GridS – GridSimples: Uma Plataforma de Computação Paralela Utilizando Arquitetura Descentralizada em Grade

André Luiz Alves Moraes[1], Luís Augusto Mattos Mendes[1]

[1]Departamento de Ciência da Computação – Universidade Presidente Antônio Carlos (UNIPAC)
Campus Magnus – Barbacena – MG – Brasil

andrebq@gmail.com, luisaugustomendes@yahoo.com.br

Resumo: *Este artigo apresenta os conceitos e fundamentos básicos para o desenvolvimento de sistemas de computação paralela, bem como detalhes do desenvolvimento e implementação de uma plataforma de computação paralela utilizando a arquitetura de processamento em grade.*

Palavras-chave: Grades; Processamento Distribuído; Sistema de Arquivos; Computação Paralela

1. Introdução

Os recursos computacionais estão crescendo em um ritmo muito acelerado, porém a demanda por processamento e armazenamento cresce muito além do que o hardware tem conseguido acompanhar.

Nesta visão muitos pesquisadores e desenvolvedores voltaram-se para o desenvolvimento de plataformas distribuídas. Baseados no conceito que o todo é maior que a soma das partes, os pesquisadores vêm desenvolvendo ferramentas que permitem alcançar processamento e armazenamento por um custo acessível tornando a utilização de *mainframe* um termo restrito para certos tipos de aplicações.

A computação nas nuvens ou *cloud computing* consiste em utilizar os recursos disponíveis na *WEB* como se estes estivessem no computador do usuário, um exemplo é o *Google Docs* onde o usuário pode editar documentos *on-line* e acessá-los em qualquer computador, sendo necessário apenas um navegador compatível.

Seguindo esta linha de pensamento este trabalho visa desenvolver um sistema computacional cuja principal característica é utilizar um sistema de arquivos distribuído que permita armazenamento redundante e, sobretudo, deverá ser simples de instalar, utilizar e modificar.

As seções seguintes descrevem conceitos e características de sistemas semelhantes, na seção 2 são listados conceitos relevantes para o entendimento dos sistemas distribuídos, a seção 3 apresenta duas plataformas existentes (*BOINC* e *Globus*). Nas seções 4 e 5 são discutidas as características da ferramenta desenvolvida e

detalhes da sua implementação respectivvamente, logo após na seção 6 é apresentada uma tabela comparativa das ferramentas discutidas ao longo do artigo.

2. Conceitos e Definições

A computação paralela/distribuída apresenta uma mudança de paradigma considerável quando comparada ao modelo serial/centralizado, sendo assim alguns conceitos devem ser expostos para o melhor entendimento do trabalho.

Abaixo estão as classificações que impactam diretamente no projeto do sistema e, portanto, seu entendimento é de vital importância.

2.1. Classificação Quanto a Forma de Processamento

Segundo a taxonomia de Flynn os computadores podem ser classificados considerando o número de fluxos de instrução e dados em que trabalham, abaixo estão os quatro tipos de classificação propostos por Flynn. (ABD-EL-BARR, 2005)

- Uma Instrução Um Dado (*Single Instruction Single Data – SISD*): Os computadores nesta classe apresentam a arquitetura de computadores de Von Neummann processando apenas um fluxo de instruções em um único fluxo de dados.
- Uma Instrução Muitos Dados (*Single Instruction Multiple Data – SIMD*): Os computadores *SIMD* possuem a capacidade de executar uma instrução em diversos valores ao mesmo tempo. Considere a operação: "adicionar um valor fixo a uma série de elementos em um vetor". Nos computadores *SISD* cada elemento do vetor seria operado em um instante *t* de tempo resultando em um tempo total de $n * t$ onde *n* é o número de elementos do vetor, porém a mesma operação realizada através de uma máquina *SIMD* efetuaria a adição de todos os elementos do vetor em um único instante *t*.
- Muitas Instruções Um Dado (*Multiple Instruction Single Dada – MISD*): Nestes computadores vários fluxos de instruções poderiam operar sobre um único fluxo de dados. Atualmente não existem máquinas que apresentem este modo de operação.
- Muitas Instruções Muitos Dados (*Multiple Instruction Multiple Data* – MIMD): Este modo de operação é o mais comum e será o modelo utilizado para a realização deste artigo. No modo *MIMD* de operação, o sistema é composto por diversas unidades com seus processadores independentes e memória privada. Este modelo de computação é o mais adaptado para a utilização em sistemas nos quais as unidades processadoras encontram-se separadas por distâncias físicas consideráveis (ou seja, não estão montadas sobre a mesma placa de circuito). O modelo MIMD será utilizado na implementação do GridS pois permite a execução de aplicativos *bag of tasks*[1]

2.2. Classificação Quanto ao Tipo de Memória

Nos sistemas *MIMD* cada processador é composto por uma tripla (controladora, processador, memória), assim como nos outros modelos o *MIMD* também necessita de

[1] O termo indica uma classe de tipos de programas que podem ser executados. Será definido na seção 4 deste artigo.

formas de compartilhar informação. Sendo assim, também é necessário classificar os modelos de acesso à memória.

- Memória Compartilhada (*Shared Memory*): Neste modelo a comunicação é feita através do acesso a uma memória global única, compartilhada por todas as unidades processadoras. Sua principal vantagem é o rápido acesso à memória (para sistemas pequenos) e a facilidade de implementação, pois os programadores não precisam se preocupar com o acesso à memória (TANENBAUM, 2001). O principal problema deste modelo é o fato de não ser escalável dificultando o acréscimo de poder computacional (ABD-EL-BARR, 2005).
- Troca de Mensagens (*Message Passing*): No modelo de troca de mensagens não existe memória global centralizada somente a memória local dos processadores está presente. Quando faz-se necessário a comunicação entre as unidades uma solicitação é enviada ao detentor do endereço desejado e este responde com os dados solicitados. A vantagem deste modelo é sua escalabilidade possibilitando o aumento ou diminuição de processadores com grande facilidade. Sua desvantagem é a necessidade de que os programadores criem as chamadas manuais de solicitação de memória exigindo maior conhecimento por parte dos mesmos. (TANENBAUM, 2001)
- Memória Distribuída Compartilhada (*Distributed-Shared Memory*): Esta arquitetura de memória para sistemas paralelos une as vantagens existentes nos modelos acima. Nesta arquitetura o acesso à memória é feito utilizando-se o modelo compartilhado, liberando assim os programadores das tarefas de solicitação de memória. Porém tal acesso ocorre apenas de forma lógica sendo necessária uma camada inferior capaz de controlar os pares de requisição/resposta necessários para permitir a troca de mensagens e tornar o sistema escalável. (TANENBAUM, 2001)

O GridS não irá possuir comunicação entre processos através da utilização de memória principal. Essa decisão foi feita para simplificar o projeto e pelo fato do mesmo ser projetado para trabalhar com tarefas do tipo *bag of tasks*.

2.3. Classificação Quanto à Rede de Conexão

A classificação da rede de conexão pode ser feita em quatro sub-classificações: (ABD-EL-BARR, 2005)

- Modo de operação: determina se a comunicação é sincronizada por uma entidade central ou se o processamento é feito de forma completamente assíncrona. O modelo síncrono apesar de seguro contra problemas de diferença de velocidades dos processadores pode apresentar um desempenho muito inferior aos sistemas assíncronos. Já os sistemas assíncronos apresentam um desempenho melhor, pois os processadores podem trabalhar em toda velocidade, porém processadores mais lentos podem ser inviáveis de serem utilizados.
- Controle dos componentes: Indica como os diversos componentes do sistema são controlados: podendo ser centralizados ou descentralizados. Em sistemas centralizados o computador central determina a performance geral do sistema, podendo gerar gargalos se mal projetado.

- Técnicas de comutação: determina se o sistema é baseado em comutação de circuitos ou de pacotes.
- Topologia: Indica como as unidades estão conectadas entre si. As topologias podem apresentar comportamentos estáticos ou dinâmicos. Nos modelos estáticos as conexões entre unidades são imutáveis e podem ser: linear, anel, grade, árvore, hiper-cubo. Nas topologias dinâmicas as conexões entre os processadores podem mudar arbitrariamente. Em uma analogia, as topologias estáticas seriam aquelas utilizadas para conectar máquinas em uma rede local, já as redes dinâmicas se assemelham às redes de roteadores da internet.

2.4. Definições para a comparação entre plataformas

Na seção 6 deste artigo é apresentado um quadro comparativo das plataformas consideradas. Para o melhor entendimento do quadro os seguintes termos devem ser considerados:

- Processamento distribuído: Consiste em dividir o processamento de uma ou mais tarefas entre diversas unidades. Normalmente divide o problema em pequenas frações que são enviadas para processamento nas máquinas remotas e posteriormente os resultados são coletados e agrupados, uma abordagem de dividir e conquistar.
- Conjunto de ferramentas para sistemas distribuídos: Significa uma seleção de ferramentas importantes para disponibilizar e administrar recursos distribuídos, normalmente envolvendo organizações (criando a idéia de Organizações Virtuais), utiliza esquemas de autenticação de usuários e máquinas, controle de acesso, *firewall*, criptografia de conexões, administração e tarifação do uso.
- Sistema de arquivos distribuídos: Consiste em utilizar o mesmo conceito atual de diretórios (diretórios *UNIX* ou pastas *Windows*) em um sistema sem unidades centralizadoras. O sistema deve permitir a modificação de arquivos e diretórios de modo transparente ocultando a localização física dos arquivos. Outro ponto importante é a redundância dos dados garantindo que mesmo a falha de uma máquina não torne um arquivo indisponível.

3. Plataformas Existentes

Em busca de permitir a utilização da computação paralela em GridS, diversos projetos foram criados com focos distintos. Alguns deles apresentam apenas bibliotecas para facilitar a comunicação entre os computadores que compõe os nós, outros como é o caso do MPI buscam compilar o código nativo de modo a permitir sua execução em diversas máquinas.

Dentre os inúmeros projetos, os projetos *BOINC* e *Globus* destacam-se pela grande comunidade que os utiliza e mantém.

3.1. BOINC - *Berkeley Open Infrastructure For Network Computing*

Atualmente hospedado pela Universidade da Califórnia e apoiado pela *National Science Foundation* o projeto *BOINC* é a evolução de um dos mais bem sucedidos projetos de computação distribuída em escala global o SETI@Home. Sendo criado a partir da necessidade de poder computacional suficiente para processar a imensa quantidade de dados captados pelo radiotelescópio de Arecibo em Porto Rico, o poder computacional

do projeto SETI@Home era gerado através dos inúmeros computadores espalhados pelo globo terrestre os quais cediam voluntariamente seu tempo ocioso para baixar, processar e enviar os resultados das captações de rádio.(ANDERSON, 2008)

Em vista da grande representatividade obtida pelo projeto SETI@Home o projeto BOINC foi criado no intuito de disponibilizar um *framework* para o desenvolvimento, operação e gerenciamento de projetos científicos cujo processamento é feito através de computadores interligados através de uma rede de comunicação. Vale citar que atualmente o projeto SETI@Home utiliza o *framework* BOINC para efetuar o gerenciamento do projeto. (ANDERSON, 2008)

Por tratar-se de um *framework* o BOINC é capaz de adaptar-se as mais diversas aplicações, porém algumas restrições são feitas quanto ao design das aplicações para que as mesmas possam utilizar a tecnologia BOINC: (ANDERSON, 2008)

- Aplicações devem ser projetadas para serem divididas em inúmeros *jobs* (tarefas para processamento) os quais poderão ser processados em paralelo e de forma independente.
- Aplicações que gerem muito tráfego nas redes de dados podem tornar-se inviáveis financeiramente, pois a comunicação entre os computadores pode utilizar redes comerciais. Segundo as informações do projeto BOINC uma aplicação que consome ou gera mais de 1GB de informação por dia não seria indicada para utilizar a computação voluntária[2].

Para iniciar um projeto BOINC e obter os benefícios da computação distribuída são necessários recursos de infra-estrutura caros, segundo informações do web-site do projeto BOINC (http://boinc.berkeley.edu/), um servidor de médio porte com muita memória e espaço em disco. Também é necessária uma conexão com a internet em alta velocidade (T1[3] ou mais rápida).

3.1.1. Estrutura do BOINC

Os componentes físicos mais importantes para o funcionamento do BOINC são: um servidor capaz de rodar um servidor web (para disponibilizar informações sobre o projeto), um banco de dados que é responsável por manter o estado de todos os Jobs e espaço de armazenamento para guardar os arquivos de entrada e seus posteriores arquivos de saída. (ANDERSON, 2008)

O funcionamento básico do BOINC consiste dos seguintes passos: (ANDERSON, 2008)

- Criação do projeto no servidor: Esta etapa consiste em iniciar o projeto, registrá-lo no banco de dados, criar e disponibilizar os arquivos de entrada.
- Instalação na máquina cliente do software que irá executar o processamento dos arquivos de entrada.
- Carga e processamento dos dados pela máquina cliente, que ao término do processamento envia os resultados ao servidor.
- Validação dos dados de saída: realizada para garantir que a saída dos dados foi feita com sucesso (por exemplo, validação de precisão dos valores informados).

---

[2] Consiste na doação voluntária de recursos computacionais para uma determinada tarefa. Por exemplo, o projeto SETI@Home utilizava computadores de indivíduos que não recebiam nenhum pagamento pelo tempo de processamento disponibilizado.

[3] "T1: Conexão telefônica para transmissão de dados a uma velocidade de 1,544 Mbps, composta por 24 canais de 64 Kbps cada."(COBOL 2008)

Através desta estrutura, o BOINC é capaz de manter registros da ordem de teraflops (trilhões de cálculos em ponto flutuante por segundo).

3.2. Globus

O *Globus toolkit* é um conjunto de ferramentas para criação e manutenção de serviços em Grid. Baseado no padrão *OGSA* (*Open Grid Services Architecture* – Arquitetura Aberta de Seriços de Grid) e sendo uma implementação de código-livre da *OGSI* (*Open Grid Services Infrastructure* – Estrutura Aberta de Serviços de Grid), o Globus é a ferramenta de computação em grid que mais utiliza padrões conhecidos em sua implementação. (GAWOR 2008)

O *Globus toolkit* utiliza uma arquitetura altamente modular, permitindo ao seu usuário utilizar a plataforma modificando-a para atender suas necessidades.

3.2.1. Estrutura do Globus

O *Globus* é composto por três elementos básicos necessários para permitir o funcionamento do grid, são estes: (ALBUQUERQUE 2008)

- Gerente de alocação de recursos do Globus – *GRAM* (*Globus Resource Allocation Manager*): Responsável por gerenciar a execução dos jobs e sua monitoração. Baseado no protocolo *HTTP* ele é o responsável por gerenciar as máquinas que formam o grid.
- Infra-estrutura de Segurança em Grid – *GSI* (*Grid Security Infrastructure*): Sua responsabilidade é garantir o acesso seguro e autenticado aos recursos do grid. Como em algumas situações a utilização do grid pode ser tarifada é necessário garantir a identificação dos usuários. Uma vez autenticado, o usuário deve ser capaz de utilizar todos os recursos do grid sem que novas autenticações sejam feitas.
- Serviço de monitoração e descobrimento – *MDS* (*Monitoring and Discovery Service*): Este é o serviço de localização e monitoração dos recursos do grid. Dado o ambiente dinâmico no qual os grids são construídos, este serviço é responsável por identificar todos os recursos disponíveis no ambiente e indexá-los para que possam ser utilizados.

O único ponto negativo do *Globus* é que devido à sua grande modularidade e alta customização criam uma estrutura muito complexa de ser configurada e instalada.

4. Plataforma Proposta

Sem o intuito de questionar o desempenho das plataformas acima, ambas apresentam um problema em comum: Implementação complexa.

O BOINC exige equipamentos caros e conexões rápidas, no caso de projetos para a internet. Já o Globus é altamente modular e mesmo com o utilitário de instalação fornecido pelo Globus muitos passos são necessários para iniciar o funcionamento do grid.

Sendo assim, a plataforma GridS (Grid Simples), que será implementada, visa o fácil aprendizado por parte dos usuários, disponibilizar a utilização do grid para resolução de problemas *bag of tasks*[4] e criar um sistema de arquivos distribuído.

---

[4] Aplicações projetadas de tal forma que seja possível dividir sua execução em várias partes independentes e sem interatividade com o usuário. Após a execução das atividades independentes, outra rotina é responsável por efetuar a combinação de todos os resultados obtidos.

4.1. Definições para a implementação

Abaixo estão alguns termos que serão utilizados para a especificação das funcionalidades da plataforma:

- Grade (Grid): Representa um sistema virtual unificado, onde todos os computadores participantes podem compartilhar arquivos, memória e processamento.
- Nó (Node): Representa uma máquina conectada a outras máquinas através de uma rede de dados. Além de serem os componentes físicos do Grid os nós também possuem a responsabilidade de hospedar os serviços do grid (sistema de arquivos, memória compartilhada, processamento)
- Serviço (Service): Representa uma determinada atividade executada ininterruptamente por um ou mais nós do grid. Dentre os serviços que deverão ser disponibilizados estão: sistema de arquivos do grid, execução de jobs.
- Trabalho (*Job*): Representa uma tripla (aplicação, arquivos de entrada, arquivos de saída). O *job* representa a execução de uma determinada aplicação de maneira independente de outras execuções concorrentes. Os *jobs* na sua maioria são executados em modo não interativo.
- Procedimento Remoto (Remote procedure): consiste em um bloco de instruções que são executados em um computador diferente do computador que efetuou a chamada. Seu funcionamento é muito semelhante à uma chamada de procedimento local.

4.2 Funcionalidades Desejadas

No intuito de reduzir a curva de aprendizagem da plataforma, somente algumas funcionalidades básicas serão implementadas, dentre elas estão:

- Sistema de arquivos distribuído (GridFs): os nós componentes do grid poderão contar com a distribuição de arquivos pela rede permitindo o acesso a grandes volumes de dados. O sistema de arquivo será projetado para permitir o acesso aos arquivos sem que o usuário tenha a obrigação de conhecer a localização física dos dados. Maiores detalhes sobre a implementação do GridFs serão discutidos na seção 5 deste artigo.
- Execução de *jobs* (*bag of tasks*): Esta funcionalidade refere-se a capacidade de um cliente do grid solicitar a execução em paralelo de uma aplicação. A aplicação, os arquivos de entrada e os arquivos de saída deveram estar disponíveis através do sistema de arquivos distribuído do grid.

5. Projeto e Implementação

Quanto à linguagem de programação utilizada, em um primeiro momento a escolha foi *C++* por apresentar bom desempenho, porém uma característica foi determinante para que esta linguagem fosse abandonada: dificuldade em encontrar bibliotecas bem documentadas e portáveis. A *Platform SDK* da Microsoft era capaz de suprir todas as necessidades do projeto e possuía documentação bem acabada, porém não era portável, já o projeto *BOOST* foi capaz de suprir as necessidades do projeto, exceto no que diz respeito ao gerenciamento de processos do sistema operacional, e apresentou a portabilidade desejada, porém a documentação não segue um padrão o que aumentaria o tempo de desenvolvimento para níveis que iriam extrapolar o cronograma do projeto.

Entraram em cena as linguagens *Java* e *C#*, ambas possuidoras um bom desempenho e capazes de atender à necessidade de uma biblioteca bem documentada e portável (a portabilidade da linguagem *C#* é disponibilizada pelo projeto *Mono* patrocinado pela *Novell*). Devido à característica de utilizar chamadas de procedimentos remotos para a comunicação entre as unidades do grid, mais estudos serão feitos para identificar qual tecnologia será utilizada: Java utilizando a tecnologia *RMI*[5] ou C# utilizando a tecnologia *.NET Remoting*[6].

Após testes feitos com ambas plataformas e nos sistemas operacionais Windows XP SP2 e Ubuntu Studio 8.04, detectou-se que a plataforma Java apresentou um desempenho superior nos dois ambientes de teste. Sendo assim foi escolhida para a implementação.

Os códigos fonte bem como a documentação do sistema estão disponíveis no serviço de hospedagem de código fonte fornecido pela Google, o *Google Code*. O endereço do site do projeto é: http://code.google.com/p/grids-gridsimples/. A disponibilidade dos arquivos depende da Google continuar disponibilizando o serviço gratuitamente.

5.1. Sistema de Arquivos Distribuído

Batizado de *GridFs* o sistema de arquivos utilizado será construído sobre dois aspectos fundamentais: disponibilidade dos arquivos e facilidade de uso. O primeiro diz respeito à necessidade de que os arquivos estejam disponíveis para uso mesmo após uma falha ou desconexão de um dos computadores e o segundo refere-se ao fato de que para o usuário a complexidade gerada pela distribuição dos arquivos por diversas máquinas esteja oculta.

Baseando-se no sistema de arquivos GFS (Google FileSystem) o GridFs apresenta três elementos básicos importantes:

- Mestre de Diretório (*Directory Master - DM)*: Este servidor é o responsável por manter a estrutura de diretórios integra. Ele mantém uma lista de todos os outros DMs e também controla um vetor de servidores de arquivos. Sua estrutura de diretório está replicada em todos os outros DMs e em alguns dos servidores de arquivos controlados por ele.
- Servidor de Arquivos (File Server - FS): Este servidor armazena os dados dos arquivos em um diretório no seu disco. Cada arquivo é replicado um número N de vezes (determinado pelo DM). O FS é responsável apenas por responder a requisições de escrita e leitura nos arquivos solicitados pelos DM's (o DM funciona como uma ponte para o cliente, sendo assim, o cliente nunca acessa o FS diretamente). Cada arquivo é identificado por um GUID (Global Unique Id) que é um vetor de 24 bytes contendo um valor único em toda uma intranet.
- Clientes (Clients): Os clientes efetuam solicitações de manipulação de diretórios (criar, excluir, listar, pesquisar) para os DMs e requisições de manipulação de arquivos (ler, escrever) para os DM que possuem os arquivos indicados pelo GUID informado pelo cliente.

De forma semelhante ao GFS, o GridFs separa o gerenciamento de diretórios do gerenciamento de arquivos, permitindo que um servidor de diretório (DM) possa

---

[5] Tecnologia da plataforma Java para a invocação e utilização de métodos remotos.

[6] Equivalente .NET para a tecnologia RMI.

gerenciar diversos servidores de arquivos (FS). A grande diferença é que no GFS existe apenas um servidor de diretório que é replicado em outras máquinas. Já no GridFs podem existir inúmeros servidores de diretórios que por sua vez podem gerenciar seus servidores de arquivos.

No GridFs a unidade básica de armazenamento é o arquivo enquanto no GFS a unidade básica é o *chunk* (um *chunk* é uma parte de um arquivo, os *chunks* possuem no máximo 64MB) a adoção de *chunks* com um tamanho grande foi feita pois em média os arquivos da Google possuem 100MB de informações e os acessos de leitura são, em sua maioria, seqüenciais e os de escrita são de concatenação (*append*).(GHEMAWAT, 2008) Já o GridFs é projetado para trabalhar com arquivos pequenos, leituras e escritas randômicas.

```
package andre.grids.filesystem.common;

import andre.grids.guids.IGuid;
import java.rmi.*;

/**
 *
 * @author andre
 */
public interface DirectoryOperations extends IRemoteOperations<DirectoryOperations>,
Remote {
    public void createDirectory(String path) throws RemoteException;
    public void deleteDirectory(String path) throws RemoteException ;
    public IGuid createFile(String filePath) throws RemoteException;
    public IGuid openFile(String filePath) throws RemoteException;
    public void deleteFile(String file) throws RemoteException;
        public void setDirectoryMaster(String path, String hostName) throws
RemoteException;
    public String getDirectoryMaster(String path) throws RemoteException;
    public String[] listContents(String path) throws RemoteException;
    public boolean exists(String path) throws RemoteException;
}
```

```
package andre.grids.filesystem.common;


import andre.grids.guids.*;
import java.rmi.*;

/**
 *
 * @author andre
 */
public interface FileOperations extends IRemoteOperations<FileOperations>, Remote {
    public void createFile(IGuid fileGuid) throws RemoteException;
    public void deleteFile(IGuid fileGuid) throws RemoteException;
      public WriteOperation write(IGuid file, long fileStart, byte[] buff) throws
RemoteException;
       public ReadOperation read(IGuid file, long startPos, int count) throws
RemoteException;
}
```

Figura 1 - Interfaces principais do GridFs

O GridFs utiliza como base para sua implementação a tecnologia RMI da plataforma Java e as funções principais são disponibilizadas através de duas interfaces principais. A Figura 1 mostra as duas interfaces principais existentes no GridFs, sendo suas responsabilidades as seguintes:

- A interface *DirectoryOperations* lista as funções que permitem a manipulação da estrutura de diretórios mantida pelos DMs, os computadores clientes devem utilizar todos os métodos propostos na interface exceto o método *setDirectoryMaster*, pois este é apenas para configurar as rotas entre os DM's. A interface *DirectoryOperations* é implementada pela classe *DirectoryOperationsImpl*.
- A interface *FileOperations* é implementada em dois locais distintos: primeiro nos DM's ela funciona como uma ponte ocultando a existência dos FS's dos clientes do GridFs isso aumenta a segurança e aumenta a tolerância a falhas pois apenas o DM teria de se adaptar para mudar de FS; já nos FS a interface funciona acessando os arquivos físicos no computador e transferindo os dados para o DM que fez a solicitação.

Para banco de dados será utilizada a ferramenta DB4O, pois trata-se de um banco de dados completamente orientado à objetos, é embarcável[7], possui licença GPL e apresenta desempenho superior às outras soluções consideradas (MySQL, HSQL, Hibernate+RDBMS).

O GridFs reflete as atualizações para todos os elementos do sistema, isso significa que quando um usuário cria um diretório em um DM, este por sua vez atualiza todos os outros DM's conhecidos. O mesmo vale para as atualizações nos arquivos, sempre que um arquivo é modificado (criado, atualizado ou excluído) o DM responsável informa a todos os FS que contém uma réplica do arquivo. Esta implementação é mais lenta, porém garante a integridade dos dados.

O sistema mantém um registro de todas as operações efetuadas, assim se uma máquina perder a conexão com o *grid* poderá ser sincronizada e voltar a funcionar normalmente.

Existem seis operações principais no *GridFs*, sendo elas:

- Criar diretório: Esta é a operação mais complexa, primeiro o DM que recebeu a solicitação verifica quem é o *master* para o diretório pai do solicitado, feito isso ele verifica com o *master* se o diretório desejado já exista, caso não exista ele cria o registro na base de dados e notifica os outros DM's.
- Excluir diretório: Exclui o registro no banco, notifica os outros DM, caso contrário repassa a chamada para o *master*.
- Criar arquivo: O DM é responsável por selecionar três servidores de arquivos e criar as réplicas nos mesmos.
- Alterar arquivo: O DM responsável pelo arquivo seleciona as três réplicas e aplica as alterações nas três. Caso ocorra algum erro na atualização os servidores de arquivo depois efetuam a sincronização.
- Ler arquivo: O DM responsável pelo diretório solicita as informações de uma dos servidores de arquivo que contém réplicas do arquivo solicitado.
- Excluir arquivo: O DM responsável seleciona as réplicas, apaga os registros do banco de dados, notifica as réplicas e depois aos outros DM.

Um detalhe é que em todas as operações acima, exceto a criação de diretórios, caso o DM solicitado não seja o *master* para o diretório ou arquivo solicitado o mesmo repassa a chamada para o DM responsável, isso esconde por completo do usuário a localização física dos arquivos.

5.2. Execução de programas em paralelo

O foco principal deste artigo foi a implementação do GridFs, porém para exemplificar o seu funcionamento será desenvolvida uma pequena aplicação para processamento paralelo.

A aplicação funciona sobre o GridFs e está altamente acoplada ao mesmo, não sendo possível utilizar outro sistema de armazenamento. Seus componentes básicos são:

- Diretório de aplicação: consiste na estrutura que contém as informações para o processamento, contém o seguintes diretórios: */usr/gridApp/<appName>; /usr/gridApp/<appName>/bin; /usr/gridApp/<appName>/input; /usr/gridApp/<appName>/output*;

[7] Um sistema embarcado não necessita de instalação no cliente, basta instalar a aplicação e a funcionalidade do banco de dados já está disponível.

- Arquivos de entrada: são os arquivos que estão dentro da pasta */usr/gridApp/<appName>/input* um computador obtém um arquivo de entrada para processsar.
- Arquivos de saída: após processar um arquivo os resultados são colocados no diretório: */usr/gridApp/<appName>/output*. Os arquivos de saída são depois baixados pelo usuário e processados pelo mesmo.

6. Comparativo entre GridS, BOING e Globus

As três ferramentas comparadas buscam encontrar soluções para a computação em grade. A Tabela 1 mostra algumas das características presentes nas respectivas ferramentas.

Tabela 1 - Comparativo GridS, BOINC, Globus

Funcionalidade	BOINC	Globus	GridS
Processamento Paralelo	Sim	Sim	Sim
Foco	Processamento distribuído	Conjunto de ferramentas para computação distribuída	Sistema de arquivos distribuído
Utilização	Comercial ou Universidades	Comercial ou Universidades	Educacional, demonstrar os conceitos de sistemas distribuídos.
Sistema de Arquivo Distribuído	Não	Não	Sim
Padrões	Sim	Sim	Sim

A ferramenta *BOINC* é sem dúvida a mais madura de todas, tendo começado com o projeto SETI@home, os projetos que a utilizam alcançam velocidades de processamento altíssimas, porém sua maior deficiência é a falta de um sistema de arquivos o que a deixa restrita somente ao campo de processamento paralelo, utiliza a WEB e seus protocolos para funcionamento (basicamente o HTTP).

O projeto *Globus* visa criar as ferramentas necessárias para permitir a utilização da computação distribuída através de organizações diferentes, permitindo que o compartilhamento seja feito sob o controle das organizações. O Globus por si só é apenas uma caixa de ferramentas, ao administrador cabe a tarefa de montar e organizar as ferramentas. O *Globus* é altamente modularizado e utiliza diversos padrões, esse alto grau de modularidade gera dificuldades na instalação. Para compartilhamento de arquivos utiliza uma versão modificada do protocolo FTP.

Estando nos estágios iniciais de desenvolvimento, a ferramenta GridS permite facilitar o aprendizado pelos seus usuários, o ponto principal da ferramenta é a simplicidade da mesma. Essa simplicidade aliada a um código fonte relativamente pequeno (aproximadamente cinco mil linhas) permite que os alunos possam aplicar seus conhecimentos modificando-a e visualizando os resultados de suas alterações.

7. Considerações finais

O aprimoramento contínuo dos recursos de hardware não está acompanhando a demanda de processamento e armazenamento das aplicações atuais. Nesta linha de raciocínio muito se tem pesquisado sobre as soluções de processamento distribuído. Estas soluções visam redistribuir as aplicações em várias máquinas e assim aumentar a capacidade de um sistema acrescentando processamento paralelo. Nesta linha de pesquisa duas ferramentas se destacam: *BOINC* e *Globus*.

Ao término da produção deste artigo, muito pode ser aprendido sobre o desenvolvimento de sistemas distribuídos. Desenvolver ferramentas para computação distribuída sem o auxílio de componentes que facilitem tal tarefa é realmente muito desgastante, o programador pode acabar se preocupando tanto com a infra-estrutura a ser utilizada e acabar dando menos atenção às regras do negócio que está sendo desenvolvido.

A ferramenta Globus reduz drasticamente o tempo gasto no desenvolvimento dos programas, mesmo sendo a mais complexa de todas. A plataforma BOINC é imbatível na velocidade de processamento que alcança, muito embora dependa de boa propaganda para conseguir níveis elevados.

Neste contexto, a plataforma GridS aparece para tornar o aprendizado das tecnologias de processamento distribuído mais simples para aqueles que nunca estudaram sobre o assunto.

Muito ainda deve ser feito para que a plataforma GridS possa ser utilizada fora das salas de aula e laboratórios, porém por ter sido desenvolvida utilizando apenas ferramentas livres (OpenJDK, NetBeans, Ubuntu) ela permite que toda alteração seja feita e novas possibilidades encontradas.

Como sugestão de trabalhos futuros na plataforma GridS, alguns pontos se destacam dada sua importância para melhoria da qualidade do sistema:

- Autenticação: O GridFs não possui autenticação, portanto isto seria interessante para manter um controle de acesso.
- Conexões seguras: A utilização de conexões TCP seguras sob a plataforma RMI pode ser feita de modo transparente ao restante da aplicação.
- Recuperação automática do sistema: O sistema ainda exige muita interação manual caso ocorra uma falha, dificultando a administração.
- Ferramentas para facilitar a administração do sistema.

8. Bibliografia

ABD-EL-BARR, Mostafa, EL-REWINI, Hesham. *Advanced Computer Architecture And Parallel Processing*. Hoboken: John Wiley & Sons, 2005.

ALBUQUERQUE, Márcio Portes de, et al. *Instalação e Configuração do Globus Toolkit para Computação em Grid*. Disponível à url http://mesonpi.cat.cbpf.br/grid/PDF/nt01203.pdf. Acesso em 9 de junho de 2008.

ANDERSON, David P. *BOINC: A System for Public-Resource Computing And Storage*. Disponível à url http://boinc.berkeley.edu/grid_paper_04.pdf. Acesso em 14 de dezembro de 2008

COULORIS, George, DOLLIMORE, Jean, KINDBERG, Tim. *Sistemas Distribuídos: conceitos e projetos*. Tradução João Tortello. 4ªed. Porto Alegre: Bookman, 2007.

COBOL – *Dicionário de Informática – Letra T*. Disponível à url http://www.cadcobol.com/tt.htm. Acesso em 1 de Novembro de 2008.

GAWOR, Jarek, SANDHOLM, Thomas. *Globus Toolkit 3 Core – A Grid Service Container Framework*. Disponível à url http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf. Acesso em 9 de junho de 2008.

GHEMAWAT, Sanjay, GOBIOFF, Howard, LEUNG, Shun-Tak. *The Google File System*. Disponível à url http://labs.google.com/papers/gfs-sosp2003.pdf. Acesso em 1 de julho de 2008

KARNIADAKIS, George Em; KIRBY, Robert M. II. *Parallel Scientific Computing in C++ and MPI*. Cambridge: Cambridge University Press, 2003.

TANENBAUM, Andrew S.. *Organização Estruturada de Computadores*. Tradução Nery Machado Filho. Rio de Janeiro: LTC, 2001. 398p.