# CONSTRUÇÃO DE UM BANCO DE DADOS DISTRIBUÍDO PARA O LIMA

**Jun Kei Silva¹; Nilton Cézar de Paula²**

¹Acadêmico e bolsista do Curso de Ciência da Computação da UEMS; E-mail: 018926@comp.uems.br

²Professor e orientador do Curso de Ciência da Computação da UEMS; E-mail: nilton@comp.uems.br

Universidade Estadual de Mato Grosso do Sul – UEMS

Área CNPq: Banco de Dados

**Resumo**

Esta proposta de Iniciação Científica teve por objetivo a construção de um banco de dados distribuído com informações sobre recursos de computação para ser utilizado pelo LIMA (***L****ight-we****I****ght* ***M****onitoring* ***A****rchitecture*). As informações sobre os recursos de computação podem vir de diferentes gerenciadores de recursos locais, tais como: Condor, Ganglia ou PBS (*Portable Batch System*). A construção do banco de dados distribuído consistiu na organização de informações em várias bases de dados e no oferecimento de um conjunto de funcionalidades para manipular as informações armazenadas. Para isto, conceitos de fragmentação e alocação de dados foram os principais assuntos da pesquisa.

**Palavras-chave:** Funcionalidade. Fragmentação e alocação de dados. API.

**Introdução**

Uma grade computacional pode disponibilizar transparentemente recursos de computaçãoespalhados fisicamente e pertencentes a diferentes domínios administrativos (FOSTER; KESSELMAN; TUECKE, 2001). Para tanto, um sistema de monitoramento de recursos deve coletar informações sobre os recursos e as disponibilizar a outros sistemas tais como: seleção de recursos, alocação e gerenciamento de execução de aplicações.

Vários sistemas de monitoramento de recursos foram propostos como o Hawkeye, o Ganglia, o Monalisa e o LIMA (DE PAULA, 2009). O LIMA foi concebido com o objetivo de reduzir o uso das redes de comunicação e facilitar a seleção de recursos de uma grade de computação. Ele suporta provedores de informações como o Condor (THAIN; TANNENBAUM; LIVNY, 2003), o Ganglia e recentemente o PBS (PBS, 2011).

Para disponibilizar as informações do PBS de maneira mais eficiente, um estudo teórico/prático de bancos de dados distribuídos (KIM, 1995) foi realizado e construímos uma interface para acesso aos dados distribuídos.

**Materiais e Métodos**

Na primeira etapa foram estudados os conceitos relacionados com bancos de dados distribuído, principalmente as arquiteturas existentes, fragmentação e alocação de dados e operações para inclusão, alteração e exclusão de dados. Esse estudo foi realizado através de bibliografia básica do assunto (ÖZSU; VALDURIEZ, 1999) e documentações existentes na Internet. Em seguida, foram projetadas tabelas distribuídas e foram descritas as interfaces para realizar as operações básicas (inclusão, alteração e exclusão) sobre bancos de dados distribuídos. Para a implementação das interfaces foi utilizado o NetBeans IDE versão 6.9.1 e o JDK (*Java Development Kit*) versão 6 para a programação em Java versão 1.6. O sistema de gerenciamento de banco de dados utilizado foi o MySQL versão 5.1 (MySQL, 2011) usando a linguagem SQL (*Structured Query Language*). Antes de iniciar a implementação das interfaces em forma de API (*Application Programming Interface*) foi necessário preparar o ambiente onde foi realizada a instalação e configuração do MySQL, NetBeans IDE e o JDK . Depois, foi desenvolvida a API composta pelas seguintes classes: "Server" e "Client". Por fim, foi gerada a documentação do programa através da ferramenta Javadoc com o objetivo de dar suporte a outros trabalhos.

**Resultados e discussões**

A API desenvolvida é composta por duas classes: "Client" e "Server" (Tabela 1).

Tabela 1. Principais classes da API.

|  |  |
| --- | --- |
| **Client** | Estabelece para cada usuário (cliente) uma nova conexão com o servidor podendo então executar funções para criação, manipulação e gerenciamento de informações do banco de dados. |
| **Server** | Contém as funcionalidades para criação, manipulação e gerenciamento de informações do banco de dados e de usuários (clientes). |

A classe "Client" procura estabelecer uma conexão com o servidor, possibilitando assim a comunicação cliente/servidor. Já a classe "Server" oferece um conjunto de funcionalidades para criação, manipulação e gerenciamento de informações do banco de dado, sendo necessário manusear os usuários de forma que não ocorra conflito enquanto cada um estiver realizando suas operações.

Quando a classe "Server" é executada, o programa estará aguardando por novas conexões e ao mesmo tempo estará realizando operações de usuários conectados, até que seja interrompida a conexão. Quando a classe "Client" for executada e encontrar um servidor com a mesma porta de conexão, o usuário deverá preencher alguns campos para se conectar com o servidor, onde será chamado o método de conexão com o MySQL (Figura 1): “*setaConexaoBD(String Host, String Porta, String nomeBD, String Usuario, String Senha);*”, retornado uma variável do tipo *Connection*. Na chamada deste método, deve-se carregar e acionar a classe do *driver* JDBC (*driver* Java para conexão com o banco de dados).

Conforme a Figura 1, o *host* é a máquina onde o banco de dados está configurado, “*3306*” é a porta de conexão com o banco, *db* é o nome do banco de dados (ou vazio para *default*), *login* é o nome do usuário e *passwd* é a senha do usuário para acessar o 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 = setaConexaoBD (host, "3306", db, login, passwd);*

Figura 1. Exemplo de conexão com o MySQL.

Na medida em que o servidor aceita novos usuários, é realizada a autenticação para verificar se tal usuário já não está conectado. Caso não esteja, o usuário e a sua correspondente conexão são adicionados a um *hash table* no servidor, para facilitar o gerenciamento e manipulação de clientes.

Após a conexão, foi criada uma tabela de dados e inserido valores aleatórios com os campos que o PBS disponibiliza (PBS, 2011), como pode ser observado na Figura 4, tais como: quantidade de processadores ou memória RAM disponível, espaço em disco, plataforma de execução de aplicações, disponibilidade do recurso, entre outras informações.

A execução de operações SQL sobre o banco de dados é realizada através da comunicação cliente/servidor. O método utilizado é: “*comandosSQL (msg, login);*” conforme demonstra a Figura 2. O campo *msg* é a mensagem lida pelo servidor e na qual será executada, contanto que esteja dentro dos padrões da linguagem SQL, e *login* é o usuário que enviou a mensagem. Este método retorna uma string para *resp* que poderá ser equivalente a *msg*, o que significa que não houve problemas, ou irá conter uma mensagem de erro.

A Figura 3 demonstra como o servidor se comporta durante a execução do programa enquanto o cliente “*Jun*” se conecta e executa alguns comandos SQL (criar tabela, inserir valores e selecionar tabela), realizadas na Figura 4. O servidor sempre mostra quem está executando e qual comando foi solicitado.

*resp = comandoSQL(msg, login);*

Figura 2. Exemplo de método para execução de um comando SQL*.*

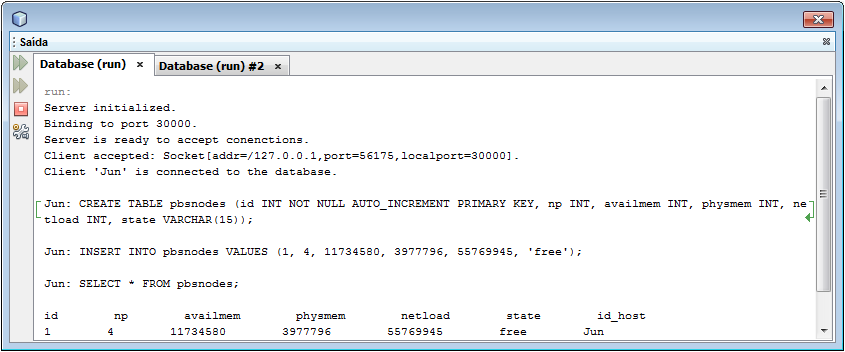


Figura 3. Exemplo de execução no Servidor.

Na parte do cliente (Figura 4), foi solicitado ao usuário inserir suas informações para poder se conectar com o servidor, conforme os campos necessários para realizar o método da Figura 1. Caso ocorra algum erro, o usuário deverá responder novamente. Em seguida, alguns comandos básicos de SQL (criação de uma tabela com nome *pbsnodes*, inserção de valores nessa tabela e selecioná-la) foram enviados ao servidor para ser executado.

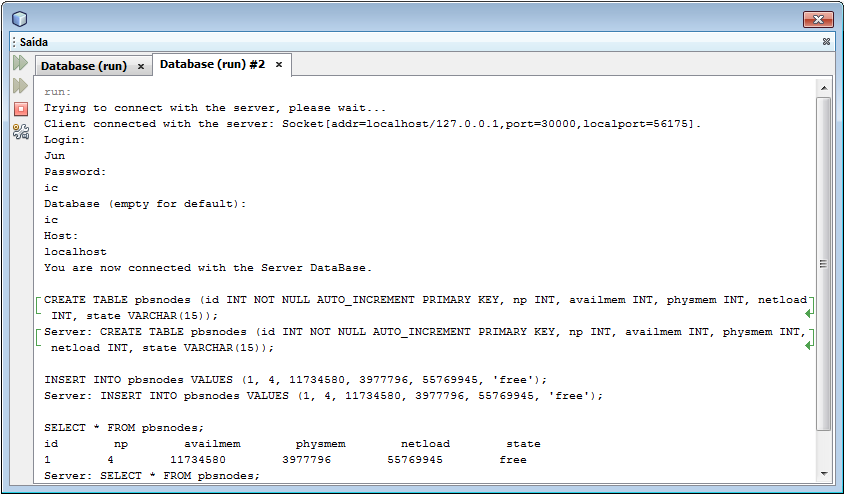


Figura 4. Exemplo de execução no Cliente.

O programa suporta mais de um usuário para se conectar ao servidor, contudo não aceita iniciar uma nova sessão com uma conta que já está sendo usada ou que não cadastrada. Também cada usuário poderá apenas visualizar e alterar os seus dados, não podendo modificar ou excluir os dados das outras contas. Algumas operações irão depender dos privilégios dado ao usuário durante a criação de sua conta, podendo limitar assim o seu uso. Porém, se o usuário for *root*, então este estará livre para utilizar qualquer comando sem nenhuma restrição.

Além dos comandos básicos da linguagem SQL, uma nova funcionalidade foi adicionada no programa, a fragmentação de tabelas. Essa função procura facilitar a criação de uma nova tabela com apenas fragmentos da tabela original com que o usuário gostaria de trabalhar. Essa operação possui uma sintaxe semelhante à operação SELECT, mas é necessário especificar o nome da nova tabela, como demonstra o exemplo na Figura 5.

FRAGMENT id, np, state FROM pbsnodes AS pbs WHERE state='free';

Figura 5. Exemplo de uso da operação FRAGMENT.

O exemplo ilustra a criação da tabela fragmentada "pbs" a partir da tabela original "pbsnodes", com apenas os campos "id", "np" e "state" e os registros que possuem "state" como "free". Esse é um tipo de fragmentação mista.

Todas as mensagens enviadas do cliente para o servidor são processadas para verificar se o usuário encaminhou um comando *exit* para finalizar o programa. Ao encerrar uma conexão "Cliente" o usuário é removido do *hash table* do servidor. Ao encerrar o "Servidor", as conexões com todos os clientes também são finalizadas.

**Conclusão**

O presente trabalho apresentou um estudo teórico e prático de banco de dados distribuído assim como o trabalho foi proposto. Durante o seu desenvolvimento, possibilitou a aquisição de conhecimentos para a manutenção dos dados do PBS, distribuídos ao longo de um sistema de redes de computadores, para serem utilizados pelo sistema LIMA. Além disso, auxiliou ainda mais no aprendizado sobre redes de computadores e ambientes cliente/servidor.

Com o desenvolvimento da API, há uma maior facilidade na construção do banco de dados distribuídos, porém é um sistema mais complexo podendo gerar outros trabalhos extras para garantir que a natureza da distribuição do sistema seja transparente. Além disso, é necessário muito esforço para manter a integridade em sistemas distribuídos e garantir a segurança, exigindo determinação na parte do desenvolvedor na organização de informações, seleção de recursos, criação de aplicações e funcionalidades que venham a contribuir para melhorar a manipulação às informações armazenadas.

Os resultados alcançados neste trabalho poderão dar suporte a estudos mais avançados em pesquisas futuras e auxílio a outros trabalhos.

**Agradecimentos**

Agradeço ao meu orientador pela oportunidade em realizar esta pesquisa e a Universidade Estadual de Mato Grosso do Sul pelo apoio financeiro que foi concedido.

**Referências bibliográficas**

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

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

KIM, W. 1995. Modern Database Systems: **The Object Model, Interoperability, and Beyond. Reading**, MA, Addison Wesley/ACM Press, 705p.

MySQL. **Conexão a um banco de dados MySQL**. Disponível em: http://netbeans.org/kb/docs/ide/mysql\_pt\_BR.html (último acesso em: 26/04/2011).

ÖZSU, M. T.; VALDURIEZ, P. 1999. **Principles of Distributed Databases Systems**, 2nd edition, Prentice Hall, 666p.

# PBS. An introduction to PORTABLE BATCH SYSTEM (PBS). Disponível em: http://hpc.sissa.it/pbs/index.html (último acesso em: 29/01/2011).

THAIN, D.; TANNENBAUM, T.; LIVNY, M. 2003. **Condor and the Grid, chapter in Grid Computing:** Making the Global Infrastructure a Reality. John Wiley & Sons, Ltd.