UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

RICARDO JOSÉ SABATINE

IMPLEMENTAÇÃO DE UM GERENCIADOR DE REDES OVERLAY PARA O GRIDSIM

SÃO CARLOS
2010

RICARDO JOSÉ SABATINE

Implementação de um gerenciador de redes overlay para o Gridsim

Dissertação apresentada ao programa de Pós-Graduação em Física do Instituto de Física de São Carlos da Universidade de São Paulo para obtenção do Título de Mestre em Ciência.

Área de Concentração: Física Aplicada
Opção: Física Computacional

Orientador: Prof. Dr. Gonzalo Travieso.

São Carlos
2010

Ficha catalográfica elaborada pelo serviço de biblioteca e informação IFSC/USP

Sabatine, Ricardo José

Implementação de um gerenciador de redes overlay para o GridSim / Ricardo José Sabatine; orientador: Prof. Dr. Gonzalo Travieso.-- São Carlos, 2010.

97 f.

Dissertação (Mestrado em Ciência – Área de concentração: Física Aplicada – opção Física Computacional) – Instituto de Física de São Carlos da Universidade de São Paulo.

1. Rede de overlay. 2. Computação em grade. 3. Simuladores e Linguagem Java I. Titulo.

FOLHA DE APROVAÇÃO

Ricardo José Sabatine

Dissertação apresentada ao Instituto de Física de São Carlos da Universidade de São Paulo para obtenção do título de Mestre em Ciências.
Área de Concentração: Física Aplicada - Opção: Física Computacional

Aprovado(a) em: 11.11.2010

Comissão Julgadora

Prof(a). Dr(a). Gonzalo Travieso

Instituição: IFSC/USP Assinatura_____

Prof(a). Dr(a). Francisco Isidro Massetto

Instituição: UFABC Assinatura_____

Prof(a). Dr(a). Calebe de Paula Bianchini

Instituição: Mackenzie Assinatura_____

AGRADECIMENTOS

À divina providência pelo amparo em todos os momentos.

Aos meus pais, meus irmãos, minha avó, aos meus tios e primos, a minha sobrinha e aos meus vizinhos.

Ao meu Orientador Gonzalo Travieso, pelo apóio, orientação.

Ao Adriano (pelo companheirismo e pelo rodízio de pastel e pizza (bordas do maldonado), Rodrigo (pelas engraçadas e emocionantes brigas com a Carol), Rafael Frota, Rafael/Vinicius Durelli, Leonardo (sócio da Brasile...), Gustavo (galão do lab), Raul (ex-galão do lab), Jeremias (grande conhecedor do IFSC e futuro galão do lab) e outros, pela amizade e ajuda nestes últimos anos.

À Michele Desaparecida Noda, Giulianna Comadre Marega e Silvia Figurinha Helena, pessoas pelo qual tenho enorme admiração e carinho. Qualquer forma de agradecimento será insignificante ao que elas realmente merecem.

À Juliana Trabalho Pera, Kelly Maria Handa, Priscila Chorona Freitas, Aline Encrenca Oliveira e a Vivian Risada Barone, Vick Alegria (futura esposa), Mariane Silva.

À Dona Maria (o nome dela não é esse) por pegar as cartas, o tio da quitanda (não devolvi até hj os cascos da coca cola), ao vizinho “estranho” da frente que pagava metade da internet e a Fernanda Baiana (Cadê o Pica nanda??).

À Honda por fabricar a nega veia de todos os dias e a Volks pelo lendário Apollo 13 (altas emoções).

Aos professores, aos funcionários, às faxineiras, aos jardineiros, aos cozinheiros do bandejão, ao pessoal da coordenação, aos vigias do IFSC/ICMC, e principalmente ao contribuinte paulista e a todos que, de alguma maneira contribuíram para a elaboração deste trabalho.

À FAPESP pelo apoio financeiro

RESUMO

SABATINE, Ricardo José. **Implementação de um gerenciador de redes overlay para o GridSim**. 2010. 97 p. Dissertação (Mestrado em Ciência) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2010.

Computação em grade tem se estabelecido como um importante paradigma de computação, por permitir lidar com grandes quantidades de cálculos e dados e a colaboração de participantes geograficamente distribuídos. Esses sistemas devem ser organizados de forma completamente distribuídas, com cada participante mantendo informações sobre outros participantes, e as informações necessárias ao funcionamento do sistema circulando pela rede de overlay resultante. Quando novas propostas de algoritmos, protocolos ou infra-estruturas para a grade são apresentadas, sua avaliação efetiva implica considerar sua operação com uma grande quantidade de participantes, o que invariavelmente significa que simulações devem ser realizadas. Este trabalho apresenta um sub-sistema de simulação de redes de overlay integrado à plataforma de simulação de computação de grade GridSim, de forma a facilitar o estudo desse tipo de estruturas e o desenvolvimento de novas propostas de protocolos e algoritmos para seu uso em grades de computadores. A metodologia adotada resultou no desenvolvimento de um *Java package* no GridSim com classes e interfaces que representam os conceitos básicos de redes de *overlay* e da interface dos clientes com essas redes. A partir dele foi possível desenvolver protocolos para redes estruturadas e não estruturadas no simulador e simulá-los utilizando cenários de grade de dados. Com os resultados obtidos foi possível observar que, os protocolos implementados no simulador estão de acordo com o que é encontrado na literatura.

Palavras-Chave: Rede de overlay. Computação em grade. Simuladores e Linguagem Java.

ABSTRACT

SABATINE, Ricardo José. **Implementation of an overlay network manager for GridSim**. 2010. 97 p. Dissertação (Mestrado em Ciência) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2010.

Grid Computing has been established as an important computing paradigm, since it allows dealing with a large quantity of computations and data and the collaboration of geographically distributed participants. Those systems must be organized in a completely distributed way, with each participant knowing about some other participants, and the needed information to the functioning system circulating through the resulting overlay network. When new algorithm proposals, protocols or infrastructures to the grid are presented, its evaluation implies to consider its operation with a large number of participants, which invariably means that simulations must be done. This work presents a subsystem of overlay network simulation integrated to the GridSim simulation platform, in order to facilitate the study of that type of structures and the development of new protocols and algorithms for use in grid computers. The adopted methodology led up to the development of a Java package with classes and interfaces that represent the basic concepts of overlay networks and of the clients interface with those networks. Using this package, it was possible for develop protocols to structured and non-structured networks in the simulator and simulate them using data grid scenarios. With the obtained results it was possible to observe that the implemented protocols in the simulator agree with what is found in the literature

Keywords: Overlay network. Grid computing. Simulators. Java.

LISTA FIGURAS

LISTA DE TABELA

LISTA DE ABREVIATURAS E SIGLAS

P2P	*Peer-to-Peer*
LHC	*Large Hadron Collider*
Cern	Organização Européia para Pesquisa Nuclear
DHT	*Distributed Hash Table*
CA	*Certificate Authority*
GSI	*Globus Security Infrastructure*
TLS	*Transport Layer Security*
SSL	*Secure Sockets Layer*
IP	*Internet Protocol*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
MTU	*Maximum Transmission Unit*

SUMÁRIO

INTRODUÇÃO

O surgimento dos computadores revolucionou a ciência por possibilitar a solução de novos tipos de problemas que não podiam ser resolvidos outrora. Além disso, com os recentes avanços em redes de computadores foi possível permitir a interligação de recursos computacionais em escala mundial.

A computação em grade é realizada pela interligação de recursos computacionais sob administração de diferentes organizações e indivíduos, muitas vezes participando em uma base voluntária em escala uma mundial. Esse tipo de sistema pode ser utilizado para a realização de cálculos complexos (*computational grids*), para o armazenamento de grandes quantidades de dados (*data grids*) ou para facilitar a colaboração em ambientes geograficamente dispersos (*collaborative grids*) (1).

Devido à grande quantidade de participantes, o sistema deve ser organizado de forma completamente distribuída, sem elementos que determinem centralização. Devido ao caráter eminentemente dinâmico do sistema, com constantes entradas e saídas de novos participantes, a estrutura de comunicação e distribuição de recursos na rede não pode ser rígida. Tudo isto aponta para o uso de técnicas *peer-to-peer* na implementação de serviços da grade (2, 3, 4).

Em um sistema *peer-to-peer* os participantes podem se conectar e desconectar dinamicamente. Por questões de eficiência e tolerância a falha, as informações sobre os recursos disponíveis devem ser armazenadas de forma distribuída. Os participantes devem então manter informações sobre outros participantes, para poder realizar busca dos recursos necessários. A interligação determinada por essas relações entre participantes e pelo correspondente fluxo de informações é chamada "rede de *overlay*" (3) cuja topologia se adapta dinamicamente, e que servirá de base para a comunicação entre os diversos participantes durante a operação do sistema. As redes de *overlay* são normalmente classificadas em estruturadas e não estruturadas. No primeiro caso, a topologia da rede é controlada, como: os sistemas Chord (5), CAN (6), Pastry (7) e Tapestry (8). Enquanto, no segundo caso, é livre e determinada apenas pelo processo dinâmico de conexão dos participantes na rede, por exemplo: Gnutella (35). Cada um desses tipos possui suas vantagens e desvantagens (3).

Sistemas para grades de computadores envolvem uma grande complexidade de

implementação devido à necessidade de lidar com diversos tipos de protocolos de rede, cuidar de aspecto de segurança, apoiar diversos tipos de sistemas operacionais, lidar com sistemas heterogêneos, entre outros (9, 10). Por essa razão, qualquer aspecto relacionado com algoritmos ou políticas para grades deve ser avaliado com cuidado antes de sua implementação em um sistema em funcionamento.

É esperado que esses algoritmos ou políticas sejam escaláveis para o grande número de nós ou *peers* que se vislumbra em uma grade de computadores, isto é, o regime principal de funcionamento é para um número muito grande de nós. Infelizmente, é difícil convencer um grande número de usuários a instalar uma ferramenta experimental em seus computadores, com o único intuito de avaliar o desempenho dessa ferramenta. Deste modo, a única forma adequada de realizar esse tipo de avaliação é através de simulação. Com o uso de simuladores, pesquisadores podem se concentrar na implementação do algoritmo ou política de interesse no momento, sem se preocupar com a complexidade adicional do software de grade e a avaliação pode ser efetuada considerando um grande número de nós. Além disso, tem-se maior facilidade de estudar o efeito do tráfego na rede no desempenho ou as falhas de equipamento, entre outras. Por estas razões o uso de simulação para avaliação de grades computacionais é prática estabelecida (10, 11).

Para a simulação de computação em grade, a plataforma GridSim (11) é bastante utilizada. Esta plataforma de simulação disponibiliza toda a infra-estrutura de simulação baseada em eventos, geração de números aleatórios de acordo com diversas distribuições de simulação de falhas, políticas de reserva e acesso a recursos, topologia de rede (para comunicação) incluindo geração de tráfego de *background*, entre outras funcionalidades. Entretanto não são fornecidas ferramentas para auxiliar a integração de redes de *overlay* na infra-estrutura da grade.

Objetivos

O GridSim é um sistema muito difundido para simulação de grades, porém não prove recursos para simulação de redes overlay em ambiente simulado de computação em grade.

O objetivo deste trabalho foi de permitir a integração das funcionalidades do

GridSim com as do pacote de redes de *overlay* desenvolvido, proporcionando assim uma capacidade mais ampla e integrada de simulação de infra-estrutura de grades.

Isto irá beneficiar a grande base de usuários do GridSim, além de permite a implementação de protocolos de rede overlay de forma confiável, flexível e fácil.

Para isso, foi desenvolvido neste trabalho um sub-sistema ou pacote de simulação de redes de *overlay* integrado à plataforma de simulação de computação de grade GridSim, que fornecer condições e mecanismos que possibilitam a construção de sistemas *peer-to-peer* O pacote de redes *overlay* está disponível para a infra-estrutura de grade de dados do GridSim. Foi possível desenvolver protocolos para redes estruturadas e não estruturadas no simulador e simulá-los utilizando cenários de grade de dados.

Organização do texto

O texto deste trabalho está organizado em outros cinco capítulos descritos a seguir:

O primeiro capítulo traz a revisão bibliográfica, contextualizando quanto aos aspectos envolvidos como ambientes típicos de Grade e de sistemas P2P.

O segundo apresenta aspectos de simulação e os ambientes de simulação para Grades de Computadores mais utilizados atualmente realizando um comparativo entre os mesmos.

O terceiro capítulo apresenta detalhes de implementação do pacote de *rede de overlay* desenvolvido.

O quarto capítulo apresenta uma descrição dos protocolos implementados no simulador e os resultados obtidos.

O quinto apresenta uma conclusão dos trabalhos desenvolvidos.

1 COMPUTAÇÃO EM GRADE E PEER-TO-PEER

O uso de computadores é essencial na solução dos mais variados tipos de problemas, tanto em ciência quanto em tecnologia. A evolução em termos de *hardware*, como computadores cada vez mais poderosos e redes de computadores mais avançadas, permitiram a interligação em uma escala mundial de uma grande quantidade de recursos computacionais disponível para operação em conjunto. A interligação de recursos existentes e distribuídos em termos geográficos e administrativos para formar um sistema computacional virtual recebe o nome de "computação em grade" (*grid computing*) (12, 9, 13, 14).

O termo original em inglês, "*Grid Computing*", surgiu na segunda metade da década de 1990, sendo uma analogia com a rede elétrica. Assim como no sistema de energia elétrica, no qual os usuários usam a energia sem conhecimento de sua origem e complexidade de transmissão, as grades computacionais são capazes de prover um serviço semelhante, ou seja, o usuário tem acesso a serviços e recursos compartilhados de maneira universal e transparente (15, 16).

Algumas das motivações na utilização da computação em grade são: a busca por maior poder computacional, um melhor aproveitamento de supercomputadores como MPPs que estão atualmente ociosos em diversas instituições, a confiabilidade obtida devido à existência de múltiplos recursos, e o acesso transparente a recursos remotos como bases de dados e softwares (15). Portanto a computação em grade adota o compartilhamento e a reutilização de recursos computacionais como fonte para obtenção de alto desempenho. Dentre os exemplos de grade em desenvolvimento destacam-se: Large Hadron Collider (LHC) (17), Folding@Home e Genome@Home (18).

Contundo, a exploração eficiente do seu poder computacional apresenta elevada complexidade, devido ao seu comportamento dinâmico e instável. Alguns aspectos que caracterizam grades computacionais devem ser observados (19):

- Heterogeneidade: recursos que formam a infra-estrutura tendem a ser extremamente heterogêneos, portanto, tanto em termos de *software* quanto de *hardware*;

- Distribuição geográfica em elevada escala: grades podem ter escala global, portanto, recursos podem estar distribuídos em locais geograficamente distantes.
- Múltiplos domínios administrativos: recursos das grades são gerenciados por diversas organizações, portanto, cada domínio pode definir regras distintas de acesso e usos de seus serviços e recursos.
- Compartilhamento de recursos: os recursos não são dedicados, são compartilhados entre todos os participantes, portanto, pode ocorrer elevada variação no poder computacional disponível em uma grade.

1.1 Funcionalidades de grade computacional

Podem-se classificar as grades computacionais conforme suas funcionalidades em três categorias (1):

- Grade Computacional (*Computing Grid*): integra recursos computacionais para disponibilizar uma maior capacidade de processamento na realização de cálculos complexos. Exemplo: Projeto *OurGrid* (20) e o Projeto *InteGrade* (21);
- Grade de Dados (*Data Grid*): gerencia o armazenamento e o acesso de grande quantidade de dados distribuídos em diversos repositórios. Exemplo: Projeto *DataGrid* (22);
- Grade de Serviços (*Service Grid*): fornece serviços viabilizados pela integração de diversos recursos computacionais. É subdivida em: colaborativa (*collaborative*), sob-demanda (*on-demand*) e multimídia (*multimedia*). Exemplo: Projeto *National Grid Service* (23).

1.2 Grades, *clusters* e *cloud computing*

Os principais aspectos que diferenciam uma grade de um *cluster* são: a dispersão geográfica, a heterogeneidade do *hardware*, os múltiplos domínios administrativos e a

disponibilidade dos recursos computacionais.

Devido ao caráter dinâmico, a disponibilidade de recursos computacionais em uma grade é variável, uma vez que, a computação em grade é realizada pela interligação de recursos de diferentes organizações e indivíduos, participando em muitas vezes, em uma base voluntária. Os recursos computacionais por apresentarem uma estrutura descentralizada (dispersa geograficamente), tendem a ser heterogêneos, pois cada recurso tem sua própria arquitetura (processador e memória) e normalmente são interconectados por meio de uma rede de baixa velocidade.

Os *cluster*s são fisicamente centralizados, portanto, concentrados em uma mesma área física e, normalmente os vários computadores apresentam o mesmo tipo de arquitetura (processador e memória), todos interconectados por uma rede de alta velocidade, seus recursos computacionais são dedicados e há somente um domínio administrativo (16).

Há também diferenças com relação aos tipos de aplicações: no *cluster* as aplicações típicas são aquelas com forte acoplamento, isto é, cada processo depende de outro para continuar a execução, e em uma grade as aplicações ideais a serem executadas são aquelas com fraco acoplamento (mínima a dependência entre os processos).

Um novo paradigma de computação é designado pelo termo *Cloud Computing*, que segundo Foster, pode ser definido como: "*um paradigma de computação em larga escala que tem como foco proporcionar economia de escala, em que um conjunto abstrato, virtualizado, dinamicamente escalável de poder de processamento, armazenamento, plataformas e serviços são disponibilizados sob demanda para clientes externos através da Internet*". (24).

As grades computacionais têm como foco a colaboração e compartilhamento de recursos tendo seu uso concentrado no mundo acadêmico, *Cloud Computing* tem como princípio gerar economia de escala e disponibilizar aplicações sob a forma de serviços que estarão disponíveis na Internet e poderão ser utilizados por clientes externos de forma semelhante, sendo muito mais utilizado no setor corporativo. *Cloud Computing* lida com mais transparência, facilidade e simplicidade o acesso aos seus serviços quando comparado a grades (24).

1.3 Arquitetura

Inicialmente considerando aspectos de *hardware* e *software* é possível dividir uma arquitetura de grade em três níveis que se comportam como um único e bem integrado sistema computacional (25):

- Infra-estrutura: componentes de *software* e *hardware*, integrados por uma única rede de recursos;
- *Middleware*: composto por ferramentas e serviços que oferecerem transparência de acesso aos recursos disponíveis, gerenciamento e controles da infra-estrutura e das aplicações;
- Aplicações: aplicações desenvolvidas para explorar os recursos fornecidos pela grade.

O desenvolvimento de sistemas para grades de computadores possui um alto grau de complexidade na implementação. Para facilitar a sua implementação, Foster, Kesselman e Tuecke (9), propuseram uma arquitetura na qual os componentes do ambiente fossem organizados em camadas, sendo que cada camada fornece funcionalidades específicas.

Essa arquitetura apresenta um formato semelhante ao de uma ampulheta, conforme ilustrado Figura 1. A camada central da ampulheta possui um número pequeno de abstrações de programação e protocolos, já as camadas superiores são direcionadas aos usuários e suas aplicações, consistindo de muitas funções de alto nível que são mapeadas sobre a camada central, a camada inferior, por sua vez, consiste de diferentes tecnologias mais direcionadas a questões do hardware em geral (9, 26).

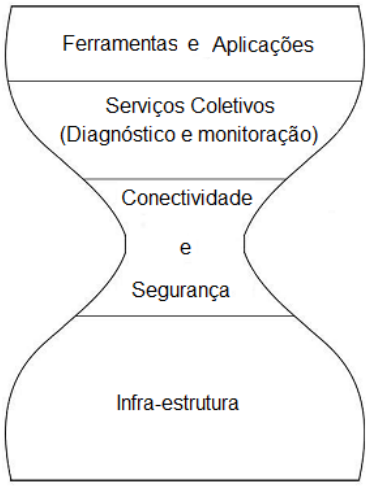


Figura 1 - Arquitetura de uma grade organizada em camada (26)

Uma breve descrição das funcionalidades de cada camada:

- camada de infra-estrutura: formada com diversos recursos compartilhados, como, computadores usando sistemas operacionais tais como Unix, Linux ou Windows, redes, cluster de computadores, sistemas de arquivos, entre outros (16).
- camada de conectividade: define um conjunto de protocolos de comunicação (IP, TCP e UDP) e autenticação (criptografia, X.509, SSL/TLS, CA (*Certificate Authority*)) exigidos para as operações de transações de uma grade. Provê acesso seguro aos recursos da grade.
- camada de serviços coletivo: é construída sobre os protocolos de comunicação e autenticação da camada de conectividade, possui ferramentas para serviços de alocação e monitoramento de recursos, escalonamento, entre outros.
- camada de aplicação: camada que o usuário efetivamente "enxerga". Constituída das mais variadas aplicações que farão uso dos recursos disponíveis pela grade.

Um conceito importante para o desenvolvimento de um ambiente de computação em uma grade é a padronização. A utilização de padrões é um requisito que auxilia a resolver problemas de interoperabilidade entre os sistemas de grade. Os padrões estão

sendo desenvolvidos pelo *Global Grid Forum*, apresentados a seguir (9, 13):

- *Open Grid Services Architecture* (OGSA): define quais serviços e toda a estrutura que pode ser fornecida em um sistema de grade, ou seja, define o modelo da arquitetura e identifica os serviços básicos. É baseada nos padrões já definidos para *web services*, porém com algumas particularidades (13).

- *Open Grid Services Infrastructure* (OGSI): é uma especificação que descreve detalhadamente o comportamento da arquitetura e dos serviços esquematizada pela OGSA. Baseado nas tecnologias de grades e *web services* define como construir as interfaces básicas, gerenciar e expandir um serviço (13).

Globus Tookit

O projeto *Globus,* iniciado em 1996, foi desenvolvido em parceria entre Laboratório Nacional de *Argonne*, a Universidade da Califórnia e a Universidade de Chicago. A primeira versão 1.0 do *Globus Toolkit* foi lançada em 1997, atualmente o *GlobusToolkit* está na versão 4.0 e inclui várias outras universidades e centros de pesquisa (27).

O *Globus Toolkit* é um conjunto de ferramentas de *software* e protocolos que fornecem suporte à arquitetura e às aplicações em uma grade. Além disso, fornece ferramentas para a monitoração e movimentação de dados na grade, descoberta e gerência de recursos, interface de autenticação e detecção e tolerância de falhas (28, 27).

Na sua última versão do Globus 4.0, os serviços são oferecidos por meio dos conceitos de *WebServices* e do padrão *WebServices Resource Framework,* entretanto há serviços que se encontram fora desse padrão, como o serviço responsável pela movimentação/transferência de dados (28).

O conjunto de ferramentas e protocolos em que o *Globus Toolkit* consiste, está dividido em cinco camadas: segurança, gerenciamento de dados, gerenciamento de execução, informações de serviços e sistemas de execução comuns. Dentre os principais protocolos e serviços, os principais são apresentados a seguir (25, 27, 28).

GRID SECURITY INFRASTRUCTURE

Em qualquer ambiente de grade é importante a existência de bons mecanismos de segurança. No *Globus Toolkit*, o serviço de segurança responsável pela autenticação de usuários na grade é o GSI. O GSI utiliza um esquema de chaves públicas e privadas para fornecer autenticação e comunicação segura. Uma vez autenticado o usuário recebe uma certificação que o habilita a acessar os recursos sem a necessidade de se autenticar novamente.

GridFTP

O GridFTP é responsável pela movimentação de dados, considerado um serviço de FTP mais robusto do que o serviço de FTP comum, pois é capaz de mover grandes quantidades de dados, de forma mais rápida, segura e confiável (utiliza vários canais TCP paralelos para transmissão).

GRID RESOURCE ACCESS AND MANAGEMENT (GRAM)

Responsável por tornar possível a submissão de processos locais ou remotos, o GRAM realiza o monitoramento da execução de um processo, controla a entrada e a saída de arquivos de dados utilizados para a execução dos processos e a submissão de um processo ao escalonador local.

Quando uma requisição é submetida ela é analisada pelo *Gatekeeper* que faz o gerenciamento desta requisição. O *Gatekeeper* tem as funções de efetuar a autenticação do usuário e criar um *Job Manager* que fará a submissão do processo para os recursos da grade e o controle dos estados dos processos desta requisição.

1.4 Modelo Peer-to-Peer

Com a rápida evolução da internet e sua utilização em milhões de computadores, espera-se que a demanda por serviços aumente em uma escala muito grande (29).

Diferente do tradicional modelo cliente-servidor, onde há uma entidade central (servidor), surgiram os sistemas *peer-to-peer* (P2P), que tem como objetivo permitir o compartilhamento de dados e recursos sem a exigência de uma organização centralizada, ou seja, todos os nós ou *peers* interagem de forma direta de maneira uniforme, podem atuar como clientes ou como servidores, compartilhando recursos sem a existência de qualquer hierarquia centralizada (29).

Os sistemas *peer-to-peer* são sistemas distribuídos consistindo de *peers* interconectados capazes de se auto-organizar em topologias de rede de *overlay.* Com este propósito é possível compartilhar recursos tais como: conteúdo, ciclos de CPU, armazenamento e largura de banda, de se adaptarem a falhas e acomodar populações variáveis de *peers* enquanto mantém conectividade aceitável e desempenho, sem necessitar da intermediação ou apoio de um servidor centralizado. (3). O modo distribuído com as informações sobre os recursos disponíveis são armazenadas, permite que o sistema P2P garanta eficiência e tolerância a falhas.

Em um sistema *peer-to-peer* os participantes podem se conectar e desconectar dinamicamente, havendo uma constante entrada e saída de *peers* da rede. Os participantes devem, então, manter informações sobre outros participantes, para que possa realizar busca dos recursos necessários. A interligação determinada por essas relações entre participantes e pelo correspondente fluxo de informações é chamada "rede de *overlay*" (3) cuja topologia se adapta dinamicamente, e que servirá de base para a comunicação entre os diversos participantes durante a operação do sistema.

Os principais sistemas *peer-to-peer* normalmente são elaborados baseando-se nos conceitos de redes *overlay*, que fornecem uma estrutura para armazenamento de dados de forma distribuída e mecanismos de consulta para a localização dos dados requeridos, sendo que é possível dividir as operações de uma rede de *overlay* em entrada e saída de *peers* e de consulta aos dados (30).

1.5 Redes de *overlay*

Uma rede de *overlay* é uma rede "virtual" criada sobre uma rede física existente, ou seja, cria-se uma topologia lógica virtual sobre uma topologia física.

Cada *peer* se conecta diretamente com outro, tendo como objetivo criar uma

estrutura de nível mais elevado de abstração que permita uma solução mais simples e adequada de determinados problemas (31, 30). Portanto, existem conexões virtuais entre os *peers* pertencentes à rede que não estão necessariamente ligados pela rede física de forma direta.

Geralmente as redes *overlay* são criadas na camada de aplicação, isso permite delegar à mesma, uma parcela da complexidade do roteamento com intuito de facilitar buscas e conexões na rede, permitindo que cada aplicação crie sua própria rede conforme suas necessidades.

Pode ser observado na Figura 2 um exemplo de rede de *overlay*.

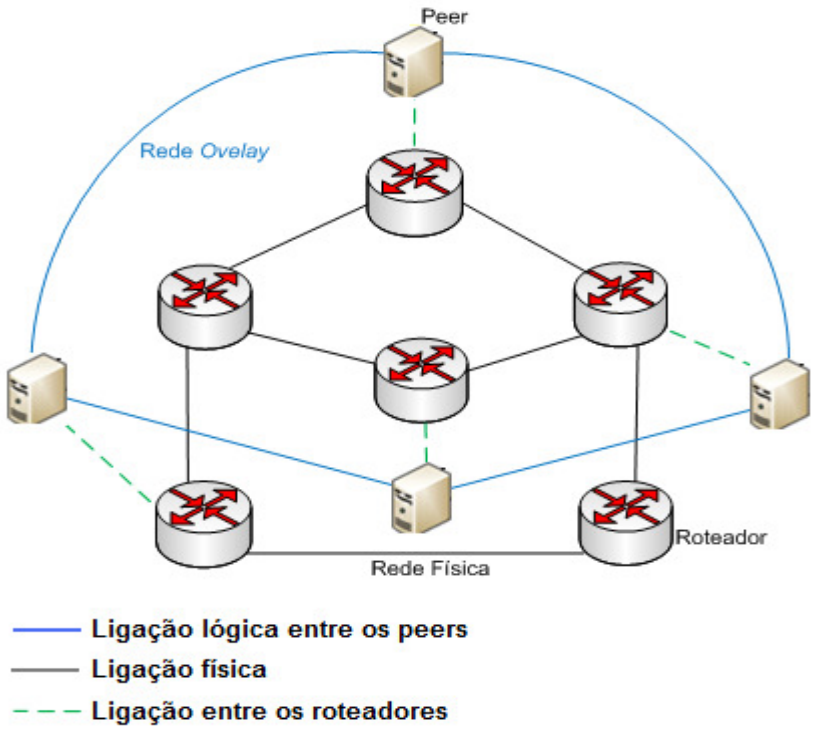


Figura 2 - Rede de *Overlay com 4 peers*

Em sua essência, redes de *overlay* são totalmente descentralizadas, porém na prática é possível observar a existência de sistemas com variados graus de centralização, na qual, é possível identificar três categorias (3):

- **Puramente Descentralizadas:** todos os *peers* da rede realizam as mesmas tarefas e atuam como servidores e clientes. Não há coordenação centralizada de suas atividades.
- **Parcialmente Centralizadas:** semelhante às redes puramente descentralizadas, contudo alguns *peers*, que são denominados *supernodes*,

assumem um papel mais importante que outros e atuam como índices centralizados para localização de arquivos compartilhados pelos *peers*, neste caso, os *supernodes* não constituem um ponto de falha crítico na rede, pois são escolhidos dinamicamente e podem ser substituídos.

- **Centralizadas:** há a presença de um servidor central que facilita a interação entre os *peers*, neste caso, os servidores centrais constituem um ponto de falha crítico na rede.

O desempenho e estabilidade do sistema dependem da organização da *rede de overlay*, portanto esta recebe bastante ênfase na literatura. As redes de *overlay* para sistemas *peer-to-peer* são classificadas em duas categorias principais, estruturadas ou não-estruturadas, de acordo com os mecanismos utilizados para a sua construção (3, 32). A diferença fundamental entre essas categorias se resume na condição da organização dos *peers* permitir, ou não, a localização determinística de um de dado (32).

Redes Não Estruturadas

As redes não estruturadas possuem a topologia livre e determinada apenas pelo processo dinâmico de conexão dos participantes na rede, as regras para a conexão dos nós são mais flexíveis e são intuitivamente mais fáceis de serem implementadas e mantidas.

Um novo *peer* é inserido na rede apenas se efetuar comunicação com um *peer* que já pertença a esta rede, após esta ação, deve publicar os dados que irá disponibilizar aos outros membros da rede. A descoberta de outros *peers* existentes na rede pode ser feita por meio de mecanismos como *multicast*. Os *peers* são conectados de forma aleatória e, não há restrição sobre o número de participantes. Os *peers* que não desejam mais utilizar a rede podem sair a qualquer momento e em alguns sistemas sem a necessidade de informar aos demais membros da rede. Em alguns casos, *peers* podem deixar a rede de forma abrupta, como em casos de falta de energia, perda de conectividade da rede, entre outros.

Identificar o *peer* da rede responsável por um determinado dado é um fator importante, pois após entrar na rede, um *peer* emite uma consulta para algum outro *peer* da rede. Essa consulta é, então, propagada por vários *peers* através de um protocolo de roteamento, até chegar ao seu destino. Quando a consulta é bem sucedida, o valor

retornado será alguma identificação do *peer* (por exemplo, endereço IP) onde a consulta foi satisfeita. Nesse ponto, o *peer* solicitante pode obter os dados diretamente do *peer* destino.

Os mecanismos de consulta para a localização dos dados incluem algoritmos como: *flooding* (33), *random walks* (34) e entre outros métodos. O mecanismo de *flooding* não é eficiente para localização de dados quando o tamanho da rede cresce muito, pois necessita de um grande poder de processamento e largura de banda. Sistemas como Gnutella e Kazaa utilizam essa arquitetura.

***Flooding*:** quando um *peer* precisa de um dado especifico, ele envia uma mensagem de *query* a todos os seus vizinhos, cada *peer* que recebe a mensagem verifica se tem o dado requisitado, se tiver ele envia uma mensagem contendo sua identificação (normalmente o endereço IP) para o *peer* requisitante, caso contrario, incrementa o número de *hops* e repassa a requisição recebida para todos os seus vizinhos. Esse processo repete-se em cada *peer* da rede até que a requisição seja atendida ou que o TTL (*Time To Live*) (utilizado em alguns sistemas como Gnutella) que representa o número de hops que a requisição pode ter seja atingido, conforme ilustrado na Figura 3.

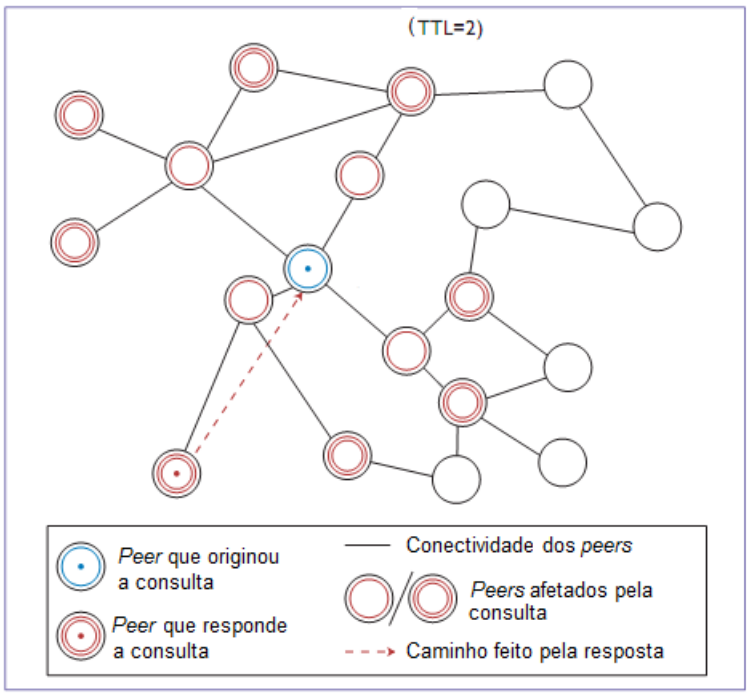


Figura 3 - Exemplo de busca em uma rede não estrutura, utilizando *flooding* (30)

Redes Estruturadas

As redes estruturadas possuem a topologia da rede controlada, as regras de organização para a conexão dos *peers* são mais rígidas, utilizam regras para distribuir os dados na rede de modo que facilite a posterior localização dos mesmos.

Os principais protocolos utilizados são baseados na noção de uma tabela de indexação distribuída DHT (*Distributed Hash Table*). A principal vantagem dessa abordagem é a localização eficiente e escalável dos dados, embora ao custo de uma maior sobrecarga no sistema e dificuldade em suportar populações de *peers* muito dinâmicas. Exemplos: Chord, CAN, Pastry e Tapestry.

DHT: os identificadores dos *peers* e dos dados são calculados utilizando uma função *hash*. Os dados são armazenados nos *peers* que possuem o identificador igual, ou o mais próximo ao seu. Cada *peer* possui conexões com apenas um pequeno conjunto de outros *peers* e repassa as mensagens para apenas um deles. Quando uma consulta por um dado vai ser realizada, primeiro o identificador associado ao dado é calculado e a consulta é passada de *peer* em *peer*. O *peer* escolhido será o que possui o identificador mais próximo do identificador pesquisado, portanto, requisições são encaminhadas com base no identificador do dado até que ele seja encontrado, permitindo que um dado seja encontrado em um pequeno número de passos (5, 30), conforme ilustrado na Figura 4.

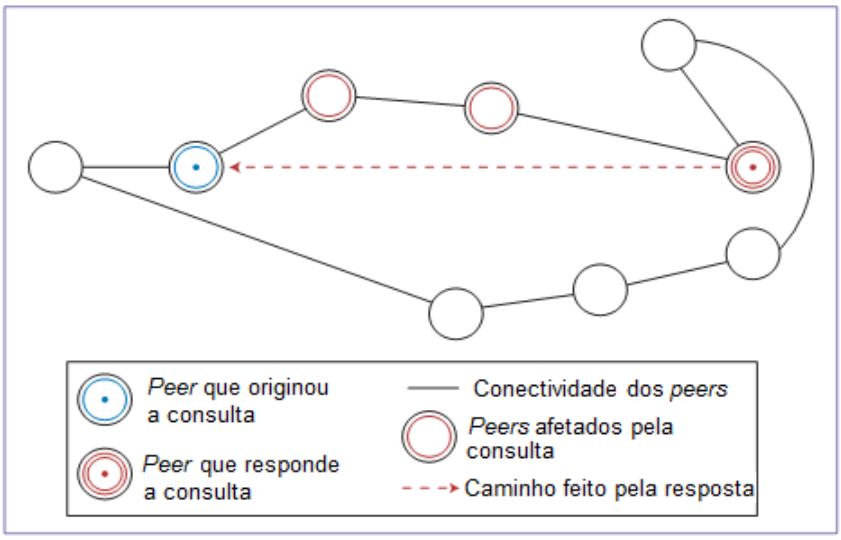


Figura 4 - Exemplo de uma busca em uma rede estruturada (30)

1.6 Sistemas P2P

O desenvolvimento de sistemas e aplicativos *peer-to-peer* teve inicio com o serviço de compartilhamento de música Napster, depois surgiram os sistemas de compartilhamento de arquivos, incluindo Freenet, Gnutella, Kazaa e Bit-Torrent, porém estes sistemas apresentam uma ineficiência na localização de arquivos, os sistemas baseados em DHT surgem para tentar localizar eficientemente o *peer* que armazena um item de dados particular.

A seguir, são apresentados um exemplo de sistemas não estruturado (Gnulella) e um sistema estruturado (Chord).

Gnutella

O protocolo Gnutella é um sistema de compartilhamento de arquivos (qualquer tipo) na qual não existe um controle sobre a topologia da rede. Todos os *peer* apresentam funcionalidades idênticas, sendo denominados de *servents*, um acrônimo para "*server*" e "*client*".

Cada *peer* mantém informações a respeito de alguns *peers* vizinhos, para executar uma consulta, o protocolo Gnutella, utiliza algoritmo de *flooding* baseado em BFS (*Breadth First Search*). O protocolo é composto basicamente por cinco tipos de mensagens (35):

- *Ping*: mensagem usada para descobrir outros *peers* na rede. O *peer* que recebe uma mensagem do tipo *ping* deve responder com uma ou mais mensagens de *pong* ao remetente.
- *Pong*: resposta à mensagem *ping*. A mensagem *Pong* contém o endereço do *peer*, a porta, e a quantidade de dados que o mesmo compartilha
- *Query*: Mensagem utilizada para realização da uma consulta na rede. Um *peer* que recebe uma mensagem *query*, deve realizar uma consulta interna, e

se tiver a informação especifica na mensagem *query*, deve responder com uma mensagem de *query* hit ao remetente.

- *Query* hit: é a mensagem de resposta a uma *query*. *Query hit* contém o conteúdo necessário para o acesso da informação.
- *Push*: é a mensagem que permite ao *peer* enviar arquivos para a rede, é utilizada para que o *Firewall* permita o tráfego de mensagens de troca de arquivos.

Chord

O protocolo Chord (5), é baseado no mecanismo DHT (*Distributed Hash Table*), ou seja, utiliza chaves para mapear, localizar e remover *peer* e dados na rede. O protocolo Chord se responsabiliza pela localização das chaves (dados), gerenciamento de entrada e saída de *peers* da rede e pela recuperação de falhas nos *peers* existentes, disponibiliza somente a função *lookup* a camada de aplicação.

O posicionamento dos *peers* e dos dados é dado por uma função *hash* consistente, o que permite equilibrar a carga, uma vez que cada *peer* recebe aproximadamente o mesmo número de chaves (dados).

O mapeamento de um *peer* é feito aplicando-se a função *hash* ao seu endereço IP, a função *hash* retorna o chamado identificador do *peer*, os identificadores são organizados em um círculo de identificadores de tamanho 2^m, onde m é número de *peers*.

Cada *peer* mantém uma tabela de roteamento (*Finger Table*) com m entradas, onde m é o número de *peers* da rede, essa tabela possui informações sobre outros *peers* (*peers* sucessores), os *peers* sucessores são definidos utilizando a função *successor*($n + 2^{i-1}$), onde n é o número do próprio *peer* e i é número da entrada na tabela (5). As tabelas são atualizadas automaticamente sempre que houver a entrada ou saída de um *peer* da rede, isso assegura que um *peer* responsável por uma chave possa ser sempre encontrado.

A Figura 5 demonstra uma rede Chord com três *peers*: 0, 1, 3 e com três dados: 1, 2 e 6. Uma chave (dado) k é atribuída ao primeiro *peer* cujo identificador é igual ou segue k no espaço de identificadores, por exemplo, a chave 1 é armazenada no *peer* com identificador 1, pois neste caso o identificador é igual, já a chave 6 é armazenada no primeiro *peer* sucessor ativo neste caso o *peer* com identificador 0.

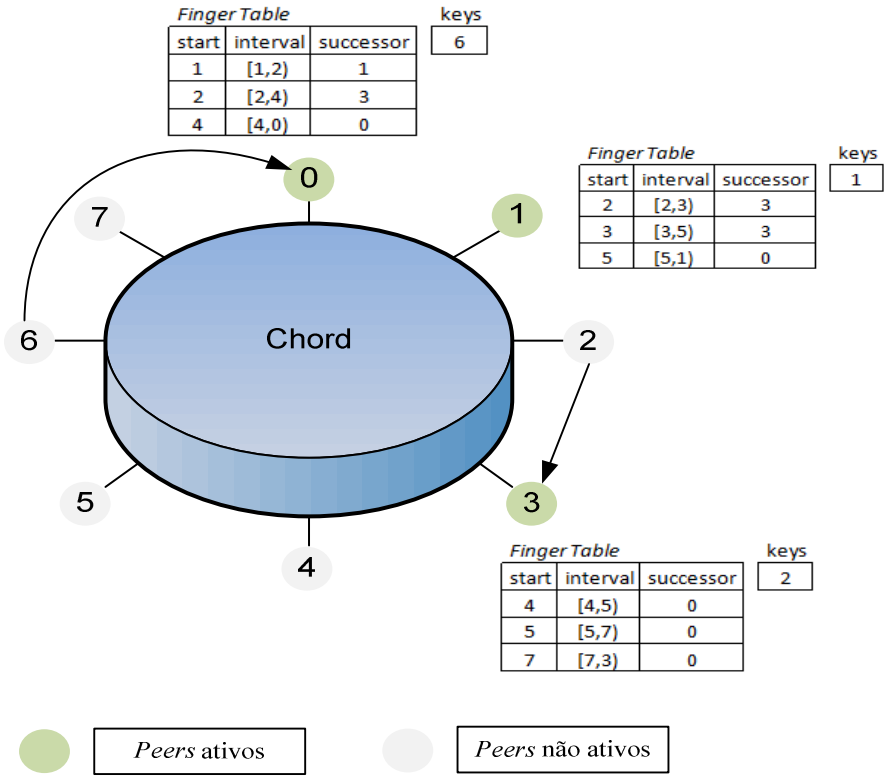


Figura 5 – Exemplo de uma rede de *overlay* estrutura Chord

1.7 Computação em grade e sistemas P2P

Atualmente, grades de computadores e sistemas *peer-to-peer* apresentam diferenças significativas. São baseadas em comunidades de usuários diferentes, onde grades têm como objetivo resolver problemas complexos, que normalmente envolvem muito poder de processamento, já as redes *peer-to-peer* visam principalmente resolver problemas de complexidade mais simples, porém que sejam solicitados por um número muito grande de usuários, como é o caso das redes *peer-to-peer* para compartilhamento de arquivos (4, 16, 2).

A busca de recursos e informações em uma grade de computadores torna-se cada vez mais difícil com o crescente volume de informação e recursos em redes de computação. A descoberta rápida e eficiente dos recursos disponíveis e a manutenção dos estados dos recursos são os principais requisitos para uma grade de computadores possa atingir uma ótima utilização do sistema e do equilíbrio de carga entre os computadores

participantes (36).

A descoberta de recursos e informações em grades é baseada principalmente em modelos hierárquicos ou centralizado. Sistemas como Globus ou Condor utilizam soluções baseados em servidor para atribuir tarefas para os recursos disponíveis. Essa abordagem funciona bem para pequenas e médias grades, no entanto, para grades de grande dimensão, até de escala global, esta abordagem não é eficiente e não escala bem. Além disso, mesmo para pequenas redes, uma solução centralizada terá um único ponto de falha (36).

Para descobrir recursos de maneira mais dinâmica, em grande escala, técnicas de *peer-to-peer* tem sido utilizado em grades de computadores. Redes de *overlay* geram implicitamente equilíbrio de carga e escalam muito bem para grandes números de *peers* e dados (37). No entanto, a tecnologia *peer-to-peer* traz novos problemas, no caso de sistemas *peer-to-peer* não estruturado baseado em *flooding,* não há uma boa escala em termos de *overhead* de mensagem, para este caso, *random walks* pode ser uma alternativa a *flooding*, uma vez que reduz a quantidade de tráfego de mensagem na rede, já os sistemas baseados em DHT têm-se mostrado escalável e eficiente, no entanto, uma característica é a falta inerente apoio a consultas complexas (37).

Projeto OurGrid

Apresentaremos um exemplo do uso de *peer-to-peer* em grades.

O *OurGrid* é um projeto de computação em grade, desenvolvido no LSD/UFCG (Universidade Federal de Campina Grande) com apoio da HP (*Hewlett-Packard*), tem como objetivo aproveitar recursos computacionais ociosos de laboratórios de pesquisa de pequeno e médio porte (20). Utiliza uma abordagem P2P na qual, qualquer um pode participar, ou seja, é baseado em uma rede P2P onde cada laboratório de pesquisa é um *peer* da grade (20). O *OurGrid Toolkit* contém três componentes principais: *MyGrid*, *OurGrid Peer* e o serviço de segurança SWAN (38).

O componente *MyGrid* fornece abstrações de alto nível, o que permite ao usuário utilizar os recursos da grade. Foi projetado para ser transparente e completo, sendo responsável pelo escalonamento das aplicações nos diversos recursos em que o usuário tenha permissão de acesso. O *OurGrid Peer* é responsável por alocar recursos tanto localmente quanto em outras comunidades, ele verifica se o recurso satisfaz os requisitos

para o trabalho.

Conforme pode ser observado na Figura 6, uma instância da Comunidade *OurGrid* é composta por um conjunto de *peers,* sendo que cada *peer* representa um site. Cada usuário utiliza uma instância local MyGrid para interagir com o *OurGrid Peer* local. Quando o usuário submete um *job* pelo MyGrid, o mesmo solicita ao *OurGrid Peer* local a montagem de um conjunto de máquinas da rede que satisfaçam os pedidos, caso o *OurGrid Peer* local não tenha todos os recursos necessários para satisfazer os pedidos na sua rede local, ele solicita recursos aos outros *peers* da comunidade *OurGrid* (38). O OurGrid utiliza um sistemas denominado Rede de Favores que premia os *peers* que mais contribuem, ele serão favorecidos quando necessitarem de recursos (38).

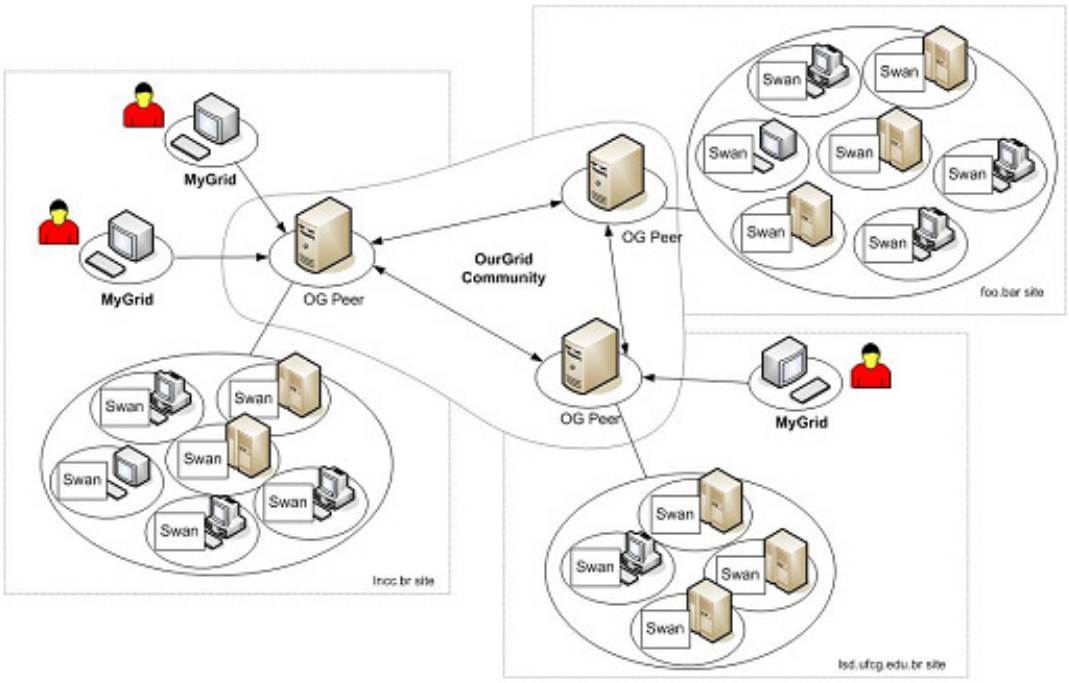


Figura 6 - Comunidade OurGrid (38)

O *OurGrid* suporta aplicações paralelas do tipo *Bag-of-tasks*, nas quais as tarefas são formadas por subtarefas executadas seqüencialmente sendo: uma tarefa inicial, uma tarefa grade e uma tarefa final. As tarefas, inicial e final, são executadas na máquina do próprio usuário que está submetendo a aplicação, e têm como objetivos, preparar os arquivos de entrada e coletar os resultados finais, respectivamente. As abstrações do *MyGrid* possibilitam que as subtarefas sejam construídas sem conhecimento de detalhes sobre os recursos a serem utilizados para a execução.

2 AMBIENTE DE SIMULAÇÃO

A simulação tem sido amplamente utilizada na modelagem e avaliação de sistemas do mundo real. Assim, ao longo dos anos, a modelagem e a simulação têm emergido como um importante fator para o desenvolvimento de vários padrões e aplicações de componentes e de tecnologias (11).

Simuladores são ferramentas de alto nível que permitem abordar comportamentos específicos de um sistema, sendo muito úteis na medida em que possibilita a análise de diversos cenários diferentes, o que permite uma economia preciosa de tempo, custo e esforço (39).

Os resultados recolhidos a partir da simulação indicam como o sistema real irá se comportar, permitindo assim uma melhor concepção e compreensão por parte dos desenvolvedores, sem a necessidade da execução propriamente dita do sistema real (39).

A avaliação de proposta de protocolos, algoritmos ou metodologias para grades envolve dificuldades relacionadas com o grande porte (em número de participantes e de extensão geográfica) que se espera em uma grade.

Isto significa que os resultados de testes em pequena escala, realizados em apenas um laboratório, trazem pouca informação sobre como uma dada proposta se comportará durante a operação em situações reais, pois envolvem uma quantidade muito menor de participantes, uma latência menor (e mais uniforme) nas comunicações e maior confiabilidade.

Dada a escala, a complexidade e a heterogeneidade inerente de uma grade computacional, é difícil produzir de forma repetitiva e controlada uma avaliação do desempenho.

Além disso, grades experimentais são limitadas, e criar uma grade com dimensão experimental adequada possui alto custo e exige muito tempo. Com isso, antes da efetivação de uma grade computacional, existe a possibilidade da utilização de simuladores computacionais com o intuito de obter, de forma prévia e com custos reduzidos, informações relevantes sobre o comportamento da futura estrutura física e lógica. Portanto, é mais fácil usar simulação como um meio de estudar cenários complexos.

2.1 Ferramentas de simulação para grades

O uso de simulação para avaliação de grades computacionais é prática estabelecida, existem algumas ferramentas para a simulação de aplicação em ambientes de grades computacionais disponíveis, tais como, OptorSim (10), MicroGrid (40), SimGrid (41) e GridSim (11) simulador que está relacionado com este trabalho e que abordaremos com mais detalhes.

2.2 SimGrid

O SimGrid toolkit foi desenvolvido em 1999, num projeto de pós-doutorado desenvolvido por Henri Casanova na Universidade de Califórnia em *San Diego*, consiste de uma ferramenta de código aberto implementada em linguagem C para a simulação orientada a eventos. Provê um conjunto de abstrações e funcionalidades básicas que permitem construir simulações para aplicações de domínio específico, visando o estudo de algoritmos de escalonamento para aplicações paralelas em plataformas computacionais distribuídas. Utiliza arquivos XML como entrada para definir a topologia de rede e as características e responsabilidades dos recursos. Usando a API SimGrid, tarefas podem ser atribuídas aos recursos, dependendo da política de escalonamento a ser simulada (41).

O SimGrid oferece ao usuário quatro interfaces (41): GRAS (*Grid Reality And Simulation*) é um framework que possibilita o desenvolvimento e o teste de aplicações reais para grade; o SMPI tem como finalidade oferecer suporte a execução de programas MPI; o SIMDAG permite a execução de tarefas paralelas interdependentes; e o MSG permite maior praticidade na construção de simulações, sendo desnecessária a implementação da aplicação.

O SimGrid não é capaz de modelar todos os fenômenos envolvidos (por exemplo, erros de rede e topologias de rede complexas), entretanto apresenta uma boa

documentação, permite a modelagem de tráfego e computação de fundo, proporciona um nível adequado de abstração, além de produzir resultados precisos e uma simulação eficiente.

2.3 OptorSim

O Optorsim é um simulador de grades direcionado para grades de dados, implementado em linguagem Java, ele permite que sejam estudadas estratégias de escalonamento que são baseadas em replicação. Para tornar a simulação a mais realista possível, a arquitetura do OptorSim foi baseada no projeto EU *Data Grid*. O OptorSim simula o comportamento da grade, conforme os parâmetros definido pelo usuário, como topologia, estratégia de replicação, política de escalonamento, tipo de recursos disponíveis, existência ou não de tráfego de fundo e conjunto de tarefas (10).

No OptorSim a grade é formada por diversos sites que provêm nenhum ou vários Elementos de Computação (CE) e Elementos de Armazenamento (SE). O CE executa tarefas que utilizam os dados armazenados nos SEs. O *Resource Broker* (RB) controla o escalonamento de tarefas nos sites, o RB tenta distribuir todas as tarefas em todos CEs, para evitar ociosidades em alguns CEs.

O Replica *Manager* (RM) é responsável pela manipulação dos dados em cada site, como por exemplo, movimentação de dados entre os sites; já o Replica *Optimizer* (RO) que pertence ao componente RM, contém o algoritmo de otimização de réplicas que decide pela criação automática ou pela eliminação de réplicas.

O OptorSim apresenta algumas limitações, por exemplo: a modelagem da rede dentro dos sites é limitada e a pouca flexibilidade na modelagem da plataforma da grade, entretanto, permite a modelagem de tráfego de fundo e é portável (10).

2.4 MicroGrid

O MicroGrid é um emulador de grades seguindo a concepção do Globus *Grid*. É uma ferramenta de código aberto, implementado em linguagem C que utiliza arquivos XML como entrada para definir as características dos recursos, bem como para especificar as tarefas a serem executadas (40).

O objetivo do projeto MicroGrid é desenvolver e implementar uma ferramenta de simulação que permita o estudo científico e sistemático das interações entre aplicações, *middleware*, recursos e redes em uma grade. Em suma, o MicroGrid fornece uma infra-estrutura de grade virtual que permite a realização de experimentos de forma controlável e reproduzível. A virtualização dos recursos é feita de forma transparente ao usuário.

O MicroGrid possibilita a avaliação de aplicações reais, uma vez que a ferramenta suporta aplicações que fazem uso da infra-estrutura do *Globus Grid*.

Para o funcionamento do MicroGrid faz-se necessário a instalação da biblioteca de troca de mensagem MPI, visto que, a execução do MicroGrid é paralela.

2.5 GridSim

O GridSim (11) é uma ferramenta que permite a modelagem e a simulação de diferentes classes de recursos heterogêneos, usuários, aplicações e escalonadores (*brokers*), fornecendo primitivas para criação, gerenciamento e mapeamento de tarefas nos recursos, podendo ser usado para simular escalonadores de tarefas para sistemas distribuídos com um (*clusters*) ou vários domínios administrativos (grades).

O GridSim permite a modelagem de uma ampla gama de recursos heterogêneos tais como monoprocessadores, multiprocessadores de memória compartilhada, máquinas de memória distribuída, tais como SMPs e clusters com diferentes capacidades e configurações. Os recursos podem ser modelados utilizando diferentes políticas de alocação como espaço compartilhado (*space-shared*) ou tempo compartilhado (*time-shared*). É possível definir a arquitetura dos recursos, sendo que a capacidade dos mesmos pode ser definida na forma de MIPS (milhões de instruções por segundo).

A ferramenta de simulação permite ao usuário criar sua própria topologia de rede, permite também a geração de tráfego de fundo e a incorporação de diferentes níveis de serviços para o envio de pacotes. A arquitetura de rede existente permite que as entidades do GridSim possam ser conectadas usando *links* e roteadores, com políticas de escalonamento de pacotes tais como FIFO (*First-In, First-Out*), WQF (*Weighted Fair Queuing*) e SCQF (*Self Clocked Fair Queuing*) implementadas nativamente no GridSim.

O roteamento pode ser feito através de tabelas estáticas ou métodos dinâmicos, como o *Routing Information Protocol* (RIP) e *Open Shortest Path First* (OSPF).

Principais características do GridSim incluem (11):

- Os recursos podem ser localizados em qualquer zona de fuso-horário, ou seja, permite o ajuste do relógio para fusos horários diferentes para simular a distribuição geográfica dos recursos;
- Fins de semana e feriados podem ser mapeados em função da hora local do recurso para modelar carga de trabalho não referente à grade (carga de trabalho local);
- Os recursos podem ser reservados antecipadamente para sua utilização durante um determinado período de tempo;
- Tarefas podem ser heterogêneas e podem ser essencialmente de computação (CPU *intensive*) ou de dados (I/O *intensive*) ou ambos;
- Não há limite no número de tarefas que podem ser submetidos a um recurso e vários usuários podem submeter tarefas para execução simultânea no mesmo recurso;
- Suporta simulação de escalonadores estáticos e dinâmicos;
- Suporte a grades de dados;
- Estatísticas da totalidade ou de operações selecionadas podem ser gravadas e então serem analisadas usando métodos de análise estatística fornecido

pelo GridSim.

Arquitetura GridSim

A primeira arquitetura proposta para o GridSim é dividida em multi-camadas de forma modular, conforme pode ser observado na Figura 7, isso permite a integração fácil de novos componentes ou camadas (11).

A primeira camada está relacionada com a máquina virtual Java, chamada JVM (*Java Virtual Machine*), cuja implementação está disponível para sistemas monoprocessados e multiprocessados.

A segunda camada procura fornecer uma infra-estrutura básica de simulação baseada em evento discreto. Para isso o pacote de simulação SimJava serve como base para o GridSim, visto que trata-se de uma implementação muito difundida de infra-estruturas de simulação baseada em eventos discretos disponível em Java.

A terceira camada está relacionada ao núcleo do GridSim, ou seja, a modelagem e simulação das entidades básicas da grade, como recursos e serviços de informação. Essa camada oferece um arcabouço para a criação de entidades de nível superior.

A quarta camada se preocupa com a simulação de agregadores de recursos chamados de resource *brokers* ou escalonadores. A terceira camada oferece uma infra-estrutura que permite a implementação dos algoritmos de escalonamento na quarta camada.

A camada final é dedicada à aplicação e à modelagem de recursos com diferentes cenários utilizando os serviços disponíveis pelas duas camadas de nível inferior, ou seja, é nessa camada que se define as configurações do ambiente de simulação da grade, como requerimento do usuário (quantidade de tarefas, tamanho das tarefas, etc), configurações dos recursos (quantidade de recursos, quantidades de máquinas para cada recurso, quantidades de processadores em cada máquina, dentre outros), topologia de rede, entre outras características.

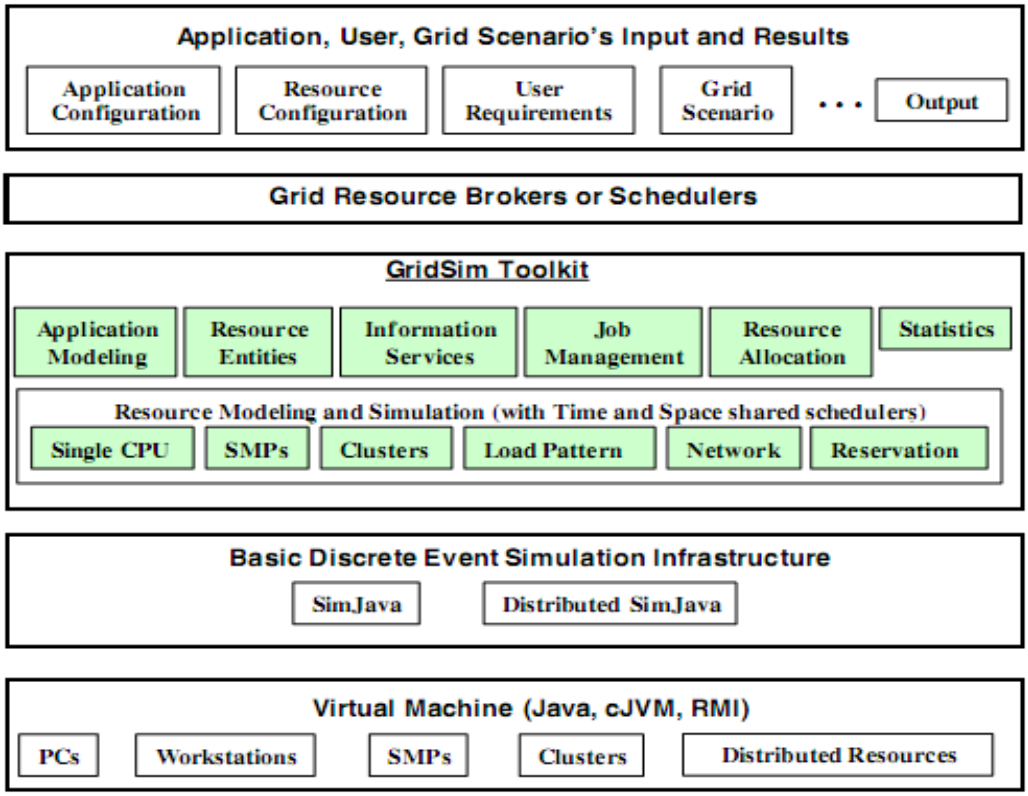


Figura 7 - Arquitetura do GridSim (11)

Na versão 4.1, devido à extensão para a simulação de grades de dados, o GridSim teve sua arquitetura inicial modificada (42), conforme pode ser observado na Figura 8.

As camadas mais inferiores (primeira e segunda) na antiga arquitetura foram unidas, dando origem a uma nova camada denominada SimJava *Simulation Kernel* (Núcleo de Simulação SimJava).

A antiga terceira camada foi desmembrada dando origem a novas camadas: camada *Core Elements* (Elementos do Núcleo), camada de *Grid* Computational (Grade de Computação) e camada de *Data Grid* (Grade de Dados).

A camada de Elementos de Núcleo ficou responsável pela modelagem dos elementos de infra-estrutura como recursos, enlaces de rede, entre outros. As camadas de Grade de Computação e Grade de Dados ficaram responsáveis pela modelagem e simulação de grades de computação e grades de dados respectivamente, entretanto, serviço de informação e gerenciamento de tarefas ficaram comuns em ambas as camadas. Não houve modificação nas duas camadas mais superiores.

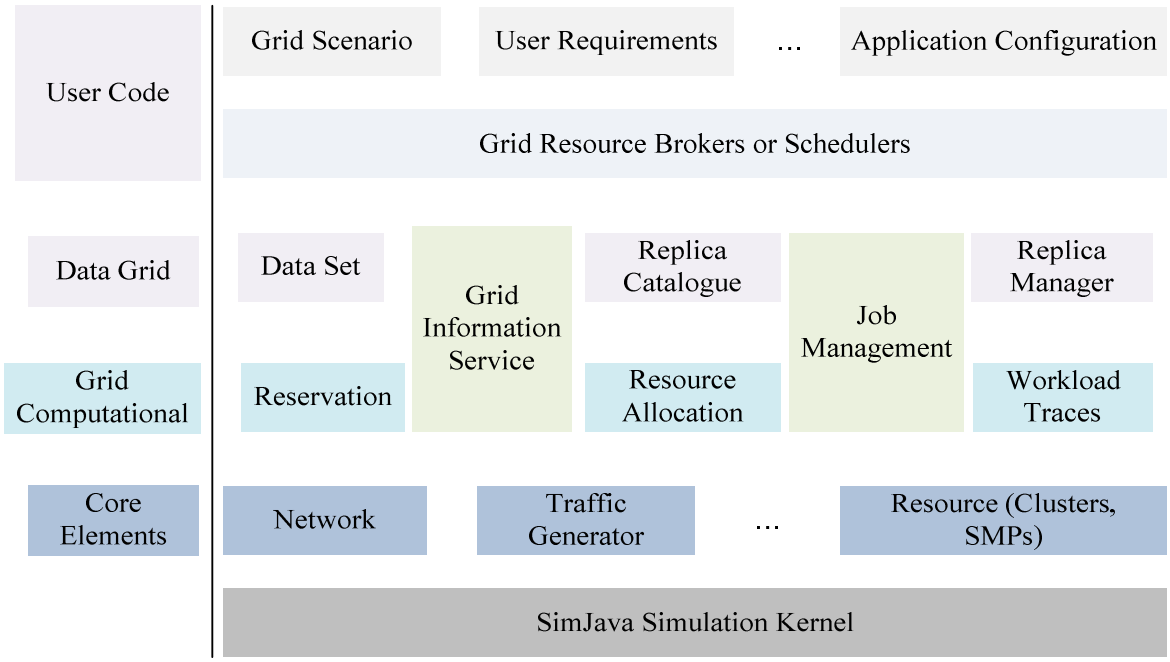


Figura 8 - Nova Arquitetura do GridSim (42)

SimJava

O SimJava é um pacote de simulação baseado em eventos discretos de propósito geral implementado em Java. Simulações em SimJava contêm um número de entidades, cada uma sendo executadas em paralelo na sua própria thread. As entidades têm seus comportamentos descritos em Java usando o método body(). Entidades são conectadas utilizando portas, o que permite o envio e o recebimento de eventos. Todos os eventos criados são incluídos na fila *Future* para serem processados pelo método process_event() (11).

As entidades têm acesso a um pequeno número de primitivas de simulação:

sim_schedule() envia evento a outras entidades, através das portas;
sim_hold() adia o tempo de simulação;
sim_wait_for() espera que um determinado evento ocorra;
sim_wait() espera que um evento ocorra.

Entidades do GridSim

Durante a simulação o GridSim cria uma série de entidades *multi-threaded*, cada uma executando em paralelo em seu próprio segmento. O comportamento de uma entidade deve ser descrito no método body(). As entidades básicas definidas no GridSim são (11):

Usuário: Cada instância da entidade *User* representa um usuário na grade, sendo que cada usuário pode ser diferente dos demais usuários com relação às seguintes características:

- tipo das tarefas criadas, por exemplo, tempo de execução da tarefa, número máximo de replicas, etc;
- estratégia de otimização de escalonamento, por exemplo, minimização de custos, do tempo, ou ambos;
- taxa de atividade, por exemplo, quantas vezes ele cria uma nova tarefa;
- fuso horário

Broker: Cada usuário está conectado a uma instância da entidade *broker*. Toda tarefa de um usuário é submetida ao seu *broker* e, em seguida, o *broker* escalona a tarefa de acordo com a política de escalonamento adotada. Antes de escalonar as tarefas, o *broker* recebe dinamicamente a lista de recursos disponíveis para tentar otimizar a política de escalonamento do usuário.

Recurso: Cada instância da entidade *Resource* representa um recurso da grade, sendo que cada recurso pode ser diferente dos demais recursos no que diz respeito às seguintes características:

- número de processadores;
- custos de processamento;
- velocidade de processamento;
- política interna de escalonamento de processos, por exemplo, tempo compartilhado ou espaço compartilhado;
- fator de carga local e fuso horário.

Serviço de informação da grade: Fornecer serviço de registro de recursos, mantendo uma lista dos recursos disponíveis na grade.

Entrada/Saída: O fluxo de informações entre as entidades do GridSim acontece através das entidades de *Input* e *Output*. Cada entidade do GridSim tem canais I/O ou portas, que são utilizadas para estabelecer um elo entre a entidade e sua própria entidades de entrada e saída. O uso de entidades separadas para entrada e saída permite que as demais entidades modelem canais de comunicação *full-duplex* e comunicação paralela entre vários usuários.

2.6 Considerações finais

Neste capítulo 2 foram apresentados alguns ambientes de simulação e suas características com destaque para o GridSim.

Os simuladores desenvolvidos em Java apresentam portabilidade de plataforma de execução proporcionada pelo uso da máquina virtual JVM (*Java Virtual Machine*), esses simuladores tem como mecanismo de simulação o uso de múltiplas *threads*, o que possibilita a execução de diversas entidades de forma simultânea, porém o uso de múltiplas threads acaba limitando o número de entidades possíveis no ambiente.

Na Tabela 1 há uma comparação entre os simuladores citados.

Tabela 1 - Comparação entre simuladores de Grade

Características	SimGrid	OptorSim	MicroGrid	GridSim
Tipo	Simulador	Simulador	Emulador	Simulador
Linguagem de programação	C	Java	C	Java
Portabilidade	Fraca	Forte	Fraca	Forte
Escalonamento de tarefas	Sim	Não	Sim	Sim
Mecanismo de simulação	Serial	Multithread	Paralelo	Multithread
Geração de tráfego de fundo	Sim	Sim	Sim	Sim

3 DESENVOLVIMENTO DO PACOTE DE REDE DE *OVERLAY*

O trabalho desenvolvido foi baseado na construção de um Java *package* com classes e interfaces que representem os conceitos básicos de redes de *overlay* no GridSim. Após análise das características e dos recursos fornecidos por esse ambiente de simulação, consideramos interessante o desenvolvimento do pacote de redes *overlay* voltado para a infra-estrutura de grade de dados do GridSim.

O GridSim permite a simulação de grades de dados de forma robusta e bastante flexível, o que possibilita a realização de estudos sobre estratégias de replicação. Algumas funcionalidades disponíveis são: replicação de dados em vários recursos, consulta para a localização dos dados replicados, acesso aos dados replicados, além de permitir consultas complexas sobre os atributos dos dados (42).

Na Tabela 2 há uma comparação entre alguns dos principais simuladores existentes atualmente levando em consideração algumas funcionalidades básicas para simulação de grades de dados. É possível observar que os ambientes de simulação MicroGrid e SimGrid não têm como foco a simulação de grades de dados.

O OptorSim foi desenvolvido como parte do projeto de EU *DataGrid* e visa estudar a eficácia das estratégias de replicação de dados, entretanto, não possui algumas funcionalidades desejáveis para simulação em grade de dados.

O Monarc tem suporte para replicação de dados e permite a consulta de dados, entretanto não possui um gerador de tráfego de rede em *background* o que não permite que as simulações possam levar em conta a concorrência dos canais de comunicação pelos recursos e usuários.

Tabela 2 - Comparação entre Simuladores adaptado de (42)

Funcionalidades	GridSim	OptorSim	MicroGrid	SimGrid	Monarc
Replicação de dados	Sim	Sim	Não	Não	Sim
Overhead de E/S	Sim	Não	Sim	Não	Sim
Consulta de dados	Sim	Não	Não	Não	Não
Reserva de CPU de um recurso	Sim	Não	Não	Não	Não
Traços de execução baseado em cargas de trabalho reais	Sim	Não	Não	Não	Não

O GridSim possui algumas entidades básicas para grades de dados, sendo que as principais são:

- *Dataset*: representa uma coleção de dados que pode ser acessado como uma unidade.
- *Replica Catalogue*: provê informações sobre a localização de um *Dataset* e de suas réplicas em uma grade de dados
- *Data Grid Resource*: extensão da entidade *Grid Resource* é responsável pelo armazenamento dos *Datasets* e pela execução dos *jobs* submetidos pelos usuários da grade.
- *Data Grid User*: extensão da entidade *Grid User* é responsável pela submissão de *jobs*, registro de eventos, busca e transferência de *Datasets* nos *Data Grid Resource*.

Para a grade de dados, o GridSim faz uso de um catálogo de réplica (*Replica Catalogue*) que armazena uma lista dos dados disponíveis em um *Data Grid Resource* ou *Data Centers*. Somente os atributos dos dados são mantidos nos catálogos de réplica. Quando um usuário necessita de um determinado dado, ele consulta o catálogo de réplica que fornece informações sobre a localização física do dado e suas réplicas na grade, ou seja, provê um mapeamento entre o nome de um dado e sua localização física.

Uma grade de dados no GridSim pode ter um ou vários catálogos de réplica, geralmente utilizando diferentes topologias de rede, tais como modelos centralizados (só existe um catálogo de réplica) e hierárquicos (existem vários catálogos de réplica organizados em uma estruturada de árvore), para melhorar a escalabilidade e a tolerância a falhas (42).

Os catálogos de réplica podem ser configurados em vários níveis, como LocalRC (a nível de recurso) ou RegionalRC (a nível de grade), ambos devem ser conectados a um nível superior, o TopRegionalRC, que atua como um servidor de catálogo de réplica, conforme ilustrado na Figura 9.

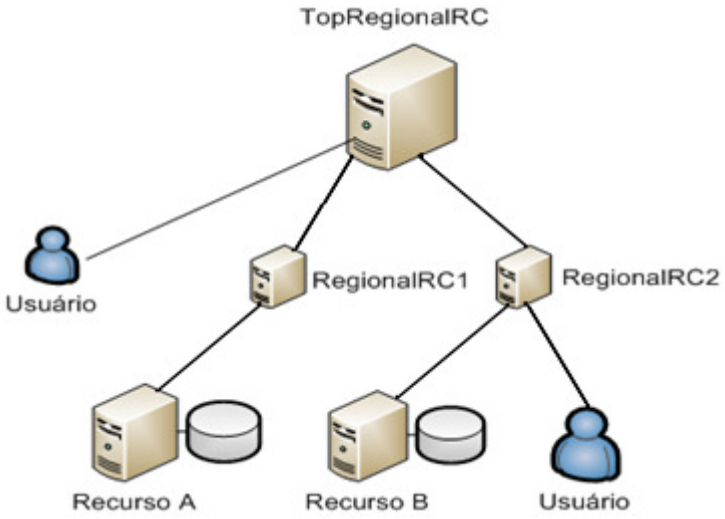


Figura 9 - Modelo de organização de Catálogo de Replica hierárquico

Como o objetivo deste trabalho é fornecer condições e mecanismos que possibilitem a construção de sistemas *peer-to-peer* para serem testados na infra-estrutura de grade de computadores, propomos a substituição do uso dos catálogos de réplicas na estrutura do GridSim por uma estrutura de redes de *overlay*, ou seja, uma alternativa aos modelos de topologia especificados no GridSim para a organização de uma grades de dados, conforme ilustrado na Figura 10.

Com um amplo uso de herança e polimorfismo foi possível gerar um pacote básico de redes de *overlay* o qual permite que a *Data Grid Resource* (recurso) e a *Data Grid User* (usuário) sejam conectados diretamente, sem a necessidade do uso de uma entidade de catálogo de réplica.

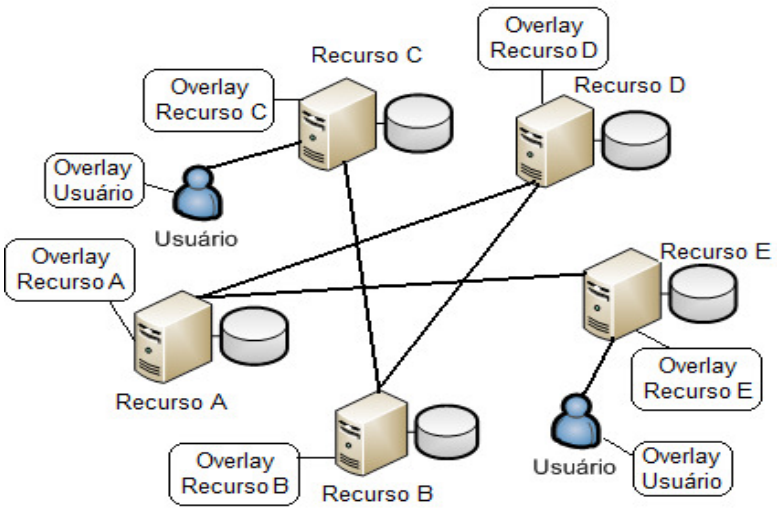


Figura 10 - Modelo utilizando redes de *overlay*

Uma questão importante é que o GridSim não permite que sejam criados os denominados *servents*, um acrônimo para "*server*" e "*client*", visto que, ele diferencia com funcionalidades distintas as entidades *Data Grid User* e *Data Grid Resource.*

3.1 Entidades do pacote de rede de *overlay*

A estrutura do pacote desenvolvido permite facilitar o trabalho e a implementação de diferentes propostas de topologias de redes, protocolos e algoritmos. O pacote de redes *overlay* é constituído de um conjunto de classes que provê meios para o desenvolvimento desses algoritmos.

Algumas entidades básicas definidas no GridSim para grades de dados como *Data Grid User* e *Data Grid Resource* tiveram que ser redefinidas em novas entidades com novas funcionalidades para dar suporte ao pacote de redes de *overlay* desenvolvido. É importante ressaltar que o código fonte original do GridSim não sofreu nenhuma alteração.

As entidades básicas do pacote de redes de *overlay* são:

- *Data Grid User Peer*: extensão da entidade *Data Grid User,* é responsável pela requisição de busca, geração de dados, entre outras funcionalidades;
- *Data Grid Resource Peer*: extensão da entidade *Data Grid Resource,* é responsável pelo armazenamento e pelo processamento dos dados;
- *Overlay Statistics*: fornece dados estatísticos do resultado de uma simulação;
- *Overlay*: principal entidade do pacote, fornece funcionalidade e um nível de abstração para implementação de protocolos *peer-to-peer*.

As funcionalidades dessas entidades foram especificadas em classes, sendo que as classes que compõem o pacote de rede de *overlay* podem ser observadas (utilizando a modelagem orientada a objetos UML) na

Figura 11.

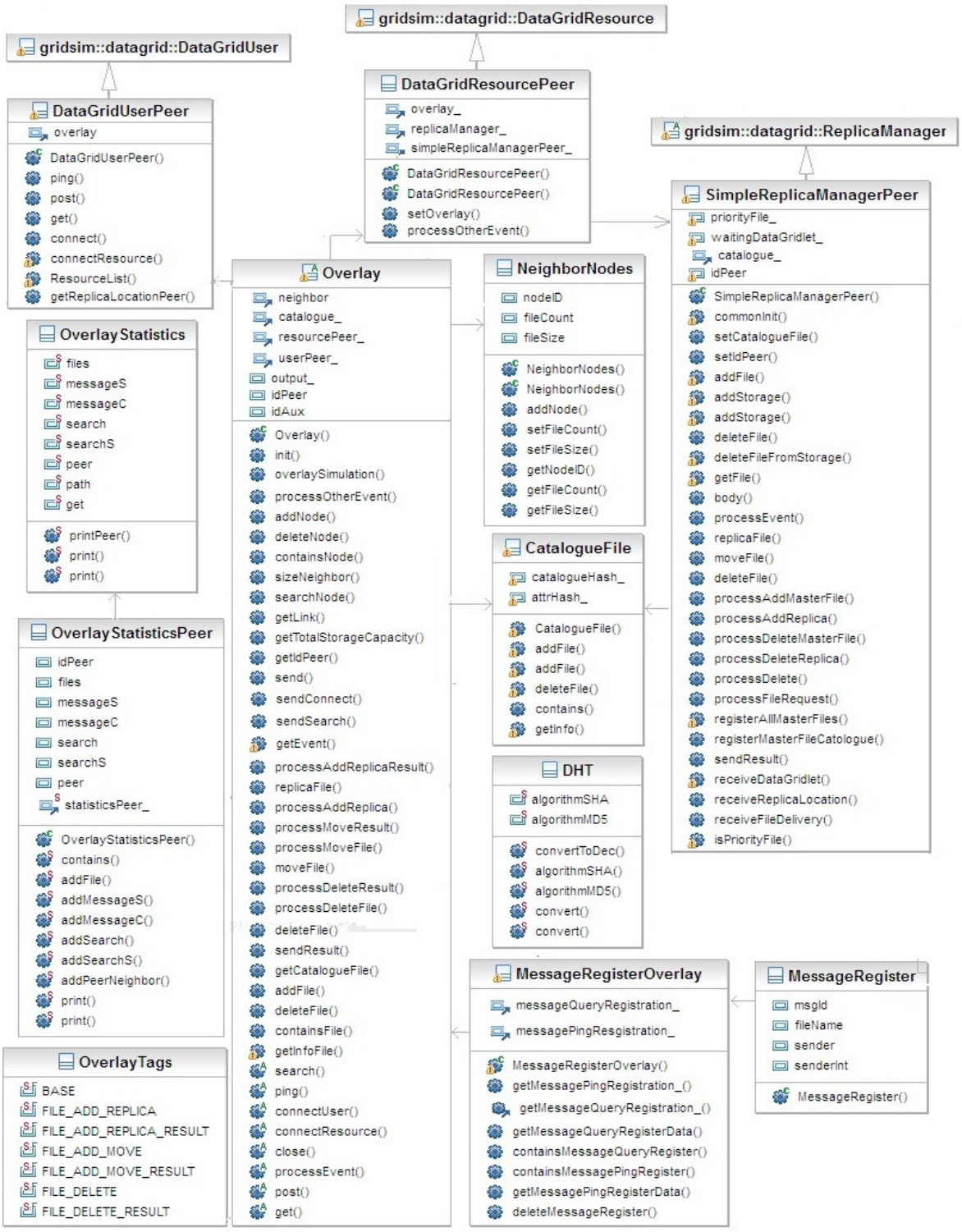


Figura 11 - Diagrama de classe do Pacote de Rede de *Overlay*

3.2 Usuário Peer – Entidade Data Grid User Peer

No GridSim um usuário (*Data Grid User*) pode ser uma aplicação ou um *broker* (escalonador). É um componente capaz de efetuar consultas, de remover e de transferir um conjunto de dados entre os recursos (*Data Centers*). A entidade *Data Grid User* é implementada através da criação de um objeto da *class DataGridUser*. Essa entidade foi redefinida em um nova entidade *Data Grid User Peer* (*class DataGridUserPeer)* para que seja possível a consultas de dados utilizando a estrutura *overlay* desenvolvida, e não mais os catálogos de réplicas. Portanto, a entidade *Data Grid User Peer* além de contar com todas as funcionalidades da entidade *Data Grid User*, conta também com novas funcionalidades que têm como objetivo permitir o acesso aos mecanismos de um protocolo de rede de *overlay*, como: conexão, desconexão, busca de novos *peers* vizinhos e busca de dados.

3.3 Recurso Peer – Entidade Data Grid Resource Peer

Na computação em grade, qualquer componente de *hardware* como um *cluster*, um supercomputador ou um repositório de armazenamento de dados é chamado de recurso.

O recurso (*Data Grid Resource*) pode ser associado com um ou vários repositórios de armazenamento (*Storage*), sendo que a manipulação dos dados nos repositórios, como inclusão, remoção e transferência de um dado é feita pelo gerenciador de réplicas (*Replica Manager*).

O gerenciador de réplicas (*class SimpleReplicaManager*) teve que ser redefinido em uma nova classe (*class SimpleReplicaManagerPeer*) para dar suporte ao pacote de redes *overlay* e não mais os catálogos de réplicas (*Local Replica Catalogue*). Pelo mesmo motivo, a classe principal da entidade *Data Grid Resource* (*class DataGridResource*) também sofreu alterações e foi redefinida em uma nova entidade *Data Grid Resource Peer* (*class DataGridResourcePeer*).

A Figura 12 ilustra a entidade *Data Grid Resource* original e a nova entidade *Data Grid Resource Peer*, sendo que é possível notar que na nova entidade não existe mais

o catálogo de réplicas, dessa forma, todas as informações referentes aos dados existentes em uma *Data Grid Resource Peer* agora são armazenados no *Catalogue File* da entidade *Overlay* do pacote desenvolvido.

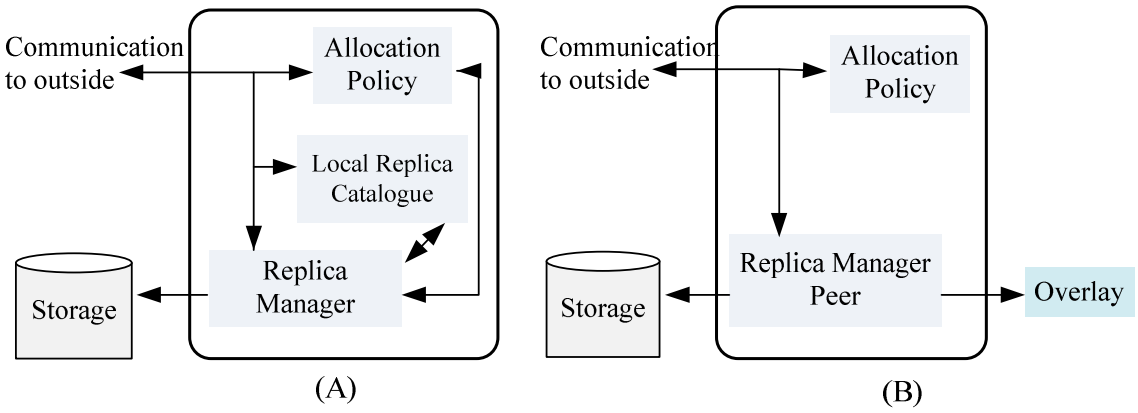


Figura 12 - (A) Data Grid Resource (B) Data Grid Resource Peer

3.4 Entidade *Overlay Statistics*

A entidade *Overlay Statistics* é responsável por registrar dados estatísticos que ocorrem durante a simulação e ao final grava os mesmos em um arquivo no formato *txt*, permitindo consultar as estatísticas gravadas para a geração de relatório.

Os dados estatísticos contidos no arquivo são: número de arquivos gerados, número total de buscas, número de buscas efetuadas com sucesso, caminho médio das buscas, número total de mensagens de conexão e desconexão de *peers*.

No arquivo consta também dados estatísticos separados por *peer,* como: número de mensagens de busca recebidas, número de requisições atendidas, números de mensagem de conexão e desconexão.

3.5 Entidade *Overlay*

A entidade *Overlay* é a principal do pacote desenvolvido, sendo que, suas

características foram definidas em diversas classes. A principal delas, a *class Overlay* é uma classe abstrata e uma extensão da *class* Sim_entity (do pacote SimJava), ou seja, a estrutura *Overlay* é uma entidade no GridSim, possui assim, sua própria *thread* de execução.

As redes de *overlay* estruturadas e não estruturadas, possuem funcionalidades básicas em comum como: busca, conexão, desconexão e atualização da rede. Estas funcionalidades foram especificadas em métodos abstratos definidos na *class Overlay*, como *search()*, *connectUser(), connectResource()*, *processEvent()*, *ping(), post(), get() ,close()*, o que permite que o desenvolvedor dos algoritmos de rede de *overlay* possa implementá-los da forma mais adequada para o seu sistema.

Na *class Overlay* foram definidas outras funcionalidades para:

- Comunicação: métodos que permitem o envio de mensagem, recebimento de mensagem com *timeout*, fornecem informação referente à largura de banda, identificador (algo semelhante ao IP), portas de comunicação de E/S e entre outros.
- Gerenciamento de Recurso e Usuário: métodos para adicionar, remover, transferir e replicar dados de recurso para outro, métodos para obter informação sobre o número de arquivos armazenados, capacidade total de armazenamento, capacidade de processamento, número de processadores e entre outros.

As demais funcionalidades que compõem a entidade *overlay* foram especificadas em outras classes, na qual iremos definir como estruturas auxiliares:

- Estrutura *Neighbor Node*: responsável pelo gerenciamento de *peers* vizinhos.
- Estrutura *Catalogue File*: responsável pelo registro de informações dos *Datasets* (dados) armazenados em um *Data Grid Resource Peer.*
- Estrutura *Message Register*: responsável pelo registro e pelo gerenciamento das mensagens de conexão ou de busca que passaram pelos *peers*.

- Estrutura DHT: estrutura com funções de *Hash* amplamente utilizadas em protocolos *peer-to-peer* estruturados como MD5 e SHA-1.

A Figura 13 ilustra a relação entre essas estruturas (classes no pacote *overlay*). É possível observar que a entidade *Overlay* sempre estará associada à entidade *Data Grid Resource Peer* ou a *Data Grid User Peer* e que a comunicação externa é feita por essas entidades, ou seja, para comunicar com outras entidades a entidade *overlay* utiliza as portas de entrada e de saída das entidades *Data Grid Resource Peer* ou do *Data Grid User Peer*.

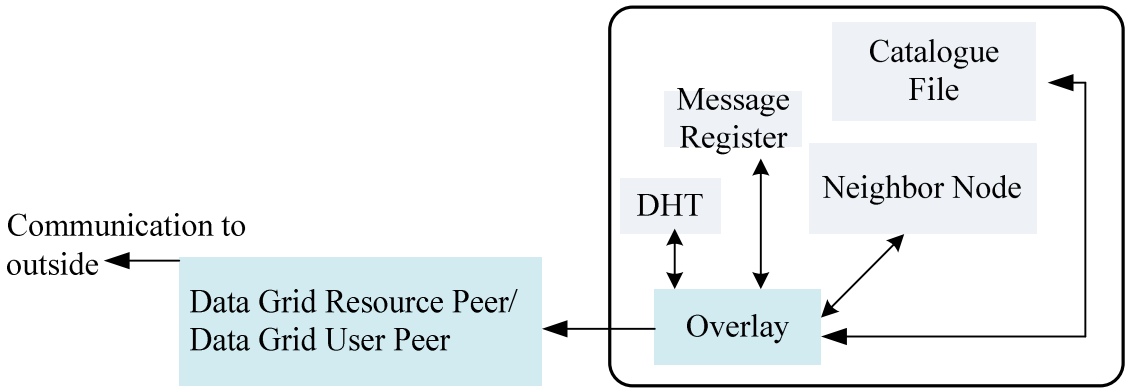


Figura 13 - Entidade *Overlay*

Operações de manipulação dos dados

Em muitos casos em redes de *overlay* estruturadas, para manter a distribuição das chaves de forma equilibrada faz-se necessário o uso de operações como remoção, replicação e transferência de um conjunto de dados entre os recursos. Essa operação se torna necessária principalmente na entrada e na saída de um *peer* da rede. No GridSim a transferência, a replicação e a remoção de um conjunto de dados (*Datasets*) só é possível com a intermediação de uma entidade usuário (*Data Grid User*), uma vez que as entidades de recursos *(Data Grid Resource)* do GridSim não foram implementadas para efetuar essas operações diretamente entre elas.

A Figura 14 ilustra essa situação. Neste caso, não é possível replicar o arquivo do Recurso B para o Recurso A de forma direta, para isso, o Usuário A requisita uma cópia do arquivo para o Recurso B e em seguida envia a cópia do arquivo para o Recurso A.

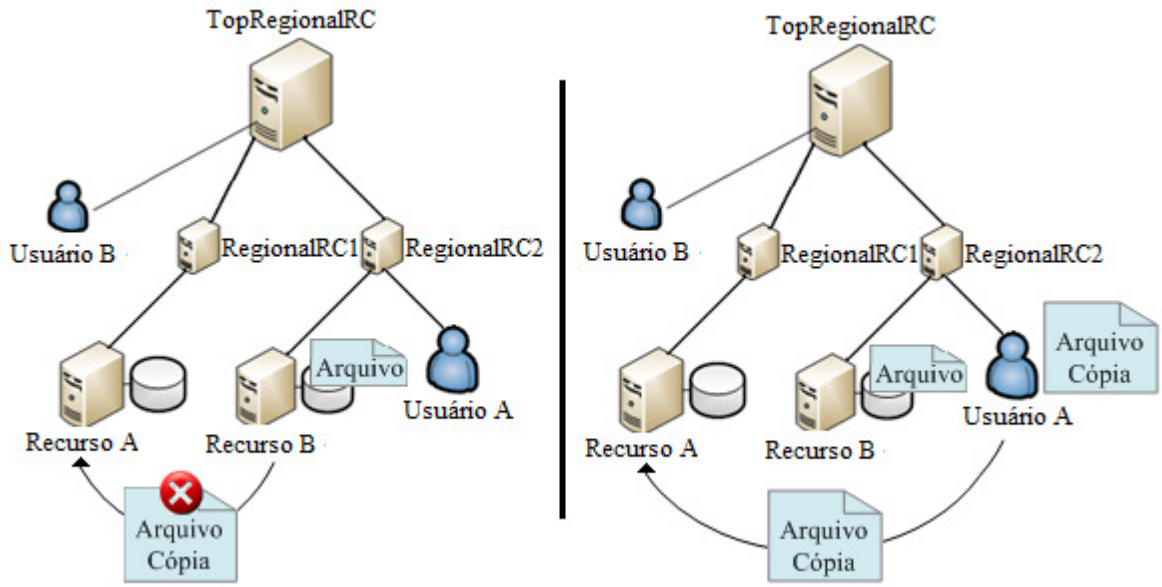


Figura 14 - Exemplo de uma replicação de dados no GridSim Original

Para permitir essas operações de maneira mais fácil e flexível e que as mesmas ocorram sem a necessidade de uma entidade usuário (*Data Grid User*), foram especificados na entidade *Overlay* essas funcionalidades (transferência, replicação e remoção de dados). Dessa forma, uma entidade Overlay do Recurso A pode solicitar a replicação de um arquivo para a entidade *Overlay* do Recurso B diretamente, como ilustrado na Figura 15.

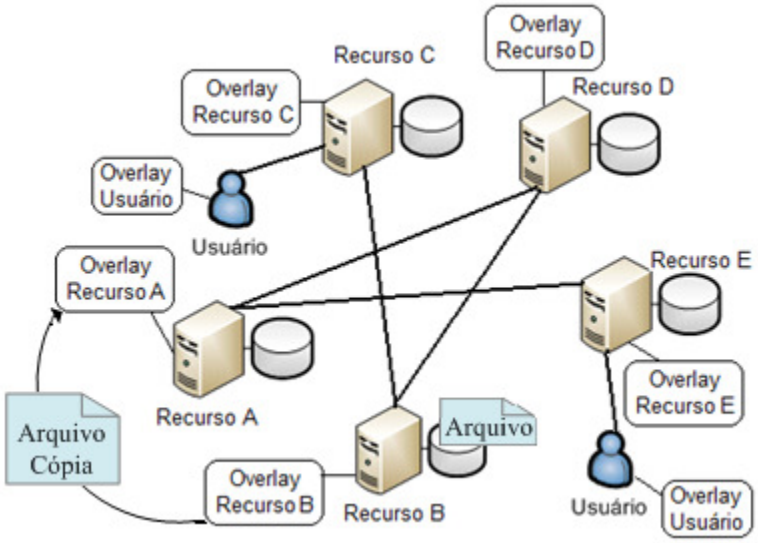


Figura 15 - Exemplo de uma replicação de dados com o pacote de overlay proposto

Conexão e desconexão dos *peers*

Na estrutura *Neighbor Node* (*class NeighborNode*) foram especificadas funcionalidades como adição, listagem e a remoção de *peer*, que visam garantir a conectividade de todos os *peer*, ou seja, permitir que um *peer* consiga, direta ou indiretamente, comunicar-se com qualquer outro *peer*. Os *peer* vizinhos são armazenados em uma lista, sendo armazenadas informações como identificador (algo semelhante ao IP) do *Data Grid Resource Peer* ou *Data Grid User Peer,* quantidades de dados armazenados, entre outras informações.

Informação sobre os dados

Como não há o uso dos catálogos de réplica, entidade responsável pelo armazenamento das informações sobre a localização física de um conjunto de dados e de suas réplicas, foi necessária a criação de uma nova entidade responsável pelo armazenamento das informações.

A estrutura *Catalogue File (class CatalogueFile)* é responsável pelo armazenamento das informação referentes a um dado ou a uma réplica em uma tabela *hash*. Nesta estrutura são disponibilizados funcionalidades para inserção, remoção e a busca de um determinado dado em um recurso. Sempre que uma entidade Overlay é instanciada, automaticamente uma entidade *Catalogue File* é instanciada e associada à entidade *Overlay*. Quando um dado ou uma réplica é armazenado em um *Data Grid Resource Peer,* os seus atributos (nome, tamanho) do mesmo são armazenados no *Catalogue File.*

Registro de mensagens

Em sistemas de rede de *overlay* não estruturados, uma mensagem de busca (*query)* pode passar várias vezes pelo mesmo *peer*. Caso não haja um controle desta mensagem, o *peer* irá verificar se tem o dado requisitado toda vez em que receber a

requisição. Para evitar essa situação, o registro e o gerenciamento das mensagens que cada *peer* recebe são de significativa importância para obter um desempenho aceitável.

Na estrutura *Message Register (class MessageRegisterOverlay)* foram especificados métodos para adicionar, remover, buscar e gerenciar as mensagens de conexão ou busca que passaram pelos *peers* durante a simulação. Uma mensagem com o identificador gerado de forma aleatória é armazenada em uma lista em todos os *peers* pela qual se propagou. No caso das mensagens de busca, informações contidas na mensagem, como nome do arquivo e remetente da mensagem, podem ser registradas na lista

Recursos DHT

As redes estruturadas, como já apresentado, possuem uma topologia de rede controlada e utilizam regras para distribuir os dados na rede de modo que facilite a posterior localização dos mesmos de forma estruturada e eficiente.

Muitos protocolos altamente estruturados (como exemplos: Chord, CAN, Pastry, entre outros) são baseados na noção de uma tabela de indexação distribuída DHT (*Distributed Hash Table*), na implementação de serviços como de busca de dados ou conexão e desconexão de *peers*, ou seja, podemos considerar esses protocolos em termos abstratos como uma DHT com suporte para operações de inserção, remoção e busca de chaves, sendo que essas chaves são obtidas utilizando uma função segura de *hash*.

O uso de funções *Hash* é bastante difundido no contexto das aplicações computacionais, por se tratar de um mecanismo de segurança e por possuir propriedades como resistência a colisões e unidirecionalidade (29). As funções mais usadas atualmente em protocolos *peer-to-peer* estruturados são a *Message Digest*-5 ou MD5 e o *Standart Hash Algorithm* ou SHA-1 (29).

Na estrutura DHT *(class DHT)* do pacote de *rede de overlay* foi especificado as funções *Hash* MD5 e SHA-1, sendo que, fica a critério do usuário do pacote de *rede de overlay* qual função utilizar no desenvolvimento de seu protocolo. Em ambas as funções é passada uma informação (nome do arquivo ou identificador do *peer*) de um tamanho arbitrário e retornado um padrão de bits inteiro de comprimento fixo que caracteriza aquela informação.

3.6 Considerações finais do desenvolvimento do pacote de rede de *overlay*

O pacote desenvolvido fornece estruturas básicas como: estrutura de comunicação, de gerenciamento de informações sobre os recursos e os usuários da grade, de operações de manipulação de dados (remoção, replicação e transferência de um conjunto de dados entre os recursos), entre outras. Portanto, temos com isso o objetivo de permitir e facilitar o desenvolvimento de protocolos de rede de overlay no GridSim.

Além disso, desenvolvemos meios para criar uma nova alternativa aos modelos de topologia para a organização de uma grade de dados, sem que o GridSim sofresse nenhuma alteração em seu código original ou que perdesse suas funcionalidades.

4 IMPLEMENTAÇÃO DOS PROTOCOLOS NO SIMULADOR

Neste trabalho foi desenvolvido um conjunto de classes que permite a implementação de protocolos e de algoritmos de redes de *overlay* (redes estruturadas e não estruturadas).

O pacote de rede de *overlay* não implementa nenhum protocolo específico, somente disponibiliza recursos para que os mesmos sejam implementados. A *class Overlay* da entidade Overlay disponibiliza métodos abstratos, ou seja, métodos que foram declarados, mas não implementados (nenhum código funcional) para conexão, desconexão, busca de dados, busca de novos *peers,* entre outros. Isso permite que esses métodos sejam implementados na classe que estenda a *class Overlay.*

Além disso, como já apresentado no Capítulo 3, são disponibilizados recursos para comunicação dos *peers*, como criação, envio e o recebimento de mensagem, entre outros.

Para testar as funcionalidades providas pelo pacote de rede de *overlay* desenvolvido, foi implementado um protocolo com base em uma versão simplificada do protocolo Gnutella 0.4 (redes não estruturadas) e outro protocolo baseado em uma versão simplificada do protocolo Chord (redes estruturadas). Temos com isso, o objetivo de certificar se as funcionalidades providas pelo pacote permitem a construção de protocolo de redes de *overlay* no GridSim. Em ambos os protocolos todo o gerenciamento dos *peers* e comunicação entre os mesmos é fornecido pelo pacote *overlay.*

Utilizamos a política de replicação de dados passiva, A replicação passiva ocorre naturalmente quando um *peer* requer a cópia de algum dado a outro *peer*, portanto, ambos tornam-se provedores do dado. A política de replicação de dados é um ponto chave na colaboração de conjuntos de dados, pois permite diminuir a latência na transferência destes conjuntos.

4.1 Implementação do protocolo de redes não estruturadas no simulador

Os protocolos de redes de *overlay* não estruturados apresentam uma topologia mais livre e flexível determinada apenas pelo processo dinâmico de conexão dos participantes na rede. As regras para a conexão dos nós são mais flexíveis e são intuitivamente mais fáceis de serem mantidas. Sistemas como Gnutella e Kazaa utilizam essa arquitetura.

Neste trabalho optamos por implementar uma versão de protocolo de rede não estruturada baseado em uma versão simplificada do protocolo Gnutella 0.4. O protocolo é composto basicamente por cinco tipos de mensagens, como descrito na Seção 1.4.2, sendo que destas, foram implementadas as seguintes:

- *Ping*: mensagem usada para descobrir outros *peers* na rede.
- *Pong*: resposta à mensagem *ping*.
- *Query*: mensagem utilizada para realização da uma consulta na rede.
- *Query hit*: é a mensagem de resposta a uma *query*.

Para criar um ambiente Gnutella, utilizando o pacote de redes *overlay* desenvolvido, foi feita uma extensão da *class Overlay*.

```
public class Gnutella extends Overlay
```

A Figura 16 ilustra a estrutura básica do protocolo Gnutella, os métodos abstratos fornecidos pela *class Overlay* para conexão, busca de dados e busca de novos *peers* foram implementados seguindo as especificações do protocolo Gnutella, sendo que outros métodos foram implementados para dar suporte a esses métodos.

```
public class Gnutella extends Overlay {

  public Gnutella() throws Exception{}

  //método é responsável por qerenciar todos os eventos de entrada
  public void body() {}

  //Processa os eventos de entrada
  public boolean processEvent(Sim_event ev) {}

  //Métodos para manutenção
  public void pingUser(){}
  public void pingResource(){}
  public void pong(){}

  //Métodos para busca
  public int search  (String lfn){}
  public void queryHit(Sim_event ev) {}
  public void query(Sim_event ev) {}

  //Métodos para Conexão e Desconexão
  public void connectResource(){}
  public void connectUser(){}
  public void close(){}
}
```

Figura 16 - Estrutura Básica do Protocolo Gnutella

Foi implementada a classe *class GnutellaMessageTag* que define constantes utilizadas para identificar as mensagem.

Foi utilizada a estrutura *Message Register* do pacote de rede de overlay com o objetivo de evitar a recepção repetitiva de mensagens de conexões diferentes, com isso cada *peer* consulta a estrutura *Message Register* que contém informações das mensagens recebidas. Caso confirmada a presença, descarta-se a mensagem.

Presente em todos os *peers*, a estrutura *Neighbor Node* foi utilizada para armazenar informações dos *peers* vizinhos de um *peer*, ou seja, essa estrutura armazena e gerencia em uma lista dos *peers* as conexões virtuais estabelecidas de um *peer* X.

A Estrutura *Catalogue File* foi utilizada para armazenar informações de dados contidos nos *peers*, uma vez que não há mais o uso dos catálogos de replica.

As classes implementadas podem ser observadas (utilizando a modelagem orientada a objetos UML) na Figura 17.

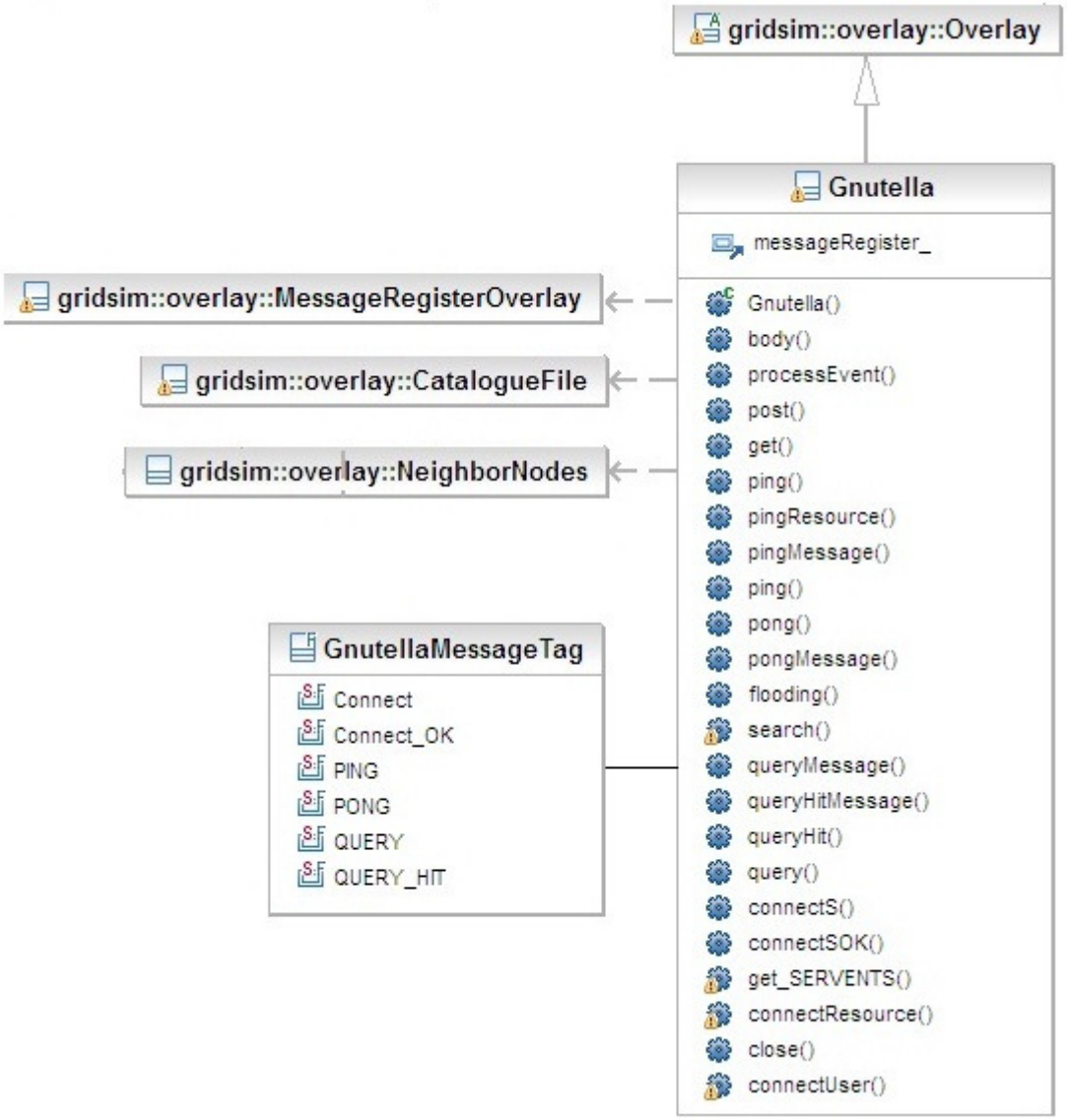


Figura 17 - Diagrama de classes Gnutella

Conexão dos Peers

A entrada dos *peers* na rede foi implementada para ocorrer de modo dinâmico e em tempo de simulação, porém, as ligações virtuais lógicas de cada *peer* foram definidas de modo estático utilizando um gerador de redes complexas (43).

Para criar uma topologia de rede lógica utilizamos o modelo de rede Barabási-Albert (44) que apresenta propriedades e características semelhantes às redes encontradas na maioria das redes de overlay como Gnutella.

Mecanismo de Busca

A Figura 18 ilustra o funcionamento do mecanismo de busca implementado. Inicialmente o *peer*, fonte da busca, cria um pacote de requisição de dados (uma mensagem Query) e encaminha cópias destinadas a seus respectivos vizinhos (recurso 1). Estes, por sua vez, se não possuírem o dado requisitado, repassam-no a outros (recurso 2), e assim sucessivamente. Quando o dado é encontrado gera um pacote com o identificador do recurso (mensagem QueryHit).

Todos os dados são armazenados na Estrutura *Catalogue File*, visto que não há mais o uso dos catálogos de réplica.

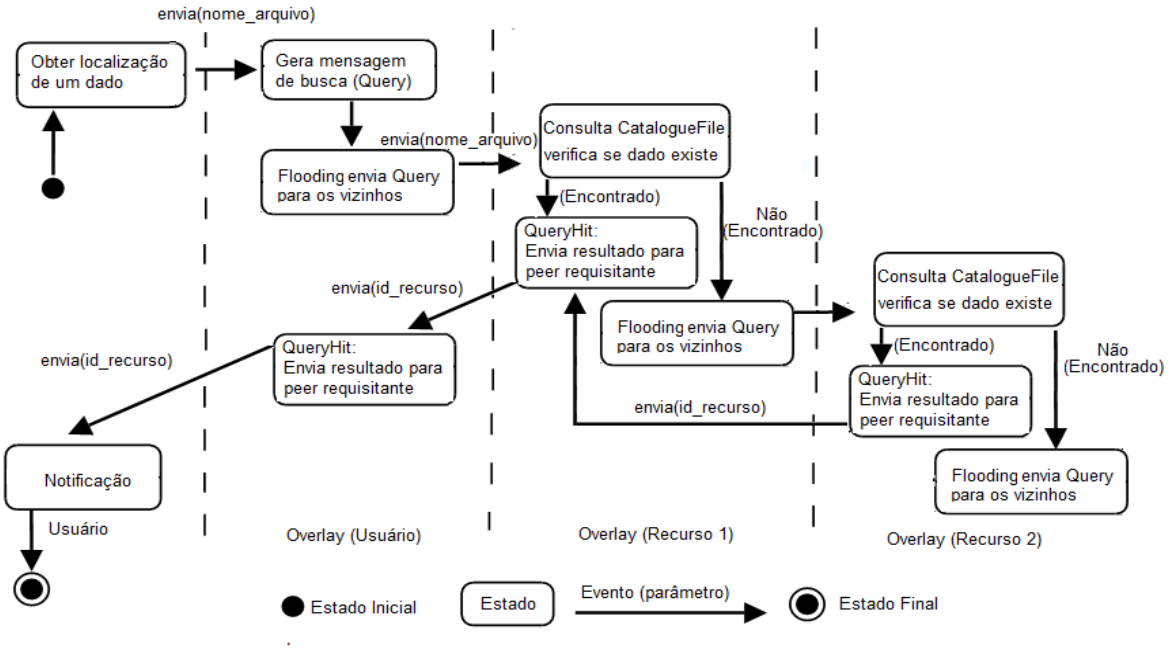


Figura 18 - Diagrama de estado referente à busca utilizando redes de *overlay*

Para se ter uma comparação da estrutura proposta, a Figura 19 ilustra a localização de um dado utilizando catálogos de réplica. É possível observar que inicialmente o usuário faz uma requisição para o catálogo de réplica (RegionalRC1) na qual está associado. Caso o dado esteja registrado nesse catálogo é enviada uma mensagem de resposta com o identificador do recurso, caso contrário é enviada uma requisição para o nível superior (TopRegionalRC), estes, por sua vez, verifica em quais catálogos de réplica o dado se encontra e envia uma lista com os mesmo para o catálogo de réplica

(RegionalRC1) que fez a requisição. Este seleciona um catálogo da lista (RegionalRC2) e envia uma requisição, ao receber a requisição esse catálogo obtém a identificação do recurso e envia diretamente para o usuário.

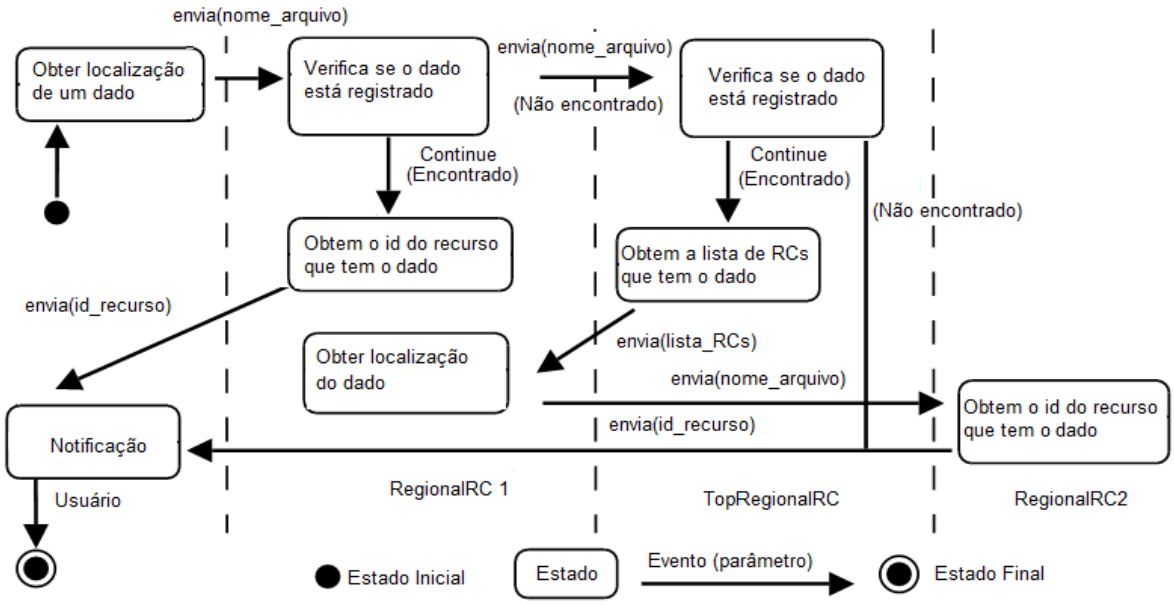


Figura 19 - Diagrama de estado referente à busca utilizando catálogos de réplica (42)

4.2 Implementação do protocolo de redes estruturadas no simulador

Os protocolos de redes de *overlay* estruturados apresentam uma topologia mais controlada, as regras para conexão e desconexão dos nós são mais rígidas, ou seja, ela não ocorre de maneira arbitrária e há regras para distribuir os dados na rede de modo que facilite a posterior localização. Sistemas como Chord, CAN, Pastry e Tapestry utilizam essa arquitetura.

Neste trabalho optamos por implementar uma versão de protocolo de rede estruturada baseado no protocolo Chord. O protocolo é composto por algumas funções que definem o comportamento do ambiente, destas foram implementadas as seguintes:

- *Stabilize*: utilizada para atualizar tabela de roteamento.
- *Join*: solicitação para entrada na rede de um novo *peer*.
- *Notify*: resposta de solicitação para entrada na rede de um novo *peer*.

- *Lookup*: localização de todos os *peers* que contém informações sobre um dado.
- *Get:* recuperar a localização física de um dado.
- *Post:* registrar as informações (nome e localização física) de um dado na rede.

Para criar um ambiente Chord utilizando o pacote de redes de *overlay* foi feita uma extensão da *class Overlay*.

```
public class Chord extends Overlay
```

A Figura 20 ilustra a estrutura básica do protocolo Chord. As funções citadas foram implementadas em métodos seguindo as especificações do protocolo Chord, estas funções foram implementadas nos métodos abstratos fornecidos pela *class Overlay*.

```
public class Chord extends Overlay {

  public Chord()throws Exception{}

  //método é responsável por qerenciar todos os eventos de entrada
  public void body() {}

  //Processa os eventos de entrada
  public boolean processEvent(Sim_event ev) { return true; }

  //Métodos para busca
  public void join(Sim_event ev){}
  public void notify(Sim_event ev){}
  public void stabilize(Sim_event ev){}
  public void replicaKeys(){}
  public void connectResource(){}
  public void connectUser(int dest_peer){}
  public void close(){}

  //Métodos para manutenção
  public void stabilization(Sim_event ev){}

  //Métodos para busca
  public int  search (String file){ return idResource: }
  public void lookup(Sim_event ev){}
  public void post(String file, int idPeer, int dest){}
  public int  get(String file, int dest){ return idResource;}
}
```

Figura 20 - Estrutura básica do protocolo Chord

Foi implementada a classe *class ChordMessageTag* que define constantes utilizadas para identificar as mensagens. O Chord utiliza uma tabela de roteamento *(finger)* presente em cada *peer*, esta tabela possui informações sobre alguns *peers* o que permite buscas mais eficientes. A *class FingerTable* representa esta tabela.

Como no Gnutella, utilizamos a estrutura *Catalogue File* para armazenar informações sobre os dados armazenados nos *peers* e a estrutura *Neighbor Node* para armazenar informações dos *peers* vizinhos.

A estrutura *DHT* do pacote *overlay* fornece funcionalidades para criação de uma tabela de indexação distribuída DHT, como a função *Hash* SHA-1.

As classes implementadas podem ser observadas (utilizando a modelagem orientada a objetos UML) na Figura 21.

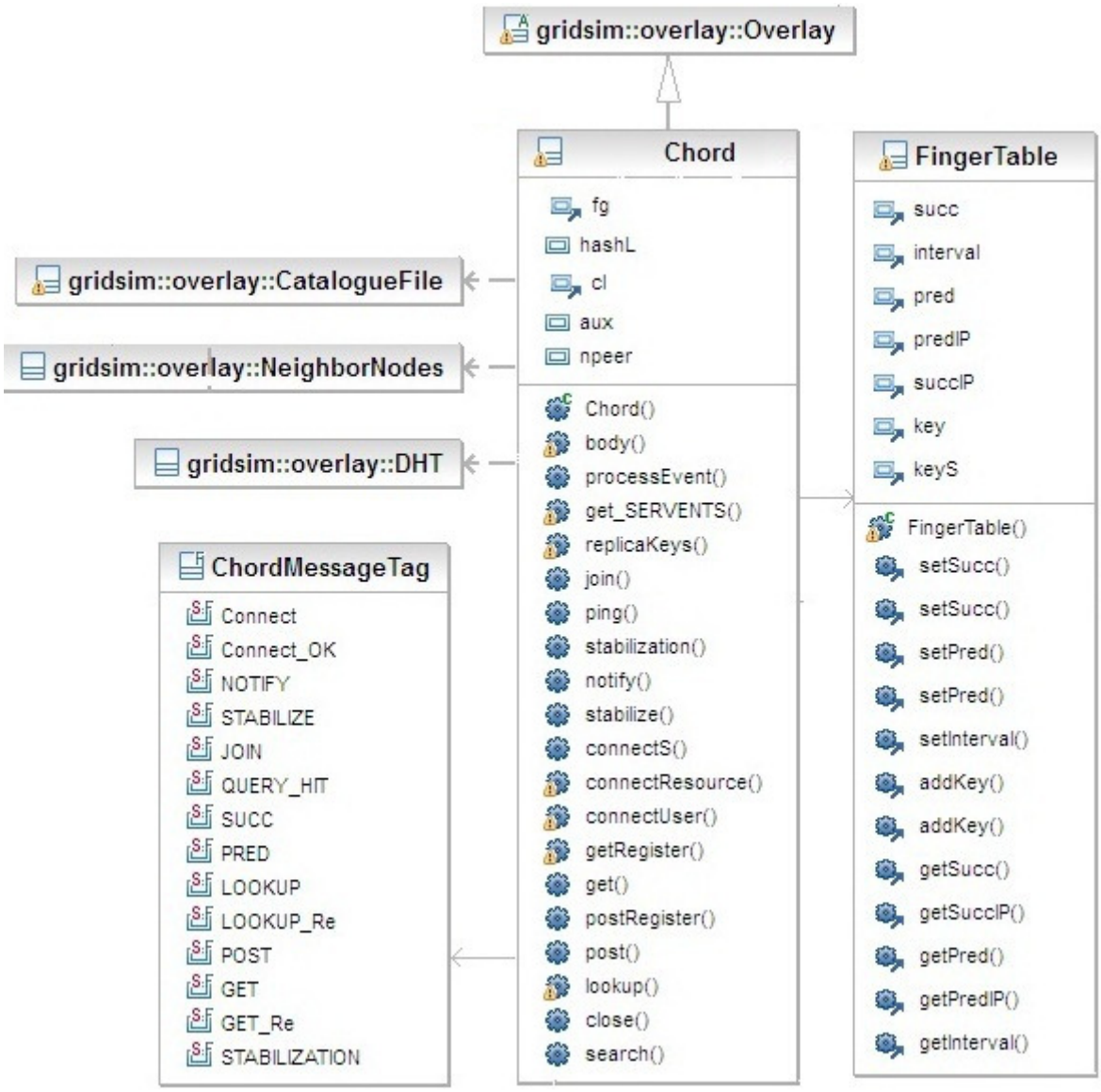


Figura 21 - Diagrama de classes Chord

Conexão dos *Peers*

A entrada de *peers* ocorre de modo dinâmico, para cada *peer* da rede uma chave para a identificação é criada aplicando a função *hash*, esta chave vai determinar a posição do *peer* na rede.

Quando um *peer* X deseja entrar na rede ele envia um *Join* para um *peer* Y que já participa da rede, como resposta ele recebe uma mensagem *Notify* indicando com qual *peer* ele deve estabelecer uma ligação, o *peer* X inicializa sua tabela de roteamento e envia mensagens para atualizar as tabelas de roteamentos de seus antecessores e sucessores de forma a refletir sua adição na rede.

Mecanismo de Busca

A Figura 22 ilustra o funcionamento do mecanismo de busca implementado. Cada *peer* é responsável por armazenar uma lista dos dados cujos identificadores são maiores ou iguais ao seu próprio identificador.

Inicialmente o *peer*, fonte da busca, consulta sua tabela de roteamento para que seja encontrado o *peer* com o identificador responsável mais próximo. É enviada uma mensagem a este *peer*, caso ele seja responsável pela localização do dado ele retorna ao *peer* requisitante a informação desejada, caso contrário, o processo de consulta à tabela de roteamento é realizado novamente a partir deste *peer*.

Após receber a informação do *peer* responsável pelo dado requerido foi utilizada a função *get* para obter a localização física deste arquivo.

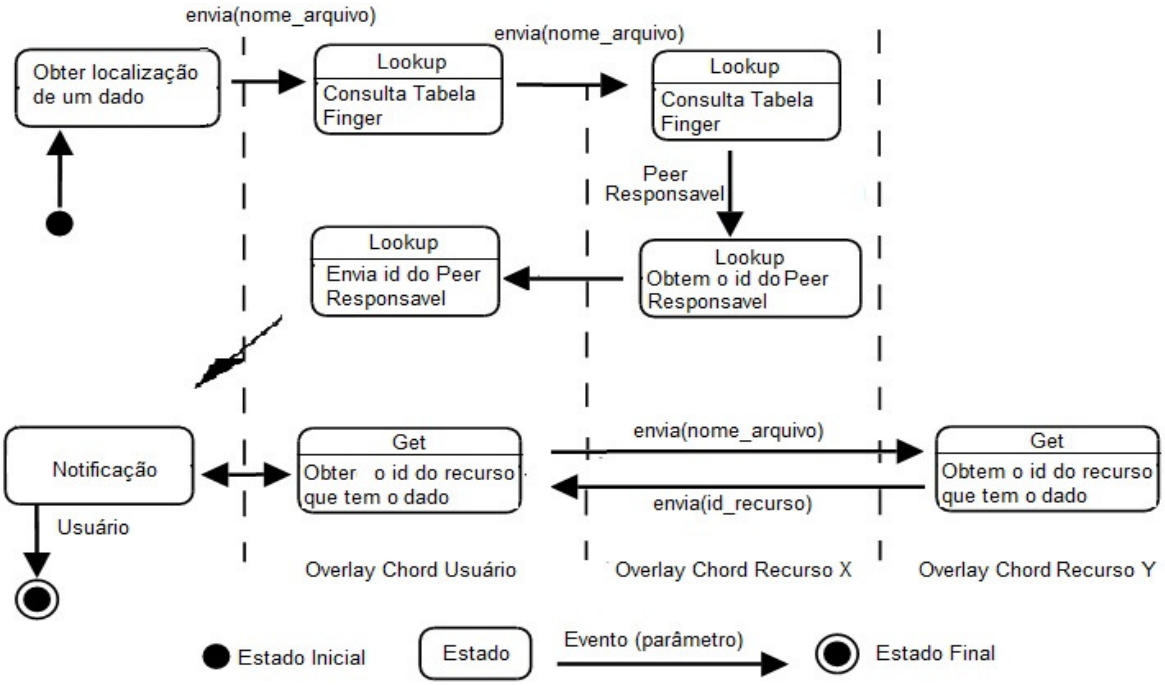


Figura 22 - Diagrama de estado referente à busca utilizando redes de overlay

4.3 Considerações finais do desenvolvimento de protocolos no simulador

Desenvolvemos protocolos para redes estruturadas e não estruturadas no simulador GridSim utilizando as funcionalidades do pacote de redes de *overlay* proposto neste trabalho, contudo para o desenvolvimento destes protocolos de forma flexível e fácil utilizando a estrutura fornecida pelo GridSim, é interessante que o desenvolvedor tenha um conhecimento básico dos principais conceitos de redes de *overlay* e do simulador GridSim.

5 RESULTADOS

Nesta fase de teste foram criados alguns cenários de teste no GridSim para verificar o funcionamento da implementação dos protocolos Gnutella e Chord utilizando as funcionalidades do pacote de rede de *overlay*

Para análise dos protocolos desenvolvidos utilizando as funcionalidades do pacote de *rede de overlay*, analisamos cinco fatores:

- **Taxa de Sucesso:** percentual de requisições atendidas.

- **Passos da busca:** caminho médio para localização de um arquivo na rede, ou seja, quantos saltos uma requisição necessita para encontrar um dado, não está incluída a fonte da busca.

- **Número de Mensagem de E/S**: número de mensagens enviadas para realizar operações de entrada e saída de *peers* na rede.

- **Número de Mensagem de Busca:** número de mensagens enviadas para todas as busca de dados.

Os valores apresentados para cada experimento correspondem à média de 30 execuções.

5.1 Cenários de teste desenvolvidos

Foram criados alguns cenários de teste somente para verificar o funcionamento da implementação dos protocolos Gnutella e Chord utilizando as funcionalidades do pacote de rede de *overlay*.

Foram criados alguns *peers* "usuários" (estendem a entidade *Data Grid User Peer)* para geração dos dados ou arquivos e para serem fontes das buscas destes dados.

Cada dado gerado por um *peer* "usuário" é distribuído de modo aleatório, ou seja, é escolhido um *peer* de forma aleatória entre os N existentes na rede para armazenar o dado. Somente *peers* "recurso" (estendem a entidade *Data Grid Resource Peer)* podem armazenar o dado. Ao realizar uma busca, o *peer* "usuário", fonte da busca, seleciona um dado aleatoriamente em uma lista de dados existentes na rede para ser procurado. A Figura 23 ilustra um cenário de teste.

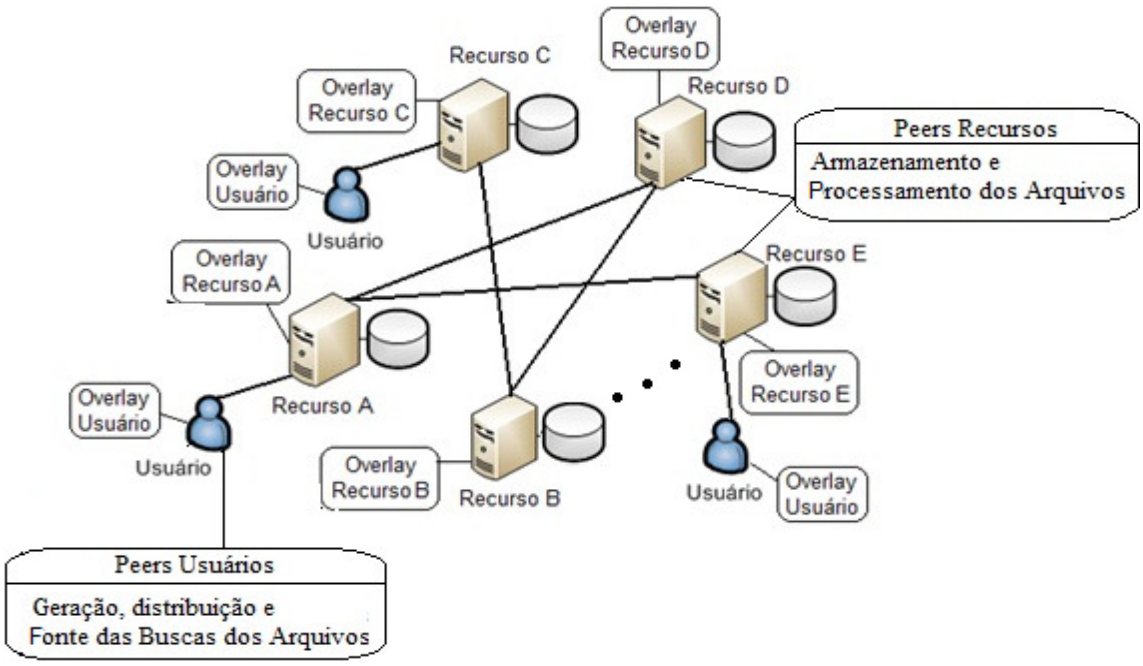


Figura 23 - Exemplo de cenário de teste

5.2 Configuração do ambiente

Rede Física

Foi criada uma rede física de alta confiabilidade, sem interferência e tendo todos os parâmetros idênticos para todos os links da rede, por exemplo, a MTU é de 1.500 bytes e a latência dos links é de 10 milissegundos.

Foram criados 30 roteadores interligados, sendo que os *peers* são conectados nestes roteadores de modo aleatório, contudo, foi definido um número máximo de vinte *peers* na qual um roteador pode estabelecer conexão.

Gnutella

Os *peers* na rede Gnutella tiveram no máximo 15 vizinhos, o TTL das mensagens foi de 5 saltos e o timeout de espera de 20s.

Os *peers* usuários começaram a executar as operações após 5 minutos (tempo de simulação), este período foi utilizado para permitir a entrada de alguns *peers* na rede, ou seja, a montagem de uma rede inicial. As operações foram executadas a cada 2 minutos (tempo de simulação).

A saída dos *peers* só ocorre no final da simulação e nenhum tipo de falha nos *peers* foi implementado.

A Tabela 3 demonstra os tipos de cenários de simulação gerados e seus respectivos parâmetros.

Tabela 3 - Parâmetros utilizados na simulação do protocolo Gnutella

N° Arquivos	N° Buscas	N° *Peers* Recurso	N° *Peers* Usuário	TTL	Timeout
100	100	30, 90, 180, 360	3	5, 7 ,9	30s
100	500	30, 90, 180, 360	3	5, 7, 9	30s
500	500	30, 90, 180, 360	3	5, 7, 9	30s
500	1000	30, 90, 180, 360	3	5, 7, 9	30s

As ligações lógicas de cada *peer* foram definidas de modo estático especificadas em um arquivo, para isso utilizamos um gerador de redes complexas que permitiu gerar o modelo de topologia de rede Barabási-Albert.

A Tabela 4 ilustra os parâmetros utilizados para estabelecer a rede virtual lógica como número total de ligações virtuais.

Tabela 4 - Parâmetros da topologia de rede Barabási-Albert

N° de *Peers*	N° de Links
30	81
90	360
180	761
360	1424

Chord

Os *peers* na rede Chord tiveram as tabelas de roteamento atualizadas a cada 30 segundos, o TTL das mensagens de 5 saltos e o timeout de espera de 20s. A função hash foi de 32 bits.

Os *peers* usuários começaram a executar as operações no primeiro cenário após 3 minutos (tempo de simulação) e no segundo cenário após 6 minutos (tempo de simulação), este período foi utilizado para permitir a entrada de alguns *peers* na rede, após este período as entradas dos *peers* ocorrem simultaneamente com as operações efetuadas pelos *peers* usuários. O momento em que um *peer* entra na rede é determinado de modo aleatório.

A saída dos *peers* só ocorre no final da simulação e não foi implementado nenhum tipo de falha ou interferência nos *peers*.

A Tabela 5 demonstra os tipos de cenários de simulação gerados e seus respectivos parâmetros.

Tabela 5 - Parâmetros utilizados na simulação do protocolo Chord

N° Arquivos	N° Busca	N° *Peers* Recurso	N° *Peers* Usuário	TTL	Timeout
100	100	30, 90, 180, 360	3	5	20s
100	500	30, 90, 180, 360	3	5	20s
500	500	30, 90, 180, 360	3	5	20s
500	1000	30, 90, 180, 360	3	5	20s

Cada *peer* mantém uma tabela de roteamento (*Finger Table*) com m entradas,

onde m pode ser até o número de *peers* da rede, essa tabela possui informações sobre outros *peers* (*peers* sucessores e antecessores).

A Tabela 6 demonstra o número máximo de entradas que cada *peer* teve em suas tabelas de roteamento de acordo com o número de *peers* na rede.

Tabela 6 - Número de entradas na tabela de roteamento

N° de *Peers*	N° Máximo Entradas na Tabela de *Finger*
30	4
90	6
180	7
360	8

5.3 Resultados obtidos Gnutella

Os gráficos nesta seção são resultados obtidos com a simulação do protocolo Gnutella.

Taxa de Sucesso

A Figura 24 demonstra o percentual de sucesso obtido, é possível notar que o protocolo Gnutella obtém um desempenho satisfatório em redes com poucos *peers*. Porém perde desempenho à medida que o tamanho da rede cresce. Isto ocorreu devido ao TTL definido não ser o mais adequado para as redes a partir de certo número de *peers*, comparando as Figuras 11 (A), (B) e (C) notamos que, com o aumento do TTL o percentual de sucesso fica mais equilibrado em todas as redes. Outro fator que influenciou no desempenho foi o timeout de espera de resposta, todas as mensagens de resposta após 20 segundos foram descartadas.

Considerando o número de Arquivo X Busca, a perda de desempenho ocorre de

modo semelhante em todos os cenários e quanto maior é o número de busca mais foi refletida essa perda de desempenho.

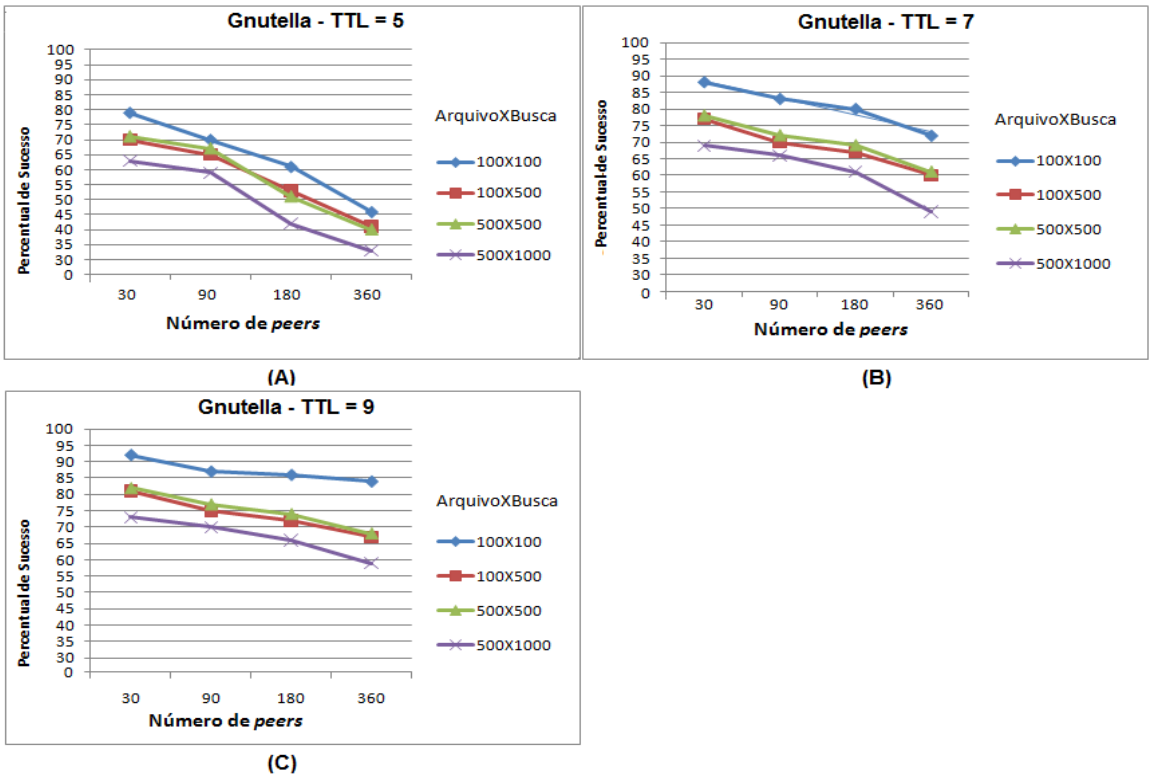


Figura 24 - Taxa de Sucesso – (A) Gnutella TTL = 5; (B) Gnutella TTL = 7; (C) Gnutella TTL = 9

Passos da busca

A Figura 25 ilustra o número de passo, ou seja, o número médio de mensagens geradas pelas buscas que obtiveram êxito. Os valores foram arredondados e é importante ressaltar que o número de saltos máximos permitidos para uma requisição foram de 5, 7 e 9.

Quando o número de *peers* na rede é baixo, os arquivos tendem a ser encontrados em *peers* mais próximos, ou seja, a rede apresenta um baixo caminho médio resultando na melhoria da busca por dados dentro da rede, isto acaba tendo grande influência na taxa de sucesso, demonstrada na Figura 24.

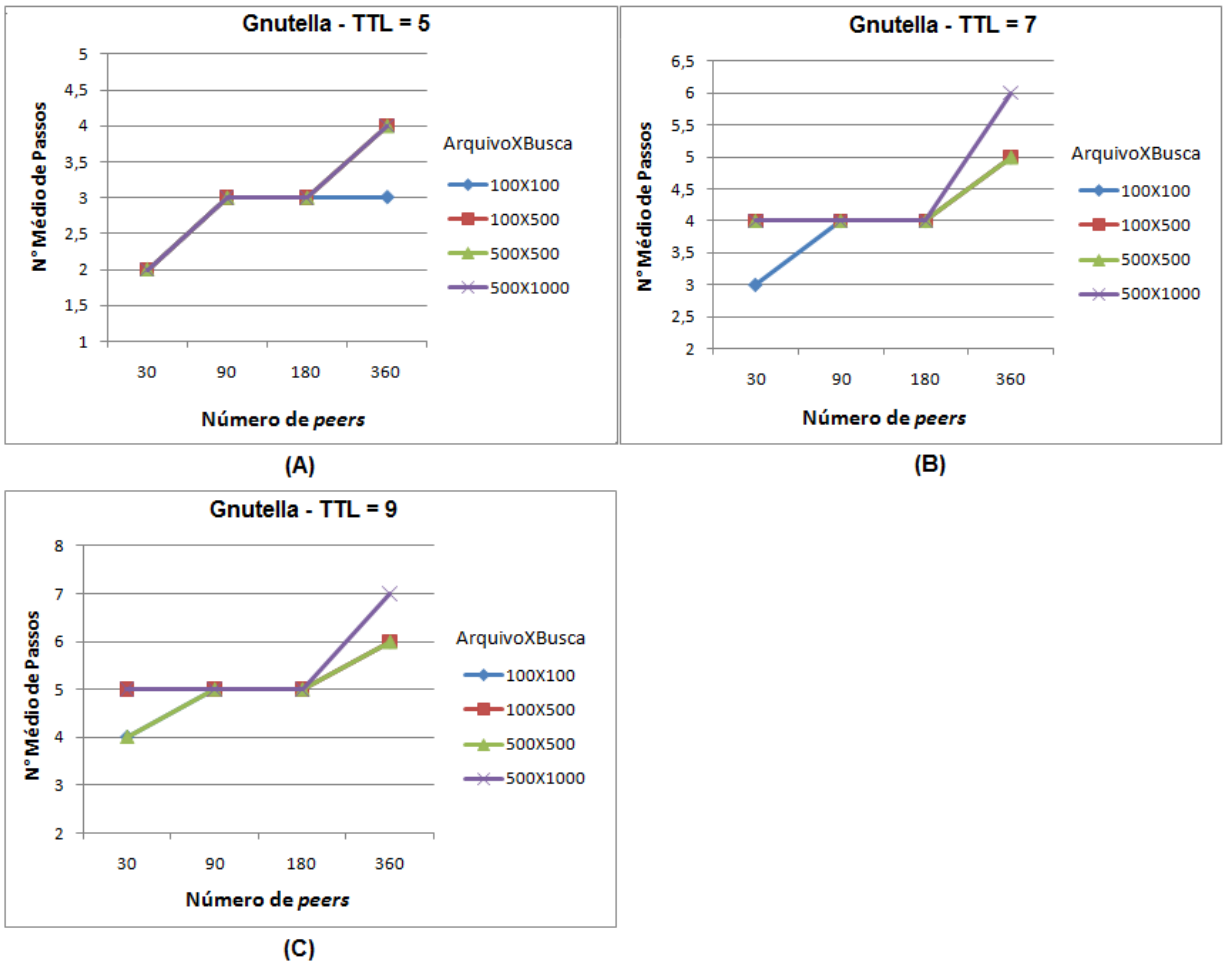


Figura 25 - Passo da Busca – (A) Gnutella TTL = 5; (B) Gnutella TTL = 7; (C) Gnutella TTL = 9

Número de Mensagem de E/S

A Figura 26 ilustra o número de mensagens de E/S, ou seja, mensagens que foram utilizadas para efetuar a entrada dos *peers* na rede. É possível notar que o número de mensagens cresce de acordo com o número de *peers* na rede e com o número de ligações virtuais lógicas (Tabela 4). Para cada ligação lógica é necessário no mínimo duas mensagem de E/S para estabelecer uma conexão entre os *peers.*

Como as ligações lógicas de cada *peer* foram definidas de modo estático o aumento do TTL não causa nenhum efeito.

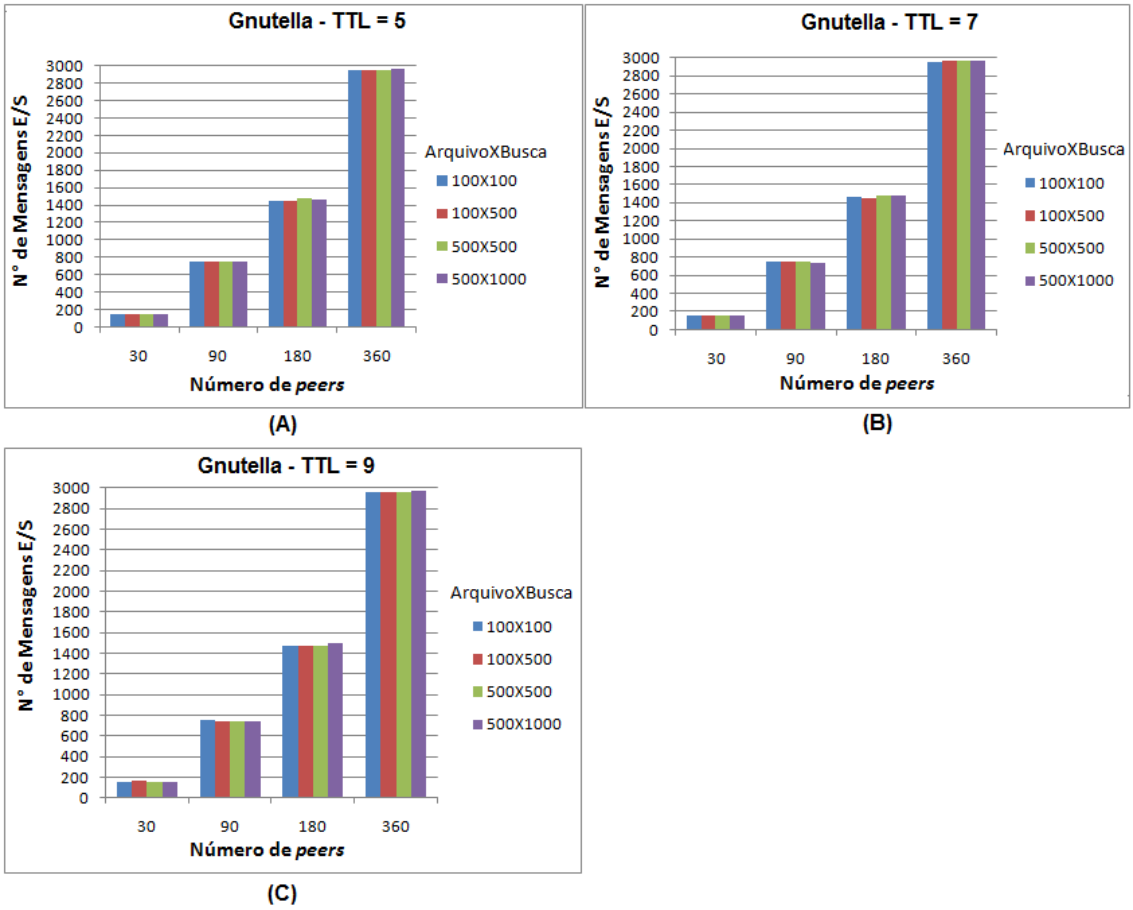


Figura 26 – Nº de Mensagens de E/S – (A) Gnutella TTL = 5; (B) Gnutella TTL = 7; (C) GnutellaTTL= 9

Número de Mensagem de Busca

A Figura 27 ilustra o número de mensagens de busca. É importante ressaltar que as mensagens de busca que não obtiveram êxito foram consideradas também. É possível notar no gráfico que o número de mensagens de busca cresce praticamente de forma linear de acordo com o número de *peers* na rede, isso ocorre, devido ao fato de que quanto maior a rede, mais dispersos os dados ficam, portanto, mais saltos são necessários para localizar um dado.

Portanto, *flooding* não foi eficiente em termo de energia gasta ou quantidade de mensagens geradas. No *flooding* quando um *peer* precisa de um dado específico, ele envia uma mensagem de busca a todos os seus vizinhos, e assim por adiante, dependendo do grau da vizinhança diversas mensagens de busca desnecessárias são geradas.

Considerando o número de Arquivo X Busca, quanto maior o número de buscas

mais mensagens de busca desnecessárias são geradas, o que eleva consideravelmente o número de mensagens e o impacto na rede.

O fato de aumentar o TTL de 5 para 7 ocasionou um impacto proporcionalmente maior do que aumentar o TTL de 7 para 9. Isso ocorre, devido ao fato que quando se aumenta o TTL consequentemente uma mensagem tende a passar mais de uma vez pelo mesmo *peer,* quando isso ocorre, descarta-se a mensagem.

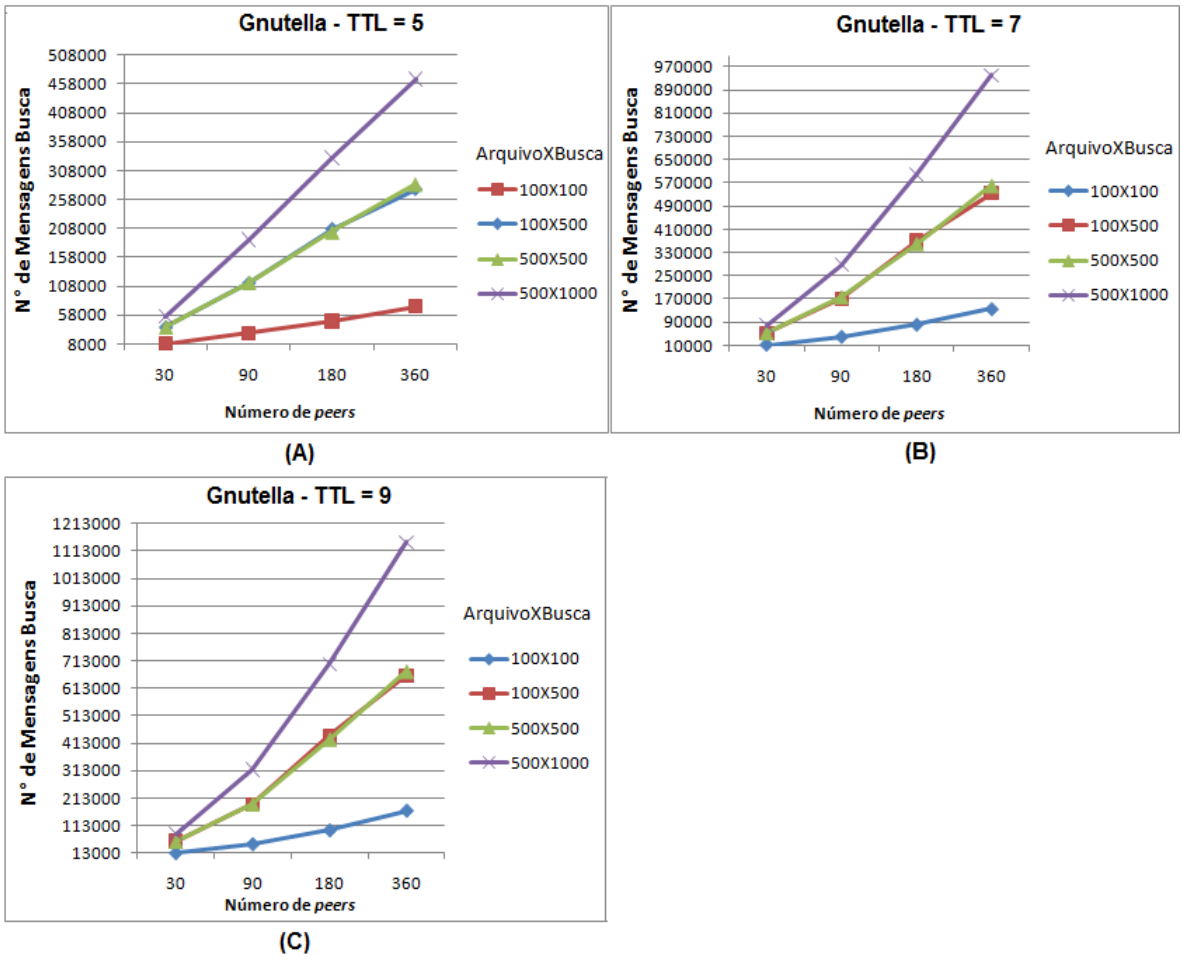


Figura 27 – Nº de Mensagens de Busca – (A) Gnutella TTL = 5; (B) Gnutella TTL = 7; (C) Gnutella TTL = 9

5.4 Resultados obtidos Chord

Os gráficos nesta seção são resultados obtidos com a simulação do protocolo Chord. Os *peers* usuários começaram a executar as operações no primeiro cenário (A) após 3 minutos (tempo de simulação) e no segundo cenário (B) após 6 minutos (tempo de

simulação).

Taxa de Sucesso

A Figura 28 (A) e (B) demonstra o percentual de sucesso obtido pelo protocolo Chord, podemos observar que foi possível obter um desempenho satisfatório, principalmente quando a rede está mais estável, Figura 28 (B).

Houve uma leve perda de desempenho à medida que o número de *peers* presente na rede cresceu, pois quanto maior for à movimentação da entrada dos *peers*, mais tempo leva a rede para se estabilizar. Portanto, a principal fonte de falha é a inconsistência gerada pela entrada e saídas de *peers*, pois a tabela de roteamento presente nos *peers* fica frequentemente desatualizada.

Considerando o número de Arquivo X Busca, os melhores resultados foram obtidos quando o número de buscas era superior ao número de arquivos, nesses cenários o feito da inconsistência nas tabelas de roteamento foi atenuado. Podemos observar também que o cenário de teste 100 Arquivos X 500 Buscas foi o mais eficiente.

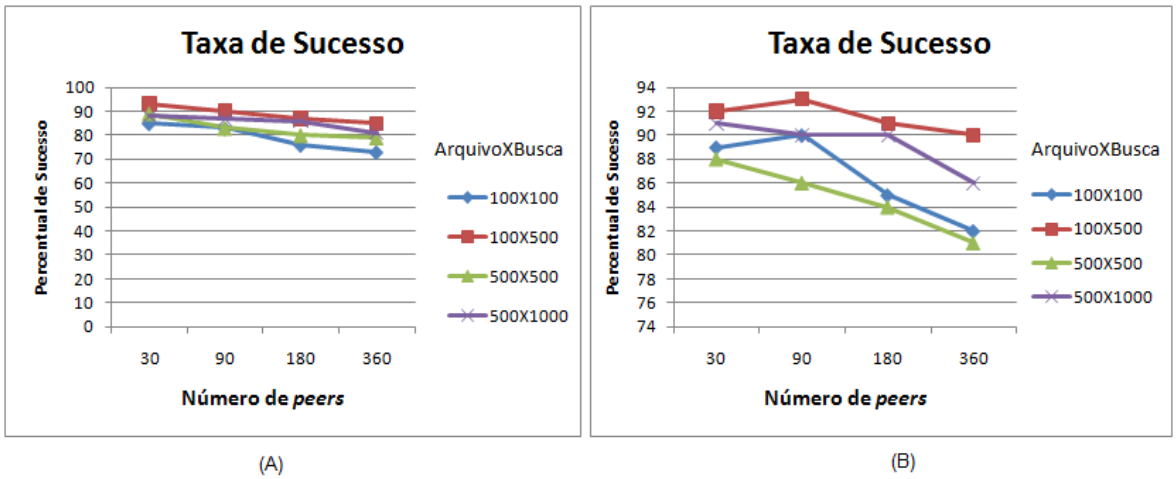
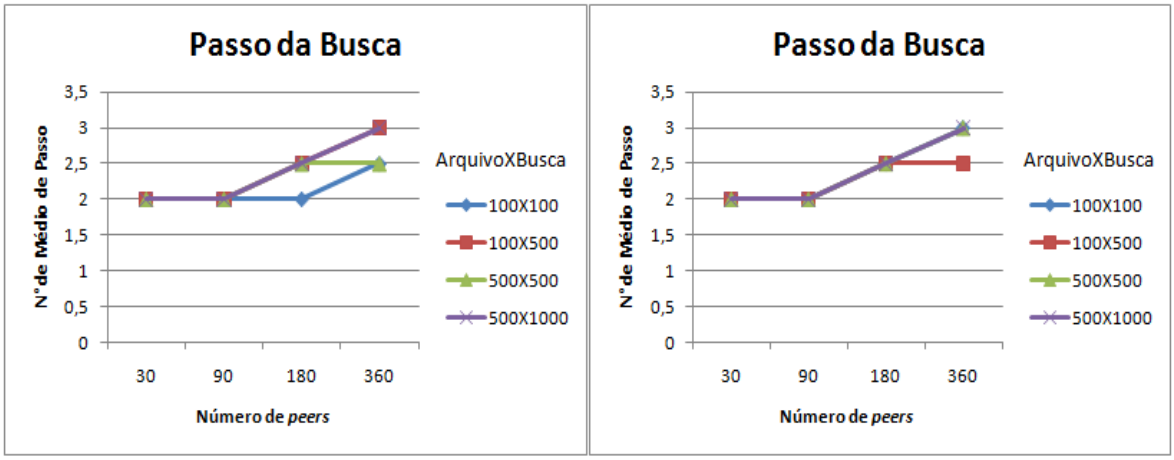


Figura 28 - Taxa de sucesso: (A) espera de 3 minutos; (B) espera de 6 minutos – Chord

Passos da busca

A Figura 29 (A) e (B) ilustram o número de passos. É possível notar que o número de passos está de acordo com o que encontramos na literatura e acreditamos que à medida que o número de *peers* presente na rede cresça o número de passos possa crescer como encontrado na literatura (logaritmicamente).



(A) (B)

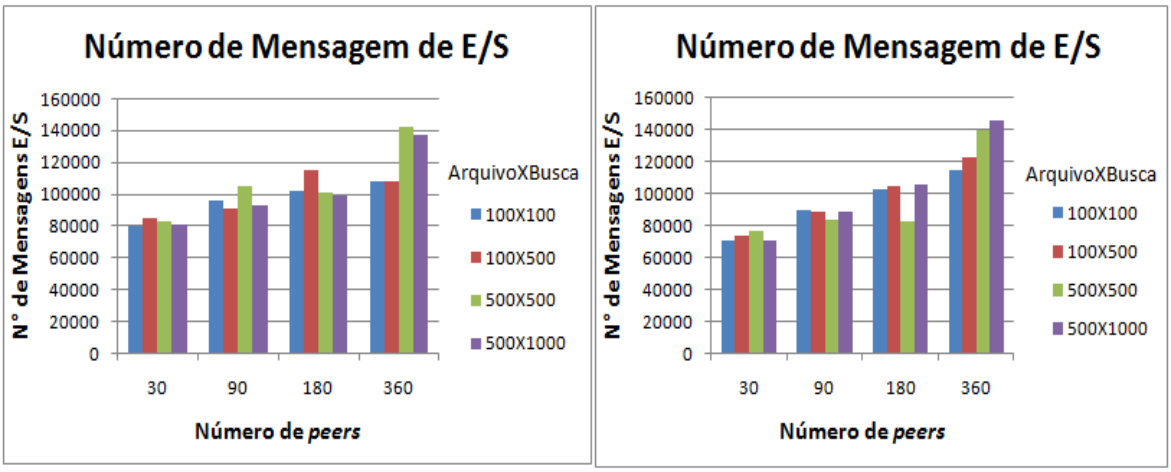
Figura 29 Passo da Busca: Espera de 3 Minutos - (B) Passo da Busca: Espera de 6 Minutos - Chord

Número de Mensagem de E/S

A Figura 30 (A) e (B) ilustram o número de mensagens de E/S, a montagem da rede e a correta informação nas tabelas de roteamento é fundamental para o sucesso das buscas. Portanto, é possível observar no gráfico que houve um grande número de mensagens de E/S, mensagens de estabilização (atualização das tabelas de roteamento, enviadas a cada 30 segundos) e mensagens para replicação de chave (mover as chaves correspondentes para um *peer* quando o mesmo for adicionado na rede). O número de mensagens para E/S está de acordo com o que encontramos na literatura.

No gráfico apresentado pela Figura 30 (B) houve mais mensagens, pois a simulação durou um tempo maior o que permitiu a entrada de mais *peers* e mensagens de

estabilização, ou seja, mais replicação de chave foi necessária, isso quando comparado com o gráfico da Figura 30 (A).



(A) (B)

Figura 30 Número de Mensagens de E/S (A) espera de 3 minutos; (B) espera de 6 minutos – Chord

Número de Mensagem de Busca

A Figura 31 ilustra o número de mensagens de busca. É possível notar no gráfico que o número de mensagens de busca cresce de forma constante independente do número de *peers* na rede. Isto demonstra que o Chord é um protocolo que permite uma boa escalabilidade.

Considerando o número de Arquivo X Busca, é possível observar que o número de mensagem cresce à medida que mais buscas são feitas, contudo é um crescimento natural e dentro de um padrão aceitável.

Houve uma pequena diferença no número de mensagens no cenário de teste 100 Arquivos X 500 Buscas que conseguiu ser mais eficiente quando comparados ao cenário 500 Arquivos X 500 Busca.

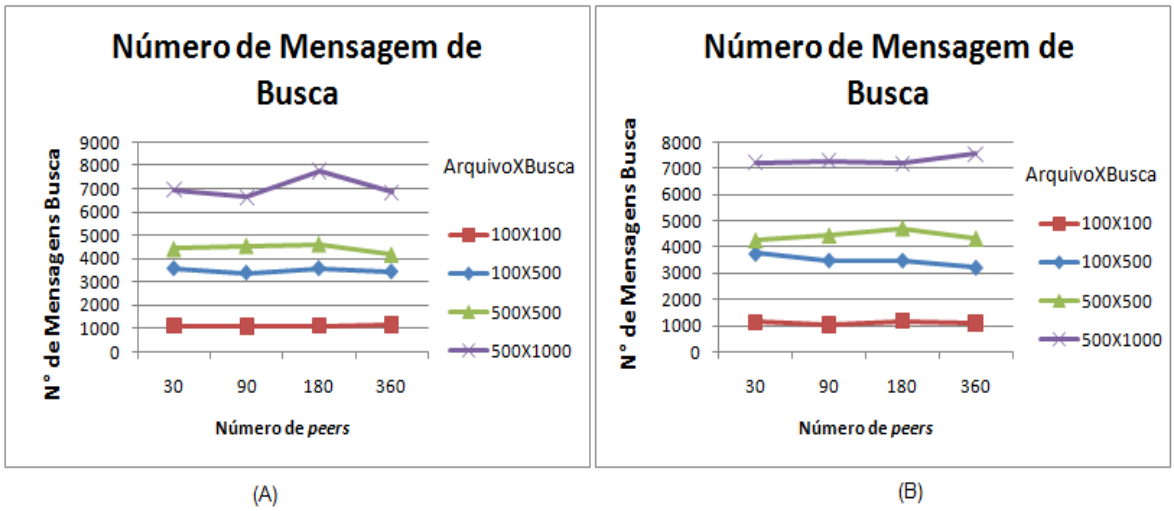


Figura 31 - Número de Mensagens de Busca (A) espera de 3 minutos; (B) espera de 6 minutos – Chord

5.5 Análise do desenvolvimento dos protocolos no simulador e resultados obtidos

O resultado mais importante que consideramos neste trabalho foi verificar se o pacote de rede de *overlay* permite a implementação de protocolos *peer-to-peer* de forma confiável, flexível e fácil, utilizando a estrutura de simulação de grade e redes fornecida pelo GridSim. Sendo assim, permitir que possa ser utilizado em pesquisas de estruturas *peer-to-peer* de forma interativa e prática, permitindo a realização de experimentos sobre os mais variados cenários de grades de dados.

De acordo com as simulações realizadas e analisando os resultados obtidos foi possível observar que os protocolos implementados no simulador estão de acordo ou próximo com o que está descrito na literatura. É bom ressaltar que as simulações ocorreram em uma rede sem interferência e estática.

A versão baseada no protocolo Gnutella utilizando a estratégia de *flooding* gerou uma grande quantidade de mensagens para efetuar buscas e o sucesso destas buscas dependeu muito do valor do TTL. Portanto, a estratégia de *flooding* foi razoável em termo de latência (ou quantidades de saltos), mas não foi eficiente em termo de quantidade de mensagens.

Seria interessante futuramente simular o protocolo Gnutella utilizando a estratégia de caminho aleatório, pois poupa mais energia apesar de apresenta perda de desempenho

em relação a latência quando os dados de interesse estão muito afastados.

A versão baseada no protocolo Chord gerou uma quantidade razoável de mensagens para entrada de *peers* na rede, porém obteve um ótimo desempenho principalmente quando houve menos inconsistência nas tabelas de roteamento. Além disso, demonstrou ter uma ótima escalabilidade na busca de dados.

Um fator limitante que o simulador apresentou foi ser pouco escalável, isso ocorre devido ao fato de que, durante a simulação, o GridSim cria uma série de entidades *multi-threaded*, cada uma executando em paralelo em seu próprio segmento. Dependendo do número de *peers* a simulação pode não suportar a quantidade de threads gerada.

Na Tabela 7 podemos observar o número de *threads* durante a simulação.

Tabela 7 – Número de thread durante a simulação

N° de *Peers*	**N° de Thread**
30	285
90	634
180	1.179
360	2.565

6 CONCLUSÃO

Este trabalho apresenta um sub-sistema de simulação de redes de *overlay* integrado à plataforma de simulação de computação de grade GridSim, visando fornecer condições e mecanismos que possibilitam a construção de sistemas *peer-to-peer* para serem testados na infra-estrutura de grade de computadores.

Para isso desenvolvemos uma alternativa aos modelos de topologia especificados na estrutura do GridSim. O pacote de redes *overlay* está disponível para a infra-estrutura de grade de dados do GridSim.

O pacote de simulação de redes de *overlay* integrado à plataforma de simulação de computação de grade GridSim foi desenvolvido com êxito. O pacote de redes *overlay* está disponível para a infra-estrutura de grade de dados do GridSim.

A partir dele foi possível desenvolver protocolos para redes estruturadas e não estruturadas no simulador e simulá-los utilizando cenários de grade de dados.

Usuários do GridSim provavelmente não devem encontrar dificuldades para utilizar o pacote de redes de *overlay.* Portanto, para um desenvolvimento satisfatório na implementação destes protocolos acreditamos ser interessante que o desenvolvedor tenha um conhecimento básico dos principais conceitos de redes de *overlay* e do simulador GridSim.

As principais conclusões obtidas dos estudos no Capitulo 6 com a análises dos resultados obtidos, foi possível observar que, os protocolos implementados no simulador estão de acordo com o que é encontrado na literatura.

6.1 Contribuição deste trabalho

A busca de recursos e informações em grades de computadores torna-se cada vez mais difícil com o crescente volume de informação e recursos em redes de computação e para descobrir recursos de maneira mais dinâmica, em grande escala, técnicas de *peer-to-*

peer têm sido utilizadas em grades.

O GridSim é um sistema muito difundido para simulação de grades, porém não provê recursos para simulação de redes de *overlay* em ambiente de computação em grade. Para isso desenvolvemos uma alternativa aos modelos de topologia especificados na estrutura do GridSim.

A contribuição deste trabalho foi de permitir a integração das funcionalidades do GridSim com as do pacote de redes de *overlay*, proporcionando assim uma capacidade mais ampla e integrada de simulação de infra-estrutura de grades. Isto irá beneficiar a grande base de usuários do GridSim e facilitar o estudo desse tipo de estruturas e o desenvolvimento de novas propostas de protocolos e algoritmos para seu uso em grades de computadores.

Com isso, seremos capazes de projetar protocolos mais eficientes que tirem proveito das propriedades topológicas, melhorando o desempenho de busca de dados dentro da rede. Além disso, será possível obter modelos ou cenários mais precisos para efetuar simulações.

6.2 Trabalhos futuros

Foram identificados pontos em que são necessários estudos mais profundos, deste modo, são sugeridos como possíveis trabalhos futuros:

- Implementar mais dinamismo e simular os protocolos com diferentes parâmetros, por exemplo, diferentes valores para TTL e *Timeout*.

- Desenvolver outros tipos de protocolos, tanto para redes estruturadas como para não estruturadas.

- Estender as funcionalidades do pacote de rede de *overlay* para todos os tipos de grades suportado pelo GridSim.

- Averiguar formas de deixar o simulador GridSim mais escalável.

- Realizar simulações com maiores detalhes de redes para averiguar se esse tipo de interferência compromete os resultados.

- Fornecer recursos mais flexíveis para criação ou alteração de políticas de replicação e substituição de dados.

REFERÊNCIAS

1 KRAUTER, K.; BUYYA, R.; MAHESWARAN, M. A taxonomy and survey of grid resource management systems for distributed computing. **Software - Practice and Experience,** v. 32, n. 2, p. 135–164, 2002.

2 FOSTER, I.; IAMNITCHI, A. On death, taxes and the convergence of peer-to-peer and grid computing. In: KAASHOEK, M. Frans; STOICA, Ion (Eds.). PEER-TO-PEER SYSTEMS II, INTERNATIONAL WORKSHOP – IPTPS, 2., February 21-22, 2003, Berkeley, CA, USA. **Proceedings…** Berkeley, CA, USA, 2003. (Lecture notes in computer science, v.2735, p.118-128)

3 ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D. A survey of peer-to-peer content distribution technologies. **ACM Computing Surveys**, New York, USA, v. 36, n. 4, p. 335–371, 2004.

4 TALIA, D.; TRUNFIO, P. Toward a synergy between P2P and grids. **IEEE Internet Computing**, Los Alamitos, v.7, n.4, p. 96, 94-95, 2003.

5 STOICA, I.; MORRIS, R.; KARGER, D.; KAASHOEK, M. F.; BALAKRISHNAN, H. Chord: a scalable peer-to-peer lookup service for internet applications. **ACM SIGCOMM Computer Communication Review,** v.31, n.4**,** p**.**149-160, Oct.2001. (Proceedings of the SIGCOMM Conference, San Diego, 2001).

6 RATNASAMY, S.; FRANCIS, P.; HANDLEY, M.; KARP, R. M.; SHENKER, S. A Scalable content-addressable network. In: ACM SIGCOMM CONFERENCE ON APPLICATIONS, TECHNOLOGIES, ARCHITECTURES, AND PROTOCOLS FOR COMPUTER COMMUNICATIONS, 2001, San Diego. **Proceedings…** San Diego: 2001. p.161-172.

7 ROWSTRON, A.; DRUSCHEL, P. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In: GUERRAOUI, Rachid (Ed.). IFIP/ACM-INTERNATIONAL CONFERENCE ON DISTRIBUTED SYSTEMS PLATFORMS HEIDELBERG, Germany, November 12-16, 2001. **Proceedings...** [S.l.]: Springer 2001. (Lecture Notes in Computer Science, v.2218, p.329-350, 2001)

8 ZHAO, B. Y.; HUANG, L.; STRIBLING, J.; RHEA, S. C.; JOSEPH, A. D.; KUBIATOWICZ, J. D. Tapestry: a resilient global-scale overlay for service deployment. **IEEE Journal on Selected Areas in Communications, v.** 22, n.1, p. 41-53, 2004.

9 FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: enabling scalable virtual organizations. **International Journal of Supercomputer Applications**. v. 15, n 3, p. 200- 222, Aug. 2001

10 CAMERON, D. G.; CARVAJAL-SCHIAFFINO, R.; MILLAR, A. P.; NICHOLSON, C.; STOCKINGER, K.; ZINI, F. Evaluating scheduling and replica optimisation strategies in OptorSim. In: INTERNATIONAL WORKSHOP ON GRID COMPUTING**,** 4, 2003. Phoenix,USA. **Proceedings…** Phoenix, AZ, USA p. 52–59, 2003.

11 BUYYA, R.; MURSHED, M. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. **Concurrency and Computation: Practice and Experience**, v. 14 n. 13, p. 1175–1220, 2002.

12 FOSTER, I. **Internet computing and the emerging grid**. Nature Web Matters, December 2000. Disponível em: <http://www.nature.com/nature/webmatters/Grid/grid.html> . Acesso em: novembro 2008.

13 TUECKE, S.; CZAJKOWSKI K.; FOSTER, I.; FREY, J.; GRAHAM, S.; KESSELMAN, C.; SNELLING, D; VANDERBILT P. **Open grid services infrastructure** (ogsi), *v*ersion 1.0. 2003. Disponível em: <www.ggf.org>. Acesso em: novembro 2008.

14 BOGHOSIAN, B.; COVENEY, P. Guest editors' introduction: scientific applications of grid computing. **Computers in Science and Engineering**, v. 7, p. 10–13, 2005.

15 FOSTER, I.; KESSELMAN, C.; Computational grids. high performance computing for computational science. In: PALMA, J. M. L. M.; DONGARRA, J.; HERNANDEZ, V. INTERNATIONAL CONFERENCE ON VECTOR AND PARALLEL PROCESSING SYSTEMS AND APPLICATIONS-VECPAR, 4.,Porto, Portugal, 2000. **Proceedings…** Porto- Portugal, 2000. (Lecture Notes in Computer Science, v. 1981, 2001).

16 MINOLI, D. **A networking approach to grid computing**. New York, USA, John Wiley & Sons, 2005.

17 CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH. **The large hadron collider computing grid project.** Disponível em: <http://lcg.web.cern.ch/LCG/>. Acesso em: agosto 2009.

18 LARSON, S. M.; SNOW, C. D.; SHIRTS, M. R.; PANDE, V. S. Folding@home and genome@home: using distributed computing to tackle previously intractable problems in computational biology. In: GRANT, R. (Ed.) **Modern methods in computational biology**, [S.l.]: Horizon Press, 2003.

19 STOCKINGER, H. Defining the grid: a snapshot on the current view. **The Journal of Supercomputing**, v. 42, n.1, p.3–17, 2007.

20 OURGRID. Disponível em: <http://www.ourgrid.org/>. Acesso em: novembro 2008.

21 INTEGRADE, Disponível em: <http://www.integrade.org.br/>. Acesso em: novembro 2008.

22 DATAGRID, Disponível em: <http://www.edg.org/>. Acesso em: novembro 2008.

23 NGS - *National Grid Service*, Disponível em: <http://www.grid-support.ac.uk/>. Acesso em: novembro 2008.

24 FOSTER, I.; YONG, ZHAO; RAICU, I.; LU, S. Cloud computing and grid Computing 360-degree compared. In: GRID COMPUTING ENVIRONMENTS WORKSHOP, GCE, 8, 2008, Austin, TX. **Proceedings…** Austin: [s.n.], 2008. p.1–10.

25 FERREIRA, L. *et al.* Introduction to Grid computing with globus. Technical Report, **IBM International Technical Support Organization**, Sept. 2003. Disponível em: <http://www.redbooks.ibm.com/abstracts/sg246895.html> Acesso em: novembro 2008.

26 FOSTER, I.; KESSELMAN, C. **The Grid 2**: blueprint for a new computing infrastructure, Part II: framework. San Francisco: Elsevier Science, 2003.

27 GLOBUS. Disponível em: <http://www.globus.org/toolkit/about.html/>. Acesso em: novembro de 2008.

28 FOSTER, I. Globus toolkit version 4: software for service-oriented systems. In: JIN, H.; REED, D.; JIANG, W. (Eds.). **Lecture notes in computer science.** Berlin/Heidelberg: Springer, 2005. vol. 3779, p. 513-520 (IFIP INTERNATIONAL CONFERENCE ON NETWORK AND PARALLEL COMPUTING)

29 COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas distribuídos**: conceitos e projetos. 4. ed. Porto Alegre: Bookman, 2007.

30 DOVAL, D.; O'MAHONY, D. Overlay networks: a scalable alternative for P2P. **IEEE Internet Computing ,** v.7, n.4, p. 79-82, July-Aug. 2003

31 ANDERSEN, D.; BALAKRISHNAN, H.;, KAASHOEK, F.; MORRIS, R. Resilient overlay networks. **ACM SIGOPS Operating Systems Review**, New York, NY,v.35 n. 5, p. 131 – 145,2001.

32 LUA, E. K.; CROWCROFT, J.; PIAS, M.; SHARMA, R.; LIM, S. A survey and comparison of peer-to-peer overlay network schemes. **IEEE Communications Surveys & Tutorials**, v.7, n.2, p.72-93, 2005.

33 JIANG, H.; JIN, S. Exploiting dynamic querying like flooding techniques in unstructured peer-to-peer networks. In: IEEE INTERNATIONAL CONFERENCE ON NETWORK PROTOCOLS,ICNP, 13,2005,Washington. **Proceedings…** Washington, DC: IEEE Computer Society, 2005. p. 122–131.

34 GKANTSIDIS, C.; MIHAIL, M.; SABERI, A. Random walks in peer-to-peer networks: algorithms and evaluation volume. **Performance Evaluation**, Amsterdam, The Netherlands, v.63, n. 3, p. 241-263, 2006.

35 RIPEANU, M; **Peer-to-peer architecture case study:** Gnutella network. Chicago: University of Chicago, 2001. Technical report, TR-2001-26.

36 HAUSWIRTH, M.; SCHMIDT, R. An overlay network for resource discovery in grids. In: PROCEEDINGS INTERNATIONAL WORKSHOP ON DATABASE AND EXPERT SYSTEM APPLICATIONS-DEXA, 05,16, 2005,Copenhagen. **Proceedings…** Copenhagem; [s.n.], 2005. p. 343–348.

37 LI, JUAN; VUONG, SON. Semantic Overlay Network for Grid Resource Discovery In PROCEEDINGS OF THE IEEE/ACM INTERNATIONAL WORKSHOP ON GRID COMPUTING, 6., Nov. 2005. New York. **Proceedings…** New York, USA: Wiley Press, 2005. v.34, n. 7, p. 653 – 673.

38 ANDRADE N., COSTA L., GERMOGLIO G., CIRNE W. Peer-to-peer grid computing with the Ourgrid community. In: BRAZILIAN SYMPOSIUM ON COMPUTER NETWORKS-SBRC, 23., May 2005. Proceedings… [S.l.]; [s.n.], 2005. (4th Special Tools Session).

39 SULISTIO, A.; YEO, C. S.; BUYYA, R. A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. **Software - Practice** & Experience, New York, NY, v. 34, n. 7, p. 653 – 673, 2004.

40 SONG, H. J. et al. The microgrid: a scientific tool for modeling computational grids. **Journal Sci. Program**., v. 8, n. 3, p. 127–141, Amsterdam, Netherlands 2000.

41 LEGRAND, A.; MARCHAL, L.; CASANOVA, H. Scheduling distributed applications: the simgrid simulation framework. In**:** PROCEEDINGS OF THE 3ST INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, 3., 2003. **Proceedings…** Washington, DC, USA:IEEE Computer Society, 2003

42 SULISTIO, A., CIBEJ U., VENUGOPAL S., ROBIC B., BUYYA R. A Toolkit for Modelling and Simulating Data Grids: An Extension to GridSim, **Concurrency and Computation: Practice and Experience (CCPE)**, v. 20, n.13: 1591-1609, 2008. Online ISSN: 1532-0634, Printed ISSN: 1532-0626.

43 RODRIGUES, F. **Caracterização, classificação e análise de redes complexas**. 2007. . 157p. Tese (Doutorado em Física) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2007.

44 BARABASI, A. L.; ALBERT; R. Emergence of scaling in random networks. **Science,** v.286, n. 5439, p.509-512, Oct. 1999.