnologias de bancos de dados.

Desenvolvendo a API, há uma maior facilidade na construção do banco de dados, na organização de informações, seleção de recursos, criação de aplicações e funcionalidades que venham a contribuir para melhorar a manipulação às informações armazenadas.

Com os resultados alcançados, este trabalho poderá dar suporte a estudos mais avançados em pesquisas posteriores e auxílio a outros trabalhos.

Referências bibliográficas

CIRSTOIU, C. et al. **Monitoring, Accounting and Automated Decision Support for the ALICE Experiment Based on the MonALISA Framework**. In Proceedings of the 2007 Workshop on Grid Monitoring, p.39-44, 2007.

DE PAULA, N. C. **Um Ambiente de Monitoramento de Recursos e Escalonamento Cooperativo de Aplicações Paralelas em Grades Computacionais**. Tese (Doutorado), Universidade de São Paulo, 2009.

ELMASRI, R.; NAVATHE, S. R. **Fundamentals of Database Systems**. Addison Wesley, 3ª edição, 2000.

FOSTER, I.; KESSELMAN, C. **The Grid: Blueprint for a New Computing Infrastructure**. Morgan Kaufmann Publishers, 1999.

FOSTER, I.; KESSELAM, C.; TUECKE, S. **The Anatomy of the Grid: Enabling Scalable Virtual Organization**. International Journal of High Performance Computing Applications, v.15, p.200-222, 2001.

HAWKEYE. Disponível em: <http://www.cs.wisc.edu/condor/hawkeye/>.
Acesso em: 20 de Outubro de 2010.

MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. **The Ganglia Distributed Monitoring System: Design, Implementation, and Experience**. In Proceedings of Parallel Computing, v.30, issue 7, 2004.

MySQL. **Conexão a um banco de dados MySQL**. Disponível em http://netbeans.org/kb/docs/ide/mysql_pt_BR.html Acesso em 26 de Abril de 2011.

PBS. **An introduction to PORTABLE BATCH SYSTEM (PBS)**. Disponível em http://hpc.sissa.it/pbs/index.html Acesso em 29 de Janeiro de 2011.