Universidade Federal do Rio de Janeiro

DISSERTAÇÃO DE MESTRADO

FABRÍCIO BARROS GONÇALVES

UM ESTUDO SOBRE O USO DE REDES P2P SOCIAIS PARA O COMPARTILHAMENTO DE RECURSOS EM AMBIENTES DE E-SCIENDE

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMPATICA
NUCLEO DE COMPUTAÇÃO ELETRONICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

FABRÍCIO BARROS GONÇALVES

UM ESTUDO SOBRE O USO DE REDES P2P SOCIAIS PARA O COMPARTILHAMENTO DE RECURSOS EM AMBIENTES DE E-SCIENCE

RIO DE JANEIRO
2010

FABRÍCIO BARROS GONÇALVES

UM ESTUDO SOBRE O USO DE REDES P2P SOCIAIS PARA O COMPARTILHAMENTO DE RECURSOS EM AMBIENTES DE E-SCIENCE

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, Departamento de Ciência da Computação, Instituto de Matemática, Universidade Federal do Rio de Janeiro, como requisito parcial para obtenção do título de Mestre em Informática.

Orientadores: Vanessa Braganholo Murta
Carlo Emmanoel Tolla de Oliveira

RIO DE JANEIRO
2010

G635 Gonçalves, Fabrício Barros
Um Estudo sobre o uso de redes P2P sociais para o compartilhamento de recursos em ambientes de e-science / Fabrício Barros Gonçalves. – 2010.
190 f.: il.

Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro, Instituto de Matemática, Núcleo de Computação Eletrônica, Rio de Janeiro, 2010.

Orientadores: Vanessa Braganholo Murta; Carlo Emmanoel Tolla de Oliveira

1. Redes Sociais – Teses. 2. Ambientes E-Science – Teses 3. Compartilhamento de recursos – Teses. I. Murta, Vanessa Braganholo (Orient.). II. Oliveira, Carlo Emmanoel Tolla. (Orient.) III. Universidade Federal do Rio de Janeiro, Instituto de Matemática, Núcleo de Computação Eletrônica. IV. Título

CDD

FABRÍCIO BARROS GONÇALVES

UM ESTUDO SOBRE O USO DE REDES P2P SOCIAIS PARA O COMPARTILHAMENTO DE RECURSOS EM AMBIENTES DE E-SCIENCE

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, Departamento de Ciência da Computação, Instituto de Matemática, Universidade Federal do Rio de Janeiro, como requisito parcial para obtenção do título de Mestre em Informática.

Aprovada em Maio de 2010

_____

Vanessa Braganholo Murta, DSc, PPGI/UFRJ

_____

Carlo Emmanoel Tolla de Oliveira, PhD, PPGI/UFRJ

_____

Jonice de Oliveira Sampaio, DSc, PPGI/UFRJ

_____

Adriana Santarosa Vivacqua, Dsc, PPGI/UFRJ

_____

Maria Cláudia Reis Cavalcanti, Dsc, IME

A minha Aline com amor.

AGRADECIMENTOS

Muitas são as pessoas a quem devo agradecer por esta conquista, pois acredito que a gratidão é uma virtude. Assim, mesmo correndo o risco de esquecer alguém, não posso deixar de citar alguns nomes e expressar a minha eterna gratidão, são eles:

- A Deus, o autor e consumador da minha fé. Muito obrigado pela força para chegar até o fim, pela sabedoria para as escolhas certas, pelo auxílio das pessoas que você colocou no meu caminho para me ajudar e, por fim, por estar ao meu lado em todos os momentos;
- Aos meus pais, que foram grandes auxiliadores nesta etapa de minha vida. Vocês me deram todo o suporte emocional e físico necessário para que eu pudesse atingir este objetivo. Vocês são os meus exemplos de garra e de que devemos lutar sempre por aquilo que queremos;
- A minha amada, Aline, que me entende, batalha comigo e é participante de minhas vitórias. Você soube me entender nos momentos de dificuldade e sempre me deu força para chegar até o fim. Te amo.
- A Professora Vanessa Braganholo, que nesses últimos meses sua participação foi fundamental para a conclusão deste trabalho. Acredito que sem o seu apoio ele não teria se concretizado. Obrigado por acreditar em mim e por toda a compreensão e paciência que teve para comigo. Mais do que o auxílio de uma orientadora, ganhei também o apoio de uma grande amiga, que sempre estarei à disposição quando precisar de mim;
- Ao Professor Carlo Emmanoel Tola de Oliveira, que auxiliou no desenvolvimento deste trabalho, com grandes idéias sobre o uso de computação distribuída na forma em que foi proposto neste trabalho. Muito obrigado pela sua amizade, por acreditar em mim, por ter paciência nos momentos em que eu precisava de ajuda.
- Aos meus tios Antônio Carlos e Gessiléia, a minha prima Vanessa e seu marido Marcelo, por facilitarem a minha estada no Rio de Janeiro durante o período em precisei. Sem o apoio de vocês não seria possível ter começado este sonho;

- Aos amigos conquistados durante o mestrado. A Lívia Monerat, Felipe Meyer e Ana Paula pelos vários momentos de descontração que tivemos no Labase. Ao Luiz Gustavo e Izalmo pelo companheirismo durante o início do mestrado. Ao Professores Marcos Antônio, Marcos Athayde, Regina Helena, Solange Prado pela compreensão nos momentos em que faltei para com as minhas obrigações enquanto Coordenador do Bacharelado de Sistemas de Informaçao e do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Centro Universitário São Camilo – Espírito Santo. Ao Prof. Evandro Bolsoni que foi a pessoa que permitiu de maneira direta e indireta para que eu chegasse até aqui.
- Ao professores que coordenei enquanto coordenador de curso no Centro Universitário São Camilo – Espírito Santo. Vocês me propiciaram uma grande tranqüilidade nos momentos de crise no curso e no término deste trabalho.

Por fim, agradeço a todos que de alguma forma contribuíram para que este trabalho pudesse ser concluído. Espero um dia poder retribuir de alguma forma o apoio que me ofereceram.

"É melhor tentar e falhar,
que preocupar-se e ver a vida passar;
é melhor tentar, ainda que em vão,
que sentar-se fazendo nada até o final.
Eu prefiro na chuva caminhar,
que em dias tristes em casa me esconder.
Prefiro ser feliz, embora louco,
que em conformidade viver ..."

Martin Luther King

RESUMO

GONÇALVES, Fabrício Barros. **Um Estudo sobre o uso de redes P2P sociais para o compartilhamento de recursos em ambientes de e-science**. Rio de Janeiro, 2010. Dissertação (Mestrado em Informática) – Departamento de Ciência da Computação, Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2010.

Com o surgimento de uma nova geração de infraestruturas de computação e comunicação, membros de comunidades científicas têm mudado a maneira de criar, conduzir e administrar experimentos e, também, de compartilhar dados e colaborar uns com os outros. Apesar de existirem infraestruturas computacionais distribuídas, com grande capacidade computacional, elas apresentam alguma desvantagem relacionada à implantação ou uso de recursos compartilhados. No entanto, o problema não está necessariamente nos tipos de infraestruturas computacionais distribuídas, e sim, na forma de se compartilhar recursos. A proposta desta dissertação é o uso de redes *peer-to-peer* como infraestrutura computacional em comunidades científicas, devido aos diversos benefícios oferecidos por esse tipo de sistema distribuído, tais como compartilhamento do custo operacional da infraestrutura de computação; aumento da oferta de recursos em sistemas de computação distribuída, entre outros. O objetivo é contribuir na direção de novas políticas de computação distribuída que tratem o gerenciamento e a transparência de acesso aos recursos em um ambiente de computação altamente distribuída. O trabalho parte da premissa de que pessoas tendem a se comportar de maneira colaborativa, quando possuem interesses em comum. Então, redes sociais *peer-to-peer* podem ser formadas, desde que oportunidades de colaboração (por exemplo, projetos, trabalhos, estudos, entre outros) entre pesquisadores sejam criadas e publicadas em um ambiente distribuído para a colaboração científica. Como consequência da formação de uma rede social *peer-to-peer*, seus participantes podem compartilhar recursos, a fim de transformar a rede social em uma entidade computacional, que fornecerá acesso transparente aos recursos compartilhados entre vários membros de uma rede social *peer-to-peer*.

Palavras-chave: Computação Distribuída, Compartilhamento de Recursos, Objetos Móveis, Redes Sociais, Colaboração, *e-Science*

ABSTRACT

GONÇALVES, Fabrício Barros. **Um Estudo sobre o uso de redes P2P sociais para o compartilhamento de recursos em ambientes de e-science**. Rio de Janeiro, 2010. Dissertação (Mestrado em Informática) – Departamento de Ciência da Computação, Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2010.

With emerging of a new generation of computing and communication infrastructures, members of scientific communities have changed the manner to create, lead and manage scientific experiments and also to share data and to collaborate with each others. Despite the existence of distributed computational infrastructures with big computing power, they present some drawbacks related to the implantation or the use of shared resources. However, the problem is not necessarily in the types of distributed computational infrastructures, but on the manner to share resources. The proposal of this dissertation is the use of peer-to-peer networks as computing infrastructure in scientific communities, due to several advantages offered by this type of distributed system such as the sharing of computing infrastructure operational cost; the increase of resource offer in distributed computing systems, among others. The goal is to contribute in the direction of new distributed computing policies that allows the management and transparent access to the resources at high distributed computing environment. The work is based on the principle that people tend to behave collaboratively when they have common interests. So, social peer-to-peer networks can be formed whether collaboration opportunities (i.e. projects, works, studies among others) between researchers have been created and published in a distributed environment of scientific collaboration. As a consequence of the social peer-to-peer network creation, their members can share resources in order to transform the social network in a computing entity that will provide transparent access to the shared resources among members of social peer-to-peer network.

Keywords: Distributed Computing, Resource Sharing, Mobile Objects, Social Networks, Collaboration, e-Science

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	*Application Programming Interface*
BLAST	*Basic Local Alignment Search Too*
CAN	*Content Addressable Network*
CPU	*Central Processing Unit*
DAO	*Data Access Object*
DNA	*Deoxyribonucleic Acid*
DTO	*Data Transfer Objects*
HTTP	*HiperText Transfer Protocol*
IP	*Internet Protocol*
JEE	*Java Enterprise Edition*
P2P	*Peer-to-Peer*
PDA	*Personal Digital Assintant*
SGWfC	Sistemas Gerenciadores de Workflows Científicos
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
TTL	*Time-to-Live*
UDP	*User Datagram Protocol*
URI	*Uniform Resource Identifier*
WCG	*World Community Grid*
WWW	*World Wide Web*

SUMÁRIO

1 INTRODUÇÃO

No passado, um dos grandes obstáculos para a formação e colaboração nas comunidades científicas era o fato de que seus membros tinham que estar localizados fisicamente uns próximos aos outros. Desse modo, os membros de comunidades científicas (por exemplo, físicos, químicos, biólogos, entre outros) colaboravam usando recursos intelectuais de maneira compartilhada, tais como coleções de livros; periódicos e artigos; e/ou equipamentos caros e/ou específicos naquele tempo (FAROOQ, et al. 2009) como, por exemplo, computadores (*mainframes*), cíclotrons, microscópios eletrônicos, entre outros. Apesar do surgimento dos computadores pessoais e da Internet, experimentos científicos ainda eram realizados em centros especializados de pesquisas, sem o uso intensivo de computação.

Fazendo uma análise disto, a grande contribuição da computação foi propiciar, por meio de serviços de Internet, o compartilhamento de informações entre membros de uma mesma comunidade científica. Mas, ainda, pesquisadores eram tolhidos em relação à velocidade de transmissão de dados e à capacidade computacional dos computadores pessoais (processadores, memórias e discos). Naquele momento, ainda não era possível transmitir e armazenar um grande montante de dados, uma vez que as infraestruturas de computação e comunicação não estavam preparadas para tal fim, e, também, não existiam hardwares, em computadores pessoais, capazes de realizar o processamento e o armazenamento massivo de dados.

Hoje, com o surgimento de uma nova geração de infraestruturas de computação e comunicação, membros de comunidades científicas têm mudado a maneira de criar, conduzir e administrar experimentos e, também, de compartilhar dados e colaborar uns com os outros. Nesse caso, pesquisadores têm usado, para obtenção de resultados em suas pesquisas, softwares de computação intensiva, de natureza paralela ou distribuída, que são providos por infraestruturas computacionais, tais como *clusters* ou grades computacionais. Tais infraestruturas computacionais, então, executam algoritmos, computando e produzindo uma quantidade massiva de dados. Além disso, a colaboração entre membros de uma comunidade científica tem sido feita por meio de bibliotecas digitais, que são mantidas por *colaboratórios* (FAROOQ, et al. 2009). Por meio dos *colaboratórios*, pesquisadores podem compartilhar

recursos intelectuais, que, por sua vez, permitem a colegas de pesquisa, localizados em qualquer lugar, acessar, visualizar, manipular e discutir sobre resultados obtidos em pesquisa. Isso é possível, pois serviços computacionais avançados de aquisição de dados e colaboração são disponibilizados por meio de redes de alto desempenho, conforme . Assim, todo este conjunto de softwares e hardwares, que, por sua vez, apóiam intensivamente atividades científicas, tem sido chamado de ciberinfraestrutura (DAVID 2004). Por sua vez, as atividades científicas, apoiadas por meio de ciberinfraestruturas, têm sido chamadas de *e-Science* (DAVID 2004).

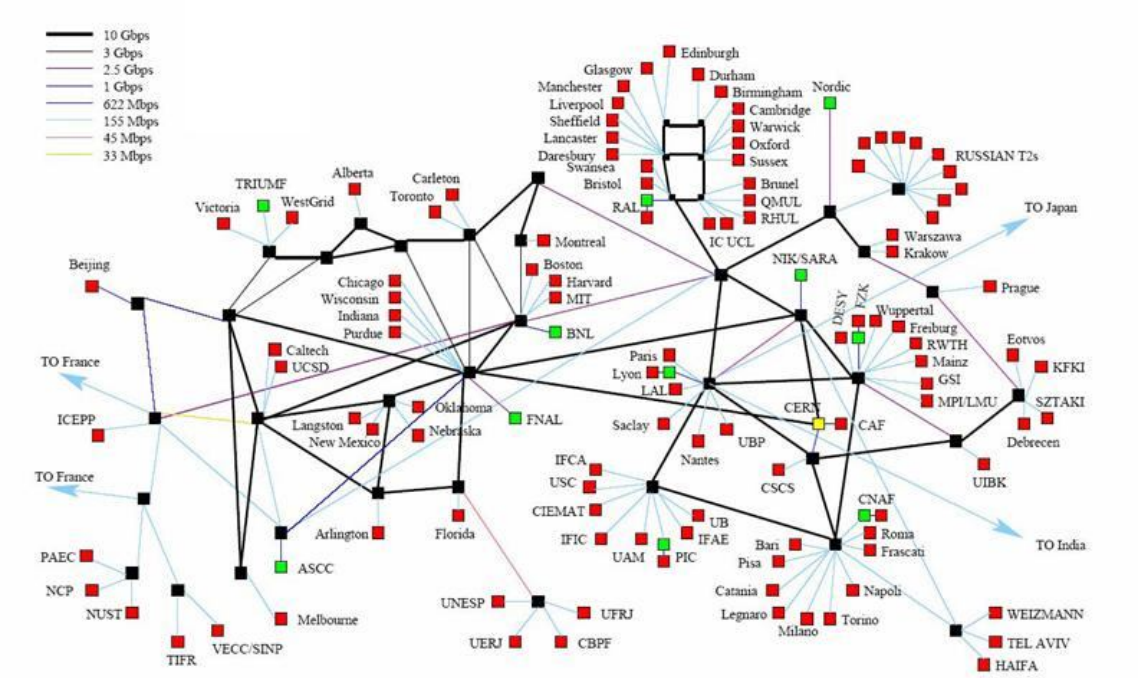


Figura 1.1**: Rede de Colaboração Científica (AL KISWANY, et al. 2007)**

Nesse sentido, usando ciberinfraestruturas ou ambientes computacionais de *e-Science*, pesquisadores definem suas hipóteses e criam experimentos, a partir de um conjunto de algoritmos e programas, que operam em ambientes de computação distribuída, permitindo a geração, a coleta e o armazenamento de um grande montante de dados experimentais. No entanto, organizar esses dados em um repositório ao qual pesquisadores possam ter acesso, a qualquer hora e de qualquer lugar, requer atividades especializadas (por exemplo, aquelas exercidas por administradores de infraestruturas computacionais) ou ferramentas computacionais para lidar com grandes volumes de dados. Essas ferramentas devem permitir a distribuição e o processamento de dados sobre os recursos computacionais compartilhados em uma ciberinfraestrutura, de modo que os pesquisadores não precisem preocupar-se com

detalhes de infraestrutura de computação (por exemplo, conexões de rede, sistemas operacionais, tecnologias de armazenamento, entre outros).

Apesar de existirem infraestruturas computacionais distribuídas, com grande capacidade computacional, como *clusters*, grades computacionais e ambientes heterogêneos de computação distribuída, elas apresentam algum tipo de desvantagem, tornando difícil a sua utilização para *e-Science* (OGASAWARA, et al. 2009). *Clusters* são soluções caras no que diz respeito à implantação e manutenção, e nem sempre são financeiramente acessíveis aos pesquisadores. Grades computacionais ainda apresentam grandes barreiras burocráticas, mas necessárias, para que um pesquisador se torne usuário desse tipo de infraestrutura (OGASAWARA, et al. 2009). Por fim, ambientes heterogêneos de computação distribuída, como *World Community Grid* (WCG 2009), disponibilizam milhões de computadores para atividades de *e-Science*. Porém, gasta-se muito tempo entre a criação, configuração, submissão e aprovação de um projeto, de modo que ele possa em seguida ser executado nesse tipo de ambiente (OGASAWARA, et al. 2009). Mesmo assim, pesquisadores utilizam os tipos de infraestruturas supracitados como meio de criar, conduzir e administrar experimentos científicos, além de compartilhar os resultados obtidos coma comunidade científica.

No entanto, o problema não está necessariamente nos tipos de infraestruturas computacionais distribuídas, e sim, na forma de se compartilhar recursos (hardware, software e conteúdo), a fim de que ambientes computacionais de *e-Science* possam escalar, de maneira autônoma, dinâmica, heterogênea e transparente, para grandes configurações, sem ignorar infraestruturas computacionais pré-existentes. O gerenciamento e a transparência de acesso aos recursos em um ambiente altamente distribuído, com uma boa relação custo/desempenho, é um problema difícil, uma vez que faltam técnicas de computação distribuída que atendam os requisitos citados acima (MATTOSO, et al. 2008).

Dadas as características do problema apresentado acima, esta dissertação propõe o uso de redes *peer-to-peer* como infraestrutura computacional, devido aos diversos benefícios oferecidos por esse tipo de sistema distribuído. Segundo Milojicic *et al*. (2002), os seguintes benefícios podem ser citados: compartilhamento do custo operacional da infraestrutura de computação; aumento da oferta de recursos em sistemas de computação distribuída; agregação e interoperabilidade de recursos entre sistemas heterogêneos; e colaboração na execução de tarefas complexas, como a manipulação de grandes quantidades de dados entre as entidades componentes do sistema distribuído.

Apesar desses benefícios, a adoção de redes *peer-to-peer* para a construção de infraestruturas de ambientes distribuídos requer cuidados (AL KISWANY, et al. 2007), pois

as técnicas de computação distribuída devem tratar requisitos de escalabilidade, características de uso e, por fim, disponibilidade de recursos (hardware, software e conteúdo). Por isto, a escolha do tipo de protocolos de redes *peer-to-peer* é fundamental, pois a opção por uma rede estruturada pode necessitar de tarefas administrativas no que tange a organização da topologia de rede (AL KISWANY, et al. 2007).A escolha de uma rede não estruturada traz uma flexibilidade maior, pois este tipo de rede não depende de uma topologia específica para organizar os *peers*, mas isto tem um custo. A natureza não determinística das redes *peer-to-peer* não estruturadas pode gerar uma grande sobrecarga e/ou um tráfego excessivo de rede, ocasionando uma possível queda de desempenho (AL KISWANY, et al. 2007).

Para minimizar tais problemas, estudos apontam para o uso de algoritmos *peer-to-peer* baseados em redes sociais (LIU, ANTONOPOULOS e MACKIN 2007; CARCHIOLO et al. 2008; SEDMIDUBSKY et al. 2008; CHANG e HU 2008). Redes sociais são formadas de maneira dinâmica e com controle descentralizado, sobre infraestruturas de computação *peer-to-peer*. Elas têm sido chamadas de redes sociais *peer-to-peer*, pois os *peers* representam pessoas, e as conexões entre os mesmos representam os relacionamentos sociais entre usuários com interesses em comum.

Esta dissertação parte da premissa de que pessoas tendem a se comportar de maneira colaborativa, quando essas possuem interesses em comum, e, por isso, elas tendem a criar relacionamentos entre si. Então, redes sociais *peer-to-peer* podem ser formadas, desde que oportunidades de colaboração (por exemplo, projetos, trabalhos, estudos, entre outros) entre pesquisadores sejam criadas e publicadas em um ambiente distribuído para colaboração científica. Neste contexto, um cálculo de similaridade é realizado para mensurar o grau de similaridade entre o perfil de um pesquisador e os requisitos de uma oportunidade. Se houver compatibilidade, uma rede social *peer-to-peer* pode, então, ser formada entre dois ou mais pesquisadores. Caso contrário, a oportunidade de colaboração é descartada pelo sistema distribuído de colaboração científica. Como conseqüência da formação de uma rede social *peer-to-peer*, seus participantes podem compartilhar recursos, a fim de transformar a rede social em uma entidade computacional, que fornecerá acesso transparente aos recursos compartilhados entre os vários componentes da rede. Desde então, hardware compartilhado, como processadores e memória, são utilizados para dar suporte à execução distribuída de tarefas. Para tanto, o mecanismo de execução de tarefas é ciente de contexto, com objetivo de identificar locais com maior disponibilidade de hardware, de modo que o processamento de dados possa ser distribuído. Por fim, o compartilhamento de um conteúdo, em uma rede social *peer-to-peer* dá-se por meio da publicação de sua disponibilidade para os membros da rede

social. Assim, se um participante da rede desejar acessar um conteúdo compartilhado, o mesmo poderá fazer o *download* deste item diretamente para o seu computador pessoal.

Fiel ao exposto acima, esta dissertação tem como objetivo contribuir na direção de novas técnicas de computação distribuída que tratem o gerenciamento e a transparência de acesso aos recursos em um ambiente de computação altamente distribuída. Para isso, é proposto um estudo sobre o uso de redes sociais *peer-to-peer* como infraestrutura para ambientes de *e-Science*, formados a partir dos interesses em comum entre pesquisadores, a fim de prover o acesso a recursos de maneira transparente e escalável, semelhante a uma nuvem de computadores (BOSS *et al.* 2007; CHAPPEL 2008). A partir deste estudo, pretende-se responder alguns questionamentos sobre esta nova abordagem para compartilhamento de recursos, apresentados a seguir. Dada uma rede *peer-to-peer*, como redes sociais *peer-to-peer* podem ser criadas, utilizando, como fio condutor dos relacionamentos sociais, os interesses comuns entre pesquisadores? Como os pesquisadores, participantes da mesma rede social, podem compartilhar o hardware de seus computadores pessoais e/ou de estruturas computacionais pre-existentes? Como software pode ser executado sobre o agregado de hardware compartilhado em uma rede social? Como pesquisadores, em uma rede social, podem compartilhar dados uns com os outros?

Para responder inicialmente os questionamentos, esta dissertação apresenta um levantamento inicial de políticas de computação distribuída, que possibilitam a criação de redes sociais *peer-to-peer*, bem como o compartilhamento de recursos no escopo destas. Para viabilizar o desenvolvimento dessas políticas, por meio de um *middleware* (MASCOLO, CAPRA e EMMERICH 2002; TANENBAUM e VAN STEEN 2007), também é proposta uma arquitetura de software cujos componentes de software foram projetados para satisfazer os requisitos impostos pelas políticas de computação distribuída.

1.1 CENÁRIO DE USO

Como já mencionado anteriormente, faltam técnicas de computação distribuída que escalem para configurações muito grandes e, ao mesmo tempo, tratem da autonomia, dinâmica e heterogeneidade dos recursos em ambientes computacionais de *e-Science* (MATTOSO, et al. 2008). Com base nisto, um exemplo de cenário de uso de redes sociais para o compartilhamento de recursos em ambientes de *e-Science* é apresentado pela Figura 1.2. Neste cenário, pesquisadores formam uma rede social sobre uma infraestrutura de computação *peer-to-peer* e, com isto, compartilham recursos de hardware (disco, processador

e memória) de maneira que os mesmo possam compartilhar conteúdos e processar grandes volumes de dados.

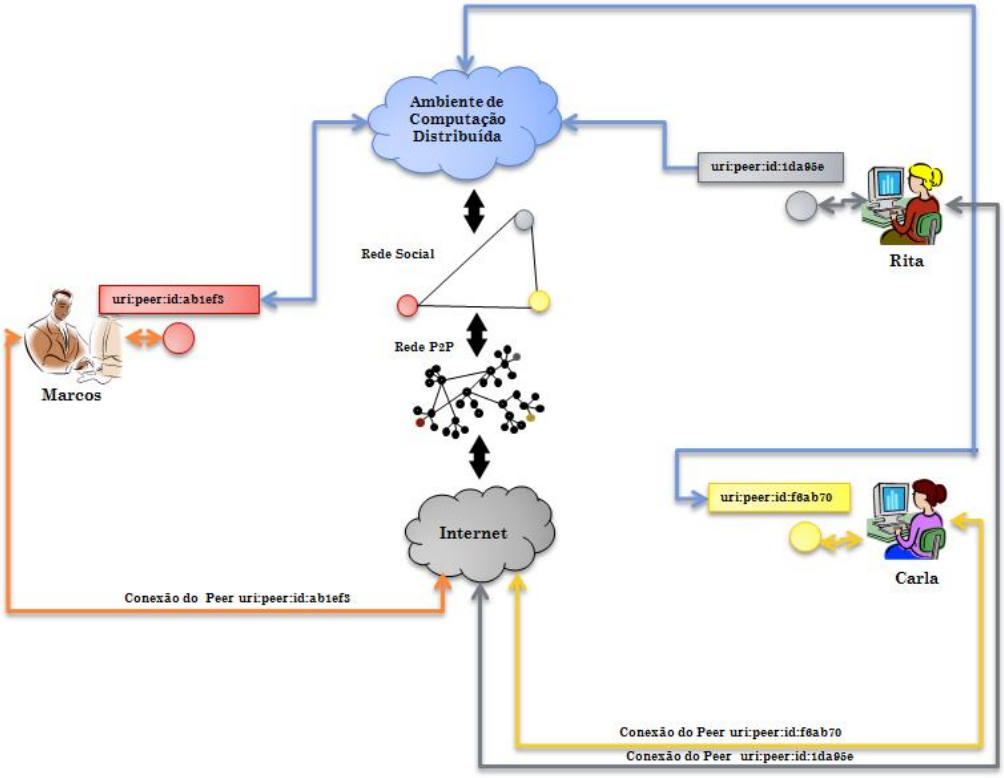


Figura 1.2: Exemplo de cenário de uso de redes sociais para o compartilhamento de recursos em ambientes de *e-Science*

Conforme a Figura 1.2, os computadores pessoais de cada pesquisador possuem conexão com a Internet. Desta forma, é possível conectar estes computadores a uma infraestrutura de computação *peer-to-peer*, fazendo com que cada computador se torne um *peer*. Isto permite que os computadores dos pesquisadores atuem como clientes e servidores ao mesmo tempo; iniciem conexões com outros computadores conectados a infraestrutura computacional *peer-to-peer* em qualquer instante de tempo; e compartilhem os recursos de hardware e/ou conteúdos existentes em seus computadores pessoais. Uma vez que recursos de hardware podem ser compartilhados em um infraestrutura de computação *peer-to-peer*, é possível agregá-los (NANDY, CARTER e FERRANTE 2005, SIMONTON, CHOI e SEIDEL 2006), para que possam utilizados para armazenamento conteúdos e execução de serviços de aplicações que necessitam de computação intensiva. Neste trabalho, o compartilhamento de recursos de hardware só é permitido aos pesquisadores, quando estes formam uma rede social sobre a infraestrutura de computação *peer-to-peer*. Por meio disto, um ambiente computacional é formado a partir dos relacionamentos entre os pesquisadores possa comportar-se semelhante a uma nuvem de computação (BOSS, et al. 2007, CHAPPEL

2009). Neste trabalho, o ambiente computacional formado pela rede social formada pelos pesquisadores é chamado de rede social *peer-to-peer* (CARCHIOLO, et al. 2008, SEDMIDUBSKY, et al. 2008, LIU, ANTONOPOULOS e MACKIN 2007).

Qualquer pesquisador, se conectado ao sistema distribuído por meio de seu computador pessoal, pode iniciar um processo de formação de rede social *peer-to-peer*. Para isto, o pesquisador deve especificar uma oportunidade de colaboração, definindo seu conjunto de interesses e, em seguida, disseminá-la no sistema distribuído. Por exemplo, um pesquisador se interessa por Bioinformática, mas isto é genérico demais e, por isso, ele deve especificar quais são as palavras-chave que delimitam o seu interesse por Bioinformática, tais como: análise genômica, DNA, genes, FASTA (FASTA 2010) e BLAST (BLAST 2010). Além disto, o usuário informa o percentual de similaridade, que é utilizado para identificar outros pesquisadores cujos interesses em seus perfis sociais sejam compatíveis com os interesses da oportunidade de colaboração. Quando uma oportunidade de colaboração é compatível com o perfil social de um pesquisador, este é notificado.

Ao receber as notificações, um pesquisador pode aceitar ou não a oportunidade publicada. Caso o usuário não a aceite, a oportunidade é descartada. No entanto, se o usuário aceitar pelo menos uma das notificações, uma rede social *peer-to-peer* é criada juntamente com o pesquisador que gerou a oportunidade de colaboração inicialmente. O aumento da rede social *peer-to-peer* se dá por meio das aceitações de notificações de compatibilidade de perfil social por parte de outros pesquisadores. Outra forma de expandir o tamanho de uma rede social *peer-to-peer* é por meio do compartilhamento de responsabilidades de publicação de oportunidades de colaboração e admissão de novos membros na rede social *peer-to-peer*. Isto permite que o a rede social *peer-to-peer* expanda, sem que o pesquisador gerador inicial da oportunidade de colaboração esteja conectado ao ambiente distribuído.

Uma vez criada uma rede social *peer-to-peer*, os pesquisadores sedem parte dos recursos de hardware disponíveis em seus computadores pessoais, a fim de formar um ambiente computacional semelhante a uma nuvem de computação, conforme apresentado pela Figura 1.1. Após isto, é possível que qualquer pesquisador, participante de uma rede social *peer-to-peer*, compartilhe conteúdos com seus demais colegas de rede social. Neste caso, cada membro da rede social, onde acontece o compartilhamento, cede parte de seus compartilhamentos de disco, de modo que o conteúdo compartilhado possa ser armazenado sobre tais espaços cedidos. Além disto, uma rede social *peer-to-peer* pode executar serviços de aplicações de maneira distribuída. Tais serviços são executados por meio de objetos móveis (tarefas) que migram para porções de recursos compartilhados cedidos por

pesquisadores em uma mesma rede social *peer-to-peer*. Assim, quando um pesquisador necessita de processar um grande volume de dados, o mesmo requisita aos seus colegas de rede social cessões de ciclos de processadores, espaço de memória compartilhado e espaço de disco compartilhado. Uma vez que as porções de recursos compartilhados tenham sido cedidas, objetos móveis migram até os computadores pessoais dos pesquisadores cedentes de recursos compartilhados e, em seguida, esperam o envio de cargas de trabalhos sejam enviadas para estas sejam processadas. Após uma carga de trabalho ter sido processada, os resultados desta são enviados até o computador pessoal do pesquisador que requisitou o processamento do serviço de aplicação. Deste modo, quando todo o volume de dados tiver sido processado, o pesquisador requisitante da execução do serviço de aplicação terá resultado deste processamento, de modo que este possa ser usado para atingir algum objetivo relacionado ao trabalho dos pesquisadores componentes da mesma rede social.

1.2 METODOLOGIA

Este trabalho, em termos metodológicos, realiza uma pesquisa exploratória, de modo que seus resultados têm como objetivo dar um novo enfoque a abordagens para o tratamento de compartilhamento de recursos (hardware, software e conteúdo) em ambientes computacionais de *e-Science*. Então, para atingir seus objetivos, a dissertação foi desenvolvida em cinco etapas, onde cada uma delas segue os procedimentos metodológicos necessários ao seu desenvolvimento, a saber: levantamento bibliográfico; especificação da arquitetura de software; especificação das políticas de computação distribuída para compartilhamento de recursos em redes sociais *peer-to-peer*, formadas sobre um ambiente computacional de *e-Science*; simulação e avaliação dos resultados das políticas de computação distribuída.

A primeira fase, o levantamento bibliográfico, tem como objetivo alcançar a fundamentação teórica, essencial para o desenvolvimento da pesquisa, que compreende desde a definição do objeto de estudo à caracterização do problema, com suas hipóteses e, por fim, uma proposta de abordagem ao problema. Essa fase compreendeu um estudo sobre o estado da prática em colaboração científica, paradigmas de computação e conceitos de redes sociais. Além disso, também foram investigados protocolos e soluções arquiteturais, importantes para a definição da abordagem proposta para o problema de compartilhamento de recursos em redes sociais *peer-to-peer*. Esses protocolos e soluções arquiteturais fazem parte dos trabalhos relacionados a esta dissertação.

Na segunda fase, foi realizada a especificação de uma arquitetura de software em camadas, onde cada componente foi concebido para satisfazer requisitos como instrumentação de hardware, comunicação *peer-to-peer*, gerenciamento de contexto, gerenciamento de compartilhamentos e gerenciamento de sessões de usuário. Já a terceira fase apresenta a principal contribuição desta dissertação, que é a proposta inicial de um conjunto de políticas de computação distribuída, a fim de contribuir na direção de uma solução para o problema de compartilhamento de recursos em ambientes computacionais de *e-Science*. Por fim, na quarta fase são apresentadas a simulação e a avaliação dos resultados das políticas de computação distribuída para o compartilhamento de recursos em redes sociais *peer-to-peer*.

A fim de evidenciar a contribuição deste trabalho junto à comunidade científica, a próxima seção apresenta os resultados esperados e as contribuições científicas desta dissertação.

1.3 RESULTADOS ESPERADOS E CONTRIBUIÇÃO CIENTÍFICA

Como resultado, este trabalho espera colaborar na direção de uma nova técnica de computação distribuída, cujo objetivo é lidar com o compartilhamento de recursos em ambientes computacionais de *e-Science*, a partir de redes sociais *peer-to-peer* formadas por pesquisadores compartilhantes de um interesse em comum. Já como contribuição, destaca-se o levantamento inicial de políticas de computação distribuída, que possibilitam a criação de um ambiente computacional para colaboração científica, com pesquisadores compartilhando hardware, software, conteúdo e processamento uns com os outros, quando esses fazem parte de uma mesma rede social. Outra contribuição é a proposta de um modelo arquitetural de software cujo objetivo é guiar o projeto de *middlewares*, possibilitando o desenvolvimento e o uso efetivo de políticas de computação distribuída na construção de ferramentas computacionais para *e-Science*.

1.4 ORGANIZAÇÃO DOS CAPÍTULOS

Os estudos e análises realizados durante o desenvolvimento desta dissertação foram organizados em 7 capítulos, como segue.

O Capítulo 2 contém um embasamento teórico sobre colaboração científica, paradigmas de computação e redes sociais. O capítulo discute como os pesquisadores têm usado a computação para realizar suas tarefas e compartilhar os seus resultados de trabalho. Além disto, são apresentados conceitos sobre o paradigma da computação distribuída, incluindo os

modelos de computação frequentemente utilizados em ambientes de *e-Science*, além do paradigma da computação orientada a serviços. Por fim, são apresentados os principais conceitos relacionados às redes sociais, no que tange às características e propriedades relacionais de uma rede social;

No Capítulo 3 são apresentados algoritmos para a construção de redes sociais *peer-to-peer*, técnicas para agregação de recursos computacionais e propostas algorítmicas e arquiteturais para utilização dos recursos computacionais compartilhados em uma rede *peer-to-peer*;

O Capítulo 4 apresenta a proposta de uma arquitetura de software para lidar com o compartilhamento de recursos em redes sociais *peer-to-peer*. Inicialmente, é apresentada a arquitetura de software. Em seguida, são apresentados, de maneira *top-down*, as camadas integrantes da arquitetura proposta. Ao longo da apresentação de cada uma das camadas, é feito um detalhamento sobre seus elementos arquiteturais internos, explicando como estes funcionam, além de seus relacionamentos, que podem ser com elementos arquiteturais da mesma camada ou com outros elementos localizados em uma camada subsequentemente abaixo.

O Capítulo 5**,** além de apresentar algoritmos relacionados à estruturação de uma rede *peer-to-peer* não estruturada, também apresenta as políticas de computação distribuída baseadas em redes sociais. Estas políticas são essenciais para a formação de redes sociais *peer-to-peer*, compartilhamento de recursos computacionais, compartilhamento e armazenamento de conteúdo sobre o agregado de recursos para armazenamento e, por fim, a para a execução de tarefas sobre os compartilhamentos de processadores e memórias;

No Capítulo 6 são apresentados detalhes sobre as configurações de simulação e cenários. Tais cenários são baseados nas políticas definidas no Capítulo 5. Além disto, também são apresentados e discutidos os resultados por meio de gráficos, que foram criados a partir dos dados gerados pelo simulador de redes *peer-to-peer*.

Por fim, o Capítulo 7 apresenta as considerações finais e contribuições, além de indicações de trabalhos futuros, visando a continuidade deste trabalho.

.

2 COLABORAÇÃO CIENTÍFICA, PARADIGMAS DA COMPUTAÇÃO E REDES SOCIAIS

Para elaboração da proposta aqui trabalhada, que é o estudo sobre o uso de redes sociais *peer-to-peer* para o compartilhamento de recursos em ambientes de *e-Science*, foi necessário realizar um estudo aprofundado sobre quais teorias podem fundamentar esse estudo. Para isso, foi estudado o estado da prática em colaboração científica, paradigmas de computação e redes sociais. No estudo do estado da prática sobre colaboração científica, foram levadas em consideração a teoria, incluindo conceituação e classificação, e a prática, apresentando como pesquisadores têm usado a computação para realizar suas tarefas e compartilhar os seus resultados de trabalho. Já o estudo sobre paradigmas de computação envolve os paradigmas da computação distribuída e da computação orientada a serviços. Sobre o paradigma da computação distribuída, são apresentados seus principais conceitos e os modelos computacionais que fundamentam este trabalho. Assim, para cada modelo computacional distribuído, são apresentados conceitos, organização arquitetural, organização física, organização lógica e requisitos para projetos de aplicações distribuídas para esses modelos. Ainda sobre os paradigmas de computação, o estudo sobre o paradigma orientado a serviços discute como ambientes distribuídos podem ser organizadas em termos de arquiteturas de software e elementos arquiteturais. Por fim, características e propriedades de redes sociais foram investigadas, no intuito de entender conceitualmente aspectos de caracterização e propriedades relacionais das redes sociais.

2.1 ESTADO DA PRÁTICA EM COLABORAÇÃO CIENTÍFICA

Segundo Sonnenwald (2007), colaboração científica pode ser definida como o comportamento humano entre dois ou mais pesquisadores, que facilita o compartilhamento e o cumprimento de tarefas. No entanto, a alocação e compartilhamento de tarefas entre pesquisadores dependem dos seguintes fatores: quantidade de tarefas, disponibilidade de recursos, interação do grupo, grau de dependência funcional entre pesquisadores e grau de dependência estratégica entre os pesquisadores.

Nesse sentido, a colaboração científica ocorre em larga escala no âmbito da ciência. As características desse âmbito são frequentemente usadas para categorizar ou classificar as colaborações. Segundo Sonnenwald (2007), tais características podem definir o foco da colaboração, a saber:

- **Foco disciplinar**: os termos colaboração intra, inter, multi e transdisciplinar enfatizam a importância do papel das áreas na colaboração científica e recorrem ao conhecimento delas, de modo que este possa ser incorporado e produzido em uma colaboração. Colaboração intradisciplinar é aquela em que os participantes têm o conhecimento da mesma área e aplicam seus conhecimentos na colaboração, produzindo novos conhecimentos na mesma área. Colaboração interdisciplinar envolve a integração dos conhecimentos de duas ou mais áreas e, tipicamente, os participantes integram seus conhecimentos para produzir novos conhecimentos. Colaboração multidisciplinar envolve o conhecimento de diferentes áreas, mas não integra e sintetiza o conhecimento. Por fim, a colaboração transdisciplinar tem sido definida como a integração de todos os conhecimentos relevantes a um problema.
- **Foco geográfico**: a localização geográfica dos pesquisadores participantes em uma colaboração fornece novos termos para classificar a colaboração científica, tais como: colaboração remota, colaboração distribuída, colaboratórios científicos e colaboração internacional. Colaboração remota e distribuída refere-se a situações em que os participantes não são alocados fisicamente no mesmo local de trabalho. Por isso, esse tipo de colaboração é definido como a separação geográfica dos pesquisadores. O termo colaboratório científico é um ambiente que fornece acesso remoto a instrumentos científicos. Segundo (SONNENWALD 2007), essa definição foi expandida, assim, colaboratórios científicos podem ser definidos como: uma facilidade baseada em rede e entidade organizacional que perpassa distâncias, apóia a interação humana orientada a uma área de pesquisa comum, estimula o contato entre pesquisadores, que conhecem ou não uns aos outros, e fornece acesso a fontes de dados, artefatos e ferramentas necessárias para o cumprimento de tarefas de pesquisa. Por fim, colaboração internacional ocorre quando participantes trabalham em diferentes países.
- **Foco organizacional e em comunidade:** aqui o foco está nos fatores que emergem das diferenças entre a academia e indústria, organizações governamentais e não governamentais, incluindo comunidades. Os termos colaboração universidade-indústria e colaboração academia-indústria são usados

> para caracterizar a colaboração entre pesquisadores da academia e pesquisadores da indústria. Esse tipo de colaboração é facilitado quando centros de pesquisas e laboratórios estão próximos ou dentro dos campi das universidades. Isto, então, promove a colaboração, transferência de conhecimento e desenvolvimento de inovação entre os pesquisadores da indústria e academia. Por fim, o termo pesquisa de ação participativa refere-se à colaboração entre pesquisadores e participantes de pesquisa em geral. O objetivo deste tipo de colaboração é criar novos conhecimentos que levem a ações sócio-efetivas, resolvendo problemas da vida real.

Atualmentea colaboração científica tem sido vista como uma atividade dependente, de maneira crítica, no que diz respeito ao acesso e compartilhamento de dados digitais de pesquisa. Outro ponto crítico na colaboração científica são as ferramentas de informação. Essas devem facilitar a estruturação dos dados, a fim de que esses possam ser armazenados, localizados, recuperados, exibidos e analisados de maneira eficiente. A partir disto, recursos de informação, codificados e arquivados, podem ser localizados prontamente e, em seguida, reutilizados para a geração de conhecimento científico confiável (DAVID 2004).

Nesse sentido, progressos, como Sistemas de Gerenciamento de *Workflows* Científicos (SGWfC) (MATTOSO, et al. 2008), têm permitido pesquisadores realizarem, de maneira quantitativa e qualitativa, a coleta, criação e inserção de grandes volumes de dados em seus experimentos científicos. Isso tem diminuído o tempo da disponibilização dos dados para análises e usos em pesquisas, além de aumentar as possibilidades práticas de integrar e transformar dados técnicos e científicos em configurações virtuais de informação, conhecimento e descoberta (DAVID 2004).

Isso tudo tem estimulado o surgimento de novas formas de colaboração distribuída e produção de conhecimento. Assim, o aumento das possibilidades e potencialidades nas pesquisas, nas áreas das ciências e engenharias, tem guiado os esforços na criação de elementos para uma infraestrutura científica global, como, por exemplo, camadas de transporte e protocolos de rede para grades computacionais; *middlewares* para *e-Science* e laboratórios virtuais.

Atualmente, todas as tecnologias, produzidas para apoiar a colaboração científica em suas mais diversas formas, têm tido como substrato abordagens computacionais cujos paradigmas utilizados são o da computação distribuída e computação orientada a serviços. Assim, de modo a permitir uma melhor compreensão sobre os paradigmas de computação

pesquisados nesta dissertação, a próxima seção apresenta um estudo sobre eles, levando em consideração seus principais conceitos e modelos arquiteturais básicos.

2.2 PARADIGMAS DA COMPUTAÇÃO

Segundo Priberam (2009), paradigma (do grego *parádeigma*) é algo que serve de exemplo geral ou de modelo. Por isto, nesta dissertação, são considerados apenas o paradigma da computação distribuída e o da computação orientada a serviços. O primeiro paradigma fornece o conhecimento necessário para, incluindo algoritmos e modelos arquiteturais, fundamentar a proposta arquitetural apresentada pelo Capítulo 4. Por isto, durante a pesquisa desta dissertação, foram investigados modelos de computação distribuída que pudessem fornecer tal aporte de conhecimento, a saber: Cliente-Servidor, *Peer-to-Peer*, Autônoma e Computação nas Nuvens. Por fim, o segundo paradigma, a computação orientada a serviços, fornece o conhecimento necessário para construção de novas aplicações distribuídas de software, cuja maneira de composição de funcionalidades se dá por meio de serviços. Assim, foram retirados da literatura apenas teorias e conceitos, sem detalhar implementações específicas para a composição de arquiteturas orientadas a serviços.

2.2.1 Computação Distribuída

A revolução ocorrida nos últimos anos nas tecnologias de informação surge como um dos principais vetores de mudança na comunicação entre as pessoas (por exemplo, pesquisadores) e na forma de trabalho. Devido à convergência dos setores de telecomunicações tradicionais e da informática, aumentou o aparecimento, a implantação e o crescimento de novos serviços de telecomunicações e aplicações de informática associadas a esses novos serviços. Além disto, o desenvolvimento de novas aplicações de software está cada vez mais voltado às infra-estruturas de telecomunicações, sejam fixas ou móveis e, com isso, essas novas aplicações tiram proveito dos serviços oferecidos, cuja característica principal é a cooperação e a distribuição. Tais características remetem ao uso de sistemas distribuídos como plataforma de computação.

Entende-se como sistema distribuído, um sistema constituído por um conjunto de componentes, distribuídos entre vários dispositivos computacionais independentes, conectados por meio de conexões de redes, que interagem uns com os outros, a fim de trocar dados e acessar serviços (TANENBAUM e VAN STEEN 2007). Esta definição de sistemas distribuídos tem dois aspectos. O primeiro trata do hardware, pois as máquinas são

autônomas. O segundo trata do software, pois o acesso aos recursos disponibilizados no sistema é transparente para os usuários que o acessam.

A definição apresentada acima se aplica tanto a sistemas distribuídos fixos quanto a sistemas distribuídos móveis. Apesar disto, existem diferenças entre esses dois tipos de sistemas que devem ser bem entendidas, pois elas influenciam na forma como o sistema distribuído será projetado e utilizado. Dessa forma, as diferenças podem ser entendidas por meio de três conceitos. São eles: o conceito do dispositivo, de conexão de rede e de contexto de execução (MASCOLO, CAPRA e EMMERICH 2002).

Quanto ao tipo de dispositivo, sistemas distribuídos fixos são compostos por dispositivos fixos, tais como: computadores pessoais, estações de trabalho e computadores de grande porte. Sistemas móveis são compostos por dispositivos móveis que variam de *notebooks* a *PDAs*. Enquanto os primeiros são em geral máquinas poderosas, com grande capacidade de memória e processadores rápidos, os últimos possuem algum tipo de limitação, como baixa velocidade de CPU, pouca memória, pouco espaço de armazenamento, limitações de bateria, entre outras características inerentes a dispositivos móveis.

Quanto ao tipo de conexão de rede, dispositivos fixos são, em geral, conectados permanentemente à rede, por meio de enlaces fixos com grande largura de banda. Assim, as desconexões, quando ocorrem, são realizadas premeditadamente por razões administrativas ou são causadas por falhas imprevistas, tais como queda de energia ou parada por defeito. Assim, essas falhas não ocorrem frequentemente e, por isso, são tratadas como exceções ao comportamento do sistema. Já os sistemas distribuídos móveis são conectados por meio de interfaces de rede sem fio, que podem obter largura de banda razoável, desde que os dispositivos móveis estejam a poucas centenas de metros de um ponto de acesso. À medida que muitos dispositivos computacionais móveis conectam-se simultaneamente, a largura de banda tende a cair rapidamente, pois tais dispositivos utilizam a mesma conexão à internet na área em que estão situados. Além disso, se o dispositivo computacional móvel se mover para uma área sem cobertura ou com alta interferência, a largura de banda pode cair a zero e a conexão pode ser perdida. A ocorrência de desconexões não programadas passa a não ser mais uma exceção, e sim, parte do comportamento esperado para sistema.

Já o conceito de tipo de contexto de execução pode ser definido como tudo o que influencia o comportamento de uma aplicação, incluindo recursos internos ao dispositivo, como a quantidade de memória, energia disponível; e recursos externos, como largura de banda, qualidade da conexão de rede, localização do dispositivo, dispositivos vizinhos, entre outros. Em um sistema distribuído fixo, o contexto é mais ou menos estático, pois a largura de

banda é alta e contínua e a localização dos dispositivos computacionais nunca muda, já que os dispositivos computacionais são adicionados ou removidos do sistema com pouca freqüência. Além disso, os serviços disponíveis podem mudar, mas a descoberta de serviços é facilmente realizada, pois os fornecedores de serviços registram-se em um serviço de localização bem conhecido pelos demais participantes do sistema.

Ao contrário dos sistemas distribuídos fixos, o contexto em sistemas distribuídos móveis é extremamente dinâmico, pois os dispositivos podem entrar ou sair do sistema com freqüência e, com isso, os serviços que estão disponíveis quando uma conexão de rede ocorre podem não estar lá quando uma nova conexão de rede ocorrer. Além disso, a busca por serviços é mais complexa, pois a localização dos dispositivos não é fixa e, dependendo de onde ele está e se há ou não movimento, a largura de banda e a qualidade de conexão podem variar muito.

A computação distribuída permite adicionar o poder computacional de diversos dispositivos interligados por uma rede de computadores, para processar colaborativamente determinada tarefa de forma coerente e transparente, isto é, como se um único computador estivesse executando a tarefa. Dessa forma, a organização das interações entre cada computador é primordial para que se possa usar um número maior de dispositivos computacionais. Além disso, os protocolos de comunicação não podem conter ou usar dados que não possam ser entendidos por certos dispositivos. Por isto, cuidados específicos devem ser tomados para que mensagens disparadas por um dispositivo sejam entregues a outro corretamente e que as mensagens inválidas sejam descartadas. Para atender a todos esses requisitos, é necessário estabelecer um modelo para que, por meio da separação das responsabilidades dos participantes do sistema distribuído e da definição de quais operações primitivas que podem ser executadas, possa-se fazer computação. Dessa forma, ao longo da evolução da computação distribuída, modelos de computação foram criados, definindo tipos de entidades de sistemas distribuídos e como essas interagem entre si, a fim de trocar dados e acessar serviços umas das outras.

As próximas seções apresentam os principais modelos de computação distribuída. Além disso, elas apresentam como cada modelo de computação distribuída contribui para este trabalho.

2.2.1.1 Computação cliente-servidor

O modelo de computação Cliente-Servidor é composto de duas partes lógicas. Dessa forma, ele descreve o relacionamento entre essas partes, que são processos executando em computadores interligados por meio de conexões de rede. Tais partes lógicas, então, são chamadas de cliente e servidor. O cliente é um processo que requisita os serviços disponíveis na outra parte lógica, o servidor. O servidor espera por requisições feitas pelos vários clientes que o acessam. Por isso, quando o servidor recebe as requisições, ele as trata e, logo após, devolve uma resposta para a origem da requisição. A interação entre o cliente e o servidor é conhecida como comportamento requisição-resposta, que é mostrada na Figura 2.1.

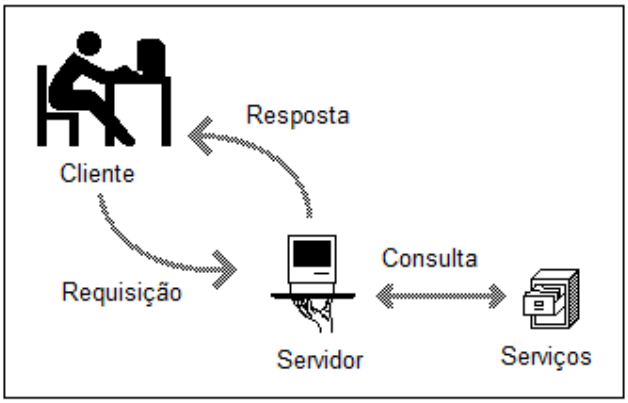


Figura 2.1: Modelo de Computação Cliente-Servidor

Segundo Tanenbaum e van Steen (2007), as aplicações construídas com base no modelo de computação cliente-servidor, em sua maioria, seguem um modelo arquitetural em camadas, que divide uma aplicação em três camadas. São elas:

- **Camada de interface de usuário:** contém um conjunto de elementos necessários para garantir a acessibilidade e a usabilidade da aplicação por parte do usuário. Além disso, é nessa camada que a lógica de uma aplicação é desenvolvida, pois o tratamento das requisições do usuário é realizado por meio de controladores, que são acionados quando o usuário interage com os controles da interface gráfica. Apesar disso, a interface gráfica com o usuário pode ser uma tela baseada em caracteres, dispensando assim um conjunto de componentes de interface gráfica, tais como janelas, caixas de diálogos, botões, menus, lista de seleção, entre outros.
- **Camada de aplicação**: contém a lógica de domínio da aplicação. Nessa camada, são implementados os códigos necessários para que a aplicação atenda aos requisitos impostos por um domínio específico. Além disso, a camada de aplicação é responsável, de maneira geral, por todo o processamento de uma aplicação.

Assim, ela é capaz de recuperar qualquer informação da camada abaixo, processá-la de acordo com a necessidade do usuário e, por fim, devolver os dados alterados para a camada abaixo.

- **Camada de dados:** gerencia os dados sobre os quais está sendo executada alguma ação. Uma importante propriedade dessa camada é que os dados costumam ser persistentes, isto é, ainda que nenhuma aplicação esteja em execução, os dados estarão armazenados em algum lugar e em algum formato para a próxima utilização. Em uma forma mais simples, por exemplo, a camada de dados é formada por um sistema de arquivos. No modelo de computação cliente-servidor, a camada de dados é implementada no lado servidor.

Fisicamente, o modelo de computação Cliente-Servidor pode ser dividido conforme a Figura 2.2. Uma máquina cliente contém apenas o software que implementa a camada de interface gráfica com o usuário. Já a máquina servidor contém as implementações da camada de aplicação e da camada de dados.

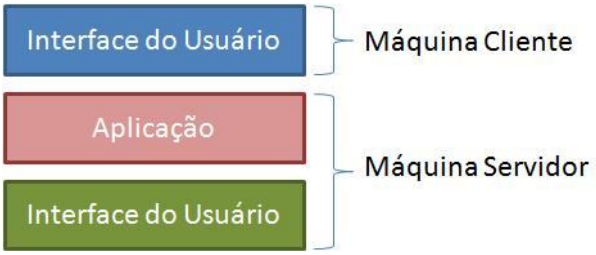


Figura 2.2: Arquitetura de software de sistemas cliente-servidor (TANENBAUM e VAN STEEN 2007)

Por fim, estruturalmente, um modelo de computação cliente-servidor pode seguir uma estrutura plana, onde todos os clientes se comunicam com um único servidor. Além dessa estrutura, o modelo discutido nesta seção pode seguir uma estrutura hierárquica, onde os servidores de um nível atuam como clientes de servidores do nível acima, com o objetivo de melhorar a escalabilidade.

2.2.1.2 Computação peer-to-peer

Com o surgimento das aplicações de compartilhamento de arquivos como Gnutella (GNUTELLA 2009), o modelo de computação *peer-to-peer* (P2P) passou a ser visto pela comunidade científica como uma alternativa ao modelo de computação cliente-servidor. Tal afirmativa baseia-se em algumas propriedades encontradas em sistemas P2P, a saber: descentralização, escalabilidade, balanceamento de carga, manutenção dinâmica, tolerância a

falha e autoestabilização (MILOJICIC 2002). Ao contrario do modelo de computação Cliente-Servidor, onde os serviços e recursos computacionais (potência computacional, dados e banda de comunicação) são ofertados por uma única entidade, o modelo de computação P2P permite que serviços e recursos computacionais sejam ofertados por um número muito grande de entidades, que são chamadas de *peers*.

Neste sentido, entende-se que o modelo de computação P2P é aquele que não possui qualquer controle centralizado ou organização hierárquica, e que todos os *peers* do sistema distribuído têm as mesmas capacidades. Assim, todos os *peers* podem em qualquer momento iniciar uma sessão de comunicação com o outro *peer*, a fim de trocar dados e acessar serviços.

Como qualquer outro modelo de computação, o modelo P2P possui características próprias. De acordo com Milojicic et al. (2002), as características são:

- **Compartilhamento de custo:** sistemas cliente-servidor servem a muitos usuários simultaneamente. Neste caso, os recursos ofertados em um servidor são compartilhados para todos os usuários que tiram proveito deles. Entretanto, à medida que a quantidade de usuários cresce, deve-se aumentar proporcionalmente a oferta de recursos. Caso a disponibilidade de recursos não seja satisfatória em relação ao crescimento de usuários, o sistema terá problemas de escalabilidade. Desta forma, quando o custo para escalar um sistema torna-se muito alto, o modelo de computação P2P pode ajudar a espalhá-lo sobre os *peers* de um sistema.
- **Escalabilidade:** com a perda de uma autoridade central, os *peers* formam uma estrutura onde cada um deles compartilha recursos com os demais integrantes da estrutura. Com isso, à medida que mais *peers* se conectam uns aos outros, a disponibilidade de recursos aumenta. Dessa forma, um sistema distribuído não depende do compartilhamento de recursos de uma única entidade, e sim, de um conjunto amplo de entidades com autonomia para compartilhar recursos entre elas.
- **Agregação de recursos**: uma abordagem descentralizada proporciona naturalmente a agregação de recursos. Aplicações que se beneficiam de uma quantia enorme de recursos, como plataformas para simulação de sistemas ou sistemas de arquivos distribuídos, naturalmente necessitam de uma estrutura que agregue suficientemente tais recursos para resolução de problemas complexos.
- **Autonomia:** em muitos casos, usuários de um sistema distribuído não confiam em qualquer fornecedor de serviços centralizados, pois eles preferem que todos os seus dados e trabalhos sejam realizados localmente. Sistemas P2P apóiam esse

nível de autonomia, pelo fato do dispositivo computacional fazer todo o trabalho para o seu usuário.

- **Anonimato e privacidade**: o usuário pode não querer qualquer um ou algum fornecedor de serviços. Com um servidor central, é difícil assegurar isso, pois tipicamente os servidores identificam os clientes que estão conectados a eles. Por meio de uma estrutura P2P, as atividades são realizadas localmente e, se quiserem, os usuários poderão fornecer suas informações para os demais usuários do sistema.
- **Dinamismo**: sistemas P2P são altamente dinâmicos. Quando uma aplicação é planejada para um ambiente com tal característica, deve ser levada em consideração a freqüência de entrada e saída de *peers*, pois a disponibilidade de recursos e serviços pode variar com isso. Consequentemente, sistemas P2P devem se adaptar às mudanças ocorridas em sua estrutura e, por isso, devem ser capazes de redistribuir tarefas para *peers* ativos e que possam processá-las, para assegurar que uma requisição para utilização de um serviço ou recurso não seja perdida, quando algum *peer* se desconectar do sistema.
- **Comunicação ad-hoc e colaboração:** sistemas P2P ajustam-se melhor do que sistemas cliente-servidor em ambientes onde as conexões e desconexões de participantes são constantes, porque eles levam em consideração as mudanças em sua infra-estrutura, uma vez que esta não é estável. Dessa forma, um *peer* conecta-se espontaneamente em qualquer outro *peer* e, a partir disso, eles passam a colaborar com o sistema, entregando mensagens para seus destinatários ou, se for necessário, enviando mensagens para outros *peers* que conheçam os destinatários delas.

Em computação P2P, as aplicações são construídas a partir de arquiteturas de software em camadas, que encapsulam os detalhes de baixo nível e, com isto, oferecem operações específicas em cada nível arquitetural. Segundo Molojicic et al. (2002), uma arquitetura P2P pode ser composta de cinco camadas: comunicação, gerenciamento de grupo, robustez, classe específica e aplicação específica. Além disso, para cada camada, Molojicic et al. (2002) propõem funcionalidades para lidar com as responsabilidades específicas do nível arquitetural correspondente. Toda essa organização é apresentada na Figura 2.3.

Figura 2.3: Arquitetura de software para sistemas distribuídos P2P (MILOJICIC 2002)

Com base no modelo exposto acima, segue a explicação sucinta de cada camada e seus respectivos componentes:

- **Camada de comunicação:** tem como objetivo tratar de maneira transparente o acesso aos tipos de conexões de rede, sejam elas fixas ou móveis. Para isso, o componente de comunicação deve cobrir um amplo espectro de tipos de comunicação. Em virtude disto, esse espectro é formado por máquinas *desktops* com conexões de alta velocidade e por pequenos dispositivos com conexão sem fio, como PDAs e *smartphones*. O componente de comunicação também deve ser capaz de tratar problemas relacionados à natureza instável de um sistema P2P e, por isso, ele deve ser responsável por conectar os *peers* ao sistema, para que uma infra-estrutura possa ser criada.
- **Camada de gerenciamento de grupo:** essa camada tem a responsabilidade de organizar logicamente os *peers* em um sistema P2P. Uma vez que a camada de comunicação interliga os diversos *peers* por meio das conexões de rede de seus dispositivos computacionais, a camada de gerenciamento de grupo define uma topologia lógica sobre a infra-estrutura mapeada pela camada de comunicação. Para isso, podem ser utilizados dois tipos de abordagens, que são: organização estruturada ou organização não estruturada. Para organização estruturada, podem ser utilizados protocolos ou algoritmos como Chord (STOICA, et al. 2001), CAN (RATNASAMY, et al. 2001), Pastry (DRUSCHEL and RWOSTROW 2001) e Tapestry (ZHAO, KUBIATOWICZ and JOSEPH 2001). Já para a organização

não estruturada, opta-se pela utilização do protocolo Gnutella (GNUTELLA 2009). Por fim, os componentes descoberta e localização e roteamento seguem as especificações das primitivas definidas nos protocolos ou algoritmos citados anteriormente.

- **Camada de robustez:** tem a responsabilidade de manter a integridade do sistema P2P, fazendo com que aplicações P2P executem em ambiente seguro, capaz de agregar recursos e ser tolerante a falhas. Para isto, existem três componentes essenciais para manter a robustez de um sistema P2P: segurança, agregação de recursos e confiabilidade. A segurança é um dos grandes desafios para sistemas P2P. Apesar do modelo de computação P2P permitir que *peers* funcionem como clientes e servidores simultaneamente, esse modelo insere alguns riscos ao sistema (SIT and MORRIS 2002). Além da segurança, a utilização de recursos compartilhados também é muito importante. Por isto, o componente de agregação de recursos é responsável por angariar uma grande variedade de recursos compartilhados por diversos *peers* em um sistema P2P. Dessa forma, mecanismos para aumentar a confiabilidade do sistema distribuído podem ser implementados, a fim de atender às funcionalidades do componente de confiabilidade. Por exemplo, em aplicações de computação intensiva, tarefas podem ser reiniciadas em outras máquinas. De forma alternada, a mesma tarefa pode ser inicialmente associada a múltiplos *peers* (OGASAWARA, et al. 2009). Em aplicações de compartilhamento de arquivos, os dados podem ser replicados através de muitos *peers* (AKBARINIA, PACITTI e VALDURIEZ 2007). Finalmente, em aplicações de mensagens, mensagens perdidas podem ser reenviadas ou podem ser enviadas ao longo de muitos caminhos simultaneamente
- **Camada de classe específica:** enquanto os componentes apresentados acima são aplicáveis a qualquer arquitetura P2P, funcionalidades abstratas de componentes de aplicações específicas são disponibilizados para cada classe de aplicação P2P. Escalonamento aplica-se às aplicações paralelizáveis e de computação intensiva. As tarefas de computação intensiva são quebradas em partes, que são escalonadas através dos *peers.* Metadados aplicam-se às aplicações de gerenciamento de conteúdo e de arquivo. Os metadados descrevem o conteúdo armazenado pelos peers e podem ser consultados para determinar a localização de uma informação desejada. Mecanismo de mensagens aplica-se às aplicações colaborativas. As

mensagens enviadas permitem a comunicação entre os *peers.* Por fim, o gerenciamento oferece mecanismos de controle da infra-estrutura P2P.

- **Aplicação específica:** ferramentas, aplicações e serviços implementam as funcionalidades de aplicação específica, que corresponde a aplicações P2P executando sobre uma infra-estrutura P2P. Casos específicos de escalonamento distribuído, compartilhamento de conteúdo e arquivo ou aplicações executando sobre sistemas colaborativos e de comunicação são exemplos de aplicações implementadas sobre uma infra-estrutura P2P.

Como mencionado anteriormente, no modelo de computação P2P os *peers* que constituem um sistema P2P são todos iguais, isto é, as funções que precisam ser realizadas no sistema distribuído estão presentes em todos os *peers* do sistema. Como conseqüência, grande parte da interação entre os *peers* é simétrica, ou seja, cada *peer* age como um cliente e um servidor ao mesmo tempo. Dado este comportamento, arquiteturas P2P desenvolvem-se em torno da questão de como organizar uma rede de sobreposição. Na literatura, existem dois tipos de redes de sobreposição, aquelas que são organizadas de maneira estruturada e outras que são organizadas de maneira não estruturada (TANENBAUM e VAN STEEN 2007)

Os sistemas P2P criados a partir de redes de sobreposição estruturadas utilizam algum procedimento para estruturação lógica dos *peers*. Com base nisso, os *peers* formam uma estrutura que respeita algum tipo de topologia. De acordo com Risson e Moors (2004), as principais topologias para sistemas P2P são: Plaxton *Tree*, Anel, Borboleta, Torus e *de Bruijn Graph.* Com base nessas topologias, algoritmos P2P são construídos a fim de formar uma rede de sobreposição, localizar elementos nessas redes e rotear mensagens entre os diversos *peers* constituintes de tal estrutura.

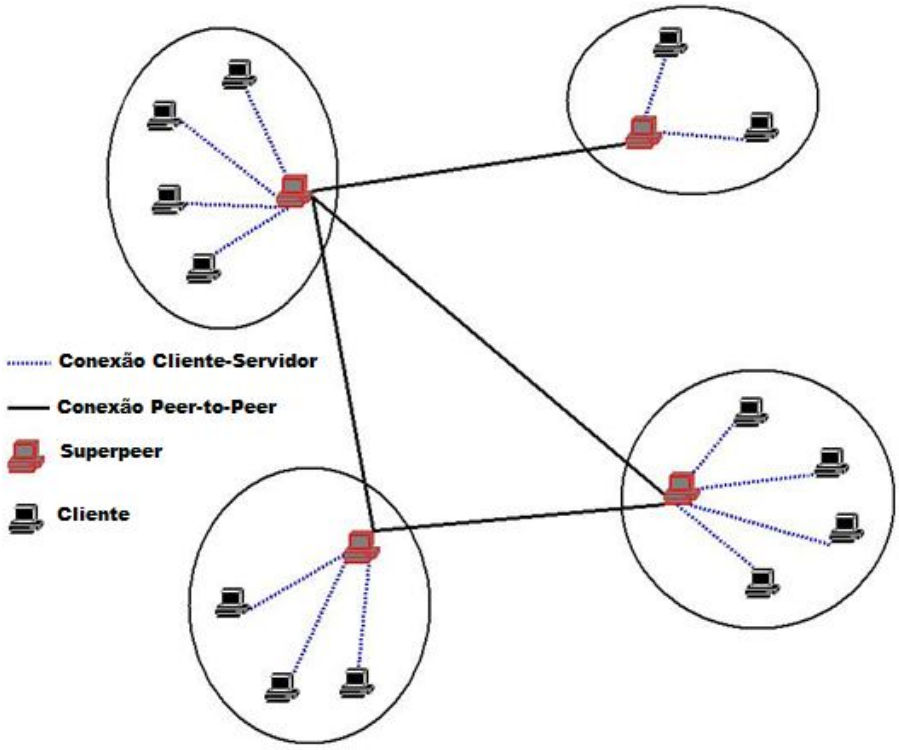


Figura 2.4: Rede de sobreposição baseada em *superpeers*

Os sistemas P2P criados a partir de redes de sobreposição não estruturadas dependem, em grande parte, de algoritmos aleatórios para construção da rede de sobreposição e, por isso, as redes de sobreposição são parecidas com grafos aleatórios. Desta forma, cada *peer* mantém uma lista com *c* vizinhos, na qual, de preferência, cada um dos vizinhos representa um dispositivo ativo escolhido aleatoriamente no conjunto de dispositivos conectados ao sistema distribuído. Apesar disto, localizar itens de dados em redes de sobreposição não estruturadas fica mais difícil à medida que a rede cresce. A razão disto é simples: como não existe nenhum modo determinístico para rotear uma requisição de consulta até um item de dado específico, a única técnica à qual um *peer* pode recorrer é enviar a requisição para todos os outros *peers*. Por isso, muitos sistemas P2P não estruturados usam *peers* especiais, com objetivo de manter um índice de dados ou atuar como intermediários. Tais *peers* são denominados *superpeers* e são organizados de forma hierárquica (SEMENOV 2005), conforme apresentado pela Figura 2.4.

2.2.1.3 Computação nas nuvens

De acordo com Boss et al. (2007), computação nas nuvens é um termo para descrever uma plataforma e tipo de aplicação. Assim, uma plataforma de computação nas

nuvens pode ser entendida como uma infra-estrutura de software e hardware que dinamicamente conecta e desconecta servidores, tanto físicos como virtuais, a fim de atender uma demanda computacional. Tal plataforma é disponibilizada na infra-estrutura da Internet como uma entidade computacional chamada de nuvem. Com isto, aplicações construídas sob o modelo de Computação nas Nuvens são aplicações *Web* que usam os recursos de uma nuvem como meio de hospedagem física e execução de suas funcionalidades. Essas aplicações têm suas funcionalidades definidas por meio de serviços *Web*.

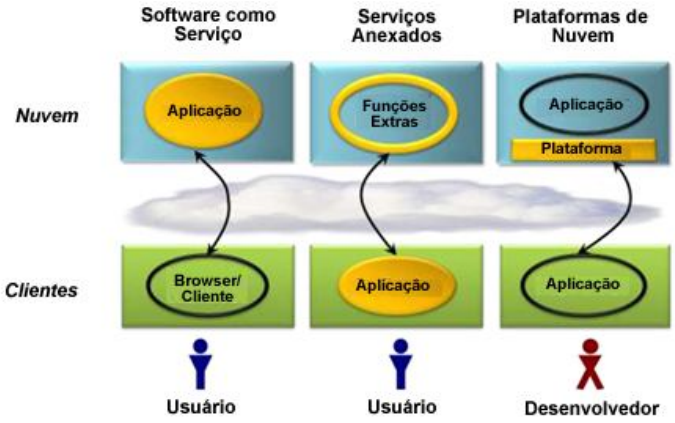


Figura 2.5: Categoria de serviços de nuvens (CHAPPEL 2009)

Atualmente, os serviços de uma nuvem podem ser agrupados em três grandes categorias (CHAPPEL 2009), a saber: software como serviço, serviços anexados e plataformas de nuvem. A Figura 2.5 mostra as categorias citadas e os relacionamentos com seus respectivos usuários diretos.

Em computação nas nuvens, uma aplicação de software como serviço executa inteiramente na nuvem. Esta é usada por usuários comuns, por meio de *browsers*, ou algum outro software cliente simples. Os serviços anexados são serviços específicos oferecidos pela infra-estrutura da nuvem que podem ser anexados a uma aplicação de usuário. Já as plataformas de nuvens fornecem serviços baseados na nuvem para criação de aplicações. Por exemplo, em vez de desenvolvedores construírem suas próprias arquiteturas, eles poderiam criar aplicações de software como serviço a partir de uma plataforma. A partir desta, então, cada desenvolvedor pode tirar proveito de componentes específicos para acesso às funcionalidades e serviços da nuvem.

Para que um software seja disponibilizado como um serviço em uma nuvem, essa deve oferecer uma plataforma de aplicação. De acordo com Chappel (2008), uma plataforma de aplicação é compreendida basicamente por três partes, que são: estrutura básica, serviços

de infra-estrutura e serviços de aplicação. O relacionamento entre essas partes e uma aplicação é apresentado por meio do modelo da Figura 2.6.

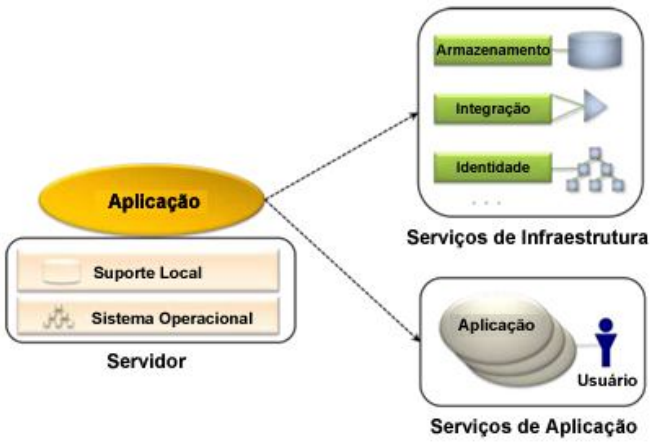


Figura 2.6: Modelo de plataforma de aplicação (CHAPPEL 2009)

A estrutura básica do modelo, apresentado pela Figura 2.6, compreende os recursos disponíveis no computador em que a aplicação executa, tais como bibliotecas padrão (.Net *framework* e Java), bancos de dados (Oracle DBMS, MySQL, Microsoft SQL Server e IBM DB2) e um sistema operacional (Windows, Linux e outras versões de Unix). Por meio da estrutura básica, uma aplicação pode acessar os serviços de infra-estrutura da nuvem (armazenamento remoto, integração, identificação, entre outros), bem como acessar serviços de outras aplicações, uma vez que essas estão tornando-se cada vez mais orientada a serviços (CHAPPEL 2009).

2.2.1.4 Computação autônoma

Os avanços na tecnologia de computação e redes de computadores, bem como em ferramentas de software têm causado uma explosão no número de aplicações conectadas a redes de computadores e a serviços de informação, presentes em todas as áreas de conhecimento e dos setores produtivos. Neste número, destaca-se um conjunto de aplicações e serviços extremamente complexos, heterogêneos e dinâmicos. Além disto, infraestruturas computacionais agregam grandes números de recursos de hardware, software e dados. Em tais infraestruturas, complexidade, heterogeneidade e dinamismos são características inerentes. Assim, a combinação entre aplicações e serviços, e infraestruturas computacionais complexas, heterogêneas e dinâmicas, tem aumentado consideravelmente a complexidade em atividades de gerenciamento, configuração e desenvolvimento de software. Por isto, a comunidade acadêmica tem investigado um novo modelo de computação distribuída, conhecido como

computação autônoma, para tratar, em ambientes computacionais distribuídos, problemas de escala, complexidade, heterogeneidade e incerteza em ambientes computacionais (KEPHART e CHESS 2003; PARASHAR e HARIRI 2005).

Dados os requisitos acima, o modelo de computação autônoma é aquele em que sistemas de software e computadores podem se gerenciar, à medida que mudanças de contexto afetam os seus comportamentos. Entende-se como contexto tudo o que influencia o comportamento de um sistema, incluindo recursos internos ao mesmo, como a quantidade de memória, energia disponível; e recursos externos, como largura de banda, qualidade da conexão de rede, localização do dispositivo, dispositivos vizinhos, entre outros (EL-KHATIB, et al. 2004). Assim, a partir das mudanças de contexto, mecanismos, presentes em uma arquitetura de computação autônoma, reagem, fazendo com que o sistema ou aplicação se recupere de uma falha e, em seguida, reconfigure-se, de modo a buscar um estado de equilíbrio, baseando-se em condições do ambiente de execução.

Aplicações e sistemas autônomos são compostos por elementos autônomos e são capazes de gerenciar seus comportamentos e relacionamentos com outros sistemas e aplicações, em concordância com as preferências de seus usuários (PARASHAR e HARIRI 2005). Assim, um elemento autônomo é um elemento arquitetural, podendo ser um software autocontido ou um módulo de sistema, com interfaces de entrada/saída e dependência de contextos específicos (PARASHAR e HARIRI 2005). Além disto, elementos autônomos também têm mecanismos para seu próprio gerenciamento, com responsabilidades de executar suas funcionalidades; exportar restrições; gerenciar seus comportamentos de acordo com o contexto e regras; e interagir com outros elementos.

Um elemento autônomo, segundo Parashar e Hariri (2005) é divido em três partes, são elas:

- **Elemento gerenciado:** é a menor parte funcional da aplicação e contém o código executável. Durante o seu tempo de execução, um elemento gerenciado pode ser afetado de diversas formas, por exemplo, falhas, indisponibilidades de recursos computacionais, baixo desempenho do ambiente de execução, entre outras.
- **Ambiente:** representa todos os fatores que podem impactar no elemento gerenciado. Qualquer mudança no ambiente faz com que uma aplicação ou sistema saia de um estado estável para um estado instável. Isso faz com que o elemento gerenciado reaja às mudanças de ambiente, de modo que a aplicação ou sistema volte a um estado estável. As mudanças de ambiente podem ser internas, quando

refletem o estado da aplicação ou sistema, ou externas, quando refletem o estado do ambiente de execução.

- **Controle:** cada elemento autônomo tem seu próprio gerenciador, que: aceita requisitos especificados pelo usuário (por exemplo: desempenho, tolerância à falha, segurança, entre outros); monitora o estado do elemento; captura o contexto do sistema ou aplicação; determina o estado do ambiente (estável ou instável); usa informações de contexto para controlar a operação do elemento gerenciado.

2.2.2 Computação Orientada a Serviços

Atualmente, a computação orientada a serviços e arquiteturas orientadas a serviços têm sido amplamente pesquisadas pela comunidade de engenharia de software e sistemas distribuídos, pois elas são as abordagens mais recentes para facilitar o projeto e desenvolvimento de aplicações em sistemas distribuídos. Por isso, o paradigma da computação orientada a serviços refere-se a um conjunto de conceitos, princípios e métodos, cuja finalidade é organizar a computação, utilizando arquiteturas orientadas a serviços. Estas, por sua vez, são compostas por componentes com interfaces bem definidas e padronizadas, chamados serviços. Os serviços podem ser vistos como elementos arquiteturais que encapsulam desde um simples método até um processo complexo (PAPAZOGLOU e GEORGAKOPOULOS 2003).

Segundo Papazoglou e Georgakopolous (2003), serviços são componentes abertos e autodescritivos que permitem a composição rápida e de baixo custo de aplicações distribuídas. Por isso, tais elementos arquiteturais devem utilizar um protocolo padrão e aberto para comunicação, como HTTP (*HyperText Transfer Protocol*), para que possam compor arquiteturas que cooperem entre si, sem depender de fornecedores de infra-estruturas de computação orientada a serviços. Serviços possuem informações que são publicadas e, posteriormente, utilizadas na descoberta, seleção, ligação e composição dos mesmos. As informações sobre serviços descrevem suas capacidades, interface, comportamento e qualidade (PAPAZOGLOU e GEORGAKOPOULOS 2003).

Apesar de arquiteturas orientadas a serviços serem frequentemente utilizadas para construção de aplicações *web* de grande porte, a tecnologia não se resume unicamente nisso. Isso se dá devido ao fato de serviços *web* serem a tecnologia de maior destaque para implementação de arquiteturas orientadas a serviços, que para tal fim não é necessariamente

obrigatória. Neste caso, existem outras tecnologias que podem ser utilizadas para implementar arquiteturas orientadas a serviços, como Jini (JINI 2009) e JXTA (JXTA 2010).

Figura 2.7 –Visão Simplificada de uma Arquitetura Orientada a Serviços

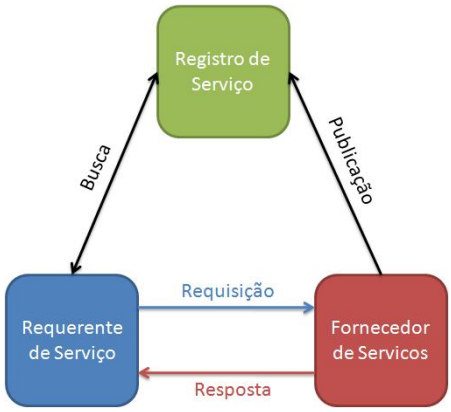


Figura 2.8 – Arquitetura Base para Computação Orientada a Serviços

Em uma arquitetura orientada a serviços, de maneira resumida, um requerente de serviço envia uma mensagem para um fornecedor de serviços. Este, por sua vez, envia uma resposta, podendo ser uma informação requisitada ou a confirmação de alguma ação realizada pelo fornecedor de serviços, conforme ilustra a Figura 2.7.

Para possibilitar o funcionamento de uma arquitetura orientada a serviços, devem existir os mecanismos de registro de serviços e descoberta de serviços, que resultam em três papéis bem definidos na arquitetura, a saber:

- **Fornecedor de serviços:** fornece serviços acessíveis por meio de uma rede de computadores e é responsável pelo gerenciamento, manutenção, publicação dos serviços, entre outros.
- **Cliente de serviços:** usa serviços de um fornecedor de serviços e, quando necessário, para conhecimento das funcionalidades, localização e requisitos específicos de um serviço, pode realizar consultas a um registro de serviços.
- **Registro de serviços:** mantém os registros dos serviços disponíveis em uma rede de computadores, bem como funcionalidades, localização, protocolos utilizados pelos serviços, entre outras informações.

A interação entre as três entidades citadas acima se dá em quatro passos, conforme a Figura 2.8. Inicialmente o fornecedor publica o serviço em um registro, fornecendo informações relevantes quanto à utilização do mesmo, tais como: localização do serviço, interfaces de interação, protocolos de transporte utilizados, formato de dados requeridos e parâmetros de qualidade de serviços (PAPAZOGLOU e GEORGAKOPOULOS 2003). Em

seguida, o cliente de um serviço solicita ao registro a busca por um serviço que satisfaça certos parâmetros. O registro responde ao cliente disponibilizando-lhe uma lista com serviços que satisfazem os parâmetros recebidos. Então, baseado em algum critério de avaliação, o cliente escolhe o serviço que mais lhe convém e, em seguida, envia uma requisição ao fornecedor do serviço, que por sua vez envia o resultado de tal requisição.

2.3 REDES SOCIAIS

Segundo Gillespie e Glisson (1992), uma rede social pode ser definida como um conjunto de unidades sociais conectadas. Dessa forma, unidades sociais são organizadas por um ou mais tipos específicos de interdependência, como valores, visões, idéias, atividades, entre outros. Esses tipos específicos de interdependência ligam pessoas umas com as outras, formando um grafo, onde unidades sociais são os vértices, e os relacionamentos entre elas são as arestas.

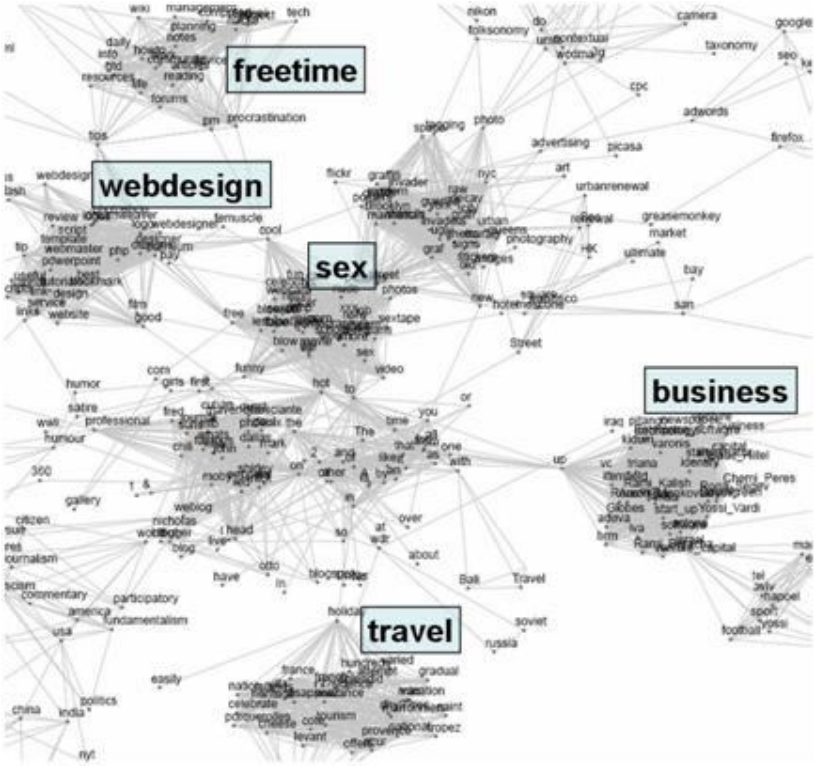


Figura 2.9. Estrutura de redes sociais Fonte bibliográfica inválida especificada.

Com base na definição acima, três aspectos podem ser destacados (GILLESPIE e GLISSON 1992). Primeiro, as redes sociais têm fronteiras, ou seja, existem critérios para determinar a associação em uma rede. Em algumas redes sociais, como família, grupos de amigos e equipes de trabalho, as fronteiras são relativamente claras e fáceis de determinar. No

entanto, redes sociais também fazem parte de grandes sistemas sociais. Assim, algumas vezes é difícil distinguir entre uma rede e seu amplo contexto social. O segundo aspecto da definição é a conectividade. Para ser parte de uma rede social, cada membro deve ter conexões atuais e potenciais com pelo menos um dos membros da rede. Tais conexões podem ser diretas ou indiretas. Enquanto alguns membros podem ser periféricos na rede ou quase completamente isolados, cada um, de certo modo, deve estar conectado a outros membros. Por fim, o terceiro elemento chave é a unidade social. Análises de rede podem ser facilmente aplicadas a um amplo conjunto de unidades sociais. Assim, essas podem ser indivíduos, no caso de redes de apoio social. Além disso, unidades sociais podem ser agências de serviços sociais, instituições sociais em comunidades locais, ou nações na economia global.

Além dos aspectos de caracterização, redes sociais possuem um conjunto de propriedades, que podem ser classificadas em duas grandes categorias: propriedades relacionais e propriedades estruturais (GILLESPIE e GLISSON 1992). As propriedades relacionais focam no conteúdo do relacionamento entre os membros da rede e na forma desses relacionamentos. Por isto, dois aspectos são estudados, são eles: conteúdo de transação e a natureza dos relacionamentos. O conteúdo de transação refere-se a tudo aquilo que flui ou é trocado nas redes. Assim, quatro tipos básicos de troca de conteúdo podem ser identificados: recursos, informação, influência e apoio social (GILLESPIE e GLISSON 1992). A natureza dos relacionamentos refere-se às qualidades inerentes entre os membros da rede.

Enquanto propriedades relacionais tratam o conteúdo das relações, propriedades estruturais descrevem a forma como os membros se unem para formar redes sociais. Nesse contexto, as propriedades estruturais podem ser divididas em três níveis de análise: membros individuais, subgrupos e redes totais (GILLESPIE e GLISSON 1992). A análise de membro individual descreve as diferenças entre as conexões deste para com os outros membros da rede. A análise de subgrupos descreve as características estruturais em uma rede total. A maioria das redes sociais contém áreas de concentração com grande quantidade de conexões entre os membros. Tais concentrações são examinadas como aglomerados (*clusters*). Por fim, as análises realizadas nesses aglomerados têm como objetivo descrever características estruturais dos subgrupos e como um determinado subgrupo se une à rede total.

2.4 CONSIDERAÇÕES FINAIS

O modelo de computação cliente-servidor é o mais conhecido e amplamente utilizado para o desenvolvimento de aplicações distribuídas, tanto que os serviços de Internet

e aplicações Web são desenvolvidos e disponibilizados com base nesse modelo. No entanto, sistemas cliente-servidor não escalam bem quando a quantidade de usuários desses sistemas cresce. Neste caso, usuários disputam por recursos computacionais e serviços ofertados por uma única entidade. Por isso, tais sistemas tendem a perder desempenho ou, até mesmo parar, quando a oferta de recurso computacional é insuficiente para atender a demanda dos usuários. Além disso, sistemas cliente-servidor necessitam de constante gerenciamento, ou seja, é necessário que um administrador possa verificar *logs*, consumo de recursos (percentual de uso do processador, espaço em disco, quantidade de memória livre, tráfego de rede, por exemplo) e também gerenciar o acesso dos usuários. Desta forma, o modelo de computação cliente-servidor torna-se caro e até mesmo difícil de manter ao longo do tempo.

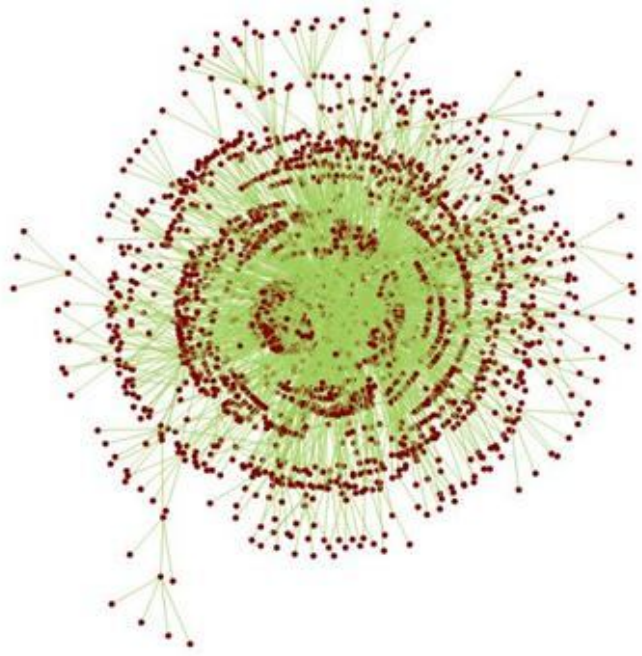


Figura 2.10: Internet como infraestrutura de rede

Nesse sentido, o modelo de computação *peer-to-peer*, em relação ao modelo de computação cliente-servidor, permite o aproveitamento dos recursos de todos os participantes do sistema. Enquanto no modelo cliente-servidor a oferta de recursos parte de um único ponto, o modelo de computação *peer-to-peer* permite que todos os integrantes do sistema compartilhem seus recursos, a fim de formar uma estrutura computacional que sirva para os mais variados propósitos. Assim, redes de sobreposição podem servir não somente para manter sistemas de compartilhamento de arquivos, mas também para manter ambientes distribuídos, onde os *peers* colaboram para a hospedagem de conteúdos e execução de software de forma cooperativa. Para isso acontecer, é necessário que os compartilhamentos de

recursos na Internet migrem do centro para a periferia da nuvem exibida na Figura 2.10, uma vez que na periferia existe um número maior de computadores que no centro. Além disto, os computadores localizados nas bordas da Internet possuem capacidade computacional maior ou igual a servidores localizados no centro da infra-estrutura da rede mundial de computadores. Entretanto, os recursos de hardware ficam ociosos, pois o modelo de computação vigente não permite a criação de uma via de sentido duplo para utilização desse potencial computacional.

Esta dissertação apropria-se dos conceitos e características estruturais do modelo de computação *peer-to-peer* para compor a camada de comunicação da arquitetura de software proposta por este trabalho. O objetivo dessa camada é compor uma infraestrutura computacional descentralizada, a partir de computadores utilizados em atividades de *e-Science*, conectados direta ou indiretamente à Internet. Com isto, recursos computacionais podem ser compartilhados e ofertados para hospedagem de conteúdo e aplicações e para execução de serviços. Uma vez que recursos tenham sido compartilhados sobre uma rede *peer-to-peer*, estes podem ser agregados a fim de compor aglomerados de recursos, de modo que possam ser utilizados para o armazenamento de conteúdo e a execução de software de forma distribuída. No entanto, aglomerados de recursos compartilhados devem ser vistos de maneira única, semelhante a uma nuvem computacional, ou seja, usuários não devem ter conhecimento sobre os detalhes de baixo nível que envolvem o compartilhamento, localização e uso dos recursos.

Finalmente, para formar agregados computacionais, esta dissertação apropria-se das dinâmicas das redes sociais no que tange a aspectos estruturais e propriedades. Com isto, as relações sociais entre os usuários de um ambiente altamente distribuído são utilizadas para a construção de redes sociais *peer-to-peer*. Assim, sobre a ótica de que pessoas tendem a se comportar de maneira colaborativa quando estas possuem interesses comuns, compartilhamento de recursos pode ser direcionado às redes sociais, a fim de formar um ambiente de computação capaz de prover acesso transparente a hardware, software e conteúdos compartilhados. Mas, para manter o comportamento de uma rede social *peer-to-peer*, semelhante a uma nuvem, é necessário utilizar políticas de computação distribuída que enderecem requisitos de computação autônoma, são eles: autonomia, heterogeneidade, complexidade e incerteza. Portanto, o software disponibilizado em uma rede social *peer-to-peer* não deve ser diferente quando se trata do desenvolvimento e disponibilização do mesmo em uma rede *peer-to-peer*. Por isso, as funcionalidades de uma aplicação distribuída devem ser disponibilizadas na forma de serviços.

O levantamento bibliográfico apresentado neste capítulo auxiliou a compreensão das teorias fundamentadoras desta dissertação. Portanto, Todo esse referencial teórico possibilitou a busca por abordagens, tanto arquiteturais quanto algorítmicas, que pudessem compor uma proposta metodológica para o compartilhamento e uso de recursos em ambientes computacionais de *e-Science*. No entanto, faz-se necessária a investigação do estado da prática, no que diz respeito ao compartilhamento de recursos em redes sociais *peer-to-peer*. Por isto, foram estudadas abordagens algorítmicas e arquiteturais que pudessem guiar esta pesquisa para a proposição de uma arquitetura de software e políticas de computação distribuída, ambos para endereçar o compartilhamento de recursos em redes sociais *peer-to-peer*. O próximo capítulo apresentaas abordagens investigadas ao longo do desenvolvimento desta dissertação.

3 COMPARTILHAMENTO DE RECURSOS EM REDES SOCIAIS *PEER-TO-PEER*

O uso de ferramentas avançadas para computação de alto desempenho (por exemplo, *clusters* e grades computacionais), e de sistemas de gerenciamento de *workflows* tem revolucionado as atividades de pesquisa em diversos domínios como química, física, biologia, medicina, entre outros. Nesse cenário, pesquisadores definem suas hipóteses e criam experimentos com base em um conjunto de algoritmos e programas que operam em ambientes de computação distribuída, permitindo, então, a geração, coleta e armazenamento de um grande montante de dados experimentais. Porém, organizar esses dados em um repositório ao qual pesquisadores possam ter acesso a qualquer hora e de qualquer local requer atividades especializadas, tais como as exercidas por administradores de infraestruturas computacionais, ou a utilização de ferramentas computacionais para lidar com grandes volumes de dados. Neste sentido, tais ferramentas devem permitir a distribuição de dados e a execução de software sobre recursos computacionais em um ambiente altamente distribuído.

No entanto, o gerenciamento e a transparência de acesso aos recursos computacionais em um ambiente altamente distribuído é, com uma boa relação custo/desempenho, um problema difícil (MATTOSO, et al. 2008). Além disso, é possível observar a falta de técnicas de computação distribuída que escalem para configurações muito grandes, e, ao mesmo tempo, tratem da autonomia, dinâmica e heterogeneidade dos recursos (ex., processadores, memória, discos, conteúdo, software, etc.) (MATTOSO, et al. 2008).

Por esses motivos, a investigação de técnicas e arquiteturas de software para computação distribuída é fundamental, dado que pesquisadores necessitam realizar experimentos e compartilhar os resultados destes com outros pesquisadores. Além disso, a infraestrutura computacional, onde experimentos são executados e dados são gerados, deve ser transparente para os pesquisadores, ou seja, a infraestrutura deve ser vista como um sistema único. As infraestruturas de *e-Science* devem facilitar a colaboração científica de modo que grupos de pesquisadores possam obter rapidamente resultados, sem precisar de longas esperas para obtenção de equipamentos ou de disponibilização de serviços de computação distribuída.

Nesse sentido, esta dissertação apresenta um estudo sobre o uso de redes sociais *peer-to-peer* para o compartilhamento de recursos computacionais em ambientes de *e-Science*. Antes disso, na Seção 3.1, serão apresentadas metodologias para a construção de redes sociais *peer-to-peer*. Técnicas para agregação de recursos computacionais, fundamentais para esta dissertação, serão apresentadas na Seção 3.2. Após a apresentação das metodologias para organização de redes sociais *peer-to-peer* e agregação de recursos computacionais, serão apresentadas, na Seção 3.3 e 3.4 propostas metodológicas e arquiteturais para utilização dos recursos computacionais compartilhados em uma rede *peer-to-peer*.

3.1 REDES SOCIAIS *PEER-TO-PEER*

A adoção de redes *peer-to-peer* para a construção de infraestruturas de ambientes distribuídos requer cuidados (AL KISWANY, et al. 2007), pois as técnicas de computação distribuída devem tratar requisitos de escalabilidade, características de uso e, por fim, disponibilidade de recursos (hardware, software e conteúdo). Além disso, as técnicas também devem ser capazes de permitir a criação e a manutenção de usuários e grupos de usuários. Para isso, a escolha do tipo de rede *peer-to-peer*, estruturadas ou não estruturadas, podem necessitar de tarefas administrativas ou gerar uma grande sobrecarga e tráfego excessivo de rede (AL KISWANY, et al. 2007).

Para minimizar tais problemas, estudos apontam para o uso de algoritmos *peer-to-peer* baseados em redes sociais (LIU, ANTONOPOULOS e MACKIN 2007; SEDMIDUBSKY *et al.* 2008). Assim, redes sociais são formadas, de maneira dinâmica e com controle descentralizado, sobre infraestruturas de computação *peer-to-peer*. Tais redes têm sido chamadas de redes *peer-to-peer* sociais, pois os *peers* representam pessoas e as conexões entre eles representam os relacionamentos sociais entre usuários com interesses em comum.

Liu, Antonopoulos e Mackin (2007) apresentam um algoritmo para descoberta de recursos em redes não estruturadas, imitando interações humanas em redes sociais. Diferente da maioria das redes *peer-to-peer* não estruturadas, nenhum overhead é necessário para obter informações adicionais de nós vizinhos. Nesta abordagem, cada *peer* conecta-se a outros *peers* com os mesmos interesses de maneira gradual, por meio dos resultados de buscas por recursos. Dessa forma, cada *peer* constrói um índice de conhecimento que armazena associações entre tópicos e outros *peers*, conforme os resultados das buscas. Se a busca é bem sucedida, o *peer* requerente atualiza seu índice de conhecimento a fim de associar os *peers*

que responderam à busca, e, em seguida, conecta-se a estes *peers*. Ao mesmo tempo, o *peer* requerente remove entradas inválidas no índice de conhecimento, conforme os resultados das buscas. Com isto, os *peers* podem aprender a partir dos resultados de buscas anteriores, permitindo que buscas futuras sejam mais focadas. Assim, quanto mais buscas são feitas, mais conhecimento pode ser coletado a partir delas. Se este processo continua, cada *peer* pode armazenar uma grande porção de conhecimento útil e utilizá-lo para fazer buscas mais rápidas no futuro.

Já Sedmidubsky et al. (2008) apresentam uma abordagem em que os *peers* são diretamente conectados uns aos outros, se eles são semanticamente similares. Cada *peer*, em particular, mantém uma lista de consultas, disparadas ou respondidas por este, chamada de histórico de consultas. O histórico representa o conhecimento do *peer* sobre a rede e é explorado pelo algoritmo proposto por Sedmidubsky et al. (2008). Cada consulta no histórico tem associado um grau de similaridade e uma lista de *peers*. O grau de similaridade corresponde à relação de compatibilidade entre um *peer* e uma consulta. Com isto, nesta abordagem, um *peer* inicia uma consulta. No histórico de consultas, ele busca por uma consulta similar àquela que está sendo realizada. Assim, o grau de similaridade é recuperado juntamente com os *peers* associados a ele. Em seguida, a consulta é passada para os *peers* associados, que realizam o mesmo procedimento realizado pelo *peer* que iniciou a consulta. Deste modo, aqueles *peers* que receberam a consulta só passam a consulta para outro *peer* caso este tenha um grau de similaridade maior com a consulta. Quando o melhor grau de similaridade é encontrado, o *peer* receptor da consulta devolve uma reposta para o *peer* que iniciou o processo.

3.2 AGREGAÇÃO DE RECURSOS EM REDES *PEER-TO-PEER*

Para transformar uma rede social criada sobre uma rede *peer-to-peer* em um ambiente de computação, é necessário que os membros de uma mesma rede social compartilhem recursos computacionais presentes em seus computadores pessoais e, se necessário, de infraestruturas computacionais. Porém, de nada valem os compartilhamentos de recursos computacionais se não existirem técnicas de computação distribuída que informem a quantidade de recursos compartilhados e suas respectivas disponibilidades. Com esse intuito, técnicas para agregação de recursos em redes *peer-to-peer* foram pesquisadas, com o objetivo de utilizá-las na composição de uma metodologia para agregação de recursos compartilhados por usuários, membros de uma rede social *peer-to-peer*. Algumas propostas de descoberta de

recursos computacionais, tais como processadores, memórias e discos já foram propostos na literatura (NANDY, CARTER e FERRANTE 2005; SIMONTON, CHOI e SEIDEL 2006).

Nandy, Carter e Ferrante (2005) apresentam um algoritmo baseado em vetores de distâncias, com objetivo de manter a distância de *peers* possuidores de um determinado recurso. Para tanto, a informação é atualizada via "*gossip*" para refletir o consumo, a adição e a exclusão de recursos no sistema. Nessa abordagem, os *peers* interagem somente com seus vizinhos imediatos e não se comunicam diretamente com qualquer outro *peer*. O algoritmo de Nandy, Carter e Ferrante (2005) é bem simples. Cada *peer* mantém uma tabela de distâncias de tamanho *K*, com o objetivo de localizar *K* diferentes tipos de recursos. Os *peers* fazem requisições para um dos *K* tipos de recursos e o algoritmo tenta localizar o *peer* mais próximo que satisfaça a requisição. Inicialmente, todos os *peers* têm suas tabelas de distâncias inicializadas, sem qualquer valor de distância para os tipos de recursos. Se um *peer* tem qualquer um dos tipos de recursos, ele configura o valor de distância para zero. Periodicamente, os nós enviam, por meio de "gossip", cópias de suas tabelas de distâncias para seus vizinhos. Quando um valor de distância é recebido por um vizinho, o mesmo atualiza a sua entrada na tabela, incrementando-a em uma unidade a partir do valor atual. Já quando um peer recebe uma requisição para um recurso, ele checa se seu valor de distância é 0, o que significa que ele tem o recurso. Caso contrário, o peer passa a requisição para um vizinho. Uma requisição que ultrapassa um *Time-to-Live*[1] (TTL) específico é descartada pelo sistema. Além disto, uma requisição pode ser também descartada se o *peer* que a recebeu não tiver o recurso e também não tiver vizinhos para o roteamento da requisição.

Simonton, Choi e Seidel (2006) apresentam um algoritmo e uma arquitetura de software como abordagens para localização de ciclos ociosos de computação, disponíveis em uma rede *peer-to-peer*. Quando se deseja localizar a disponibilidade de ciclos ociosos de computação, um *peer* gera uma consulta e a envia para um número *K* de vizinhos, escolhidos aleatoriamente. Essa consulta tem como base uma lista de destinos que foi construída a partir de mensagens de *gossip* recebidas pelo *peer*, enquanto o mesmo está ativo na rede. Antes de enviar *K* mensagens sobre a rede *peer-to-peer*, o *peer* deve anexar à mensagem o seu estado atual, em termos de ciclos de computação, e uma cópia da tabela da lista de destinos. Quando um *peer* recebe uma consulta, ele extrai as informações anexadas à mensagem e, em seguida, verifica se seu processador ainda possui disponibilidade. Se não houver disponibilidade, o *peer* roteia a consulta para outro *peer*, que é escolhido aleatoriamente. Caso contrário, o *peer*

[1] *Time to Live é* o número de saltos entre máquinas que os pacotes podem demorar numa rede de computadores antes de serem descartados (máx. 255).

reserva uma fatia de tempo com base na latência máxima da rede. Logo após, o *peer* receptor da consulta envia uma mensagem para o *peer* gerador da consulta informando que existe disponibilidade de processador.

3.3 BACKUP COLABORATIVO DE DADOS

Para compartilhar dados em uma rede social *peer-to-peer*, é necessária a definição de uma abordagem para o armazenamento colaborativo dos dados compartilhados. Esta abordagem deve permitir que pesquisadores compartilhem dados uns com os outros, desde que todos participem da mesma rede social. Na literatura existem abordagens que tratam do armazenamento colaborativo de dados, tais como a de Li e Dabek (2006) e Oliveira (2007).

Li e Dabek (2006) demonstram que redes sociais são viáveis para construção de sistemas *peer-to-peer* de backups, apresentando um modelo analítico para sistemas *peer-to-peer* onde os *peers* compartilham recursos somente entre amigos. Por isto, Li e Dabek (2006) partem do seguinte princípio: se um *peer* interagisse somente com *peers* de amigos conhecidos, os *peers* raramente deixariam o sistema de maneira não anunciada, permitindo que se diferenciassem as saídas temporárias das permanentes. Assim, ao invés de apresentar um esquema de redundância de dados para endereçar a participação intermitente de *peers*, um sistema *peer-to-peer* baseado em rede social, por sua natureza, terá uma taxa mais baixa de saídas permanentes, reduzindo a taxa de falhas a ser trata (LI e DABEK 2006). Li e Dabek (2006) avaliaram tal suposição e mostraram que a abordagem reduz a taxa de falhas e, conseqüentemente, também reduz o consumo de largura de banda. Apesar de disso, Li e Dabek (2006) não discutem como criar, evoluir e manter redes sociais; como o espaço de backup é negociado entre amigos; como garantir a recuperabilidade do backup.

Oliveira (2007) apresenta uma abordagem baseada em um índice centralizado, onde a maioria dos *peers* é igual, mas alguns *peers* têm funções avançadas e, por isso, são chamados de servidores (SEMENOV 2005). Dessa forma, o servidor é responsável pelo gerenciamento dos usuários das redes sociais e de metadados que descrevem os locais onde arquivos estão armazenados. Já os *peers*, chamados de agentes por Oliveira (2007), são responsáveis pela execução dos backups, realizando a negociação de armazenamento de cópias uns com os outros, desde que pertençam a uma rede social. Para tanto, o sistema provê mecanismos que permitem ao usuário construir sua própria rede social, fornecendo as operações básicas de manipulação à adição, remoção e consulta de amigos. Um usuário recém cadastrado possui uma rede social vazia. Todos os metadados relativos à rede social do

usuário são recuperados por meio do servidor. Esses metadados armazenam as relações de amizades estabelecidas, pendentes e removidas. Além disto, os metadados também armazenam notificações sobre o estado de uma amizade e a quantidade doada para cada amigo. Então, quando um usuário deseja realizar um backup, ele deve angariar espaço nos computadores dos amigos. A alocação de espaço de armazenamento para um amigo é um contrato de backup. O valor deste contrato é a quantidade de espaço que foi alocada. Cada usuário mantém todos os contratos de backup dos quais ele participa. No entanto, se nenhum dos contratos firmados anteriormente puder ser utilizado para cópias, o usuário pode requisitar novos contratos para backups. Na proposta de Oliveira (2007), os contratos de backup são definidos manualmente. Os usuários, por meio dos *peers*, definem a quantidade de espaço de armazenamento que será compartilhado com outros usuários. Por fim, quando o usuário deseja recuperar as cópias de backup, o sistema busca os metadados relativos ao conjunto de arquivos que se deseja recuperar. Em seguida, é consultado o estado de disponibilidade (*on-line* ou *off-line*) dos amigos que armazenam este backup. Uma vez que a localização, o estado e a disponibilidade das cópias são determinados, o sistema inicia o processo de escalonamento das operações de *download*. O escalonamento prioriza a recuperação das cópias mais recentes que estejam prontamente acessíveis, ou seja, aquelas armazenadas nos amigos *on-line.* Apesar de cobrir os pontos em aberto na abordagem proposta por Li e Dabek (2006), a abordagem de Oliveira (2007) pode gerar limites de escalabilidade, uma vez que requer servidores maiores quando o número de requisições aumenta, e mais espaço para armazenamento conforme a quantidade de usuários cresce.

3.4 MIGRAÇÃO DE SERVIÇOS

Os trabalhos já realizados sobre redes sociais *peer-to-peer*, agregação de recursos em redes *peer-to*-peer e backup colaborativo de dados, mostram que a adoção de uma abordagem baseada em serviços móveis é fundamental para realização de computação distribuída em redes sociais *peer-to-peer*. Por isto, algumas metodologias (HANDOREAN, et al. 2005) e arquiteturas de softwares (RIVA, et al. 2007) (RIVA, et al. 2007) baseadas em serviços móveis foram estudadas, objetivando a criação de uma proposta de arquitetura de software para aplicações baseadas em serviços móveis.

Handorean *et al.* (2005) apresentam uma abordagem de gerenciamento de sessões entre aplicações clientes e serviços móveis, introduzindo o conceitode *follow-me session* a arquiteturas orientadas a serviços em redes *ad-hoc*. Uma *follow-me session* é o elemento

chave da interação entre uma aplicação cliente e um serviço móvel, pois ela oculta as desconexões entre os intervalos de conectividade de dispositivos móveis. Por isto, Handorean et al. (2005) focam na interação entre um cliente e um serviço e não em um fornecedor específico de serviços ou processo. Isso permite a uma aplicação trocar seu fornecedor de serviços, se necessário, em tempo de execução. Para tanto, Handorean et al. (2005) desenvolveram um *middleware* para gerenciamento de sessões que apóia, de maneira transparente, a interação entre o *proxy* e o processo servidor onde o *proxy* se conecta. Esse *middleware* possui dois mecanismos importantes para o conceito *follow-me session*. O primeiro é um mecanismo para migração de *thread* de servidor, cujo objetivo é migrar um serviço quando o mesmo não é oferecido em um determinado dispositivo computacional. O segundo é um mecanismo de conexão sensível a contexto, cujo objetivo é permitir que um serviço seja oferecido por múltiplos servidores, de modo que um desses seja escolhido como melhor opção por uma aplicação cliente para execução do serviço. A seleção do melhor servidor é baseada em políticas especificadas na aplicação cliente, que captura, por meio do mecanismo de conexão sensível ao contexto, informações de contexto do ambiente distribuído. Então, uma vez que informações de contexto tenham sido capturadas, o mecanismo de conexão sensível ao contexto avalia um conjunto de servidores, fornecedores de um mesmo serviço e, em seguida, extrai a melhor opção. Após um servidor ter sido escolhido, o cliente inicia sua interação com o mesmo. Com isto, periodicamente, informações de contexto do servidor são extraídas e enviadas para o cliente, que as armazena localmente. Em paralelo, no lado cliente, o mecanismo de conexão sensível ao contexto avalia se o servidor atual ainda é a melhor opção de execução de um determinado serviço. Caso seja necessária a troca de servidor, a execução do serviço é interrompida e, logo após, o processo de conexão a um novo servidor é iniciado.

Zhou e Lo (2006) apresentam um sistema de computação em grade baseado em computadores *desktop*, formado a partir de uma rede *peer-to-peer* estruturada. Nesse ambiente distribuído, cada *peer* compartilha seus processadores com o objetivo de utilizá-los para o processamento distribuído de tarefas. Para tanto, uma rede CAN é dividida em várias *wavezones*. Cada *wavezone* representa vários fusos horários. Assim, à medida que *peers* desejam compartilhar ciclos de seus processadores, eles se conectam a uma *wavezone* da qual seus fusos horários façam parte. Com isto, aplicações clientes podem tirar proveito de ciclos de processadores disponíveis em uma *wavezone*. As aplicações que executam sobre a grade computacional proposta por Zhou e Lo são aquelas que necessitam executar o processamento de *jobs*. Cada *job* consiste de um número de tarefas independentes, que demandam uma

grande quantidade de ciclos de processador. Por este motivo, os escaladores dessas aplicações necessitam escolher um grupo de computadores capazes de receber a migração de tarefas e, em seguida, executá-las. Para isto, o critério de escolha do grupo de computadores usa um modelo de compartilhamento em que o compartilhamento dos ciclos de processador é limitado pelo tempo em que os usuários não estão usando seus computadores. Uma vez escolhido o grupo de computadores, o segundo critério utilizado inclui a potência do processador, tamanho da memória e o tipo de sistema operacional. Com isto, em um grupo de candidatos para migração, somente são selecionados aqueles que cumprirem os requisitos necessários para a execução de uma determinada tarefa. Por fim, analisando os resultados obtidos por Zhou e Lo (2006), a abordagem proposta apresenta bons resultados no que diz respeito a tempos de resposta, estabilidade e impacto nos *peers*.

Riva *et al.* (2007) apresentam uma arquitetura para aplicações baseadas em serviços móveis que fornece elementos arquiteturais fundamentais para o projeto de *middlewares* para serviços móveis. Estes elementos arquiteturais são responsáveis pelas seguintes funções: monitoramento e atualização de informação de contexto, gerenciamento de serviços móveis, gerenciamento de comunicação, migração de serviços, roteamento de mensagens e descoberta e vizinhança. Assim, conforme o nível de abstração das funções apresentadas anteriormente, elas foram agrupadas em camadas específicas da arquitetura de software. Com isto, a arquitetura proposta por Riva et al. (2007) foi dividida em duas camadas: *middleware* e sistema. A camada de *middleware* endereça o monitoramento e atualização de informações de contexto, controle de execução de serviços móveis e gerenciamento de comunicação entre clientes e serviços. Por fim, a camada de sistema é responsável pelo fornecimento de contexto, suporte a migração de serviços e comunicação *peer-to-peer*.

3.5 CONSIDERAÇÕES FINAIS

Durante o levantamento bibliográfico desta dissertação, uma parte fundamental foi a análise de abordagens para o compartilhamento de recursos em redes sociais *peer-to-peer*, de modo que essas pudessem compor os trabalhos relacionados deste trabalho de pesquisa.Até o momento da conclusão desta dissertação, não foi encontrada nenhuma abordagem que endereçasse o uso de redes sociais como metodologia para construção de ambientes de computação distribuída, onde os recursos compartilhados pelos membros de uma rede social são utilizados para armazenamento e execução de software de maneira distribuída. Por isso, técnicas e arquiteturas de softwares para sistemas distribuídos foram pesquisados, de modo

que esses pudessem compor uma proposta, tanto arquitetural quanto metodológica, para o compartilhamento de recursos em redes sociais, formadas sobre uma infraestrutura de rede *peer-to-peer*. Portanto, foram investigados trabalhos relacionados à organização de redes sociais *peer-to-peer*, agregação de recursos e backup colaborativo em redes *peer-to-peer*, e migração de serviços.

Todos estes trabalhos contribuíram para a elaboração de uma abordagem, tanto arquitetural, quanto algorítmica, para o uso de redes sociais *peer-to-peer* como meio de compartilhamento recursos em ambientes de *e-Science*. Assim, a proposta da abordagem, alvo do estudo desta dissertação, é dividida em duas partes. A primeira parte (Capítulo 4) apresenta uma proposta arquitetural de software cujo objetivo é fornecer elementos arquiteturais necessários para o compartilhamento de recursos e o uso desses em redes sociais *peer-to-peer*. Já a segunda parte (Capítulo 5) apresenta as políticas de computação distribuída, cujo objetivo é a proposta inicial de soluções algorítmicas para organizar um ambiente de computação distribuída, de modo que este possibilite o uso de redes sociais *peer-to-peer* como ambiente de computação.

4 ARQUITETURA PARA COMPARTILHAMENTO DE RECURSOS EM REDES SOCIAIS *PEER-TO-PEER*

Este capítulo apresenta a proposta de uma arquitetura de software para lidar com o compartilhamento de recursos em redes sociais *peer-to-peer*, já que as arquiteturas de software investigadas ao longo do desenvolvimento deste trabalho científico não fornecem elementos arquiteturais suficientes para lidar com as seguintes responsabilidades: instrumentação de recursos computacionais; comunicação entre computadores; descoberta e roteamento; disseminação de informação; fornecimento de contexto; compartilhamento de recursos; gerenciamento e execução de serviços e aplicações; armazenamento colaborativo de conteúdo; controle de sessão de usuário e interação entre usuário-aplicação e usuário-serviço.

Para melhor entender como o modelo arquitetural foi decomposto, este capítulo inicia-se com uma breve descrição do modelo na Seção 4.1. Em seguida, são apresentados, de maneira *top-down*, os níveis de abstração da arquitetura de software proposta neste capítulo, iniciando na Seção 4.2 e finalizando na Seção 4.6.

4.1 MODELO ARQUITETURAL

Desenvolver software para ambientes de *e-Science* não é uma tarefa fácil, pois aplicações lidam com um amplo espectro de padrões e tecnologia similares, causando uma dependência de interfaces com primitivas para lidar com padrões e/ou tecnologias de maneira muito específica. Fazendo uma análise sobre isso, é possível identificar dois aspectos, que, no escopo desta dissertação, dificultam o desenvolvimento de funcionalidades de aplicações para ambientes de *e-Science*. São eles:

- **Dependência de APIs –** para tirar proveito das infraestruturas computacionais de alto desempenho dos ambientes de *e-Science*, desenvolvedores utilizam um conjunto de primitivas, que é fornecido por bibliotecas de programação paralela ou distribuída, para programar funcionalidades de software para *e-Science*. Apesar de úteis, tais bibliotecas fazem com que funcionalidades de software sejam dependentes de suas primitivas, podendo trazer problemas de portabilidade, flexibilidade, extensibilidade e transparência.

- **Dependência de infraestrutura computacional –** devido ao uso de bibliotecas de programação paralela e distribuída, funcionalidades de software para *e-Science* são dependentes de infraestruturas computacionais como *clusters*, grades computacionais e ambientes heterogêneos de computação distribuída. Esta dependência pode tornar cara ou até mesmo inviabilizar uma pesquisa, quando custo e/ou tempo são variáveis críticas no projeto. Segundo Ogasawara *et al.* (2009), os motivos para isto são: *clusters* são soluções que apresentam um alto custo de implantação e manutenção; grades computacionais ainda apresentam grandes barreiras burocráticas, mas necessárias, para que um pesquisador se torne usuário desse tipo de infraestrutura; ambientes heterogêneos de computação distribuída aumentam muito o tempo de espera entre a criação, configuração, submissão, aprovação e execução de um projeto neste tipo de infraestrutura computacional.

Para lidar com isto, é necessário criar uma arquitetura de software que seja capaz de se adaptar facilmente aos diversos padrões e tecnologias utilizados no domínio de *e-Science*. Sendo assim, Buschmann *et al.* (2001) propõem o uso de uma arquitetura de software chamada de *microkernel*, que oferece um conjunto de funcionalidades básicas, permitindo que estas sejam estendidas por aplicações específicas de software. Um *microkernel* também serve como um ambiente para plugar e coordenar estas aplicações específicas de software. Isto faz com que um desenvolvedor de software se preocupe somente com detalhes específicos do domínio de aplicação, isto é, não é necessário se preocupar com detalhes de baixo nível de abstração. A justificativa para isto é que uma arquitetura *microkernel* oferece benefícios como portabilidade, flexibilidade e extensibilidade, separação de políticas e mecanismos, escalabilidade, confiabilidade e transparência. Por esses motivos, a proposta de arquitetura de software para compartilhamento de recursos em redes sociais *peer-to-peer* está organizada na forma de um *microkernel* distribuído.

Para construir o *microkernel*, foi necessário separar dois aspectos importantes: comportamento e conteúdo. Para lidar com o comportamento, foi elaborada uma arquitetura de software orientada a serviços móveis, devido à natureza heterogênea dos dispositivos computacionais fixos, formadores do ambiente de computação distribuída que estamos estudando. Essa parte do trabalho gerou duas publicações (GONÇALVES *et al.* 2002a; GONÇALVES *et al.* 2007b). Após isto, foi criada uma proposta que pudesse fornecer um arranjo de componentes para lidar com o aspecto conteúdo. Fruto disto foi uma arquitetura para compartilhamento de recursos computacionais de armazenamento em redes sociais *peer-*

to-peer (GONÇALVES, OLIVEIRA e BRAGANHOLO 2009). O principal objetivo desta última arquitetura de software é fornecer elementos arquiteturais para lidar com a criação de redes sociais sobre uma rede *peer-to-peer* e, em seguida, utilizar os compartilhamentos de discos dos participantes de uma mesma rede social para o armazenamento colaborativo de conteúdo.

Desta forma, o *microkernel* foi organizado em cinco camadas, de modo que os elementos arquiteturais pertencentes ao mesmo nível de abstração pudessem ser agrupados em uma mesma camada. Seguindo este critério de organização, o *microkernel* é composto pelas seguintes camadas:

- **Camada de Usuário –** responsável pelo recebimento e encaminhamento das requisições disparadas pelo usuário, a partir de um cliente do *microkernel*; extração de informações presentes nas requisições submetidas pelos clientes; e envio de requisições para a Camada de Sessão, de modo que esta possa iniciar o gerenciamento de alguma interação entre um cliente e o *microkernel*.
- **Camada de Sessão –** recebe as requisições e informações de contexto criadas pela Camada de Usuário; encaminha as informações de contexto para serem armazenadas na Camada Intermediária; cria sessões para as interações entre os clientes e serviços oferecidos pelo *microkernel*; e notifica a chegada de respostas às requisições criadas pela Camada de Usuário.
- **Camada Intermediária –** responsável pelo gerenciamento dos repositórios de aplicações, serviços e contextos, e dos compartilhamentos de recursos, sendo estes hardware e conteúdo; trata requisições para serviços do ambiente local e também aquelas que chegam por meio da Camada de Comunicação.
- **Camada de Comunicação –** fornece facilidades de comunicação *peer-to-peer*, por meio de mecanismos para comunicação fim-a-fim, descoberta e roteamento; fornece mecanismos que tratam da disseminação e replicação de dados, migração de serviços e fornecimento de contexto local e do sistema distribuído.
- **Camada de Ambiente –** encapsula o sistema operacional onde o *microkernel* está sendo executado; fornece informações sobre as configurações de hardware existentes no computador do usuário; e fornece informações específicas sobre cada tipo de hardware do computador do usuário

Um cliente é alguma aplicação que faz uso dos serviços disponibilizados pelo *microkernel*. Estes serviços compõem duas categorias de serviços, que são: serviços básicos e

serviços de aplicações. Serviços básicos são aqueles serviços relacionados ao funcionamento interno do *microkernel*. Esta categoria de serviços não precisa de uma aplicação para disponibilizá-los, pois os mesmos são disponibilizados pelo próprio *microkernel*. Já os serviços de aplicações são aqueles desenvolvidos para atender os requisitos de alguma aplicação. Para utilizá-los, é necessário que a aplicação seja instalada no *microkernel*. Este processo é explicado com mais detalhes na seção 4.4.

Baseado no modelo arquitetural lógico exibido pela Figura 4.1, um usuário, por meio de um cliente do *microkernel*, faz uma requisição para execução de um serviço à instância do *microkernel que* está em execução no computador pessoal do usuário. Esta perpassa pelas camadas do *microkernel* até chegar à camada responsável pelo seu tratamento. Logo após o tratamento da requisição, uma resposta é enviada ao cliente, que extrai os dados retornados pelo serviço requisitado. Os caminhos percorridos pelas requisições e respostas são exibidos na Figura 4.1.

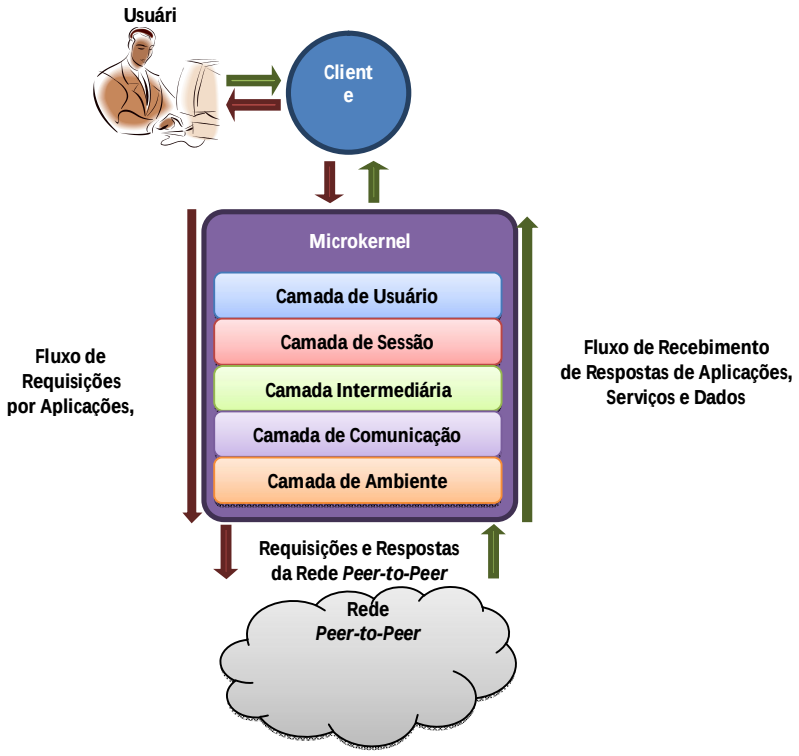


Figura 4.1: Modelo arquitetural lógico

Se a requisição for local como, por exemplo, recuperar a quantidade de espaço livre em disco do computador, ela será tratada no próprio computador do usuário e uma resposta será enviada até a camada de nível mais alto. O oposto à realização de uma requisição local se constitui quando uma requisição remota é realizada. Por exemplo, uma requisição para

compartilhar um conteúdo com os membros de uma rede social. Este tipo de requisição perpassa todas as camadas da arquitetura de software, mas diferentemente de uma requisição local, a requisição remota é submetida ao sistema distribuído. Em seguida, esta requisição é recebida por um ou mais *peers* conectados à rede *peer-to-peer*. Neste momento, cada um destes *peers* trata a requisição remota e devolve uma resposta ao *peer* dono da requisição. Isto é possível, pois os computadores pessoais de usuários conectados ao ambiente distribuído são transformados em *peers*, uma vez que cada um deles executam uma instância do *microkernel*. Quando a resposta à requisição remota chega ao seu destinatário, a mesma é tratada e, em seguida, tem seus dados encaminhados até a camada de mais alto nível do modelo arquitetural, como pode ser visto na Figura 4.1.

O encaminhamento de respostas de uma camada de baixo nível para outra de nível superior subsequente é feito por meio de *callbacks* (BUSCHMANN *et al*. (2001). Desta forma, os elementos arquiteturais de camadas de um nível mais alto de abstração podem reagir às respostas dadas por elementos arquiteturais pertencentes às camadas de um nível de abstração menor. Então, para sincronizar a comunicação entre as camadas, foi utilizado o padrão de projeto *Observer* e o *Decorator* (GAMMA, et al. 2001). Estes padrões permitem definir uma dependência entre uma camada e outra, de maneira que se uma camada de nível de abstração menor muda o seu estado, a camada de um nível de abstração acima é notificada e atualizada automaticamente. Detalhes sobre como o *microkernel* é estruturado podem ser vistos no diagrama de classes apresentado no Apêndice A.

Após toda a apresentação da estruturação do modelo arquitetural proposto por esta dissertação, as próximas seções detalharão cada camada e seus respectivos elementos arquiteturais, de modo que fique claro como se dá o funcionamento interno de uma arquitetura de software para compartilhamento de recursos em redes sociais *peer-to-peer*.

4.2 CAMADA DE USUÁRIO

A *Camada de Usuário*, dentre as demais camadas do *microkernel*, é a camada de maior nível de abstração. Devido ao seu posicionamento na organização do *microkernel*, essa camada é o primeiro elemento arquitetural a ter contato com as requisições disparadas a partir de um cliente do *microkernel*. Por este motivo, ela deve atuar como um filtro de requisições, deixando passar somente aquelas requisições relacionadas aos serviços utilizados pelo usuário, com exceção de requisições para criação e conexão de usuário no ambiente distribuído. Este nível de controle é necessário, pois ele evita a propagação desnecessária de

requisições para as demais camadas da arquitetura de software. Para tanto, é necessário que o usuário esteja conectado ao sistema distribuído, pois a camada precisa acessar informações sobre o usuário, objetivando tomadas de decisões corretas no que diz respeito à propagação das requisições para as camadas de níveis mais baixos da arquitetura de software.

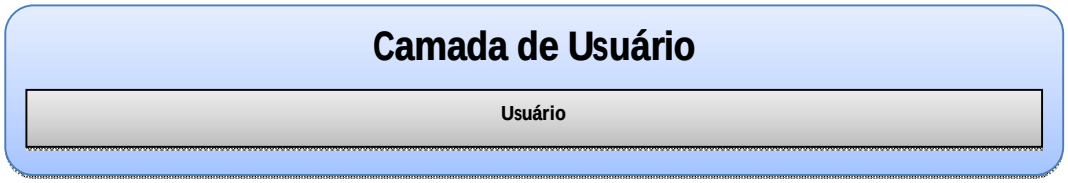


Figura 4.2: Diagrama de blocos da camada de usuário

Quando um usuário se conecta ao ambiente distribuído, a *Camada de Usuário* é notificada. Neste momento, a camada recebe todas as informações relacionadas ao usuário conectado, fazendo com que seu único elemento arquitetural seja ativado. Este, por sua vez, é chamado de *Usuário*, conforme apresentado pela Figura 4.2.

O elemento arquitetural *Usuário* tem a responsabilidade de manter a *Camada de Usuário* sempre informada sobre as atualizações de contexto do usuário. Dentre as informações de contexto do usuário, é possível destacar:

- **Informações pessoais –** mantém dados sobre a identidade do usuário para interação com o ambiente distribuído;
- **Perfil social –** mantém os interesses dos usuários. Tais informações são importantes pois as políticas de computação distribuída fazem uso destas, a fim de calcular a compatibilidade do perfil social do usuário com as restrições de perfil exigidas em uma oportunidade de colaboração. Se as informações existentes no perfil social forem compatíveis com as restrições de perfil das oportunidades, o usuário poderá construir uma rede social com outros usuários do ambiente distribuído;
- **Serviços –** mantém dados sobre os serviços do *microkernel* disponibilizados para o usuário. Nesta informação de contexto, é possível obter informações sobre quais os serviços básicos do *microkernel* estão disponíveis para o usuário. A *Camada de Usuário* utilizada essas informações para identificar se uma requisição, para alguns serviços do *microkernel*, pode ser propagada para as próximas camadas em sentido *top-down*;
- **Aplicações –** mantém dados sobre as aplicações do usuário instaladas no *microkernel*. Por esta informação de contexto, é possível obter os caminhos de aplicações e serviços. A partir delas, é possível identificar se uma

requisição para execução de uma aplicação ou serviço pode ser encaminhada ou não. Para isto, a *Camada de Usuário* verifica se o caminho do serviço e os parâmetros de uma requisição são compatíveis com algum padrão de caminho de serviço mapeado para o usuário;

- **Redes sociais –** mantém informações sobre cada uma das redes sociais em que o usuário faz parte. Nesta informação de contexto, é possível obter uma lista de membros, detalhes sobre os conteúdos compartilhados e detalhes sobre os recursos de hardware.

As informações mantidas pelo *Usuário* não podem ser alteradas na *Camada de Usuário*, pois elas são somente para leitura. Caso seja necessário alterar alguma informação, o usuário deve disparar uma requisição para um serviço do *microkernel* responsável em tratá-las. Portanto, o *Usuário* atua como um *cache*, mantendo as informações sobre o usuário mais próximas dos clientes do *microkernel*.

Para que um cliente possa acessar as informações mantidas pela *Camada de Usuário*, ele deve utilizar a interface do componente plataforma. Nesta interface, existem operações que retornam informações sobre o usuário que estiver conectado no momento. Então, ao utilizar esta operação, o cliente faz com que a plataforma solicite ao *microkernel* as informações do usuário. Consequentemente, o *microkernel* acessa a *Camada de Usuário*, que, por sua vez, retorna as informações requeridas pelo cliente.

Além de manter informações sobre o usuário, o único elemento arquitetural da *Camada de Usuário* também tem a responsabilidade de interagir com a *Camada de Sessão*, que está subsequentemente abaixo em relação à *Camada de Usuário*. Essa responsabilidade consiste na formação de caminhos que possam guiar respostas até um cliente do *microkernel*. Neste caso, a abertura de caminhos consiste em solicitar à *Camada de Sessão* a criação de sessões, com o objetivo de manter informações sobre o tratamento das requisições submetidas a partir de um cliente do *microkernel*. No entanto, não são todas as requisições que necessitam de abertura de sessões, e sim somente aquelas que são submetidas de maneira assíncrona ao ambiente distribuído. Para requisições submetidas de maneira síncrona, as respostas são dadas por meio dos retornos das operações apresentadas no diagrama de classes do Apêndice A..

A execução de serviços básicos ou de aplicações pode ser requisitada através de uma requisição por serviço, submetida de maneira assíncrona. Por este motivo, o *Usuário* verifica em suas informações se o serviço requisitado é um serviço básico ou de aplicação. Logo após a isto, o *Usuário* solicita à *Camada de Sessão* a criação de sessões adequadas para os dois tipos de serviços. Por fim, uma vez que as sessões tenham sido criadas pela *Camada de*

Sessão, o *Usuário* mantém uma lista de referências para as mesmas. Os detalhes relacionados ao gerenciamento de sessões são apresentados na próxima seção.

4.3 CAMADA DE SESSÃO

A *Camada de Sessão* tem a responsabilidade de manter informações sobre as requisições assíncronas encaminhadas pela *Camada de Usuário*. Além disto, ela também é responsável pela notificação da *Camada de Usuário* sobre modificações nas informações do usuário. A Figura 4.3 apresenta um diagrama de blocos com os elementos arquiteturais da *Camada de Sessão*.

Figura 4.3: Diagrama de blocos da camada de sessão

A *Camada de Sessão* é composta por quatro componentes, que são: *Gerente de Sessão*, *Sessão de Aplicação*, *Sessão de Serviço* e *Contexto*.

O *Gerente de Sessão* é o principal componente desta camada, pois ele é responsável pelo gerenciamento das sessões criadas, quando requisições assíncronas são encaminhadas pela *Camada de Usuário*. É importante ressaltar que o gerenciamento de sessões permite estabelecer um controle sobre o ciclo de vida das requisições assíncronas submetidas ao ambiente distribuído. Para isto, o *Gerente de Sessão* monitora o tempo em que as sessões são mantidas por ele. Quando uma sessão ultrapassa o limite de tempo permitido, o *Gerente de Sessão* interrompe imediatamente a sessão, fazendo com que esta pare de encaminhar respostas ao cliente. Além disto, o *Gerente de Sessão* solicita à *Camada de Sessão* que envie uma solicitação às camadas abaixo, a fim de que estas possam remover também alguma informação sobre a sessão expirada. Este cuidado é tomado para evitar o encaminhamento desnecessário daquelas respostas que ultrapassaram o tempo limite de chegada, diminuindo, mais uma vez, o *overhead* imposto pelo padrão arquitetural utilizado para separar os níveis de abstração do *microkernel*. Isto pode ser visto com mais detalhes em Buschmann *et al.* (2001).

O *Gerente de Sessão* mantém referências para instâncias de dois tipos de sessões, que são as sessões de aplicação e as sessões de serviços. Para representar estes dois tipos de

sessões, foram criados dois elementos arquiteturais que são mantidos pelo *Gerente de Sessão*. Estes elementos arquiteturais, por sua vez, são a *Sessão de Aplicação* e a *Sessão de Serviço*. Uma *Sessão de Aplicação* é criada quando um cliente requisita de maneira assíncrona a execução de um serviço ofertado por uma aplicação. Serviços de aplicação, invocados de maneira assíncrona, são aqueles responsáveis pelo processamento de um grande volume de dados, ou seja, estes serviços são utilizados para execução de *jobs* sobre os ciclos de processadores e memórias compartilhadas em uma rede social *peer-to-peer*. Por isto, uma *Sessão de Aplicação* mantém informações sobre o estado do *job* que está em execução sobre os recursos compartilhados em uma rede social *peer-to-peer*. Este estado mantém informações sobre a completude do *job*, além de manter os dados retornados após o processamento de cada tarefa gerada pelo *job*. Além de informar a quantidade de tarefas concluídas, a *Sessão de Aplicação* também mantém informações sobre as tarefas que ainda não retornaram seus resultados de execução. Com essas informações, é possível manter um usuário informado sobre o estado de execução do *job* submetido para processamento, pois, como mencionado anteriormente, o componente *Usuário* mantém uma lista de referências para as seções criadas a partir das requisições assíncronas encaminhadas por ele.Apesar da *Sessão de Aplicação* ser utilizada para manter informações sobre o estado de um *job*, ela pode ser utilizada para outros fins como, por exemplo, manter informações sobre os dados retornados de uma consulta a uma base de dados distribuída. Os dados mantidos em uma *Sessão de Aplicação* estão condicionados às implementações dos serviços de aplicação. Estes, ao receberem uma requisição, podem notificar a *Camada de Sessão* sobre quais os dados devem ser mantidos, ou seja, é uma decisão tomada pelo desenvolvedor do serviço.

Como mencionado anteriormente, o *Gerente de Sessão* cria sessões para manter informações sobre a execução de serviços básicos do *microkernel*. Este tipo de sessão é representado pelo componente *Sessão de Serviço*. Portanto, uma *Sessão de Serviço* é criada quando um cliente requisita de maneira assíncrona a execução de um serviço básico do *microkernel*. Serviços básicos, invocados de maneira assíncrona, são aqueles responsáveis por executar as políticas de computação distribuída que oferecem suporte para o funcionamento das redes sociais *peer-to-peer* (Capítulo 5). A criação de redes sociais *peer-to-peer* se dá de maneira assíncrona, pois as políticas de computação distribuída necessitam da interação com os usuários envolvidos neste processo. Por isto, é necessário manter sessões abertas, pois à medida que usuários confirmam ou negam requisições, em um determinado instante de tempo, eles não precisam esperar por respostas, já que suas aplicações são notificadas sobre as chegadas destas.

Outro componente importante da *Camada de Sessão* é o *Contexto*, pois este transporta informações do usuário da *Camada Intermediária* até a *Camada de Usuário*. Entende-se como contexto tudo o que influencia o comportamento de um sistema, incluindo recursos internos, como a quantidade de memória, energia disponível; e recursos externos, como largura de banda, qualidade da conexão de rede, localização do dispositivo, dispositivos vizinhos, entre outros (EL-KHATIB, et al. 2004). Portanto, quando há alguma modificação de informação de usuário na *Camada Intermediária*, esta notifica imediatamente a *Camada de Sessão*, fazendo com que o *Contexto* notifique a *Camada de Usuário* que, por fim, atualiza as informações no *Usuário*. O *Contexto* se comporta de maneira reativa, ou seja, ele só repassa informações para a camada superior à *Camada de Sessão* se ele for notificado. Por mais que pareça, o *Contexto* não é uma *thread* que fica esperando ser notifica para, em seguida, realizar alguma ação. O *Contexto* atua como um componente de mediação entre a *Camada de Usuário* e a *Camada Intermediária*, onde os dados estão armazenados.

4.4 CAMADA INTERMEDIÁRIA

A *Camada Intermediária* tem a responsabilidade de gerenciar serviços, compartilhamentos de recursos e contextos, além de tratar requisições para execução de serviços. Para tanto, a camada utiliza os componentes apresentados pela Figura 4.4.

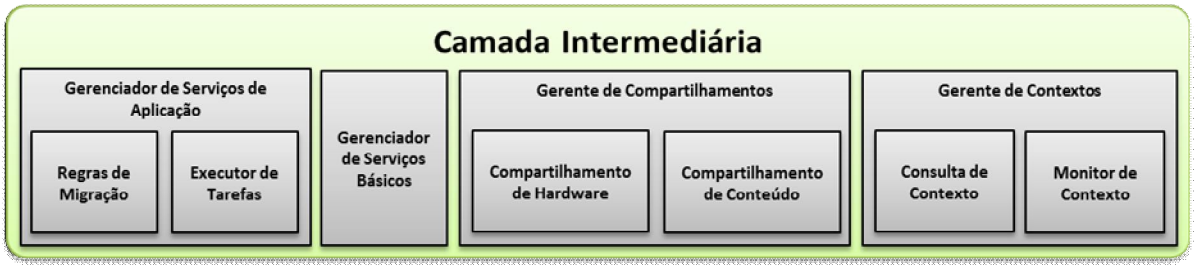


Figura 4.4: Diagrama de blocos da camada intermediária

O mecanismo para tratamento de requisições do *microkernel* é bem simples. Quando uma requisição é criada, é necessário informar o caminho do serviço. Este caminho deve ser uma *string* bem formada, a fim de não causar ambigüidade na localização do tratador de requisição relacionado ao serviço requerido. Para lidar com este requisito, foi necessário utilizar um esquema de identificação que fornecesse um modelo hierárquico, permitindo mapear um caminho de serviço em um tratador responsável por tratar requisições com este caminho. O esquema adotado segue a gramática utilizada para geração de URIs, que é: `<nome do esquema>:<parte hierárquica> [?<consulta>]`. Por meio deste esquema, serviços são identificados de maneira uniforme, ou seja, os caminhos dos serviços

só podem ser formados a partir de uma única regra. No entanto, para que o mecanismo de tratamento de requisições possa realizar os mapeamentos de forma correta, é necessária a utilização de descritores de serviços para definir o nome do esquema, as hierarquias e parâmetros de entrada dos serviços.

Para viabilizar o mecanismo de tratamento de requisições, esta dissertação lançou mão de dois padrões de projeto: *Service Configurator* (SCHIMIDT, et al. 2000) e *Reactor* (SCHIMIDT, et al. 2000). O *Service Configurator* permite que serviços sejam adicionados e removidos em tempo de execução, sem modificar ou recompilar uma arquitetura orientada a serviços. Este padrão de projeto define quatro participantes importantes para o funcionamento do mecanismo de tratamento de requisições, são eles: *Service, Concrete Service, Service Repository e Service Configurator*. O *Service* define uma interface padrão que pode ser usada para configurar, executar e fornecer informações sobre o serviço. Nesta interface, constam operações para inicialização, execução e finalização do serviço. O *Concrete Service* define a implementação de um serviço específico. O *Service Repository* gerencia todas as implementações de serviços que são configurados para uma aplicação ou uma plataforma. Este participante permite que aplicações de gerenciamento de sistemas controlem o comportamento dos serviços. Por fim, o *Service Configurator* usa o repositório de serviços para coordenar a reconfiguração dos serviços concretos. Para isto, este participante precisa de um descritor de serviços, que deve possuir uma especificação sobre que serviços serão adicionados a uma aplicação ou a uma plataforma.

O segundo padrão de projeto utilizado no mecanismo de tratamento de requisições é o *Reactor*. Este padrão de projeto permite executar os serviços de uma aplicação ou plataforma e, em seguida, enviar respostas para algum cliente. Com este objetivo, o padrão conta com seis participantes: *Handle, Synchronous Event Demultiplexer, Event Handler, Concrete Event Handler* e *Reactor*. O *Handle* representa os rótulos de eventos ou mensagens fornecidos pelos protocolos de comunicação ou por algoritmos distribuídos. Este participante é utilizado meramente como uma *string*, que permite identificar as etapas de execução de algoritmos distribuídos. Além disto, eles são utilizados para mapear tratadores de requisições. O *Synchronous Event Demultiplexer* representa uma função para esperar a chegada de uma ou mais indicações sobre a chegada de mensagens com *handles* específicos. O *Event Handler* especifica uma interface com uma ou mais operações, que são utilizadas para realizar o tratamento de requisições específicas de uma aplicação ou de uma plataforma. O *Concrete Event Handler* implementa um serviço específico que uma aplicação ou uma plataforma oferece. Este participante, neste trabalho, atua como tratador de requisições para execução de

serviços básicos e serviços de aplicações. Por fim, o *Reactor* define uma interface que permite que aplicações ou uma plataforma registrem ou removam *Event Handlers* e seus *Handles.* Um *Reactor* usa um *Synchronous Event Demultiplexer* para esperar a chegada de requisições por serviços. Quando isto acontece, o *Reactor* verifica um *pool* de *threads* que podem ser utilizadas para o tratamento da requisição. Se houver disponibilidade de *thread*, o *Reactor* captura a requisição e, em seguida, encaminha para o tratador de serviços adequado. Caso contrário, o *Reactor* espera a disponibilidade de uma*thread* no *pool.* O *pool* de *threads* deve possuir um tamanho limitado, pois isto restringe problemas relacionados ao limite de criação de *threads* em um sistema operacional.

Na *Camada Intermediária*, existem dois elementos arquiteturais que são mecanismos para tratamento de requisições. Um deles é o *Gerenciador de Serviços de Aplicações*, que tem a responsabilidade de tratar requisições para serviços que compõem as funcionalidades de aplicações instaladas no *microkernel*. Estas aplicações devem ser criadas para fornecer funcionalidades que possam tratar do processamento de grandes quantidades de dados. Assim, funcionalidades de aplicações são desenvolvidas como serviços. Estes dividem o processamento em tarefas, de modo que a execução da funcionalidade possa ser paralelizada e, em seguida, distribuída sobre os compartilhamentos de processadores em uma rede social *peer-to-peer*. As tarefas são objetos móveis que recebem como entrada de dados pequenas partes de um grande volume de dados. Para controlar o envio dos objetos móveis para executarem em processadores compartilhados, cada aplicação possui um escalonador de tarefas, que mantém o controle das tarefas que já foram executas e as que ainda faltam executar. O escalonador verifica constantemente a disponibilidade de compartilhamentos de processadores no *Gerenciador de Contextos*. Neste elemento arquitetural, o escalonador busca primeiramente informações sobre a rede social *peer-to-peer* em que o serviço de aplicação está executando e, em seguida, recupera as informações atualizadas sobre o estado dos compartilhamentos dos processadores. Após obter esta informação, o escalonador escolhe quais membros são capazes de executar tarefas e, em seguida, encaminha uma lista de membros de uma rede social *peer-to-peer* e um conjunto de objetos móveis para a *Camada de Comunicação.* Logo após, esta camada distribui os objetos móveis sobre os computadores dos membros de uma rede social *peer-to-peer*. Quando os objetos móveis chegam aos computadores dos membros de uma rede social *peer-to-peer*, eles são encaminhados até a *Camada Intermedirária* para que eles possam ser executados. Para viabilizar a execução do objeto móvel, o *Gerente de Serviços de Aplicações* fornece um elemento arquitetural chamado *Executor de Tarefas*. Por meio deste componente, objetos móveis são executados e, em

seguida, devolvidos às suas origens. Neste instante, cada objeto móvel retorna com os resultados obtidos pelo processamento dos dados de entrada, que foram configurados no objeto no momento do seu envio a outro computador. Finalmente, ao retornar ao seu computador de origem, o objeto móvel retorna os resultados obtidos durante a sua execução em outro computador, que, em seguida, são adicionados aos outros resultados já retornados por outros objetos móveis, estes gerados durante a execução do mesmo serviço de aplicação.

Além do *Executor de Tarefas*, os objetos móveis precisam de uma diretriz de execução. Estas diretrizes definem os requisitos mínimos para a execução do objeto móvel em outro computador. Portanto, tais diretrizes se referem às *Regras de Migração*. Um objeto móvel só migra para outro computador se as *Regras de Migração* forem compatíveis com a disponibilidade de recursos de hardware neste dispositivo. A partir desta condição é que são feitas as escolhas dos membros para migração dos objetos móveis. Apesar de objetos móveis poderem migrar para outro computador, eles são condicionados ao estado do ambiente em que serão executados. Então, devido a uma indisponibilidade de recursos de hardware, um objeto móvel deve ser migrado para outro computador que possa executá-lo.

Para detectar este evento, o *Executor de Tarefas* monitora informações do hardware local por meio do *Gerenciador de Contextos*. Com estas informações, o *Executor de Tarefas* é capaz de identificar se um conjunto de objetos móveis pode ser executado sobre os recursos compartilhados por um membro de uma rede social *peer-to-peer*. Se houver disponibilidade de recursos, o *Executor de Tarefas* escalona os objetos móveis para execução. Caso contrário, os objetos móveis são devolvidos às suas respectivas origens, contendo o estado de não executado.

Para saber quais objetos móveis serão executados, o *Executor de Tarefas* gera uma lista ordenada de objetos móveis que estão para ser executados, com base nos valores das regras de migração mantidas por estes objetos. Objetos móveis, além de serem configurados com dados de entrada, também recebem as *Regras de Migração* configuradas para o serviço de aplicação aos quais eles pertencem. A decisão de migrar um objeto móvel durante sua execução, no entanto, não é contemplada nesta dissertação. Este assunto deve ser estudado com mais cautela, pois isto pode causar uma queda no tempo de resposta de um objeto móvel para a sua origem. Outro motivo para uma investigação mais profunda sobre o assunto é que os trabalhos encontrados na literatura utilizam redes *peer-to-peer* estruturadas (LO e ZHOU 2004). No entanto, a proposta feita por esta dissertação está baseada em redes *peer-to-peer* não estruturadas, pois este modelo de estruturação oferece uma maior autonomia, já que a entrada de novos *peers* não exige configurações manuais nos demais *peers* do sistema

distribuído. Por fim, a divisão das tarefas para um serviço de aplicação, necessita de heurísticas que permitem dividir e escalonar tarefas de maneira ótima. Apesar disto, neste trabalho não é proposta uma heurística para divisão e escalonamento de tarefas, uma vez que já existem esforços nesta área de estudo (YANH e CASANOVA, 2003; ASSIS *et al.* 2006).

O *Gerenciador de Serviços Básicos* é o outro elemento arquitetural da *Camada Intermediária.* Ele é um mecanismo de tratamento de requisições. Este elemento arquitetural tem a responsabilidade de tratar requisições para a execução de serviços básicos do *microkernel*. Estes serviços básicos estão relacionados ao gerenciamento dos repositórios de informações de contexto, compartilhamento de hardware e compartilhamento de conteúdo.

Além dos mecanismos de tratamento de requisições, a *Camada Intermediária* ainda possui outros dois elementos arquiteturais importantes, são eles: *Gerenciador de Compartilhamentos* e *Gerente de Contextos*. O *Gerenciador de Compartilhamentos* tem a responsabilidade de manter informações sobre os compartilhamentos de recursos locais com membros de uma rede social *peer-to-peer*. Neste trabalho, o compartilhamento de recursos consiste em ceder parte do hardware local e ou disponibilizar um conteúdo para uma rede social *peer-to-peer*. Portanto, os elementos internos do*Gerenciador de Compartilhamentos* cuidam de cada tipo de compartilhamento de recursos abordados nesta dissertação. Para manter informações sobre os compartilhamentos de hardware, é utilizado o repositório chamado *Compartilhamentos de Hardware*. Nesta dissertação, são contemplados somente compartilhamentos de espaços livres em discos, ciclos de processadores e espaço livre em memória. Vale ressaltar que o registro dos compartilhamentos de hardware não é intrusivo, ou seja, não são criadas partições de discos, restrição de ciclos de processador e alocação de regiões de memória. Os compartilhamentos de hardware atuam como cotas de uso para os membros de uma rede social *peer-to-peer* e, por isso, o *microkernel* monitora o uso destes recursos compartilhados, à medida que eles são utilizados pelos membros de uma rede social *peer-to-peer.* O monitoramento do uso dos compartilhamentos se dá por meio do registro de elementos que podem consumi-los durante o ciclo de vida de uma rede social.

Então, para monitorar o uso dos compartilhamentos de processadores e memórias, são contabilizados os objetos móveis que estejam consumindo estes recursos no computador do usuário. Esta contabilidade é feita medindo o tamanho do estado do objeto móvel e o tempo em que o objeto ocupa o processador, enquanto o mesmo realiza sua tarefa. Determinar o tempo de execução de um objeto móvel em um computador é difícil. Por isso, é permitido, quando não se conhece o tempo, que um objeto móvel execute uma primeira vez. No entanto, se ele exceder um número de ciclos de processador pré-determinando, ele tem sua execução

interrompida pelo *Executor de Tarefas* e, consequentemente, é devolvido para o *peer* que o enviou. Em uma situação oposta a isto, o objeto móvel é executado e, em seguida, é obtido o tempo gasto para a execução da tarefa. Por fim, se o espaço livre de memória for menor que a quantidade de dados transportados pelo objeto móvel, este não é aceito para execução e, por isso, deve ser devolvido para o seu *peer* de origem.

Ainda no contexto de controle de recursos compartilhados, é necessário também executar o monitoramento de compartilhamento de espaço livre em disco. Os conteúdos compartilhados em uma rede social *peer-to-peer* precisam ser acessados por todos os membros de uma mesma rede. Por isto, alguns membros cedem partes livres de seus discos rígidos, a fim de que cópias do conteúdo compartilhado possam ser alocadas nestes espaços. Então, para controlar o uso dos compartilhamentos de discos, são criados registros contendo o tamanho do conteúdo alocado para um determinado espaço compartilhado. Esta área pode ser utilizada até o momento em que não for possível armazenar algo. Também não é permitido alocar conteúdos que tenham tamanho maior que a área compartilhada. O uso dos compartilhamentos de hardware em uma rede social *peer-to-peer* deve passar pelo aceite dos membros donos dos compartilhamentos. Por isto, os membros de uma mesma rede social *peer-to-peer* colaboram uns com os outros, fechando acordos de comprometimento dos seus compartilhamentos para alguma necessidade em comum dos membros.

Ainda versando sobre o *Gerenciador de Compartilhamentos*, este elemento arquitetural possui outro elemento interno que é responsável pelos registros de conteúdos compartilhados em uma rede social *peer-to-peer* e, por isso, ele é chamado de *Compartilhamento de Conteúdos*. Após o fechamento de acordos de armazenamento entre membros de uma rede social *peer-to-peer*, o *Compartilhamento de Conteúdos* registra os membros que mantêm cópias do conteúdo compartilhado. Neste trabalho, apesar de ser mencionado o uso de replicação de conteúdo, não são tratados detalhes específicos de replicação, pois este assunto não faz parte do escopo desta dissertação. Vários trabalhos que tratam do tema podem ser encontrados na literatura (AKBARINIA, PACITTI e VALDURIEZ, 2006; AKBARINIA, PACITTI e VALDURIEZ, 2007).

Por fim, o *Gerenciador de Contextos* atua como uma pequena base de dados no *microkernel*. Nele são armazenadas todas as informações utilizadas pelas camadas superiores à *Camada Intermediária*. Além disto, ele também tem a responsabilidade de armazenar informações que chegam das camadas abaixo da *Camada Intermediária*. Para isto, são utilizados elementos internos, são eles: *Consulta de Contexto* e *Monitor de Contexto*. A *Consulta de Contexto* está relacionada às informações de contexto retornadas como resultado

de uma consulta ao ambiente distribuído. Quando, por exemplo, um usuário publica uma oportunidade para formação de uma rede social *peer-to-peer*, o mesmo está consultando por outros usuários que podem contribuir para a formação da rede social. À medida que os usuários respondem à oportunidade, informações são retornadas à *Consulta de Contexto*, após a *Camada de Comunicação* recebê-las e encaminhá-las para a *Camada Intermediária*. Em seguida, o *Gerenciador de Contexto* realiza a notificação da chegada de informação para as camadas mais altas do *microkernel*. Já o *Monitor de Contexto* tem a responsabilidade de receber informações de hardware coletadas pela *Camada de Ambiente*, que as repassa para a *Camada de Comunicação* que, em seguida, entrega para a *Camada Intermediária*. Ao chegar nesta camada, as informações de hardware são transformadas em contexto e, em seguida, são armazenadas pelo *Gerenciador de Contexto*. Após isto, as informações são repassadas até chegar à *Camada de Usuário*, para que lá o componente *Usuário* possa ter suas informações atualizadas.

A *Camada Intermediária* necessita de facilidades de comunicação *peer-to-peer* e outros mecanismos úteis para interação com o ambiente distribuído. Então, a fim de detalhar como estes mecanismos funcionam, a próxima seção fará a apresentação da *Camada de Comunicação.*

4.5 CAMADA DE COMUNICAÇÃO

A *Camada de Comunicação* consiste de mecanismos para comunicação fim-a-fim, descoberta de recursos, roteamento de mensagens, disseminação de dados, controle de migração de serviços, controle de replicação de informações de contexto e fornecimento de contexto do sistema distribuído. Estes mecanismos são representados respectivamente pelos seguintes elementos arquiteturais: *Comunicação Fim-a-Fim, Descoberta e Roteamento, Disseminação de Dados, Gerente de Migração, Gerente de Replicação e Fornecedor de Contexto*. Eles são exibidos no diagrama de blocos da Figura 4.5.

Figura 4.5: Diagrama de blocos da camada de comunicação

Todos os elementos arquiteturais usam o elemento de *Comunicação Fim-a-Fim*, pois ele encapsula detalhes sobre os protocolos de comunicação disponíveis no computador do usuário. Por meio desse componente, são realizados os envios e recebimentos de mensagens entre dois computadores, desde que um possa se conectar ao outro por meio de *sockets*. Assim, quando uma mensagem precisa ser enviada para outro computador, o elemento de *Comunicação Fim-a-Fim* serializa esta mensagem, gerando um *stream*, que, em seguida, é enviado ao computador que deve receber esta mensagem. Quando este *stream* é recebido por este computador, o elemento de *Comunicação Fim-a-Fim* o recupera e o transforma em uma mensagem novamente.

Para viabilizar o funcionamento interno do elemento de *Comunicação Fim-a-Fim*, foram utilizados três padrões de projeto de software: *Wraper Façade* (SCHIMIDT, et al. 2000), *Forwarder-Receiver* (BUSCHMANN, et al. 2001) e *Client-Dispatcher-Server* (SCHIMIDT, et al. 2000). O *Wrapper Façade* é um padrão de projeto que encapsula funções e dados fornecidos por APIs não orientadas a objetos como, por exemplo, *sockets*. Por meio deste padrão, foi possível encapsular os protocolos de comunicação TCP e UDP, fornecendo interfaces bem definidas para o envio e recebimento de *streams* entre dois computadores. O *Forwarder-Receiver* é um padrão de projeto que fornece de maneira transparente a comunicação interprocesso em sistemas *peer-to-peer*. Este padrão utiliza dois componentes, *forwarder* e *receiver*, para desacoplar os *peers* dos mecanismos básicos de comunicação. Os *forwarders* são componentes responsáveis pelo envio de mensagens de um computador a outro. Antes de enviar uma mensagem para outro computador, um *forwarder* a transforma em um *stream* e, em seguida, o envia para seu local de destino. Já os *receivers* são responsáveis pelo recebimento de mensagens. Neste momento, um *receiver* recebe um *stream* e, em seguida, transforma esta sequência de bytes em uma mensagem novamente. Para desacoplar totalmente um *peer* de qualquer mecanismo de comunicação interprocesso, *forwarders* e *receivers* são decorados com *wraper façades*. Por fim, o *Client-Dispatcher-Server* é um padrão arquitetural que fornece transparência de localização de entidades do sistema distribuído, além de encapsular detalhes sobre a comunicação destas entidades. Para isto, um *dispatcher* mantém uma estrutura de dados que permite mapear identificadores em uma tupla, contendo endereço IP, protocolo de transporte e porta de comunicação. A partir destes mapeamentos, os elementos arquiteturais da camada de comunicação podem solicitar ao *dispatcher* uma conexão para envio de mensagens para algum *peer*, a fim de que este execute algum serviço. No entanto, o elemento arquitetural *Comunicação Fim-a-Fim* só manipula mensagens cujos rótulos tenham sido registrados no *dispatcher*, que, por sua vez, associa um

manipulador de mensagens a estes rótulos. Detalhes sobre protocolos e trocas de mensagens entre os *peers* são apresentados no Capítulo 5.

Uma mensagem só pode ser entregue se o destinatário da mensagem for vizinho do *peer* que a enviou ou se houver um caminho que possa ser percorrido até que a mensagem seja entregue em seu destino. Para lidar com isso, a *Camada de Comunicação* conta com o elemento arquitetural *Descoberta e Roteamento*. Este elemento arquitetural mantém uma tabela de vizinhos, que armazena referências para os *peers* que estejam imediatamente conectados a um *peer*. A partir de cada vizinho registrado na tabela, é possível adicionar rotas para *peers* que não sejam vizinhos imediatos. Este registro de rotas é constantemente alimentado, à medida que os *peers* trocam mensagens uns com outros. Por meio disto, a *Camada de Comunicação* extrai das mensagens os caminhos percorridos por elas quando passam por alguns *peers* da rede. Além disto, também é extraído o tempo gasto por uma mensagem, à medida que esta é encaminhada sobre um número finito de *peers*, até chegar ao seu destino. Estas informações são úteis, pois elas são utilizadas para avaliar o menor custo para envio de mensagens de um *peer* para outro.

Para disseminar informações sobre uma rede *peer-to-peer*, o elemento arquitetural *Disseminação de Informação* faz uso das informações mantidas pelo elemento de *Descoberta e Roteamento*. Com tais informações, é possível, de tempos em tempos, escolher aleatoriamente um vizinho para que ele possa servir de ponto de partida para disseminação de informação. É importante salientar que o mecanismo de disseminação de informação deve evitar ciclos, ou seja, uma mesma mensagem não pode passar pelo mesmo local mais de uma vez. Além disto, o caminho percorrido pela mensagem deve ser suficiente para atingir a quantidade de *peers* conectados a rede. Esses cuidados devem ser tomados, pois eles evitam que uma mensagem fique para sempre circulando na rede e permitem que todos os *peers* tenham conhecimento sobre a informação disseminada sobre a rede.

A migração de serviços é um importante requisito na arquitetura de software proposta neste capítulo. Por isto, o *Gerente de Migração* é responsável por enviar objetos móveis para serem executados sobre os recursos compartilhados pelos usuários em uma rede social *peer-to-peer*. Estes objetos móveis representam tarefas criadas, quando um *job* é submetido a partir de algum cliente do *microkernel*. Assim, o *Gerente de Migração* é acionado quando um serviço de aplicação precisa ter sua execução distribuída sobre os ciclos de processadores compartilhados em uma rede social. Neste momento, o *Gerente de Migração* recebe uma lista de *peers* capazes de executar um conjunto de objetos móveis. Esta lista é enviada pela *Camada Intermediária*, quando esta atende uma requisição para um

serviço de aplicação, conforme mencionado na seção anterior. Então, com base na lista de *peers*, o *Gerente de Migração* solicita as menores rotas para cada um dos *peers* ao elemento arquitetural de *Descoberta e Roteamento* e, em seguida, envia os objetos móveis para serem executados em cada um dos *peers* presentes na lista.

O *Gerente de Replicação* é o componente responsável pela replicação de informações de contexto entre os *peers*. Isto é necessário, pois, quando uma rede social *peer-to-peer* é criada, constantemente suas informações de contexto são alteradas, à medida em que novos membros ingressam na rede social ou o estado desta rede é alterado, conforme o compartilhamento e uso de hardware e conteúdo. Estas alterações devem ser replicadas para todos os membros de uma rede social *peer-to-peer*¸ a fim de que estes tenham sempre ciência sobre o estado atual da rede social. Outra responsabilidade do *Gerente de Replicação* é armazenar de maneira colaborativa os conteúdos compartilhados nas redes sociais. Para isto, os membros de uma rede social *peer-to-peer* precisam firmar acordos de cooperação. Isto acontece quando um usuário compartilha um conteúdo com os demais membros da rede social. Neste momento, os membros são consultados sobre a possibilidade de armazenamento do conteúdo sobre os compartilhamentos de espaços livres de discos. Conforme a resposta retornada para o usuário que está realizando o compartilhamento, o *Gerente de Replicação* recebe uma lista de *peers* que possuem espaço de disco disponível para o armazenamento colaborativo de conteúdo. A replicação, em ambientes computacionais formados pelas redes sociais *peer-to-peer*, é utilizada para suprir a falta de uma entidade centralizada de computação. Então, a replicação é feita com o objetivo de manter um conteúdo compartilhado sempre disponível para os membros de uma rede *peer-to-peer* social. No entanto, é possível que um conteúdo compartilhado se torne indisponível. Mas, para isto, os *peers* que armazenam as cópias deste conteúdo têm que estar desconectados do ambiente distribuído. Apesar disto, em atividades de armazenamento colaborativo de conteúdos em redes sociais *peer-to-peer*, pessoas tendem a colaborar, a fim de manter o conteúdo compartilhado sempre disponível, pois este é de interesse comum dos membros da rede social. Mais detalhes sobre as políticas de replicação serão explicados no Capítulo 5.

Quando alguma informação é publicada, seja esta relacionada a uma oportunidade de colaboração ou informações de contexto sobre as redes sociais *peer-to-peer*, os *peers* a capturam com o objetivo de tirar algum proveito desta. O elemento arquitetural responsável por isto é o *Fornecedor de Contexto*. Quando o elemento arquitetural *Comunicação Fim-a-Fim* recebe uma mensagem da rede *peer-to-peer*, ele identifica o rótulo da mensagem e, em seguida, encaminha para o tratador de mensagem associado ao rótulo. Assim, quando uma

mensagem relacionada à disseminação de informação chega à *Camada de Comunicação*, o elemento arquitetural de *Comunicação Fim-a-Fim* notifica o *Fornecedor de Contexto* sobre a chegada da mensagem. Em seguida, o *Fornecedor de Contexto* encaminha as informações contidas na mensagem para a *Camada Intermediária*. Ao chegar nesta camada, as informações são tratadas por um serviço de gerenciamento de contexto, que se encarrega de armazenar devidamente as informações coletadas pelo *Fornecedor de Contexto*.

Além de encaminhar informações contidas nas mensagens da rede *peer-to-peer*, o *Fornecedor de Contexto* também acessa a *Camada de Ambiente* para obter informações sobre as configurações do ambiente de computação e o estado de seu hardware. Estas informações são coletadas e seguem o fluxo normal de informações de contexto para a *Camada Intermediária.* Além disto, o *Fornecedor de Contexto* pode atuar como um filtro de mensagens contendo informações de contexto, desde que regras de filtragem sejam estabelecidas neste elemento arquitetural. Por meio disto, é possível garantir que usuários recebam somente mensagens que sejam compatíveis com seus interesses, uma vez que estes tenham sido definidos no perfil social do usuário. No entanto, no escopo deste trabalho, não é contemplado o tratamento de sobrecarga de informações, deixando isto para os trabalhos futuros relacionados à implementação de um *middleware* a partir do modelo arquitetural proposto neste capítulo.

A *Camada de Comunicação* precisa conhecer algumas informações sobre o ambiente operacional em que o *microkernel* está sendo executado como, por exemplo, endereço IP do computador do usuário e protocolos de transporte. Para fornecer estas informações, a *Camada de Comunicação* acessa informações fornecidas pela *Camada de Ambiente*. Os endereços IP e protocolos de transporte são utilizados para inicializar o elemento arquitetural para *Comunicação Fim-a-Fim*, que, por sua vez, abre portas para comunicação e, consequentemente, passa a capturar mensagens da rede *peer-to-peer*. Os detalhes sobre o funcionamento da *Camada de Ambiente* são apresentados na próxima seção.

Devido à distinção que fazemos dos *peers* em *superpeers* e *peers* simples, a camada de comunicação nestes dois tipos de *peers* se comporta de maneira diferente. Uma vez que *superpeers* intermediam a comunicação entre *peers* simples e monitoram mensagens que possuem informações de contexto a respeito do ambiente distribuído, os mecanismos da camada de comunicação de um *superpeer* tratam o mesmo conjunto de mensagens que um *peer* simples pode tratar. No entanto, os *superpeers*, quando recebem uma mensagem cujo destino não é igual ao seu identificador, a repassam para seus vizinhos *superpeers*, de modo que ela possa ser entregue ao seu destino. Além disto, quando é necessário realizar uma

disseminação de informação, os *superpeers* repassam as mensagens de disseminação de informação até que elas cheguem a um *peer* simples. Por fim, os *superpeers* monitoram mensagens que contêm requisições para criação e manipulação de informações de contextos. Isto permite que os *superpeers* mantenham as informações de contexto em índices de dados, de modo que estes possam ser consultados ou atualizados durante um processo de sincronização de informações de contexto entre os *superpeers* e *peers* simples. De maneira diferente, a camada de comunicação nos *peers* simples recebe as mensagens encaminhadas pelos *superpeers* para, em seguida, tratá-las. Se necessário, os dados mantidos nestas mensagens são utilizados para notificar os usuários dos *peers* simples, permitindo a estes usuários interagirem uns com os outros por meio de oportunidades de colaboração publicadas ou de redes sociais já estabelecidas.

4.6 CAMADA DE AMBIENTE

A *Camada de Ambiente* tem a responsabilidade de esconder detalhes sobre o ambiente de computação em que o *microkernel* está operando. Para isto, a camada utiliza elementos arquiteturais capazes de retornar informações sobre o sistema operacional, o hardware e as configurações de rede do computador do usuário. Tais elementos arquiteturais são apresentados pela Figura 4.6.

Figura 4.6: Diagrama de blocos da camada de ambiente

A *Camada de Ambiente* é formada por três componentes, como pode ser visto na Figura 4.6, são eles:

- **Sistema –** utilizado para fornecer informações sobre processos em execução no sistema operacional. Algumas APIs como, por exemplo, a WMI (*Windows Management Instrumentation*) não fornecem informações completas sobre o consumo de memória no computador do usuário. É necessário obter uma listagem de processos e capturar a quantidade de memória consumida por cada processo em execução, bem como o seu percentual consumo de CPU;

- **Hardware –** utilizado para fornecer informações sobre o hardware instalado no computador do usuário. Com isto, é possível obter informações sobre processadores, unidades de disco, interfaces de rede e memória física. Para processadores, é possível extrair a quantidade de processadores, o *clock* de processador e percentual de carga. Para unidades de disco, obtém-se quantidade de unidades de disco, capacidades de armazenamento e espaços livres. Para interfaces de rede, pode-se extrair a velocidade para transmissão e endereço IP. Por fim, para memória física, é possível obter a velocidade dos bancos e a capacidade;
- **Rede –** utilizado para obter uma lista de protocolos de transporte disponíveis no sistema operacional. Com esta lista, é possível saber ser um protocolo garante a entrega de dados, suporta *broadcasting* e ou *multicasting*.

Para recuperar todas essas informações e tornar a camada independente de sistema operacional, foram utilizados os padrões de projeto *Data Access Object* (DAO) (DEEPAK, CRUPI e MALKS 2004) e *Data Transfer Object* (DTO) (DEEPAK, CRUPI e MALKS 2004), ambos encontrados na literatura sobre padrões de projeto para aplicações JEE (*Java Enterprise Edition*) (JEE 2010) (JEE 2010). O DAO é utilizado quando o acesso a dados varia dependendo da origem dos dados. O acesso às informações sobre o computador do usuário varia muito, dependendo do tipo de sistema operacional utilizado. O DTO é um objeto utilizado para transportar dados de camadas de baixo nível para outras de alto nível. As informações contidas nos DTOs são encaminhadas para a *Camada de Comunicação* quando o *Fornecedor de Contexto* as requer. A partir disto, o *Fornecedor de Contexto* transforma os DTOs em informações de contexto e as encaminha para o *Gerenciador de Contexto* na *Camada Intermediária*.

4.7 CONSIDERAÇÕES FINAIS

Ambientes de *e-Science* são extremamente heterogêneos. Por isto, é necessário que uma arquitetura de software para compartilhamento de recursos forneça portabilidade, flexibilidade e extensibilidade, separação de políticas e mecanismos, escalabilidade, confiabilidade e transparência. Pensando nisto, este capítulo apresentou a proposta de uma arquitetura de software para compartilhamento de recursos em redes sociais *peer-to-peer*. Esta proposta fornece uma estrutura básica para construção de *middlewares* que forneçam facilidades no desenvolvimento de aplicação para *e-Science*, sem que estas dependam

fundamentalmente de APIs que causem algum tipo de dependência com relação ao hardware ou ao tipo de infraestrutura de computação distribuída.

Para dar prosseguimento ao estudo sobre o uso de redes sociais *peer-to-peer* para compartilhamento de recursos em ambientes de *e-Science*, o próximo capítulo apresenta as políticas de computação distribuída que utilizam a arquitetura apresenta por este capítulo.

5 POLÍTICAS DE COMPUTAÇÃO DISTRIBUÍDA PARA COMPARTILHAMENTO DE RECURSOS EM REDES SOCIAIS *PEER-TO-PEER*

A proposta deste trabalho, como mencionado anteriormente, é um estudo sobre o uso de redes sociais *peer-to-peer* para o compartilhamento de recursos em ambientes de *e-Science*. Neste sentido, no capítulo anterior foi apresentada uma proposta de arquitetura software, onde foram levantados os elementos arquiteturais necessários para a construção de *middleware* cuja finalidade é o auxílio no desenvolvimento de aplicações distribuídas para o domínio de *e-Science*. Estas aplicações distribuídas tiram proveito de recursos compartilhados em redes sociais *peer-to-peer* formadas por pesquisadores, quando estes se conectam em um ambiente computacional criado sobre uma rede *peer-to-peer* não estruturada. Para isto, a arquitetura de software deve oferecer serviços básicos cuja finalidade é tirar proveito de recursos locais e distribuídos.

Nesse sentido, são necessárias políticas de computação distribuída baseadas em redes sociais para viabilizar o uso dos recursos disponíveis nos computadores de pesquisadores, quando estes formam uma rede social sobre uma rede *peer-to-peer*, a fim de colaborarem uns com os outros, desde que estes tenham um motivo em comum para colaborar. Após ter sido formada, uma rede social *peer-to-peer* deve permitir que pesquisadores compartilhem recursos computacionais e conteúdo, além de executar serviços de aplicações de maneira distribuída por meio de objetos móveis. Para criar uma infraestrutura computacional distribuída que permita a satisfação dos requisitos de redes sociais *peer-to-peer*, é necessário um conjunto de algoritmos de rede *peer-to-peer* satisfaçam requisitos como: organização da infraestrutura de computação *peer-to-peer*; desconexão programada de *peers*; desconexão não programada de *peers*; descoberta de *peers*; roteamento de mensagens entre os *peers*; disseminação de informação; e sincronização de informações de contexto do ambiente distribuído. Tanto as políticas de computação distribuída quanto os algoritmos de rede *peer-to-peer* dependem de estruturas de dados, onde informações de controle da infraestrutura de computação *peer-to-peer* e contextos de rede sociais *peer-to-peer* são mantidas.

Para uma completa compreensão sobre a estruturação do sistema distribuído e o funcionamento das políticas de computação distribuída baseadas em redes sociais, este capítulo foi dividido em quatro seções. A seção 5.1 apresenta o modelo conceitual do sistema distribuído. A seção 5.2 apresenta os algoritmos de rede *peer-to-peer* utilizados no funcionamento da rede *peer-to-peer*. A seção 5.3 apresenta as políticas de computação distribuída baseadas em redes sociais. Por fim, a seção 5.4 apresenta as considerações finais do capítulo.

5.1 MODELO CONCEITUAL DO SISTEMA DISTRIBUÍDO

A proposta apresentada por esta dissertação baseia-se em uma rede *peer-to-peer* não estruturada, onde *superpeers* e *peers* simples se conectam uns aos outros, formando uma infraestrutura computacional distribuída e descentralizada. O modelo conceitual desta infraestrutura de computação é apresentado no diagrama de classes exibido pela Figura 5.1.

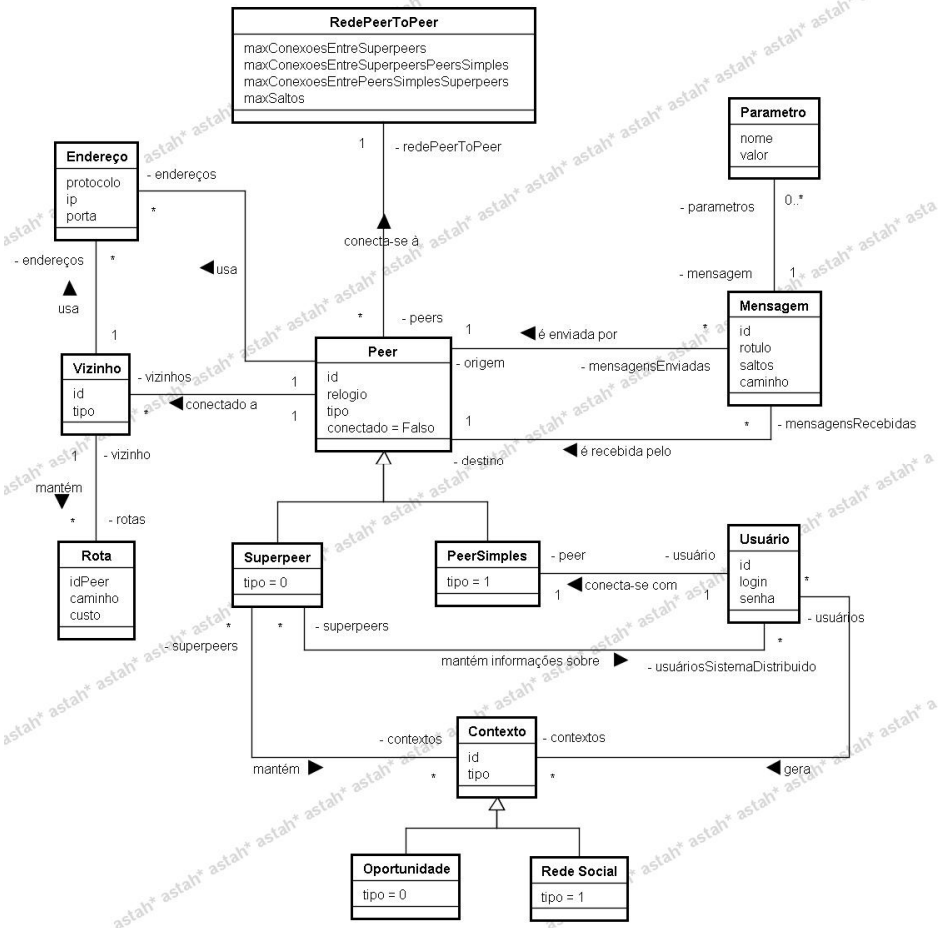


Figura 5.1: Modelo conceitual do sistema distribuído

Para organizar a formação desta infraestrutura de computação, devem ser respeitados os limites e regras para construção das conexões entre os *peers*. Os limites de conexões definem o máximo de conexões entre os tipos de *peers* definidos para a organização do sistema distribuído. Assim, o valor configurado em *maxConexoesEntreSuperPeers* define um número máximo de vizinhos *superpeers* que um *superpeer* pode suportar. Já o valor configurado em *maxConexoesEntreSuperpeersPeersSimples* define o número máximo de vizinhos *peers* simples que um *superpeer* pode suportar. Além destas configurações, ainda existe o parâmetro *maxConexoesEntrePeersSimplesSuperpeers*, que define o número máximo de vizinhos *superpeers* que um *peer* simples pode suportar. Para este trabalho, foram adotados os valores utilizados em redes Gnutella, onde um *superpeer* pode ter até 32 vizinhos *superpeers* e 30 vizinhos *peers* simples, e um *peer* simples pode ter no máximo três vizinhos *superpeers*.

De maneira geral, todo *peer* é identificado de maneira única, de modo que uma mensagem possa ser enviada a partir de uma origem e ser entregue em um destino. A identificação única dos *peers* também ajuda na configuração das informações de vizinhança dos *peers*. Além disto, cada *peer* monitora, quando conectado à rede *peer-to-peer*, valores de relógio do computador do usuário, de modo que esta informação seja utilizada em um processo de sincronização de informações de contexto no sistema distribuído. Por fim, qualquer *peer* usa endereços para enviar e receber mensagens. Cada endereço é configurado usando um protocolo de transporte, endereço IP e porta de comunicação.

As informações de vizinhança são mantidas por uma tabela de vizinhos, onde cada vizinho mantém as informações sobre *peers* conectados diretamente ao *peer* que a mantém. Tais informações consistem no identificador único, tipo e lista de endereços do *peer* vizinho. Um vizinho também é capaz de manter informações cuja finalidade é auxiliar o processo de encaminhamento de mensagens enviadas de um *peer* para outro. Estas informações compõem uma tabela de rotas, onde cada uma de suas entradas está relacionada a um *peer* conectado à rede *peer-to-peer*. Além disto, a cada entrada na tabela de roteamento está associado um conjunto de rotas, onde cada rota mantém o identificador único do *peer* de destino, o caminho a ser percorrido por uma mensagem e o custo de envio de uma mensagem sobre o caminho configurado na rota.

Quando um *peer* deseja se comunicar com outro que esteja conectado à rede *peer-to-peer*, uma mensagem é criada, contendo identificador único, rótulo, origem, destino, saltos, caminho e parâmetros. As mensagens devem conter identificadores únicos, pois esta informação é utilizada para controlar o número de mensagens encaminhadas de um *peer* para

outro, dependendo do algoritmo de rede *peer-to-peer* que estiver sendo executado em um determinado instante. O rótulo auxilia no processo de identificação e execução dos algoritmos de rede *peer-to-peer* ou na execução de mecanismos específicos da camada de comunicação, conforme apresentação no Capítulo 4. A Tabela 5.1 apresenta os rótulos de mensagens definidas para este trabalho e suas respectivas descrições.

Tabela 5.1: Rótulos de mensagens definidos para o sistema distribuído

Rótulo	**Descrição**
JOIN	Solicita conexões na rede *peer-to-peer*, a fim de criar entradas na tabela de vizinhos naqueles *peers* que aceitarem o pedido de conexão.
JOINED	Sinaliza o aceite de um pedido de conexão.
JOIN_ERROR	Rejeita um pedido de conexão.
PING	Informa a presença de um *peer* após a conexão na rede *peer-to-peer* ter sido confirmada.
PONG	Sinaliza uma resposta a mensagem cujo rótulo é PING, trazendo as informações de localização daqueles *peers* por onde uma mensagem PING passou.
ROUTE	Solicita que um *peer* passe adiante uma mensagem.
PUSH	Dissemina uma informação sobre a rede *peer-to-peer*.
ADV_OPPORTUNITY	Transporta oportunidades de colaboração criadas pelos usuários do ambiente distribuído.
DISCONNECT	Desconecta um *peer* da rede *peer-to-peer* de maneira programada.
RQS_CRT_SERVICE	Submissão de requisição para criação de informação de contexto.
RSP_CRT_SERVICE	Devolução de resposta em função do tratamento da requisição de criação de informação de contexto.
RQS_UPD_SERVICE	Submissão de requisição para atualização de informação de contexto.
RSP_UPD_SERVICE	Devolução de resposta em função do tratamento de requisição de atualização de informação de contexto.
RQS_QRY_SERVICE	Submissão de uma requisição para obtenção de informação de contexto.
RSP_QRY_SERVICE	Devolução de resposta em função do tratamento de requisição de obtenção de informação de contexto.
RQS_DEL_SERVICE	Submissão de requisição para remoção de informação de contexto.
RSP_DEL_SERVICE	Devolução de resposta em função do tratamento de requisição de remoção de informação de contexto.
RQS_DWN_CONTENT	Submissão de uma requisição para obtenção de uma parte de conteúdo.
RSP_DWN_CONTENT	Devolução de resposta em função do tratamento requisição para obtenção de uma parte de conteúdo.
RQS_MGRT_OBJ	Submissão de uma requisição para migração de um objeto móvel.
RSP_MGRT_OBJ	Devolução de resposta em função do tratamento de requisição para migração de um objeto móvel.
RQS_PROC_TASK	Submissão de uma requisição para execução de uma tarefa.
RSP_PROC_TASK	Devolução de resposta em função do tratamento de requisição para execução de uma tarefa.

Ainda versando sobre mensagens, o valor definido em saltos é utilizado para definir o número de vezes que uma mensagem pode ser repassada de um *peer* para outro. Este valor deve ser sempre igual ao tamanho do caminho percorrido pela mensagem, à medida que esta é repassada de *peer* para *peer*. Por fim, os parâmetros de uma mensagem são utilizados para transportar informações de um *peer* a outro.

Neste trabalho, os usuários se conectam no sistema distribuído por meio de *peers* simples. Além de oferecer a capacidade de conexão no sistema distribuído, os *peers* simples oferecem meios para que os usuários possam criar oportunidades de colaboração e publicá-las, de modo que estas possam requisitar a colaboração de outros usuários. Para formar uma rede social sobre a infraestrutura de computação *peer-to-peer*, basta que pelo menos um usuário colabore com aquele que criou e publicou uma oportunidade.

Oportunidades e redes sociais são contextos gerados e atualizados a partir de ações dos usuários. Assim, uma oportunidade é gerada, quando um usuário deseja conhecer outros usuários cujos perfis sociais são compatíveis com as restrições de interesses da oportunidade. Com base nisto, usuários podem formar uma rede social sobre a infraestrutura de computação e, em seguida, compartilhar as responsabilidades de publicação de oportunidade e admissão de novos membros. Isto faz com que as informações sobre uma oportunidade sejam atualizadas, à medida que um grupo de usuários se responsabiliza pela sua publicação. Os detalhes sobre a geração e atualização das oportunidades podem ser lidos nas seções 5.3.1 e 5.3.2. Após uma rede social ter sido formada por um grupo de usuários, esta possui inicialmente uma lista de usuários, cuja finalidade é manter informações de contexto sobre os membros da rede social. Á medida que os membros da rede social compartilham e usam os recursos, é necessário manter o estado de uso destes recursos, a fim de permitir a contabilidade destes. Informações de contextos sobre contabilidade de recursos compartilhados devem ser mantidas por uma informação de contexto de rede social. Apesar disto ter sido apresentado no diagrama de classes da Figura 5.1, mais detalhes sobre a composição da informação de contexto de redes sociais podem ser vistas nas seções 5.3.3, 5.3.4 e 5.3.5.

Para facilitar um processo de sincronização, estes contextos são capturados pelos *superpeers* que intermediam a comunicação entre dois ou mais *peers* simples. Além disto, os *superpeers* também são utilizados para manter informações para autenticação e autorização dos usuários no sistema distribuído. Por se tratar de um assunto amplo, nesta dissertação somente é feita uma discussão sobre questões envolvendo segurança de informação no que diz

respeito ao uso de redes *peer-to-peer* como infraestrutura de computação. Por não fazer parte do escopo deste trabalho, políticas de computação distribuída envolvendo autenticação e autorização de usuários serão propostas em trabalhos futuros.

5.2 ALGORITMOS DE REDE *PEER-TO-PEER*

Antes de apresentar as políticas de computação distribuída baseadas em redes sociais, é necessário apresentar a forma como a infraestrutura do sistema distribuído é organizada. Por isto, esta seção explica o funcionamento dos algoritmos responsáveis pelas seguintes funções: estruturação da rede *peer-to-peer*; descoberta de *peers*; roteamento de mensagens; disseminação de informações; desconexão programada de *peers*; e sincronização de informações de contexto. Todos os algoritmos apresentados nesta seção compõem os mecanismos da camada de comunicação apresentados na seção 4.5. Assim, para compor o mecanismo de descoberta e roteamento, foram definidos três algoritmos de rede *peer-to-peer*, que possuem respectivamente as seguintes responsabilidades: estruturação da rede *peer-to-peer,* descoberta de *peers,* roteamento de mensagens e desconexão de *peers*. Cada um destes algoritmos é apresentado na ordem em que foram citados anteriormente, partindo da seção 5.2.1 até 5.2.5. Além dos algoritmos de rede *peer-to-peer* relacionados ao mecanismo de descoberta e roteamento, também é apresentado o algoritmo que compõe o mecanismo de disseminação de informação da camada de comunicação do modelo arquitetural apresentado pelo capítulo 4. Este algoritmo é apresentado na seção 5.2.5. Por fim, os detalhes relacionados à autenticação e autorização no ambiente distribuído, onde cada *peer* simples necessita utilizar informações de segurança mantidas pelos seus vizinhos *superpeers*, são tratados pelo mecanismo de comunicação fim-a-fim. Vale lembrar que este mecanismo também está presente na camada de comunicação do modelo arquitetural proposta por esta dissertação de mestrado.

5.2.1 Estruturação da Rede *Peer-to-Peer*

Para criar uma infraestrutura de computação distribuída, esta dissertação optou pelo uso de redes peer-to-peer não estruturadas, pois este tipo de rede *peer-to-peer* oferece maior autonomia para ambientes de computação distribuída, uma vez que sua organização não fica limitada a uma topologia específica. Neste caso, não é necessário a intervenção do usuário, de modo que este tenha que reconfigurar toda a rede *peer-to-peer* quando esta receber um novo *peer*. Esta limitação é inerente às redes *peer-to-peer* estruturadas, pois estas são formadas a

partir de configurações pré-estabelecidas por algum administrador de sistemas como, por exemplo, a quantidade total de *peers* e os identificadores de cada *peer*. Apesar disto, redes *peer-to-peer* estruturadas, comparadas às não estruturadas, impõem uma sobrecarga menor, gerando um número menor de mensagens.

Para reduzir este problema, este trabalho utiliza o conceito de *superpeers* cujos objetivos são manter índices de metadados, comuns à rede *peer-to-peer*, e atuar como intermediários na comunicação entre *peers* simples. Para isto, é necessário, em primeiro lugar, formar uma rede entre *superpeers*, a fim de que um *superpeer* possa ter rotas para os demais *superpeers* da rede *peer-to-peer*. Complementar a isto, é necessário que *peers* simples se conectem a um ou mais *superpeers* com o objetivo de completar o processo de formação da infraestrutura de rede. Para formar a rede *peer-to-peer*, este trabalho utiliza um algortimo de rede *peer-to-peer* baseado nas especificações do protocolo de Gnutella (GNUTELLA 2009). Apesar de existir uma distinção entre os *peers* formadores da infraestrutura computacional do ambiente distribuído, o algoritmo para estruturação da rede *peer-to-peer* é o mesmo para *superpeers* e *peers* simples.

O processo de formação de uma rede *peer-to-peer* inicia quando um *peer* executa algoritmo *conectar*, que é apresentado no Apêndice C pela seção C.1. Por meio deste algoritmo, um *peer* percorre uma lista de *superpeers*, de modo que seja enviada de maneira síncrona uma solicitação para se conectar à rede *peer-to-peer*. Este envio de mensagem é parado somente quando a iteração da lista acaba ou número máximo de conexões suportado entre o *peer* e *superpeers* é atingido.

Para cada entrada na lista de *superpeers*, um *peer* envia uma mensagem *JOIN*. Ao receber esta mensagem, um *superpeer* verifica o número máximo de conexões permitido para o tipo de *peer* que a enviou. Se houver disponibilidade, o *superpeer* obtém o identificador, tipo e lista de endereços do *peer* que solicitou a conexão. Com tais informações, uma entrada na tabela de vizinhos é criada e, em seguida uma mensagem *JOINED* é enviada de volta. Caso não haja disponibilidade, o *superpeer* envia uma mensagem *JOIN_ERROR*. Este comportamento é apresentado no Apêndice C pela seção C.1. Quando um *peer* recebe uma mensagem *JOINED*, o mesmo tem a confirmação de seu pedido de conexão com a rede *peer-to-peer* por meio de um *superpeer*. Assim, o *superpeer* obtém o identificador, tipo e lista de endereços do *peer* que solicitou a conexão. Com tais informações, uma entrada na tabela de vizinhos é criada, finalizando o processo de conexão com a rede *peer-to-peer* por meio de um *superpeer*. No entanto, se um *peer* recebe uma mensagem *JOIN_ERROR*, o processo de conexão com *superpeer* emissor desta mensagem é cancelado e, em seguida, uma nova

solicitação de conexão é feita para outro *superpeer*. Ao final, se o *peer* possuir algum vizinho, o mesmo inicia o processo de descoberta de *peer* na infraestrutura de computação *peer-to-peer*, executando o algoritmo *descobrirPeers*. Este algoritmo é apresentado na próxima seção.

5.2.2 Descoberta de *Peers*

Uma vez que as tabelas de vizinhança tenham sido construídas nos *peers*, é de suma importância que cada *peer* consiga alcançar os demais *peers* conectados à rede *peer-to-peer*. Para isto, é necessário que uma rede *peer-to-peer* seja conexa, fazendo com que os peers conheçam os caminhos existentes no grafo de rede, de modo que mensagens enviadas por eles possam ser entregues a um destino com o menor custo possível. Então, para satisfazer estes requisitos de rede *peer-to-peer*, foi necessário desenvolver um algoritmo de rede *peer-to-peer* que possa viabilizar, junto com estruturas de dados apropriadas, um mecanismo de descoberta de *peers*.

A descoberta de *peers* inicia quando o algoritmo *descobrirPeers* é executado, conforme apresentado no Apêndice C pela seção C.2. A partir deste algoritmo, mensagens rotuladas com *PING* são criadas e enviadas para cada um dos vizinhos *superpeers* registrados na tabela de vizinhos do *peer*. Também é adicionado ao caminho da mensagem o identificador do *peer* gerador da mensagem.

Conforme o algoritmo apresentado no Apêndice C pela seção C.2, ao receber uma mensagem *PING*, um *superpeer* verifica se a mensagem tem seu identificador registrado no *cache* de registradores. Se verdadeiro, significa que a mensagem *PING* já foi passada adiante pelo *superpeer*. Caso contrário, o identificador da mensagem é registrado no *cache* de identificadores.

Com o objetivo de diminuir o número de saltos de uma mensagem, à medida que esta é passada adiante pelos *superpeers*, o algoritmo diminui em uma unidade o número de saltos da mensagem. Após isto, uma mensagem *PONG* é preparada com o objetivo de informar a localização do *superpeer*. Para isso, tal mensagem utiliza as informações da mensagem *PING* recebida pelo *superpeer*. Tais informações são utilizadas para guiar a mensagem *PONG* até o *peer* que originalmente criou a mensagem *PING* utilizada nesta parte do algoritmo. Além de informações como identificador, caminho e saltos, a mensagem *PONG* é configurada com um parâmetro cujo nome é *caminhoVolta*. À medida que mensagens *PING* são enviadas e, posteriormente, passadas adiante até que isto não seja mais possível, os identificadores dos *peers* são registrados em uma lista de identificadores mantida pelas mensagens. Tal lista é

chamada de caminho da mensagem. A partir desta lista de identificadores, uma mensagem *PONG* é guiada até seu destino, mas a cada vez que esta mensagem é passada adiante, um item da lista é removido. Para evitar que o caminho percorrido por uma mensagem *PONG* seja perdido, sempre que tal mensagem é enviada e, consequentemente, passada adiante, o valor removido do caminho da mensagem é adicionado no parâmetro *caminhoVolta*. Para enviar a mensagem *PONG*, o *superpeer* configura o destino da mensagem de modo que este seja o vizinho que enviou ou repassou a mensagem *PING*. Logo após isto, a mensagem *PONG* é enviada. Após enviar a mensagem *PONG*, o *superpeer* verifica se a mensagem *PING* não foi gerada por um de seus vizinhos. Se isto é verdade, é criada uma rota para o *peer* que originou a mensagem *PING* recebida pelo *superpeer*. A rota, por sua vez, é adicionada à tabela de roteamento do vizinho que repassou a mensagem para o *superpeer*, como pode ser visto no Apêndice C pela seção C.2

A propagação de mensagens *PING* pára de duas maneiras. A primeira é quando o número de saltos da mensagem chega à zero. A segunda consiste na mensagem já ter sido enviada para um conjunto de *peers*. Este segunda condição de parada foi criada para evitar ciclos no grafo de rede *peer-to-peer*. Caso o identificador de um vizinho do *superpeer* não conste no caminho da mensagem, é criada uma cópia da mensagem *PING* e, em seguida, a mesma é enviada para o vizinho. Se o vizinho for um *superpeer*, a mensagem *PING* será tratada novamente, mas no contexto de outro *superpeer*. Caso contrário, o tratamento será feito por um *peer* simples, como pode ser visto no Apêndice C na seção C.2.

Quando uma mensagem *PING* é tratada no contexto de um *peer* simples, é verificado se a mensagem *PING* não foi gerada por um de seus vizinhos. Se isto é verdade, é criada uma rota para o *peer* que originou a mensagem *PING* recebida pelo *peer* simples. A rota, por sua vez, é adicionada à tabela de roteamento do vizinho que repassou tal mensagem para o *peer* simples, como pode ser visto no Apêndice C pela seção C.2. Ao final, uma mensagem *PONG* é enviada de volta, tendo como destino o *peer* que inicialmente enviou a mensagem *PING*.

Quando um *superpeer* recebe uma mensagem *PONG*, o mesmo cria uma rota e a adiciona na tabela de roteamento do vizinho que lhe encaminhou tal mensagem. Após isto, é removido um identificador do caminho da mensagem. Este identificador é igual ao identificador do *superpeer*. Para encaminhar a mensagem *PONG* para outro *peer*, o *superpeer* verifica se o próximo identificador no caminho da mensagem é seu vizinho. Se isto for falso, o algoritmo para de executar. Caso contrário, uma cópia da mensagem *PONG* é criada, configurada e passada adiante. Se o próximo *peer* a receber a mensagem *PONG* for um *superpeer*, a mensagem será tratada novamente por um *superpeer*. Caso contrário, Se o

próximo *peer* a receber a mensagem *PONG* for um *peer* simples, o mesmo cria uma rota e a adiciona na tabela de roteamento do vizinho que lhe encaminhou tal mensagem. Detalhes de como os *peers* tratam uma mensagem *PONG*, são descritos na forma de pseudocódigo na seção C.2 no Apêndice C.

5.2.3 Roteamento de Mensagens

Após a execução dos algoritmos *descobrirPeers*, tabelas de roteamento são construídas à medida que mensagens $PING$ e $PONG$ são enviadas e repassadas entre os *peers*.

Uma vez construídas, tabelas de roteamento são utilizadas no envio de mensagens para *peers* que não são vizinhos imediatos de um emissor de mensagem. Neste caso, mensagens percorrem entre dois ou um número máximo de saltos até chegar ao seu destino. Se o destino de uma mensagem corresponde a um vizinho imediato de um *peer*, as tabelas de roteamento não são utilizadas. Para isto, o *peer* consulta sua lista de vizinhos com o objetivo de encontrar a entrada correspondente ao destino da mensagem e, em seguida, a envia ao seu destino.

Como apresentadona seção C.3 no Apêndice C, o algoritmo de roteamento em um *peer* simples não é complexo. Basicamente, o algoritmo calcula as rota cujos caminhos são definidos com o menor custo em saltos, conforme o destino da mensagem que se deseja enviar. Se existirem rotas para o destino de uma mensagem, uma delas é escolhida aleatoriamente e, em seguida, a mensagem é passada adiante por meio do vizinho responsável pela rota escolhida. Uma vez uma mensagem é encaminhada para um dos vizinhos de um *peer* simples, deve se ter em vista que o mesmo é um *superpeer*.

Sendo assim, uma vez que um *superpeer* recebe uma mensagem de um de seus vizinhos, o mesmo verifica se o rótulo da mensagem é *ROUTE*. Se for, o número de saltos da mensagem é decrementado em uma unidade e, sem seguida, é verificado se existe um vizinho do *superpeer* cujo identificador é igual ao destino da mensagem. Se verdadeiro, a mensagem é enviada diretamente para este vizinho. Caso contrário, o algoritmo calcula as rota cujos caminhos são definidos com o menor custo em saltos, conforme o destino da mensagem que se deseja enviar. Se existirem rotas para o destino de uma mensagem, uma delas é escolhida aleatoriamente e, em seguida, a mensagem é passada adiante por meio do vizinho responsável pela rota escolhida. Assim como os *peers* simples, os *superpeers* utilizam seus vizinhos *superpeers* para passar a mensagem adiante. Isto é feito até que a mensagem encontre o seu destino ou o seu número de saltos seja igual a zero. Além disso, se algoritmo de roteamento de um *superpeer* recebe uma mensagem cujo rótulo é diferente de *ROUTE*, é realizado o

cálculo para encontrar as rotas com o menor custo, como já mencionado anteriormente, e, em seguida, a mensagem que precisa ser roteada é encapsulada por uma mensagem *ROUTE* por meio de um parâmetro chamado *mensagemPeerToPeer*. Para tanto, algoritmo de roteamento cria a mensagem de roteamentento e adiciona a mensagem recebida ao parâmetro citado anteriormente. Detalhes sobre o algoritmo de roteamento dos *superpeers* podem ser vistos na seção C.3 do Apêndice C.

Por fim, quando uma mensagem *ROUTE* chega ao seu destino, é extraída da mesma a mensagem encapsulada pelo parâmetro *mensagemPeerToPeer*. Após isto, a mensagem extraída é encaminhada para o mecanismo de tratamento de mensagens, de modo que o receptor da mensagem possa reagir à mesma, realizando alguma computação e, em seguida, devolvendo uma resposta para o emissor da mensagem.

5.2.4 Desconexão de Peers

Durante o envio ou a passagem adiante de uma mensagem, um *peer* pode lidar com problemas de indisponibilidade de algum vizinho imediato, causados por uma queda de conexão, falha de equipamento, queda de energia, entre outros. Nesta situação, em que um vizinho sai abruptamente do sistema distribuído, o *peer* remove a entrada correspondente ao vizinho indisponível da lista de vizinhos e, em seguida, tenta enviar ou passar adiante a mensagem. Sempre que um *peer* (seja ele um *superpeer* ou um *peer* simples) perde algum vizinho, ele imediatamente tenta se conectar a outro vizinho. Isto é feito para evitar que o grafo de rede *peer-to-peer* se parta, causando a indisponibilidade de recursos e serviços oferecidos pelos vários *peers* pertencentes a este grafo. Portanto, quando um *peer* se conecta a outro, a fim de suprir a ausência de um vizinho, uma nova descoberta de *peers* é feita e, consequentemente, as informações de roteamento são atualizadas.

Contrário ao contexto apresentado acima, um *peer* pode se desconectar da rede *peer-to-peer* de maneira programada. Neste caso, todos os vizinhos do *peer* que está para se desconectar da rede *peer-to-peer* são notificados. Conforme o algoritmo descrito na seção C.4 do Apêndice C, a notificação dos vizinhos é feita quando o algoritmo *desconectar* é executado.

De acordo com o algoritmo descrito na seção C.4 do Apêndice C, quando a mensagem *DISCONNECT* é recebida por um *peer*, é verificado se existe um vizinho cujo identificador é igual à origem da mensagem. Se verdadeiro, a entrada na tabela de vizinhos é removida. Por se tratar de um algoritmo síncrono, uma vez que todos os vizinhos tenham

recebido a notificação de desconexão e, consequentemente, removido a entrada na tabela de vizinhos correspondente à origem da mensagem, o peer que está se desconectando da rede peer-to-peer exclui todas as entradas em sua tabela de vizinhos, conforme exibido na seção C.4 no Apêndice C.

5.2.5 Disseminação de Informação na Rede *Peer-to-Peer*

Além dos algoritmos para estruturação, e descoberta e roteamento da rede *peer-to-peer*, é necessário o uso de um algoritmo para disseminar informações sobre os usuários e oportunidades de colaboração. Portanto, uma vez que um usuário deseja publicar uma informação no sistema distribuído, é necessário que o mesmo esteja conectado a um *peer*, a fim de que a informação possa ser disseminada sobre o grafo de rede *peer-to-peer*. Para isto, é necessário existir um algoritmo e estruturas de dados apropriadas, a fim de viabilizar um mecanismo de disseminação de informação.

A disseminação de informação inicia quando o algoritmo *disseminacaoInformacao* é executado. A partir deste algoritmo mensagens contendo informações são encapsuladas por mensagens *PUSH* e, em seguida, são enviadas para a rede *peer-to-peer*, como pode ser visto na seção C.5 do Apêndice C. Para tanto, é verificado se o *peer* possui vizinhos. Se falso, o algoritmo pára de executar. Caso contrário, são executadas as seguintes ações: escolher aleatoriamente um vizinho na tabela de vizinhos do *peer*; criar uma mensagem *PUSH;* configurar a origem, destino e número de saltos da mensagem; registrar o identificador do *peer* no caminho da mensagem; e encapsular a mensagem contendo a informação para disseminação na mensagem *PUSH*. Após isto, a mensagem é enviada de maneira assíncrona para a rede *peer-to-peer* por meio do vizinho *superpeer* escolhido de maneira aleatória.

Quando uma mensagem *PUSH* é recebida por um *superpeer*, é subtraída uma unidade do valor atual do número de saltos. Após isto, se o número de saltos for igual a zero, significa que a mensagem atingiu o número máximo de saltos permitido, ocasionando a parada do algoritmo. Caso o número de saltos seja maior que zero, o *superpeer* repassa a mensagem *PUSH* para todos os vizinhos que ainda não a receberam. Esse comportamento é apresentado pela seção C.5 do Apêndice C.

Quando um *peer* simples recebe uma mensagem *PUSH*, ele compara seu identificador com o destino da mensagem. Se for igual, a mensagem contida no parâmetro *mensagemPeerToPeer* é encaminhada para tratamento. Detalhes sobre este passo podem ser vistos nos pseudocódigos apresentados na seção C.5 do Apêndice C.

5.2.6 Autenticação e Autorização

Nesta dissertação, a formação de redes sociais *peer-to-peer* depende de que usuários sejam criados no sistema distribuído, de modo que possa ser verificada a autenticidade dos mesmos, a fim de autorizar a execução de serviços e acesso a recursos compartilhados nas redes sociais *peer-to-peer*. Autorização e autenticação são conceitos fundamentais no que diz respeito a controle de acesso. Tais conceitos são distintos, mas interdependentes, de forma que a autorização de acesso a um determinado recurso ou serviço é, na verdade, dependente da autenticação. Uma vez que autenticação é o processo de determinar quem é o usuário no sistema, a autorização determina o que um usuário pode fazer. Portanto, a autorização refere-se à decisão se um usuário tem acesso garantido a um recurso ou serviço do sistema.

Tratar questões de segurança em sistemas distribuídos, cuja infraestrutura é formada por uma rede *peer-to-peer*, é uma tarefa difícil, pois as informações de usuários devem ser confidenciais. Diferentemente de sistemas distribuídos cliente-servidor, onde informações de usuários são mantidas em um índice de dados centralizado, sistemas distribuídos *peer-to-peer* precisam oferecer meios para tornar informações de controle de acesso sempre disponíveis. Isto tem como objetivo permitir que o mecanismo de controle de acesso possa autenticar o usuário e, em seguida, autorizá-lo para acessar recursos e executar serviços no sistema distribuído. Além disto, quando informações de autenticação e autorização não estão disponíveis para o mecanismo de controle acesso, usuários podem ser impossibilitados de acessar recursos e executar serviços no sistema distribuído. Para lidar com este requisito, é necessário criar um índice distribuído de informações de autenticação de usuário, de modo que os *peers* possam manter cópias deste índice, permitindo que tais informações estejam disponíveis sempre que for necessário autenticar e autorizar o usuário. Porém, esta abordagem não satisfaz os requisitos de controle de acesso plenamente, pois ela não oferece garantias de integridade e confidencialidade dos dados do usuário, uma vez que informações de autenticação de usuários podem ser obtidas por qualquer usuário, pois as mesmas não estão sob algum tipo de controle administrativo.

Para tratar isso, esta dissertação concentra informações de autenticação nos *superpeers*, pelo fato destes oferecerem suporte à cooperação entre os vários *peers* do sistema distribuído. Isto facilita o uso de técnicas para contabilidade de recursos compartilhados e autenticação de usuários do sistema. Então, pelo fato deste trabalho estar direcionado a ambientes de *e-Science*, os *superpeers* devem ser computadores que tenham alta disponibilidade e acesso restrito a pessoas autorizadas, quando se refere ao acesso a

informações de segurança. Os *superpeers*, na forma em que estamos propondo, servirão os *peers* simples normalmente, como em qualquer abordagem que separe entidades de redes *peer-to-peer* em *superpeers* e *peer* simples. Somente informações sigilosas, como aquelas utilizadas para autenticar e autorizar usuários, devem ser manipuladas por administradores de sistemas nos locais onde os *superpeers* estão fisicamente instalados. Ainda assim, existe um risco para o controle de acesso do sistema, pois arquivos de senhas ficariam expostos para os administradores.

Para evitar a violação de arquivos de senhas, este trabalho faz um *hash* irreversível dos nomes e as senhas de cada usuário do sistema. Para isto, é aplicado o algoritmo SHA-1 para criptografar o nome concatenado com a senha do usuário e, em seguida, armazenar o resultado da criptografia no arquivo de senhas. Com isto, não é possível descobrir os nomes e senhas dos usuários do sistema distribuído. Dessa forma, então, é possível fornecer um serviço básico de identificação e autorização para os usuários do sistema distribuído. No entanto, como pode ser visto na seção anterior, os *peers*, sejam estes *superpeers* ou *peers* simples, se conectam uns aos outros utilizando uma lista de *superpeers* confiáveis. Isto pode comprometer o processo de autenticação de usuário, uma vez que um *peer* nem sempre se conectará ao mesmo conjunto de *superpeers* confiáveis.

Nesse contexto, cada *superpeer* deve manter uma cópia atualizada das informações de autenticação de usuários. Porém, isto pode causar problemas de segurança ao sistema distribuído, pois as réplicas podem ser interceptadas e os dados copiados. Quando o meio utilizado para transportar informações é um meio não seguro, a informação trocada entre duas ou mais entidades computacionais deve ser criptografada e assinada digitalmente, para que se assegure a integridade e confidencialidade das informações. Isso pode ser implementado a partir do protocolo SSL, cujo objetivo é fornecer um canal seguro, ou seja, com garantia de privacidade, autenticidade dos pares e integridade de informações. Ao estabelecer a conexão, o SSL estabelece um identificador de sessão e um conjunto de algoritmos criptográficos. O algoritmo para troca de chaves será um algoritmo de criptografia de chave pública que será utilizado para enviar uma chave privada do algoritmo de criptografia de dados. Assim, o SSL utiliza-se de um algoritmo assimétrico apenas para criar um canal seguro para enviar uma chave secreta, a ser criada de forma aleatória e que será utilizada para criptografar os dados utilizando-se de um algoritmo simétrico. O algoritmo simétrico é utilizado para efetivamente criptografar os dados oriundos de alguma aplicação cliente. Por fim, o algoritmo de inserção de redundância é utilizado para garantir a integridade da mensagem. Assim, cada usuário possui um par de chaves, pública e privada utilizado no funcionamento do SSL. A chave

pública é disponibilizada a qualquer pessoa disposta a corresponder-se com o proprietário do par de chaves. A chave pública pode ser usada para ler uma mensagem assinada com a chave privada ou para criptografar mensagens que só podem ser descriptografadas com a chave privada. A segurança das mensagens criptografadas assimetricamente depende da segurança da chave privada, que deve estar protegida contra uso não autorizado.

Por se tratar de ambientes de *e-Science*, apesar deste trabalho adotar uma infraestrutura computacional descentralizada, é necessário manter atividades administrativas que promovam a segurança do ambiente. Por isto, os usuários devem ser criados por pessoas responsáveis pela administração do sistema distribuído. No entanto, apesar de terem sido investigadas as formas de implementação de mecanismos de segurança como foi mostrado acima, este trabalho não trata detalhes de segurança, ficando este aspecto para ser tratado em trabalhos futuros. Portanto, assume-se que cada usuário registrado no sistema distribuído tem suas informações de autenticação mantidas pelos *superpeers* conectados à rede *peer-to-peer*.

5.3 POLÍTICAS DE REDES SOCIAIS *PEER-TO-PEER*

Na seção anterior, apresentamos os algoritmos básicos da infraestrutura computacional criada por uma rede *peer-to-peer* não estruturada. Estes algoritmos, no entanto, não são suficientes. Deste modo, esta seção apresenta as políticas de computação distribuída que viabilizam a criação de redes sociais *peer-to-peer*, além dos compartilhamentos de hardware. Sobre estes compartilhamentos, é possível compartilhar conteúdos e armazená-los de maneira colaborativa entre alguns membros de uma rede social. Também é possível executar tarefas sobre tais compartilhamentos, a fim de processar dados que requerem computação intensiva. Cada uma das políticas apresentadas nas próximas seções é um algoritmo distribuído assíncrono, que envia requisições e recebe respostas após as requisições terem sido tratadas nos *peers*. Por se tratar de algoritmos distribuídos de um nível de abstração maior que os algoritmos de rede *peer-to-peer* apresentados na seção anterior, as políticas de computação distribuída baseadas em redes sociais são apresentadas sem que sejam abordados detalhes de baixo nível como, por exemplo, reenvio de requisições, quando respostas não são devolvidas durante um certo espaço de tempo. O tratamento deste tipo de detalhe é requisito básico para qualquer implementação de *middleware* para computação *peer-to-peer*.

5.3.1 Formação de Redes Sociais

Uma vez que a infraestrutura do sistema distribuído é formada, usuários podem se conectar a este ambiente de computação e, em seguida, formar redes sociais *peer-to-peer*. O objetivo disto é criar um ambiente de computação distribuída onde todos os membros da rede social possam compartilhar e usar tanto recursos computacionais quando conteúdos. Porém, antes de iniciar qualquer processo de formação de rede social *peer-to-peer*, cada usuário precisa registrar em seu perfil social os seus interesses. Os interesses de um usuário, como apresentado no diagrama de classes abaixo, são definidos por entradas em uma lista de interesses existente no perfil social do usuário, onde cada entrada possui um conceito utilizado para relacionar um conjunto de palavras-chaves. Por exemplo, um usuário se interessa por Bioinformática, mas isto é genérico demais e, por isso, ele deve especificar quais são as palavras-chave que delimitam o seu interesse por Bioinformática, tais como: análise genômica, DNA, genes, FASTA (FASTA 2010) e BLAST (BLAST 2010).

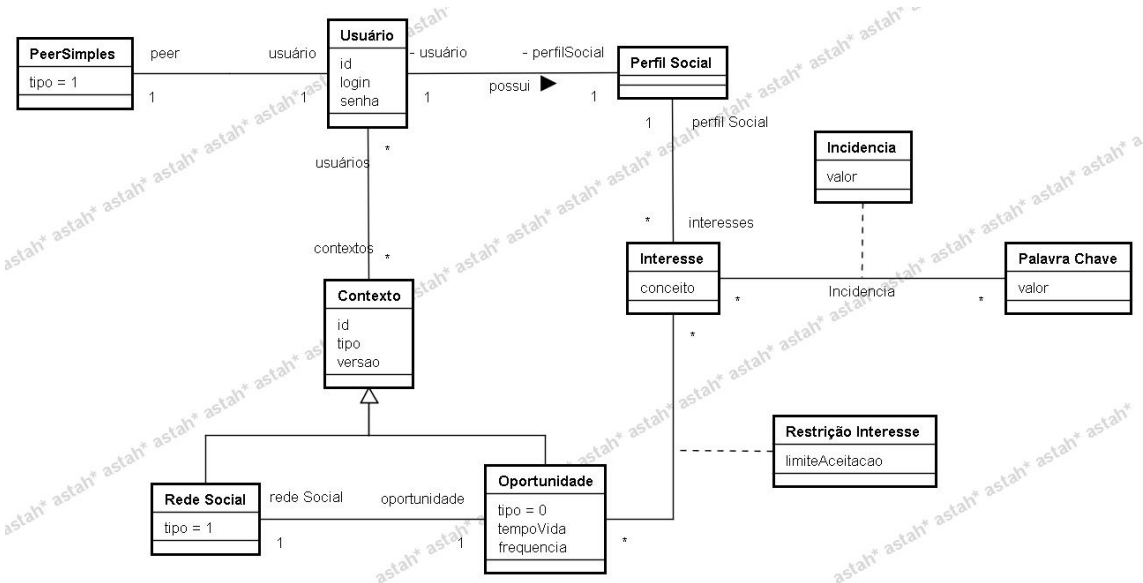


Figura 5.2: Modelo conceitual para definição de oportunidades de colaboração

A partir disso, qualquer usuário, se conectado ao sistema distribuído por meio de um *peer* simples, pode iniciar um processo de formação de rede social *peer-to-peer*. Para isto, o usuário deve especificar uma oportunidade de colaboração, definindo seu conjunto de interesses e, em seguida, disseminá-la no sistema distribuído. O modelo conceitual para definição de uma oportunidade de colaboração é definido no diagrama de classes apresentado pela Figura 5.2.

Para publicar uma oportunidade de colaboração, é criada uma mensagem *ADV_OPPORTUNITY*, já que informações para disseminação devem ser encapsuladas em uma mensagem. Por não conhecer o destino para o qual a informação será enviada e nem o tamanho do caminho percorrido, a mensagem *ADV_OPPORTUNITY* tem seu destino configurado como *Nulo* e seu número de saltos é configurado a partir do limite de saltos permitido a uma mensagem, quando esta é enviada e passada adiante de *peer* em *peer*. Como visto na seção 5.2.3, valores de destino, saltos e caminho são configurados após a mensagem ter sido extraída do parâmetro *mensagemPeerToPeer* criado em uma mensagem *PUSH*.

No instante em que uma mensagem *ADV_OPPORTUNITY* é tratada por um *peer*, é realizado um cálculo de similaridade entre os interesses contidos no perfil social do usuário do *peer* e as restrições de interesses configuradas na oportunidade de colaboração. Este cálculo tem como objetivo identificar a similaridade de pelo menos uma das restrições de interesses com um dos interesses do perfil do usuário. Se for encontrada alguma similaridade com o perfil do usuário, o usuário é notificado.

Ao receber as notificações, o usuário pode aceitar ou não a oportunidade publicada. Caso o usuário não a aceite, a oportunidade é descartada. No entanto, se o usuário aceitar pelo menos uma das notificações, é iniciado o processo de formação de uma rede social *peer-to-peer*. Quando a criação de uma rede social é requisitada, é verificado se existe uma rede social criada pela oportunidade publicada. Se não existir, a rede social é criada, os dois primeiros membros são adicionados e, em seguida, é requerida a adição da rede social no usuário requerente. Caso a rede social já tenha sido criada, um novo membro é adicionado e, logo após, é requerida atualização da lista de membros de rede social nos *peers* de cada usuário.

Por fim, detalhes sobre a descrição algorítmica da política apresentada nesta seção são descritos na forma de pseudocódigo na seção D.1 do Apêndice D.

5.3.2 Compartilhamento de Responsabilidades

Para que uma rede social *peer-to-peer* continue crescendo, é necessário que o *peer* de um usuário trate requisições para criação de redes sociais *peer-to-peer*, a fim de que possa surgir uma nova rede, caso não exista, ou adicionar novos membros à rede social. Inicialmente, o usuário responsável em criar uma nova rede social e admitir novos membros nesta rede é aquele que publicou uma oportunidade no sistema distribuído e, por isto, todas as requisições dos *peers* de outros usuários que se interessam pela oportunidade são direcionadas para um único ponto. Isto pode inviabilizar o crescimento de uma rede social *peer-to-peer*,

pois existe um único ponto responsável em tratar as requisições. Portanto, se o *peer* do usuário gerador da oportunidade sair do sistema distribuído, o crescimento de uma rede social *peer-to-peer* pode ser afetado por curto espaço de tempo ou, até mesmo, por um tempo indeterminado.

Para tratar isto,é necessário que seja adicionada uma redundância no tratamento de requisição para criação de redes sociais e adição de novos membros em uma rede. Portanto, durante o ciclo de vida de uma rede social *peer-to-peer*, aquele usuário que inicialmente gerou a oportunidade de colaboração pode convidar alguns membros da rede social para auxiliar no processo de publicação da oportunidade e tratamento de requisições para admitir novos membros.

Nesse sentido, o usuário seleciona uma das oportunidades publicadas por ele. Em seguida, envia um convite de colaboração para cada um dos membros da rede social *peer-to-peer* que ainda não compartilham as responsabilidades com o proprietário da oportunidade, a fim de que um conjunto de membros possa auxiliar na publicação e admissão de novos membros na rede *peer-to-peer*.

Quando o convite é recebido pelos membros da rede social *peer-to-peer*, cada um dos membros pode aceitar ou não o convite. Em caso de recusa do convite, a execução do compartilhamento de oportunidade é cancelada para o membro que fez a recusa. Caso contrário, é verificado se a oportunidade relacionada ao convite de compartilhamento existe como contexto no usuário. Se existir, o compartilhamento de oportunidade é cancelado, pois o usuário que recebeu este convite já compartilha a responsabilidade com o proprietário da oportunidade. Caso a oportunidade não exista como contexto no usuário, esta é, então, adicionada como contexto e, logo após, o usuário que aceitou o convite é registrado como um dos membros responsáveis pela oportunidade. Após a atualização da lista de usuários responsáveis pela oportunidade, com exceção do membro adicionado, é requerido que cada um dos membros responsáveis pela oportunidade atualize a lista de membros da oportunidade. Por fim, a oportunidade é registrada no mecanismo de disseminação de informação, que publica informações, de acordo com uma freqüência estabelecida, até que seu tempo de vida chegue à zero.

Quando cada um dos *peers* recebe uma requisição para adição de membro na lista de membros da oportunidade pela qual seus usuários são responsáveis, é verificado se a oportunidade não existe como contexto no usuário. Se verdadeiro, significa que o usuário cujo *peer* recebeu a requisição para adição de membro na lista de membros da oportunidade não é responsável pela oportunidade e, por isso, a execução do algoritmo é cancelada. Caso

contrário, a oportunidade é recuperada localmente, permitindo verificar se usuário requerente existe na lista de membros responsáveis pela oportunidade. Se após esta verificação for obtido um valor verdadeiro, a lista de membros responsáveis pela oportunidade é atualizada com o usuário requerente.

Por fim, a descrição em pseudocódigo de cada um dos passos da política descrita nesta seção pode ser vista na seção D.2 do Apêndice D.

5.3.3 Compartilhamento de Hardware

Uma vez que redes sociais *peer-to-peer* são criadas, sejam por dois ou mais usuários do sistema distribuído, estas podem se tornar ambientes computacionais, à medida que seus membros compartilham parte do hardware de seus computadores pessoais, de acordo com os objetivos da rede social. Por isto, os membros de uma rede social *peer-to-peer* podem fazer com que a rede seja um grande espaço de armazenamento, compartilhando parte dos espaços livres dos discos de cada computador pessoal. Outra possibilidade é poder fazer com que a rede social seja um grande ambiente de computação, compartilhando ciclos de processadores e espaços livres das memórias de cada um dos computadores pessoais dos membros da rede social. A Figura 5.3 apresenta uma modelagem conceitual de uma proposta de compartilhamento e uso de hardware em uma rede social *peer-to-peer*.

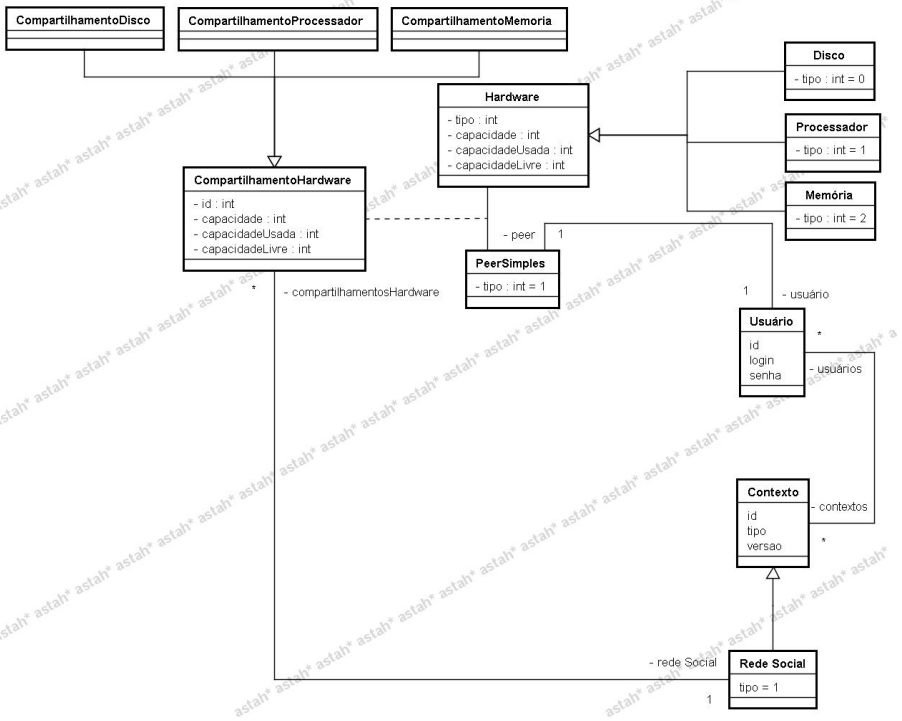


Figura 5.3: Modelagem conceitual do compartilhamento de hardware

Cada usuário pode compartilhar espaços livres de disco ou de memória, além de ciclos de processador, desde que seja membro de uma rede social. A partir disto, o usuário pode definir um compartilhamento de hardware, escolhendo o tipo de hardware e, seguida, informando a capacidade suportada pelo compartilhamento. Além disto, é necessário que o usuário escolha uma rede social, de modo que o compartilhamento seja disponibilizado para o uso na rede social escolhida. Mas, antes disto, o compartilhamento de hardware deve ser registrado no *peer* e na rede social escolhida pelo usuário, a fim de manter informações de contexto sobre uso do compartilhamento de hardware.

Para que o compartilhamento possa ser utilizado na rede social *peer-to-peer*, é necessário que todos os *peers* relacionados aos membros da rede social sejam atualizados com as informações do compartilhamento. Para tanto, para cada membro da rede social é requerida a atualização do conjunto de compartilhamentos de hardware da rede social.

Quando os *peers* dos demais membros de uma rede social recebem a requisição para atualização do conjunto de compartilhamentos de hardware, os mesmos atualizam o conjunto de compartilhamentos de hardware da rede social envolvida neste processo. Assim, é verificado primeiramente se a rede social existe para o usuário. Se falso, a execução é finalizada, pois o usuário não é membro da rede social selecionada para a adição de compartilhamento de hardware. Se verdadeiro, a informação local da rede social é obtida a partir das informações de contexto do usuário e, em seguida, o compartilhamento é adicionado ao conjunto de compartilhamentos de hardware, fazendo com que a informação de rede social seja atualizada.

Um melhor detalhamento sobre a política apresentada por esta seção é especificado por meio de pseudocódigo na seção D.4 do Apêndice D.

5.3.4 Compartilhamento de Conteúdos

Uma vez que os usuários, membros de uma mesma rede social *peer-to-peer*, tenham compartilhado parte dos espaços livres dos discos rígidos dos seus computadores, é possível armazenar conteúdos na rede social *peer-to-peer*, conforme modelo conceitual apresentado pela Figura 5.32. Então, quando um usuário deseja armazenar um conteúdo em uma rede social *peer-to-peer*, o mesmo deve requerer espaços de armazenamento para tal fim.

O usuário escolhe o conteúdo que deseja compartilhar, para, em seguida, enviar um pedido de hospedagem para aqueles membros donos de compartilhamentos capazes de manter

o conteúdo compartilhado. Para isto, é necessário que o usuário escolha uma rede social *peer-to-peer*, a fim de requerer espaços de armazenamento para o conteúdo compartilhado.

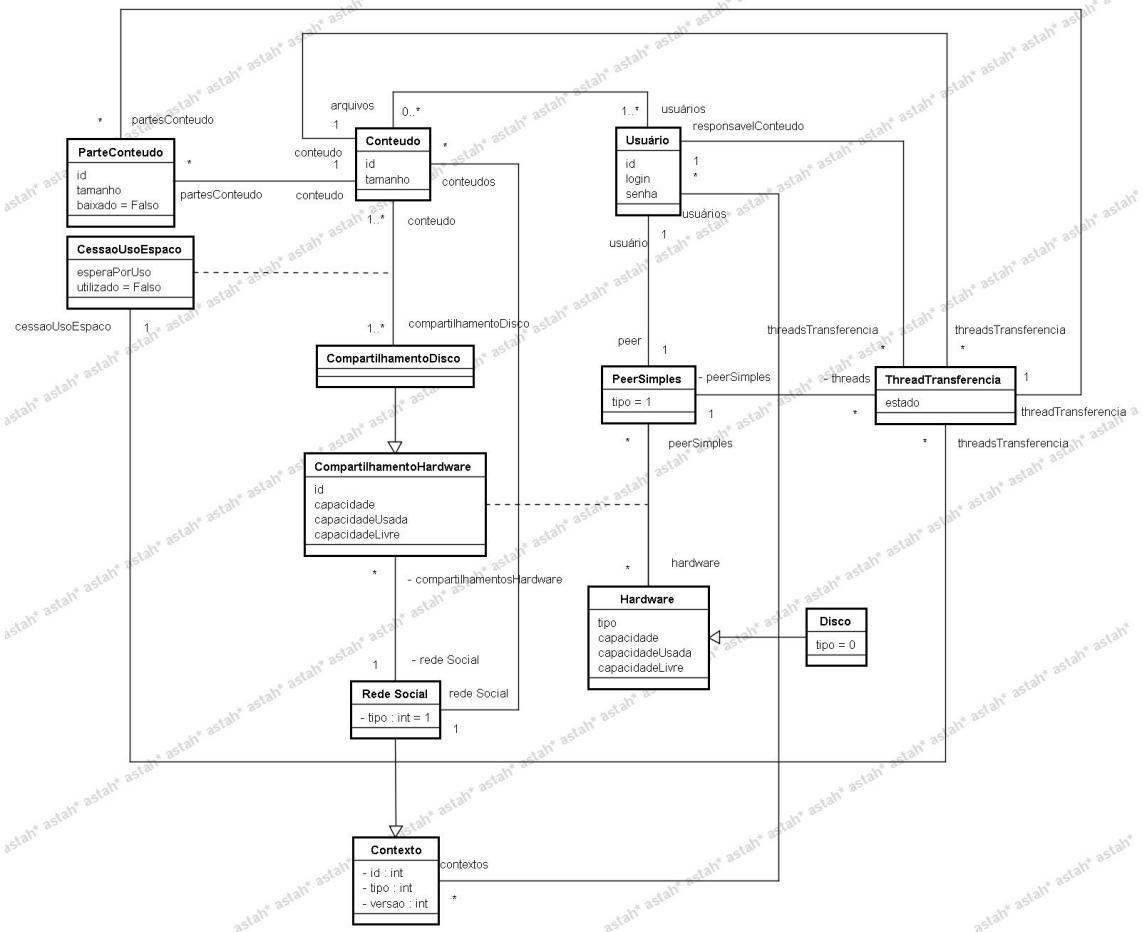


Figura 5.4: Modelagem conceitual do armazenamento de conteúdo

Uma vez que a rede social *peer-to-peer* tenha sido escolhida, é enviada uma requisição por espaço de armazenamento de conteúdo, desde que exista pelo menos um membro que tenha compartilhado espaço em disco suficiente para hospedar o conteúdo. Para cada um dos compartilhamentos de hardware da rede social *peer-to-peer,* é comparado se a sua capacidade é maior ou igual ao tamanho do conteúdo que se deseja compartilhar. Se verdadeiro, é, então, requerido ao membro responsável pelo compartilhamento de hardware a cessão de espaço para armazenamento do conteúdo.

Quando a requisição por espaço para armazenamento de conteúdo chega ao *peer* de um membro da rede social escolhida, o usuário pode aceitar ou não ceder um espaço para armazenamento no compartilhamento de disco disponibilizado por ele. Caso não aceite, é enviada para o requerente de espaço para armazenamento de conteúdo uma resposta negativa em relação à sua requisição. Mas, se o aceite for positivo, o espaço para armazenamento é

alocado no compartilhamento de disco disponibilizado pelo usuário que aceitou armazenar o conteúdo. Enquanto o conteúdo a ser armazenado não é enviado, o espaço cedido para armazenamento é configurado como não utilizado, além de ser definido um tempo de espera para utilização do espaço. Após isto, é requerido que os *peers* dos membros da rede social atualizem as informações sobre o compartilhamento de disco que teve um espaço cedido para armazenamento de conteúdo. Ao final do algoritmo de alocação de espaço de armazenamento, é iniciada a transferência do conteúdo.

Quando uma requisição para adição de cessão de uso é recebida por cada um dos *peers* dos membros da rede social, a cessão de uso de espaço é registrada no compartilhamento relacionado para atualização. Para tanto, é verificado se existe a informação de contexto da rede social. Se não existir, o algoritmo tem sua execução interrompida. Caso contrário, é feita uma nova verificação. Desta vez, é verificado se existe o compartilhamento de hardware na informação de contexto da rede social. Se falso, o algoritmo pára sua execução. Se verdadeiro, o compartilhamento de hardware é recuperado e, em seguida, a cessão de uso de espaço é adicionada.

Após o espaço de armazenamento ter sido alocado no compartilhamento, a transferência do conteúdo deve ser iniciada, de modo que cada parte do conteúdo compartilhado possa ser copiada e armazenada no espaço compartilhado. Neste caso, se dá início a transferência de conteúdo. Apesar de uma solicitação de armazenamento partir de um único usuário, o algoritmo a transferência de conteúdo pode se dar em outras ocasiões em que seja necessária a transferência de conteúdos entre os *peers*.

Durante a transferência de conteúdo, as partes de um conteúdo a ser armazenado são copiadas a partir das fontes que mantêm o conteúdo armazenado na rede social *peer-to-peer*. O número de partes de um conteúdo é calculado por meio da seguinte função:

$$calcNumPartes(tmnCntd, tmnParte) = \frac{tmnCntd}{tmnParte},$$

onde: *tmnCntd* é o tamanho do conteúdo a ser copiado e *tmnParte* é o tamanho de cada parte que será copiada do conteúdo compartilhado. Para este trabalho, *tmnParte* é configurado com valor igual a 2 KB.

Uma vez que o numero total de partes do conteúdo tenha sido calculado, deve haver um balanceamento de carga para que as partes sejam transferidas. Para este balanceamento de carga, é feito uma divisão do número de partes, de modo que cada membro mantenedor de uma cópia do conteúdo seja responsável por transferir um subconjunto de partes do conteúdo.

O cálculo do número de partes para cada membro mantenedor do conteúdo é feito por meio da seguinte função:

$$calcPartesPrpt(tmnCntd, tmnParte, numPrpts) = \frac{tmnCntd}{tmnParte} \times \frac{1}{numPrpts},$$

onde: *tmnCntd* é o tamanho do conteúdo compartilhado, *tmnParte* é o tamanho de cada parte que será copiada e *numPrpts* é o número de usuários que possuem uma cópia do conteúdo armazenado.

Uma vez que o número de partes para cada membro responsável pelo conteúdo tenha sido definido, para cada um dos *peers* destes usuários é definido um subconjunto do conjunto das partes do conteúdo. Para cada um dos *peers* responsáveis em transferir partes do conteúdo é criado uma *thread*, de modo que cada subconjunto de partes de conteúdo possa ser transferido e armazenado no espaço de disco cedido pelo usuário de um compartilhamento de disco. É importante ressaltar que cada parte de conteúdo é identificada de 1 até o número total de partes. A identificação de cada parte de conteúdo é necessária, uma vez que esta é utilizada para definir a posição inicial e final da sequência de caracteres extraída do conteúdo, quando uma parte é requerida por uma das *threads* mencionadas anteriormente.

Durante a transferência de conteúdo, o número de partes para cada membro mantenedor de um conteúdo pode mudar em duas situações: (i) o *peer* de um membro responsável por um conteúdo é desconectado da rede *peer-to-peer;* (ii) há um aumento no número de membros contendo cópias do conteúdo. Em ambas as situações, o algoritmo *distribuirPartesConteudo* é executado a fim de redistribuir o conjunto de partes de acordo com a quantidade de usuários responsáveis em manter o conteúdo. Portanto, todas as *threads* são interrompidas e removidas.

Tendo em vista que um conteúdo é um arquivo e este, por sua vez, é um grande conjunto de *strings*, uma parte é recuperada por meio de uma função que retorne um subconjunto de caracteres. Para isto, é necessário calcular a posição do primeiro e do ultimo caractere, pois elas determinam o início e o fim da *substring* que se deseja obter. A posição do primeiro caractere é calculada por meio da seguinte função:

$$calcPosicaoPrim(idParte, tmnParte) = \left((tmnParte \times idParte) \quad tmnParte\right) \mid 1,$$

onde: $idParte$ é o identificador da parte que se deseja copiar e $tmnParte$ é o tamanho da parte. Para encontrar o último caractere, é utilizada a seguinte fórmula:

$$calPosicaoUlt(idParte, tmnParte) = tmnParte \times idParte,$$

onde: $idParte$ é o identificador da parte que se deseja copiar e $tmnParte$ é o tamanho da parte. Por fim, quando o primeiro e o ultimo caractere são obtidos, uma *substring* é extraída e enviada para o *peer* que a requereu como parte de conteúdo.

Quando uma parte chega ao *peer* que a requereu, é verificado se o espaço cedido pelo usuário proprietário do compartilhamento de disco ainda está configurado como não utilizado. Caso isto seja verdade, o espaço cedido muda de estado, passando de não utilizado para utilizado. Além disto, o tempo de espera para uso é configurado com valor zero. Logo após, toda a rede social *peer-to-peer* é notificada e atualizada em função da atualização feita nas propriedades do espaço cedido pelo usuário proprietário do compartilhamento de disto. Após esta etapa, a parte recebida pelo *peer* é armazenada, assim como as outras que serão recebidas até o término da transferência do conteúdo. Por fim, após uma parte ter sido armazenada, a *thread* atualiza seu subconjunto de partes, removendo a entrada correspondente à parte recebida e armazenada no *peer*. A transferência de conteúdo é finalizada quando todas as *threads* recebem todas as partes de conteúdo relacionadas às entradas de seus subconjuntos de partes de conteúdo.

Após o término da transferência de conteúdo em um *peer*, é necessário atualizar a lista de conteúdos armazenados pela rede social *peer-to-peer*, de modo que todos os membros da rede social possam ser informados do novo conteúdo mantido pela rede. Por isto, é verificado localmente se a rede social possui informações de contexto relacionadas ao conteúdo recém armazenado. Se verdadeiro, para cada *peer* cujos usuários são membros da rede social é requerida a adição de um novo usuário na lista de membros responsáveis em manter o conteúdo armazenado. Caso contrário, é criada uma informação de contexto relacionada ao conteúdo, tendo o requerente do armazenamento do conteúdo e o usuário cujo *peer* terminou a transferência como os dois primeiros membros responsáveis pelo armazenamento do conteúdo. Em seguida, a informação de contexto é enviada para cada um dos *peers* cujos usuários são membros da rede social, de modo que ela seja adicionada a lista de conteúdos armazenados na rede social.

Por fim, todo o pseudocódigo, que especifica a política de computação distribuída descrita até aqui, é apresentado na seção D.4 do Apêndice D.

5.3.5 Execução Distribuída de Serviços de Aplicação

Uma vez que os usuários, membros de uma mesma rede social *peer-to-peer*, tenham compartilhado espaço livre em disco, ciclos de processadores e espaço livre em memória, é

possível que os mesmos possam ser utilizados para executar tarefas, de modo que a rede social *peer-to-peer* possa atuar como ambiente de processamento distribuído. Então, quando um usuário deseja executar tarefas sobre os compartilhamentos de processadores e memórias em uma rede social *peer-to-peer*, o mesmo deve solicitar aos donos destes compartilhamentos o uso de parte ou total do hardware compartilhado.

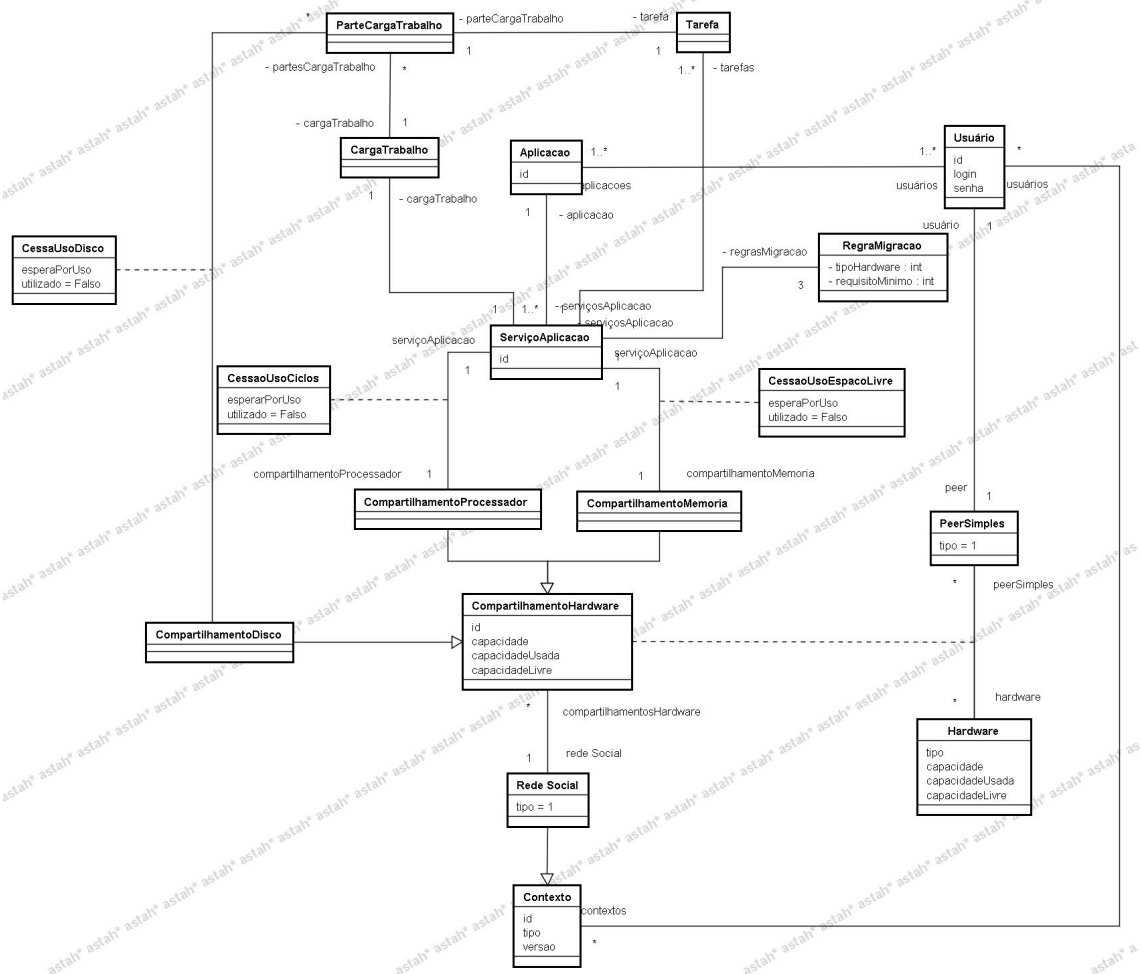


Figura 5.5: Modelagem conceitual da execução distribuída de tarefas

Semelhante ao compartilhamento de conteúdos, onde membros de uma rede social *peer-to-peer* cedem parte dos compartilhamentos de disco, a execução distribuída de serviços de aplicação necessita que membros de uma rede social *peer-to-peer* cedam parte dos compartilhamentos de espaço de disco, ciclos de processador e espaço de memória. Para isto, é necessário que o usuário escolha uma rede social *peer-to-peer*, a fim de enviar uma requisição por processamento distribuído aos demais membros da rede social, que, neste caso, serão futuros executores de tarefas. Assim, o usuário deve selecionar o serviço de aplicação que terá suas tarefas distribuídas e executadas sobre os compartilhamentos de ciclos de

processadores e espaços de memórias. Além de ciclos de processadores e espaço em memória, as tarefas necessitam de espaço de disco, pois as mesmas precisam armazenar temporariamente as partes de carga de trabalho que elas processam.

Uma vez que um usuário tenha escolhido um serviço, as informações deste são recuperadas a partir de uma aplicação, conforme o modelo conceitual para execução distribuída de serviços de aplicações apresentado pela Figura 5.5. Após o serviço ter sido selecionado pelo usuário, é requerido aos membros de uma rede social cessões de uso de recursos compartilhados. Para tanto, é verificado se existe algum membro na rede social que satisfaça as regras de migração do serviço de aplicação. Se o membro de rede social for encontrado, é requerida a cessão de uso de compartilhamentos de hardware.

Quando um membro de rede social recebe o pedido de cessão de uso de compartilhamento de hardware, o mesmo pode aceitar ou não ceder os recursos necessários para execução de tarefas em seu *peer*. Caso não aceite, é enviada para o requerente de recursos para processamento uma resposta negativa em relação à sua requisição. Mas, se o aceite for positivo, os recursos são alocados, de modo que atendam às regras de migração do serviço de aplicação.

Enquanto as tarefas não são enviadas, as cessões de compartilhamento de hardware são configuradas como não utilizadas, além de serem definidos os tempos de espera para utilização dos compartilhamentos. Após isto, é requerido que os *peers* dos membros da rede social atualizem as informações sobre os compartilhamentos cedidos para processamento de serviço de aplicação.

Após receber as confirmações de cessão de compartilhamentos de hardware, o usuário interessado em submeter um serviço de aplicação seleciona a carga de trabalho que será processada, de modo que esta possa ser dividida em partes, com o objetivo de serem processadas pelos *peers* que cederam recursos para processamento do serviço de aplicação. Então, quando o usuário solicita a execução do serviço de aplicação com a carga de trabalho selecionada, primeiramente, os *peers* dos membros cedentes de recursos para processamento são configurados com os objetos móveis que contêm o código para tratar partes da carga de trabalho. Uma vez que todos os *peers* tenham sido configurados, começa o processo de envio de tarefas para serem processadas por cada um dos objetos móveis existentes nos *peers* dos cedentes de recursos.

Quando a primeira tarefa é recebida por cada um dos *peers* responsáveis em processá-las, é feita a confirmação das cessões de uso de compartilhamento de hardware nas informações de contexto da rede social *peer-to-peer*, onde o serviço de aplicação está sendo

executado de maneira distribuída. Este processo é semelhante ao realizado no compartilhamento de conteúdo na rede social *peer-to-peer*. A diferença é que no compartilhamento de conteúdo esta atualização é feita somente para cessões de compartilhamento de disco e, para execução distribuída de serviços de aplicação, a atualização é feita para as cessões de compartilhamento de disco, ciclos de processador e memória. Quando tarefas são enviadas para os *peers* responsáveis em processá-las, as mesmas são: (i) armazenadas nos espaços de disco cedidos para uso no processamento distribuído do serviço de aplicação; (ii) carregadas em memória pelos objetos móveis, que processam as partes da carga de trabalho submetida durante a requisição de execução de um serviço de aplicação; (iii) processadas pelos objetos móveis, fazendo com que os ciclos de processadores cedidos para processamento distribuído sejam utilizados; (iv) removidas após o término da computação realizada pelos objetos móveis que as processaram.

Após a submissão das tarefas, o *peer* em que se encontra a carga de trabalho comporta-se como *master*, esperando os resultados dos processamentos de tarefas, enquanto os *peers* que recebem as tarefas comportam-se como *workers*, processando as partes de cargas de trabalho que lhes são enviadas. Quando o resultado de uma tarefa não é retornado pelo *peer* responsável em processá-la, é realizado um número determinado de tentativas e, caso não haja sucesso, a tarefa é enviada para processamento em outro *peer*.

Uma vez que a carga de trabalho tenha sido totalmente processada, os resultados obtidos devem ser agregados, a fim de compor algum resultado esperado pela execução de um serviço de aplicação. Além disso, os recursos cedidos podem ser utilizados para executar novamente os mesmos serviços de aplicação para o qual foram destinados. Uma vez que os recursos cedidos não são mais necessários, os mesmos são desalocados e, em seguida, as informações de contexto da rede social são atualizadas novamente, a fim de que outros serviços de aplicação possam ser beneficiados com novas cessões de recursos compartilhados de hardware.

5.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a principal colaboração deste trabalho, que é a definição de políticas de computação distribuída baseadas em redes sociais, de modo que estas possam permitir: a formação de redes sociais *peer-to-peer*, o compartilhamento de hardware, o compartilhamento e armazenamento de conteúdos em uma rede social *peer-to-peer* e, por fim, a execução distribuída de serviços de aplicação.

Para atingir os objetivos desta dissertação, é necessário avaliar experimentalmente os principais algoritmos de rede *peer-to-peer* e as políticas de computação distribuída baseadas em redes sociais, objetivando a obtenção de resultados que permitam considerar ou não o uso de redes sociais *peer-to-peer* como uma alternativa para o compartilhamento de recursos em ambientes computacionais de *e-Science*.

6 AVALIAÇÃO EXPERIMENTAL

Durante o desenvolvimento deste trabalho de pesquisa, foi necessário implementar um simulador de eventos discretos, a fim de permitir a prototipagem e simulação dos algoritmos de redes *peer-to-peer* e políticas de computação distribuída baseada em redes sociais. A partir desse simulador, foi possível realizar os experimentos descritos neste capítulo, a fim de obter resultados que pudessem atender às expectativas com relação aos objetivos do trabalho de pesquisa desta dissertação, que é o estudo sobre o uso de redes sociais *peer-to-peer* no compartilhamento de recursos em ambientes de *e-Science*.

Portanto, neste capítulo, serão apresentados os detalhes sobre a preparação dos experimentos, no que tange a definição do tamanho da infraestrutura de rede *peer-to-peer*, a geração aleatória da topologia da infraestrutura do sistema distribuído, limites de conexões entre os *peers*, numero máximo de saltos, tamanho da banda utilizada pelos *peers*, disponibilidade de banda, tamanhos de discos, capacidade de processadores, tamanho de memórias, disponibilidade de espaço livre em disco, disponibilidade de ciclos de processadores, disponibilidade de espaço livre em memória, distribuição dos tempos de chegada de novos *peers*, distribuição dos tempos de sessão no sistema, distribuição dos tempos em que os *peers* ficam fora da rede e, por fim, a duração da simulação. Além disso, também são apresentados os resultados dos experimentos juntamente com suas respectivas análises e conclusões obtidas.

6.1 SIMULADOR

Para avaliar o comportamento e obter uma prova de conceito de uma ambiente de computação distribuída, construído a partir da arquitetura de software, algoritmos de rede *peer-to-peer* e políticas de computação distribuída baseadas em redes sociais apresentados ao longo desta dissertação, foi desenvolvido um simulador de redes *peer-to-peer* que oferecesse suporte à prototipagem e simulação de redes sociais *peer-to-peer*. Diversos motivos motivaram o desenvolvimento de um simulador próprio, são eles: garantir todos os requisitos do tipo de ambiente distribuído que se deseja criar a partir dos resultados desta dissertação; permitir a implementação dos componentes arquiteturais de software propostos na arquitetura

apresentada no Capítulo 4; permitir a implementação de protocolos de comunicação *peer-to-peer*; permitir implementar políticas de computação distribuída baseadas em redes sociais *peer-to-peer*; facilitar o encadeamento de todas as fases envolvidas na simulação e obtenção de resultados de acordo com os objetivos definidos para este trabalho de pesquisa.

O simulador foi desenvolvido em Python, sendo constituído por três módulos principais, a saber: gerador de eventos, motor de simulação e rede *peer-to-peer*. O gerador de eventos permite basicamente a criação de eventos cujo objetivo é determinar os instantes em que alguma ação acontece durante o período configurado para simulação. Com este objetivo, o módulo de geração de eventos fornece componentes básicos que permitem distribuir instantes de tempos, baseados em distribuições de probabilidades, de modo que uma topologia de rede possa ser gerada a partir das entradas freqüentes de *peers*.

A geração dos instantes em que novos *peers* entram pela primeira vez no sistema distribuído determina somente em que ponto da linha de tempo definida para a simulação o evento irá acontecer. Os detalhes sobre a organização da topologia de rede são tratados pelo módulo de rede *peer-to-peer*. A princípio, os eventos de entrada de novos *peers* no sistema distribuído seguem uma distribuição de Poisson (POISSON 2010), que por sua vez permite a configuração da média e a freqüência com que os *peers* chegam pela primeira vez ao sistema distribuído.

Além dos eventos de chegada de novos *peers*, o módulo gerador de eventos conta com mais dois geradores de eventos: um para geração de eventos de desconexão e outro para geração de eventos de retorno ao sistema distribuído. O gerador de eventos de desconexão distribui aleatoriamente os instantes de tempo em que os eventos de desconexão de *peers* irão acontecer, seguindo uma distribuição de Weibull (WEIBULL 2010, STUTZBACH e REJAIE 2006) . A partir dos parâmetros desta distribuição, são estipulados os tempos de vida que cada *peer* terá, a partir do momento em que este se conecta ao sistema distribuído. Para gerar os tempos em que os *peers* se mantêm desconectados, o módulo de geração de eventos oferece um gerador de eventos capaz de distribuir os instantes de tempo em que os *peers* se mantêm *off-line*, segundo uma distribuição de Pareto (PARETO 2010, STUTZBACH e REJAIE 2006). Os números gerados aleatoriamente por meio desta distribuição permitem calcular o momento em que os *peers* podem se reconectar ao sistema distribuído. A partir dos tratamentos dos eventos gerados por cada um dos geradores de eventos oferecidos pelo simulador, são gerados os dados de saída de simulação, permitindo a extração de dados que futuramente poderão auxiliar na composição de uma geração de resultados.

O motor de simulação é responsável pela execução do processo de simulação propriamente dito, retirando e submetendo para tratamento os eventos de simulação gerados pelo módulo gerador de eventos. Dessa forma, os eventos são retirados conforme a evolução do relógio de simulação e, em seguida, são submetidos para tratamento, ocasionando a execução de algum comportamento programado.

O módulo de rede *peer-to-peer* é responsável pelo fornecimento de componentes e algoritmos relacionados às operações de redes *peer-to-peer* e políticas de computação distribuída. Além disto, também são oferecidos componentes que permitem gerenciar a topologia da rede *peer-to-peer* que tenha sido criada por algum protocolo de rede configurado na simulação. Atualmente, o módulo de rede *peer-to-peer* contempla somente um protocolo de rede não estruturada, baseado no Gnutella (GNUTELLA 2009). Apesar disso, é possível desenvolver outros protocolos, de modo que estes possam ser utilizados nos processos de simulação que necessitem de topologias específicas para redes *peer-to-peer*.

Outro ponto importante, no que diz respeito ao módulo de redes *peer-to-peer,* é a possibilidade de criar protocolos para aplicações ou políticas de computação distribuída. Isto é feito através da implementação de mensagens com rótulos específicos e tratadores de mensagens, que permitem a execução de algum comportamento quando um *peer* recebe uma mensagem de um de seus vizinhos.

Por fim, o módulo de rede *peer-to-peer* ainda contempla um conjunto de componentes que implementam de maneira básica o funcionamento dos recursos computacionais existentes em um *peer*. A partir destes componentes de recursos computacionais, são configuradas as quantidades disponíveis para cada tipo de recurso existente em um *peer*. Nesta versão do simulador de rede, foram criados somente componentes para implementar as funcionalidades de discos, processadores e memórias.

Além dos módulos, o simulador de rede também oferece um conjunto de parâmetros de entrada para configuração das simulações, são eles:

- Número de *superpeers*;
- Número de *peers* simples;
- Número de conexões entre *superpeers*;
- Número de conexões entre *superpeers* e *peers* simples;
- Número de conexões entre *peers* simples e *superpeers*;
- Largura de banda dos *superpeers*;
- Largura de banda dos *peers* simples;

- Tamanhos de discos em bytes;
- Tamanhos de memórias em bytes;
- Capacidades de processadores em ciclos por segundo;
- Disponibilidade de largura de banda;
- Disponibilidade de espaço livre em disco;
- Disponibilidade de espaço livre em memória;
- Protocolos de rede *peer-to-peer*;
- Número máximo de saltos dados por mensagens na rede *peer-to-peer*;
- Taxa de chegada de *superpeers*, segundo uma distribuição de poison;
- Taxa de chegada de *peers* simples, segundo uma distribuição de poison;
- Distribuição dos tempos de vida dos *peer* simples, segundo uma distribuição de Weibull*;*
- Distribuição dos tempos em que *peers* simples ficam *off-line*, segundo uma distribuição de Pareto;
- Duração de uma simulação;

Portanto, por meio dos módulos e dos parâmetros de configuração de simulação do simulador de rede, foi possível preparar cada um dos cenários de execução dos experimentos apresentados neste capítulo.

6.2 PREPARAÇÃO DOS EXPERIMENTOS

Para a execução dos experimentos no simulador, foi necessário definir previamente os objetivos dos experimentos e planejar execução de cada um deles, selecionando os geradores de eventos para cada cenário de simulação que se deseja executar, além dos parâmetros relacionados a cada um dos cenários criados e executados.

6.2.1 Objetivos

Antes de executar um experimento é necessário definir os objetivos que se deseja alcançar com o mesmo, a fim de garantir que os seus resultados sejam relevantes. Uma vez que estes objetivos são definidos, as simulações são criadas e executadas e, em seguida, os resultados destas são coletados e analisados. Portanto, os objetivos foram definidos, a fim de avaliar os seguintes aspectos:

a. **Quanto ao ambiente distribuído:**

1. Avaliar o *overhead* gerado pelos algoritmos de rede *peer-to-peer* e políticas de computação distribuída baseadas em redes sociais, capturando o número de mensagens geradas;
2. Avaliar o consumo de banda causado pelas mensagens criadas durante a execução dos algoritmos de redes *peer-to-peer* e políticas de computação distribuída, contabilizando a quantidade de bytes transferidos sobre a disponibilidade da banda dos *peers*;
3. Avaliar o tempo em que as mensagens ocupam a banda disponível nos *peers*, à medida que os algoritmos de redes *peer-to-peer* e políticas de computação distribuída são executados ao longo da simulação;
4. Avaliar a média de saltos dados pelas mensagens, à medida que as mesmas são trocadas entre os *peers*.

b. **Quanto à rede social *peer-to-peer***:

1. Avaliar a quantidade de recursos agregados à medida que o tamanho da rede social *peer-to-peer* aumenta;
2. Avaliar o impacto causado pela variação do número de *peers* no sistema distribuído na formação e evolução de uma rede social *peer-to-peer*;
3. Avaliar o tempo gasto para transmissão de um conteúdo compartilhado entre membros de uma rede social *peer-to-peer*, variando a disponibilidade de banda e o número de membros detentores do mesmo conteúdo;
4. Avaliar a número de tarefas por segundo que uma rede social *peer-to-peer* é capaz de executar, variando a disponibilidade de banda entre os membros da rede social.

A partir desses objetivos, foi possível nortear todo o planejamento dos experimentos, permitindo elaborar a metodologia de execução, que é repetida para cada experimento realizado neste trabalho.

6.2.2 Definição dos Cenários de Simulação

Analisando os objetivos na seção 6.2.1, foram projetados sete cenários, sendo dois destes com variações, devido à necessidade de analisar o impacto de saídas e retornos dos *peers* ao sistema distribuído. Portanto, os cenários são os seguintes:

1. **Estabilização da rede *peer-to-peer***: os *peers* formam a infraestrutura do ambiente de computação, sem apresentar falhas. Com este cenário, pretende-se obter os seguintes resultados: (i) quantidade de mensagens geradas à medida que os *peers* se conectam à rede *peer-to-peer*; (ii) consumo de banda disponível nos *peers*, à medida que mensagens são trocadas entre os *peers* ao longo do processo de estabilização; (iii) tempo em que as mensagens consomem a banda disponível nos *peers*;
2. **Formação de uma rede social *peer-to-peer* em uma rede *peer-to-peer* estável, sem compartilhamento de responsabilidade**: uma oportunidade de colaboração será criada e disseminada sobre o grafo de rede *peer-to-peer*, a fim de arregimentar *peers* para a formação e expansão de uma rede social. Neste cenário, pretende-se obter resultados que permitam determinar um número médio de membros de rede social e a quantidade de recursos compartilhados por estes membros, uma vez que a publicação da oportunidade de colaboração e a admissão de membros na rede social serão feitas por um único *peer*;
3. **Formação de uma rede social *peer-to-peer* em uma rede *peer-to-peer* estável, com compartilhamento de responsabilidade**: uma oportunidade de colaboração será criada e disseminada sobre o grafo de rede *peer-to-peer*, a fim de arregimentar *peers* para a formação e expansão de uma rede social. A partir deste cenário, pretende-se obter resultados que permitam determinar um número médio de membros de rede social e a quantidade de recursos compartilhados por estes membros, uma vez que a publicação da oportunidade de colaboração e a admissão de membros na rede social serão feitas pelos membros da rede social;
4. **Formação de uma rede social *peer-to-peer* em uma rede *peer-to-peer* instável, sem compartilhamento de responsabilidade**: uma oportunidade de colaboração será criada e disseminada sobre o grafo de rede *peer-to-peer*, a fim de arregimentar *peers* para a formação e expansão de uma rede social. Com este cenário, pretende-se obter resultados que permitam determinar um número médio de membros de rede social e a quantidade de recursos compartilhados por estes membros, uma vez que a publicação da oportunidade de colaboração e a admissão de membros na rede social serão feitas por um único *peer*.

5. **Formação de uma rede social *peer-to-peer* em uma rede *peer-to-peer* instável, com compartilhamento de responsabilidade**: uma oportunidade de colaboração será criada e disseminada sobre o grafo de rede *peer-to-peer*, a fim de arregimentar *peers* para a formação e expansão de uma rede social. Com este cenário, pretende-se obter resultados que permitam determinar um número médio de membros de rede social e a quantidade de recursos compartilhados por estes membros, uma vez que a publicação da oportunidade de colaboração e a admissão de membros na rede social serão feitas pelos membros da rede social.
6. **Compartilhamento de conteúdo entre membros de uma rede social *peer-to-peer*:** um conteúdo será compartilhado na rede social *peer-to-peer* e será realizado o *download* do mesmo por parte dos membros durante um período de tempo, até que todos os membros da rede tenham armazenado de maneira colaborativa o conteúdo compartilhado. Com este cenário, pretende-se obter resultados que permitam avaliar o desempenho do uso de uma rede social *peer-to-peer* em um processo de compartilhamento de conteúdo, incluindo a transferência de um conteúdo compartilhado para espaços de armazenamento na rede social *peer-to-peer*.
7. **Execução de tarefas sobre os compartilhamentos de tempos de processamento e espaços de memória:** será submetido para processamento distribuído um *job* com carga de trabalho divisível, que terá suas tarefas executadas sobre os compartilhamentos de tempo de processamento e espaços de memória em cada um dos computadores pertencentes aos membros da rede social. Neste cenário, pretende-se obter resultados que permitam apontar a viabilidade do uso de uma rede social *peer-to-peer* como infraestrutura computacional para execução de tarefas.

A partir dos objetivos traçados e dos cenários projetados, podemos definir uma metodologia de execução dos experimentos, a fim de obter resultados capazes de serem analisados de acordo com os objetivos definidos.

6.2.3 Metodologia de Execução

Para que os resultados dos experimentos sejam relevantes aos objetivos apresentados anteriormente, é necessário elaborar um planejamento das atividades para que seja possível extrair os resultados e, em seguida, analisá-los.

Nesse sentido, a metodologia de execução dos experimentos consiste na execução das simulações de cada um dos cenários apresentados na Seção 6.2.2,de maneira repetitiva, de modo que cada cenário e/ou situações específicas de cenário sejam executados dez vezes, totalizando 160 execuções de simulação.Como visto antes, cada cenário possui um propósito, o que permitirá a comparação dos resultados, a fim de permitir fazer uma avaliação do desempenho das políticas de computação distribuída baseadas em redes sociais. Então, para cada execução de simulação, são gerados arquivos de texto com os resultados gerados na simulação. Este formato pode ser aberto em uma planilha eletrônica para análise dos dados e criação de gráficos dos resultados, como foi feito neste trabalho.

6.2.4 Configuração das Simulações

As configurações utilizadas nas simulações foram criadas levando em consideração a infraestrutura computacional apresentada na Tabela 6.1.

Tabela 6.1: Configuração da infraestrutura do ambiente distribuído

Parâmetro	**Configuração**
Protocolo da rede *peer-to-peer*	Gnutella
Nº de *superpeers*	100 *superpeers*
Nº de *peers* simples	1000 *peers* simples
Conexões entre *superpeers*	32 conexões
Conexões entre *superpeers* e *peers* simples	30 conexões
Conexões entre *peers* simples e *superpeers*	3 conexões
Banda passante dos *superpeers*	1 Gbps
Banda passante dos *peers* simples	1 Mbps
Tamanhos dos discos dos *peers* simples	120 GB, 160 GB, 250 GB e 320 GB
Velocidades dos processadores dos *peers* simples	1.5 GHz, 2.0 GHz, 3.0 GHz, 4.0 GHz
Tamanhos das memórias dos *peers* simples	1 GB, 2 GB, 3 GB e 4 GB
Número máximo de saltos dados por uma mensagem	7 saltos

Para definir os instantes de entrada de novos *peers* no ambiente distribuído, foram utilizadas duas configurações distintas, uma para a entrada de novos *superpeers* e outra para

entrada de novos *peers* simples. Dessa forma, a entrada de novos *superpeers* segue uma distribuição de Poisson com uma média que varia de 1 a 3 *superpeers* a cada minuto e a entrada de novos *peers* simples também segue uma distribuição de Poisson, mas com uma média que varia de 1 a 10 *peers* simples por minuto. Já a configuração para geração dos instantes de saída dos *peers*, como já mencionado anteriormente, segue uma distribuição de Weibull, cujo parâmetro de forma *k* é 0.8526 e o parâmetro de escala λ é 819.2167. Por fim, para configurar os instantes em que os *peers* retornam ao sistema distribuído, fui utilizada uma distribuição de Pareto, onde o fator de variabilidade *a* varia de 0.1 a 0.3. Todos os instantes de tempo serão gerados entre 1 a 86400 s, sendo este último valor a duração de cada rodada de simulação, o que equivale a 24h.

Além dos tipos de eventos já mencionados até agora, também são gerados outros eventos, cuja finalidade é definir os instantes em que oportunidades de colaboração são publicadas no ambiente distribuído simulado e o momento em que os *peers* compartilham hardware. A geração destes dois tipos de eventos não se dá pelo uso de uma distribuição de probabilidades, ou seja, a mesma acontece quando um evento primário é tratado ou no momento em que os *peers* reagem ao recebimento de uma mensagem enviada por algum outro *peer* que esteja ativo durante a simulação. Assim, quando uma oportunidade de colaboração é criada, são gerados eventos de publicação desta oportunidade, de modo que os mesmos aconteçam de maneira freqüente. Já os eventos de compartilhamento de hardware são criados após um *peer* ingressar em uma rede social *peer-to-peer* juntamente com outros *peers*.

Além das configurações para geração de eventos, também são configuradas as disponibilidades de recursos, onde cada *peer* simples dispõe de 70% de espaço livre em disco, 90% de capacidade de processamento e 70% de espaço livre em memória. Dentro da disponibilidade de cada um dos tipos de hardware, o percentual de compartilhamento varia de 10% a 90%, ou seja, um *peer* simples não compartilhará totalmente a quantidade de recurso disponível, tendo em vista que o mesmo precisa de uma quantidade de recursos disponíveis para manter seu funcionamento. Tais percentuais podem ser observados à medida que um usuário comum utiliza seu computador pessoal, semelhante ao uso de computadores pessoais por cientistas. Por fim, a disponibilidade de banda de toda a rede *peer-to-peer* foi configurada para variar em 10%, 25% e 50% da banda passante configurada para cada tipo de *peer*.

Para gerar automaticamente os perfis sociais em cada *peer*, foi necessário coletar conceitos e palavras-chaves, ambos relacionados às atividades e ambientes computacionais de *e-Science*. Por meio da distribuição de tais conceitos e palavras-chaves, perfis sociais são formados em cada *peer* simples no momento de sua criação, de modo que este represente um

pesquisador conectado ao sistema distribuído. De maneira básica, neste trabalho, o perfil social dos pesquisadores define sua experiência com determinados assuntos relacionados às atividades de *e-Science*. Todos os conceitos e palavras-chaves utilizadas nas simulações são apresentados pela Tabela 6.2.

Tabela 6.2: Conceitos e palavras-chaves relacionados as atividades de *e-Science*

Conceito	**Palavras-Chaves**
bioinformatics	*workflow,blast, fasta, gene, ontology, DNA, RNA, phylogeny, biology, health, genetics, tools e genome*
distributed computing	*grid, boinc, distributed algorithm, parallel processing, parallel programming, globus, soa, cluster, colaboration, p2p computing, resource sharing, infrastructure.*
Workflow	*vistrails, provenance, kepler, XML, taverna e ontology.*

Com base nos conceitos e palavras-chaves apresentadas pela Tabela 6.2, foi selecionado um subconjunto dos mesmos, a fim de definir as restrições de uma oportunidade de colaboração. Estas restrições serão utilizadas no processo de formação de uma rede social *peer-to-peer*, utilizando um limite mínimo para admissão de membros. Assim, à medida que estas restrições são publicadas sobre o grafo da rede *peer-to-peer*, por cada *peer* em que elas passam, é executado o algoritmo de combinação social apresentado na Seção 5.3.1 desta dissertação. Portanto, se o perfil social de um *peer* for maior ou igual ao limite de aceitação definido, o *peer* ajudará a compor uma rede social *peer-to-peer*. A configuração das restrições da oportunidade de colaboração pode ser vista na Tabela 6.3.

Tabela 6.3: Restrições de oportunidade de colaboração

Conceito	**Palavras-Chaves**	**Limite de Aceitação**
bioinformatics	*ontology,DNA, RNA, blast, fasta, genome*	70%
distributed computing	*soa,globus, boinc, grid, infrastrucure*	70%
workflow	*taverna, provenance, vistrails*	70%

Além das configurações de infraestrutura computacional e das utilizadas no processo de formação da rede social *peer-to-peer*, foi necessário definir as configurações para o cenário de compartilhamento de conteúdos. Este cenário foi configurado de maneira que um conteúdo, com tamanho de 2 GB, seja compartilhado em uma rede social *peer-to-peer* com 110 membros. Como explicado na Seção 5.3.4, o conteúdo compartilhado é divido em partes, de modo que estas sejam transferidas uma a uma para um *peer* requisitante de *download*.

Assim, como parâmetro de configuração de simulação, cada uma das partes do conteúdo compartilhado terá uma tamanho de 2 KB.

Para gerar resultados relevantes para o uso de redes sociais *peer-to-peer* no compartilhamento de recursos em ambientes de *e-Science*, o conteúdo é inicialmente compartilhado a partir de uma única fonte para *download* e, em seguida, o número de membros requisitantes do *download* aumenta, à medida que os usuários solicitam o *download* do conteúdo, seguindo uma distribuição de Poisson cuja média varia de 1 a 10 usuários a cada 10 segundos. Isto fará com que o número de partes por segundo caia drasticamente, enquanto vários usuários baixam o mesmo conteúdo ao mesmo tempo, e, à medida que os downloads são concluídos, mais partes de conteúdo são oferecidas.

Tabela 6.4: Parâmetros de configuração para execução de tarefas.

Parâmetro de Configuração	**Valor**
Tamanho da Carga de Trabalho Divisível	1,5 GBytes
Tamanho da Carga de Trabalho	2 MBytes
Tamanho da Carga de Trabalho Compactada	3 KBytes
Tamanho da Mensagem *Peer-to-Peer*	256 Bytes
Tamanho da Resposta de Processamento	3 KBytes
Tempo para Divisão Total da Carga de Trabalho Divisível	307,8 s
Tempo de Compactação da Carga de Trabalho	0,01 s
Tempo de Compactação da Mensagem *Peer-to-Peer*	0,001 s
Tamanho do Objeto Móvel	3 KBytes
Tempo de Espera em Fila	0,1 s
Tempo de Descompactação da Mensagem *Peer-to-Peer*	0,001 s
Tempo de Descompactação da Carga de Trabalho	0,01 s
Tempo para Escalonamento do Sistema Operacional	0,01 s
Tempo para Armazenamento de Resposta de Processamento	0,8 s
Taxa de Vazão de Processador de 1.5 GHz x Carga de Trabalho	0.8 s
Taxa de Vazão de Processador de 2.0 GHz x Carga de Trabalho	0.5 s
Taxa de Vazão de Processador de 3.0 GHz x Carga de Trabalho	0.3 s
Taxa de Vazão de Processador de 4.0 GHz x Carga de Trabalho	0.1 s

Para simular a execução dos objetos móveis (tarefas) sobre os compartilhamentos de recursos em uma rede social *peer-to-peer* com 110 membros, o cenário foi configurado de maneira que uma carga de trabalho com 1.5 GB fosse dividida em partes de 2 MB e, em seguida, submetida para processamento na rede social *peer-to-peer*. Definir a taxa de vazão

para experimentos com cargas de trabalho divisíveis não é tarefa fácil, pois estes valores variam de computador para computador. Uma vez que uma rede social *peer-to-peer* é um ambiente computacional heterogêneo, foi necessário arbitrar valores de taxa de vazão de acordo com os *clocks* dos processadores configuradas aleatoriamente em cada *peer* da rede social. Portanto, o tempo em que os *peers* processam 2 MB, de acordo com o *clock* de processadores, e outros parâmetros de configuração de simulação são expressos na Tabela 6.4. Além destes tempos, também foram definidos tempos relacionados à compactação das cargas de trabalho e mensagens *peer-to-peer*. Estes valores foram definidos a partir do uso da biblioteca zLib (ZLIB 2010).

Na simulação de execução de tarefas, os objetos móveis só migrarão para os *peers* que oferecem disponibilidade de recursos compartilhados. Neste caso, será avaliado o tempo de processamento da carga de trabalho, tendo como base a disponibilidade para processamento encontrada nos *peers* participantes da rede social *peer-to-peer*. Portanto, foram executadas simulações com 10%, 25%, 50% e 100% de disponibilidade para processamento. A disponibilidade de processamento consiste na capacidade de um *peer* armazenar temporariamente uma carga de trabalho e, em seguida, processá-la com o objetivo de gerar uma resposta de processamento.

6.3 ANÁLISE DOS RESULTADOS

Como mencionado anteriormente, os experimentos foram executados a partir do simulador de redes *peer-to-peer* desenvolvido ao longo deste trabalho de mestrado. Os resultados foram coletados a partir dos *logs* de cada uma das simulações. As entradas nos *logs* eram geradas à medida que eventos eram tratados por componentes do motor de simulação.

É importante destacar que a implementação do simulador de redes *peer-to-peer* teve como objetivo a realização de uma prova de conceito. Portanto, os resultados que serão apresentados podem ser melhorados em versões futuras de cada um dos algoritmos *peer-to-peer* e das políticas de computação distribuída baseadas em redes sociais ou em implementações destes em algum protótipo no futuro.

6.3.1 Cenário 1: Estabilização da Rede *Peer-to-Peer*

A partir das dez rodadas de simulação realizadas para o cenário de estabilização da rede *peer-to-peer,* foi possível obter resultados relevantes quanto ao sistema distribuído, uma vez que a infraestrutura deste é formada por uma rede *peer-to-peer* não estruturada.

Inicialmente, para este trabalho, adotamos como protocolo de rede *peer-to-peer* o Gnutella, que não apresentou bons resultados no que diz respeito à quantidade de mensagens geradas de acordo com a chegada de *peers* na rede; ao consumo de banda disponível nos *peers;* e ao tempo em que as mensagens consomem a banda disponível no sistema. A quantidade de mensagens geradas pelo Gnutella cada vez que um *peer* se conecta a rede *peer-to-peer* é exibida na Figura 6.1 (note que o número de mensagens é exibido com escala logarítmica).

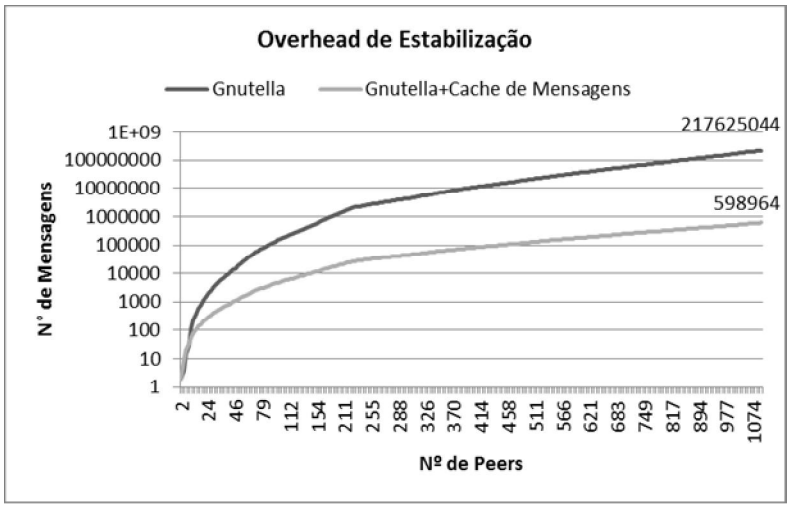


Figura 6.1: Overhead de estabilização da rede *peer-to-peer*

Na implementação padrão do Gnutella, quando um *peer* se conecta à rede de maneira não determinística, são disparadas mensagens PING, a fim de notificar todos os outros *peers* conectados à rede. À medida que cada PING é enviado e, em seguida, repassado sucessivamente até atingir o número máximo de saltos, mensagens PONG são enviadas de volta, de modo que o *peer* recém conectado à rede possa ter informações sobre outros *peers* da rede. O grande problema no uso do Gnutella não são as mensagens de *feedback* que os *peers* já conectados enviam para aquele recém chegado à rede, mas a quantidade de mensagens PING que inundam a rede.

Uma solução inicial para controlar tal problema é manter um *cache* temporário de identificadores de mensagens em cada *peer*. Dessa forma, à medida que mensagens PING chegam a cada um dos *peers*, os mesmos registram os identificadores destas mensagens e, em seguida, as repassa para seu conjunto de vizinhos. Caso um *peer* receba novamente uma mensagem cujo identificador já tenha sido registrado no *cache* de identificadores, a mensagem não é repassada para o conjunto de vizinhos. O efeito deste controle pode ver visto na Figura 6.1.

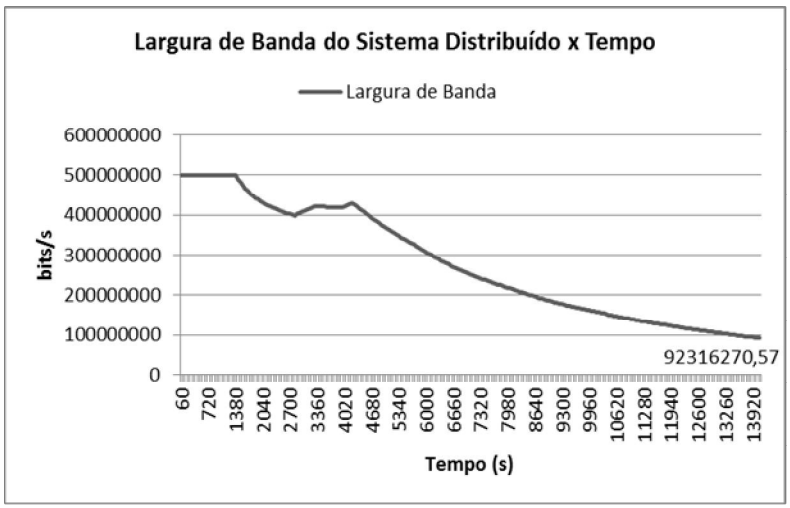


Figura 6.2: Disponibilidade de banda no sistema x tempo.

Outro benefício obtido com o uso de *cache* de identificadores é a redução do consumo da banda de comunicação disponibilizada pelos *peers*, uma vez que a banda disponível para o sistema distribuído diminui à medida que novos *peers* se conectam à rede *peer-to-peer*. A disponibilidade de banda do sistema distribuído pode ser vista no gráfico exibido pela Figura 6.2. A diminuição da largura de banda do sistema distribuído é causada pela entrada de *peers* simples, uma vez que estes possuem largura de banda menor que as dos *superpeers*. Cada um dos *superpeers* conectados à rede *peer-to-peer* compartilha sua banda passante com todos os demais *peers*. Assim, o ligeiro aumento na largura de banda do sistema distribuído é causado pela chegada de novos *superpeers*. Concomitantemente, novos *peers* simples também se conectam à rede, até o término de chegada de *superpeers*. Após este período, a largura de banda tende a cair, pois os *peers* simples continuam se conectando no sistema distribuído até a quantidade de *peers* chegue a 1100 *peers*. O crescimento da quantidade de *peers* pode ser visto no gráfico exibido pela Figura 6.3.

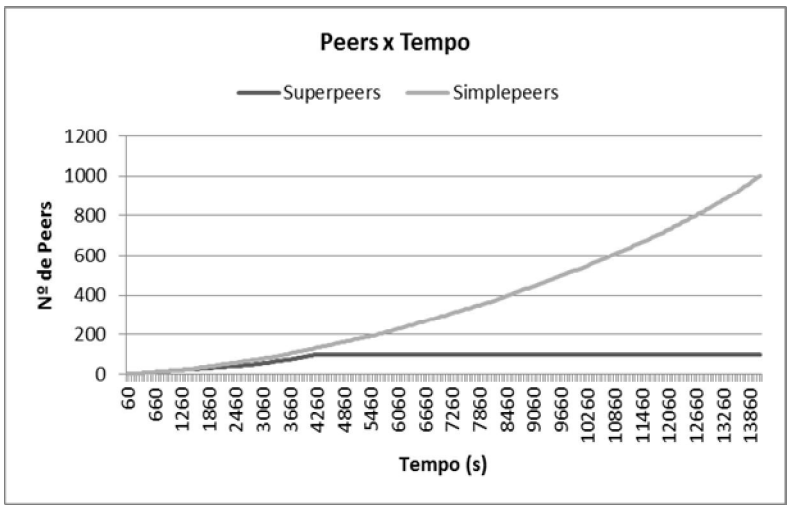


Figura 6.3. Crescimento da quantidade de *peers*

Assumindo que os tamanhos das mensagens PING e PONG são 256 bytes respectivamente, conforme a especificação do protocolo Gnutella, foi analisado o tempo em que as mensagens passam a consumir toda a banda disponível no sistema distribuído. Além disto, também foi analisado o período de tempo em que a banda do sistema distribuído é consumida integralmente. Portanto, foram simulados cenários onde a disponibilidade da largura de banda do sistema distribuído está em 10%, 25% e 50% e, por isso, foram feitas comparações entre a implementação padrão do Gnutella e a implementação do Gnutella com cache de identificadores de mensagens. Os gráficos comparativos são exibidos pela Figura 6.4, Figura 6.5 e Figura 6.6, onde são apresentados os resultados das simulações com as disponibilidades de banda mencionadas anteriormente.

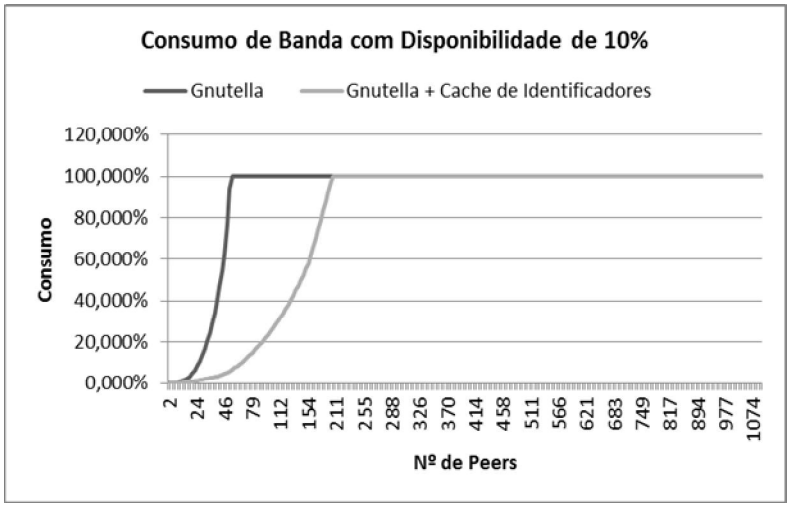


Figura 6.4: Consumo de Banda com Disponibilidade de 10%

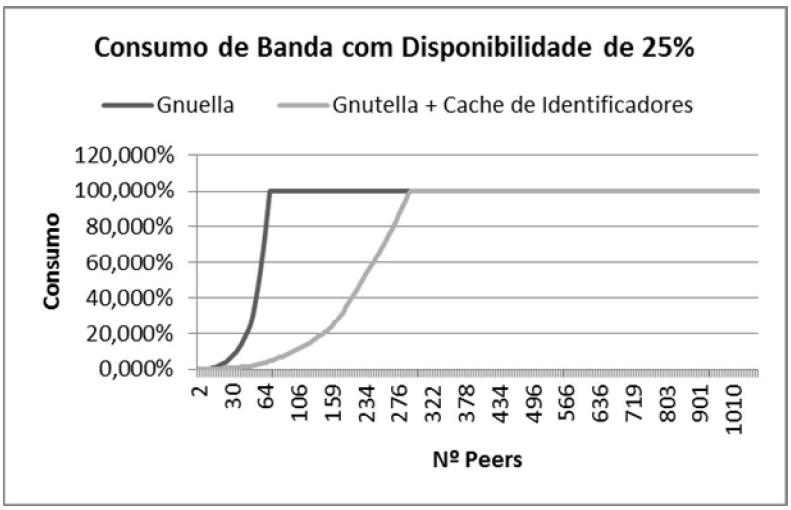


Figura 6.5: Consumo de Banda com Disponibilidade de 25%

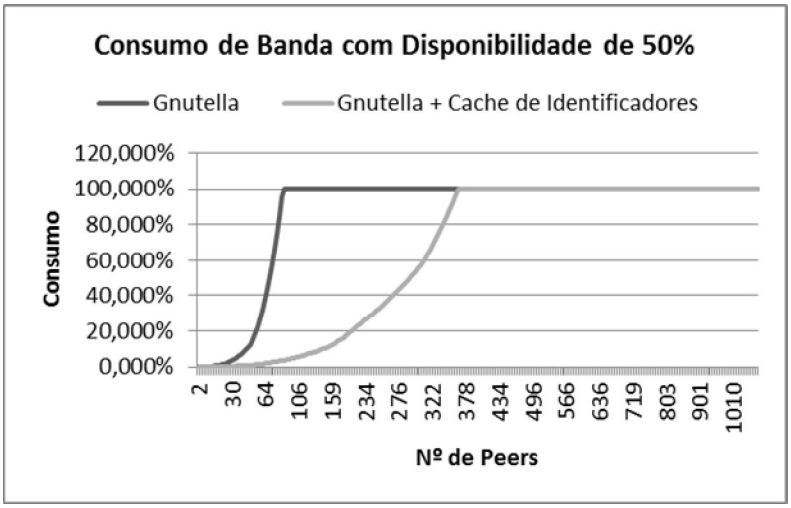


Figura 6.6: Consumo de Banda com Disponibilidade de 50%

Com a implementação padrão do protocolo Gnutella, a banda disponível no sistema distribuído é rapidamente consumida nos três cenários apresentados, sem grandes variações na quantidade de *peers* conectados à infraestrutura do sistema distribuído. Já com a adição de um cache de identificadores de mensagens, é necessário uma quantidade maior de *peers* para consumir totalmente a banda disponível no sistema distribuído. Em uma rede *peer-to-peer* não estruturada é inevitável o alto consumo da banda passante disponível. No entanto, é preciso avaliar o tempo em que tal recurso fica ocupado de acordo com o tamanho da rede *peer-to-peer*. Os tempos de consumo de banda podem ser observados na Figura 6.7, Figura 6.8 e Figura 6.9.

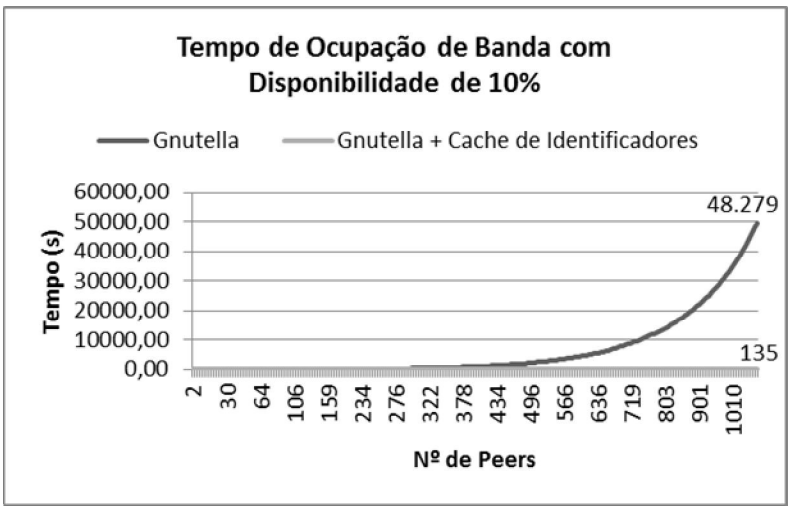


Figura 6.7: Tempo de ocupação de banda com disponibilidade de 10%

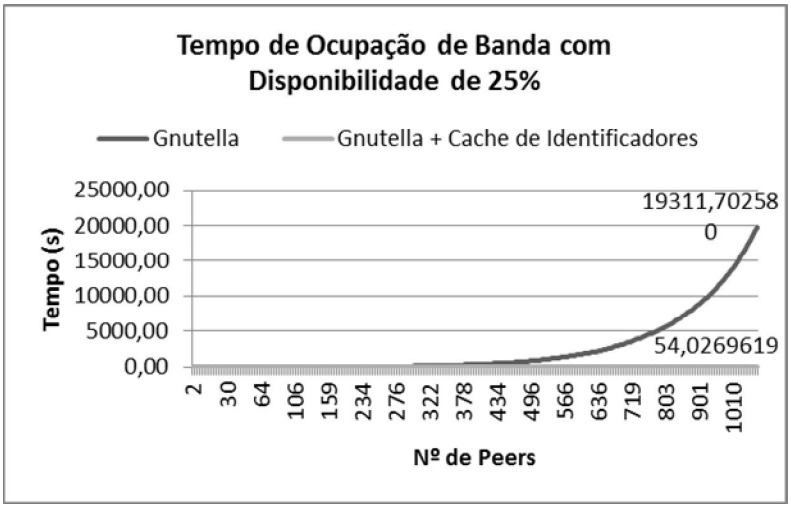


Figura 6.8: Tempo de ocupação de banda com disponibilidade de 25%

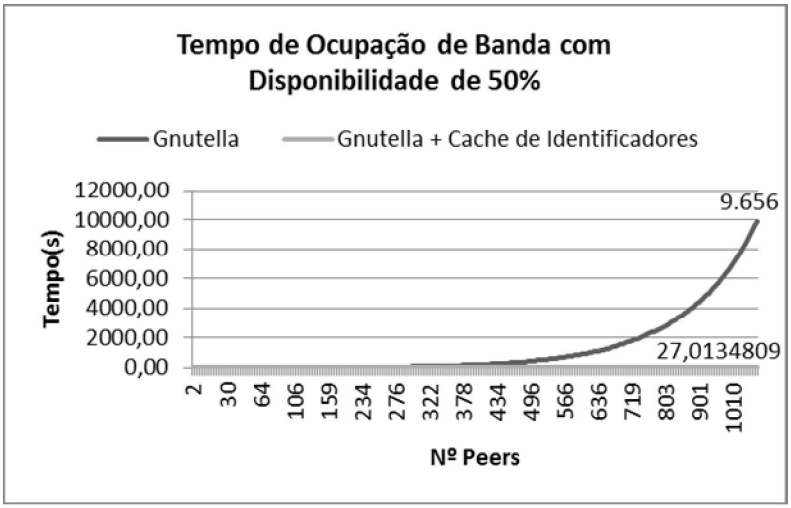


Figura 6.9: Tempo de ocupação de banda com disponibilidade de 50%

Ao final de cada uma das linhas dos gráficos apresentados pela Figura 6.7, Figura 6.8 e Figura 6.9, são exibidos os tempos de consumo de banda, quando a rede *peer-to-peer* atinge 1100 *peers.* Apesar de não ter sido o foco deste trabalho, controlar a sobrecarga e o consumo de banda é de suma importância para o estudo proposto aqui, pois ambientes de *e-Science*, apesar de conectados a *backbones* com alta velocidade na transmissão, manipulam grandes volumes de dados. Portanto, é possível concluir que o alto número de mensagens e um grande tempo de ocupação da banda passante do sistema distribuído podem afetar drasticamente as taxas de transferência de dados em uma ambiente de *e-Science* totalmente distribuído.

6.3.2 Cenário 2: Formação de uma Rede Social Peer-to-Peer em uma Rede Peer-to-Peer Estável, sem Compartilhamento de Responsabilidade

A formação de redes sociais *peer-to-peer* sobre uma infraestrutura computacional formada por uma rede *peer-to-peer* não estruturada é uma das principais colaborações desta dissertação. Para obter resultados relevantes quanto ao uso de uma rede social *peer-to-peer* como meio de agregação de recursos compartilhados, inicialmente foram criados dois cenários, de modo que pudessem ser coletados dados. A partir desta coleta, os dados foram utilizados como ponto de comparação com outros cenários criados para avaliar a formação de uma rede social *peer-to-peer*. Estes cenários serão apresentados nas Seções 6.3.3, 6.3.4 e 6.3.5.

No cenário desta seção, a publicação da oportunidade de colaboração, criação da rede social e admissão de membros na rede social são de responsabilidades de um único *peer*. Neste cenário, a rede *peer-to-peer* não apresenta oscilações no número de *peers*, ou seja, durante o período de simulação, os 1100 *peers* estão conectados à rede *peer-to-peer*. Como mencionado anteriormente na Seção 6.2.4, dos 1100 *peers* conectados à rede, 1000 são *peers* simples. Portanto, à medida que uma oportunidade de colaboração chega a estes *peers*, é executado o algoritmo de combinação social apresentado no Capítulo 5 e, se houver pelo menos um *peer* compatível com a restrição definida na oportunidade de colaboração, será formada um rede social *peer-to-peer* com aquele *peer* que publicou a oportunidade.

A partir dos resultados obtidos pelas simulações deste cenário, foi possível atingir escalas muito altas em quantidade de recursos compartilhados. Estes resultados foram atingidos reunindo uma média de 205 membros em uma rede social *peer-to-peer*, após o cálculo da média dos totais das simulações do cenário apresentado por esta seção.

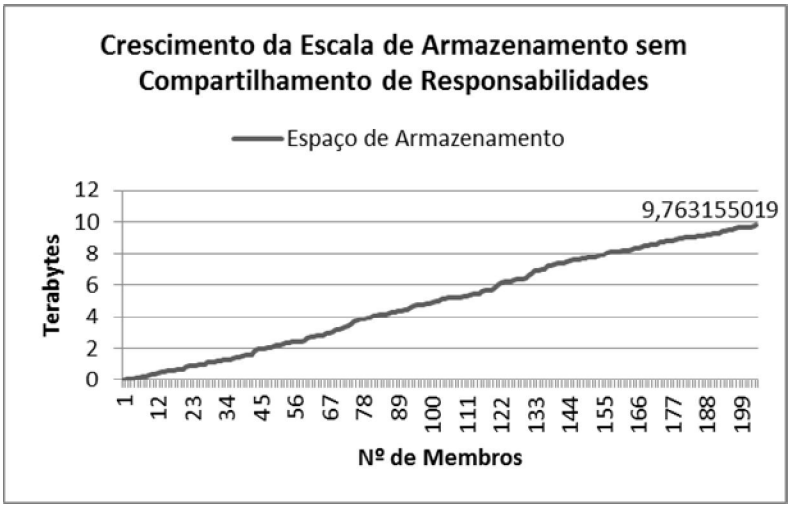


Figura 6.10: Crescimento da escala de armazenamento sem compartilhamento de responsabilidades

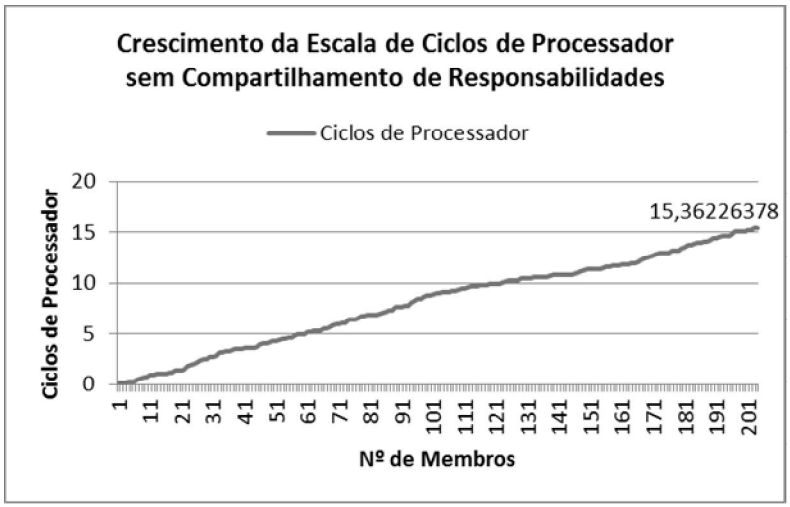


Figura 6.11: Crescimento da escala de ciclos de processador sem compartilhamento de responsabilidades

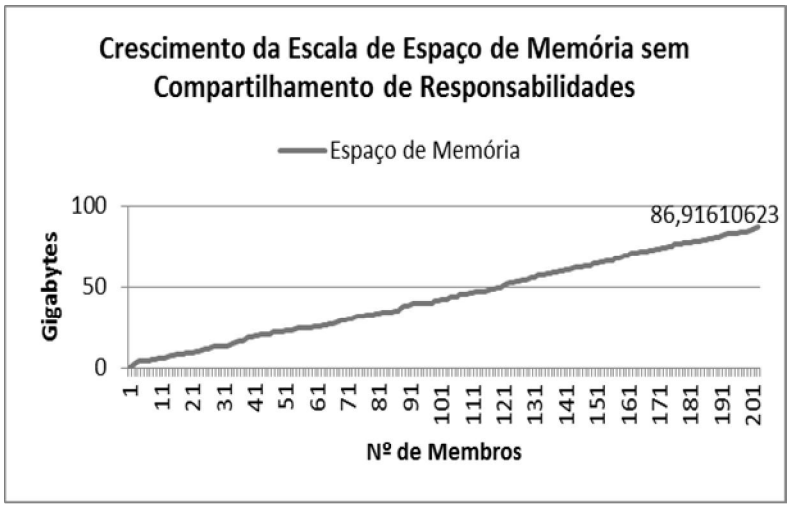


Figura 6.12: Crescimento da escala de espaço de memória sem compartilhamento de responsabilidades

Os resultados exibidos pela Figura 6.10, Figura 6.11 e Figura 6.12, mostram que é possível agregar uma grande quantidade de recursos compartilhados utilizando o modelo de computação *peer-to-peer*. Para este primeiro cenário, criar uma rede social *peer-to-peer* e admitir novos membros na rede social, foi possível obter uma média de 9,7 Terabytes de espaço de armazenamento, 15,36 bilhões de ciclos de processador e 86,92 Gigabytes de espaço em memória.

6.3.3 Cenário 3: Formação de uma Rede Social Peer-to-Peer em uma Rede Peer-to-Peer Estável, com Compartilhamento de Responsabilidade

A fim de realizar uma comparação dos resultados obtidos no primeiro cenário de formação de rede social *peer-to-peer* (Cenário 2,Seção 6.3.2), foi simulado um outro cenário onde um *peer* inicialmente possui a responsabilidade de publicar a oportunidade de colaboração, criar a rede social *peer-to-peer* e admitir membros na rede social e, logo após, as responsabilidades de publicação de oportunidade de colaboração e admissão de novos membros na rede social é compartilhada entre os integrantes da rede social (Cenário 3).

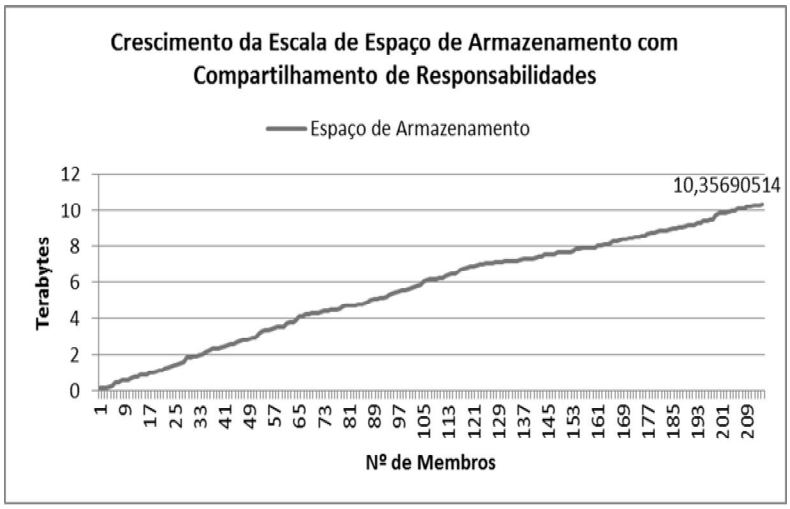


Figura 6.13: Crescimento da escala de espaço de armazenamento com compartilhamento de responsabilidades

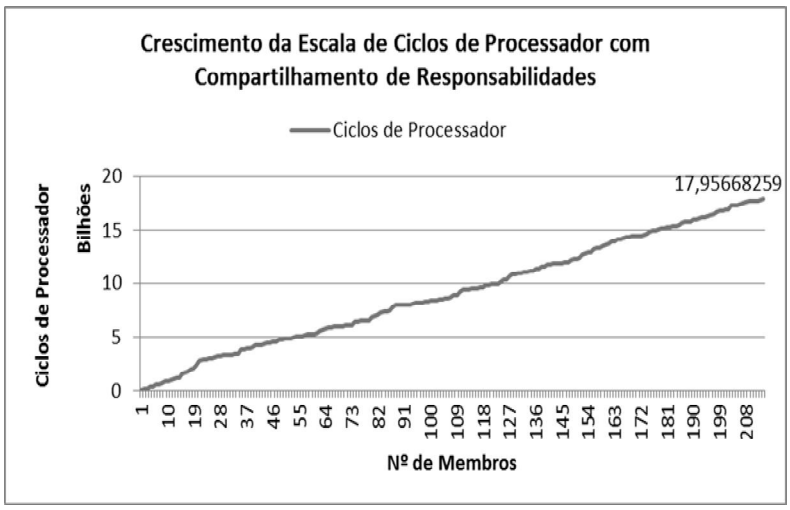


Figura 6.14. Crescimento da escala de ciclos de processador com compartilhamento de responsabilidades.

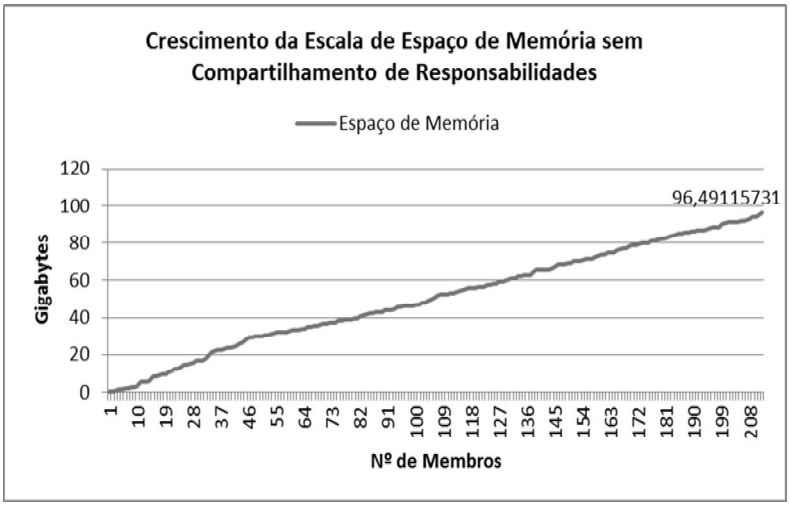


Figura 6.15: Crescimento da Escala de Espaço de Memória com Compartilhamento de Responsabilidades

Após as simulações do Cenário 3 para formação de redes sociais *peer-to-peer*, resultados muito próximos, em termos de escala, foram obtidos se comparados com o Cenário 2. A comparação entre o Cenário 2 e o Cenário 3 é apresentada na

Tabela 6.5. Além disso, a abordagem de compartilhamento de responsabilidades em uma rede *peer-to-peer* estável faz com que o número de mensagens de publicação de oportunidade cresça proporcionalmente ao número de membros da uma rede social. Caso haja um grande número de redes sociais criadas sobre a rede *peer-to-peer*, o número de mensagens aumenta de acordo com o envolvimento dos *peers* em um número de redes sociais.

Tabela 6.5: Quadro comparativo entre o Cenário 2 e Cenário 3.

	Cenário 2	Cenário 3	Aumento
Nº de Membros	205	214	4%
Espaço de Armazenamento	9,70	10,36	6%
Ciclos de Processadores	13,36	17,96	16%
Espaço de Memória	86,92	96,49	11%

Fazendo uma análise, o número de mensagens de publicação por *peer* se dá de acordo com o número de redes sociais em que um *peer* é membro. Portanto, é possível concluir que o compartilhamento de responsabilidades sobre uma rede *peer-to-peer* não influi tanto na arregimentação de membros para uma rede social. Neste caso, um número a mais de mensagens é gerado para se obter quase os mesmos resultados. É importante lembrar que as publicações de oportunidade de colaboração possuem um tempo de vida estipulado pelo usuário e, por isto, mensagens de publicação não ocupam a banda passante disponível no sistema distribuído de maneira desnecessária, após uma rede social *peer-to-peer* atingir um número de membros cujos usuários acham suficiente para realizar alguma atividade com os recursos compartilhados por eles.

6.3.4 Cenário 4: Formação de uma Rede Social Peer-to-Peer em uma Rede Peer-to-Peer Instável, sem Compartilhamento de Responsabilidade

Os cenários 2 e 3, apresentados respectivamente pelas Seções 6.3.2 e 6.3.3, foram criados de maneira que seus resultados fornecessem valores que pudessem ser comparados com outros dois cenários, ambos tendo como base uma rede *peer-to-peer* instável, onde o número de *peers* oscila em função dos tempos gerados pelas distribuições de Weibull e Pareto. Como mencionado anteriormente, a distribuição dos tempos em que os *peers* se mantêm conectados à rede *peer-to-peer* segue uma distribuição de Weibull, cujo parâmetro de forma k é 0.8526 e o parâmetro de escala λ é 819.2167. Por fim, para configurar os instantes em que os *peers* retornam ao sistema distribuído, fui utilizada uma distribuição de Pareto, onde o fator de variabilidade a varia de 0.1 a 0.3. A média das simulações para cada fator de variabilidade é exibida pela Figura 6.16.

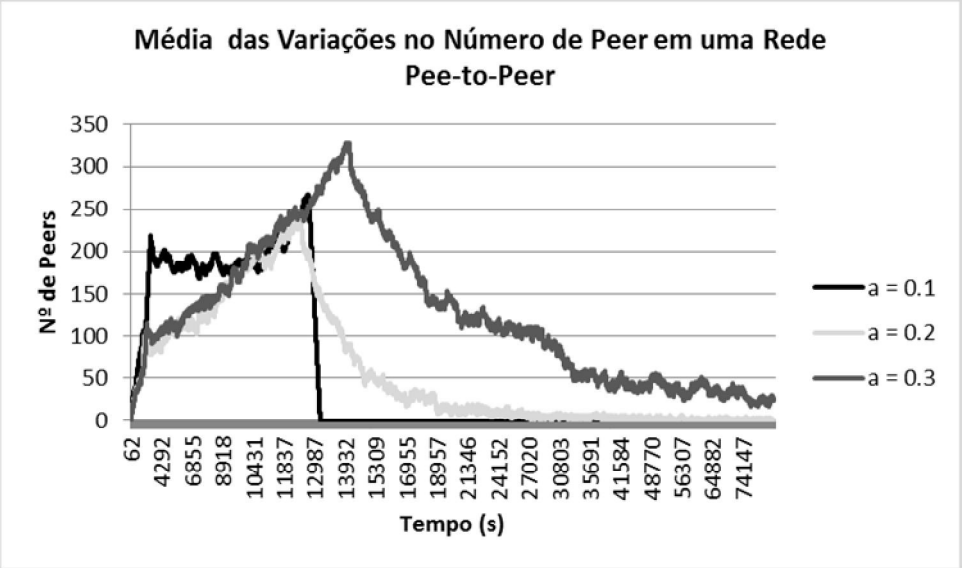


Figura 6.16: Média das variações do número de *peers* em uma rede *peer-to-peer*

O cenário avaliado nesta seção utiliza a mesma abordagem do Cenário 2: um único *peer* é responsável em publicar a oportunidade de colaboração, criar uma rede social e admitir membros na rede social. Neste cenário, é avaliado o impacto que as oscilações da rede *peer-to-peer* têm no processo de formação de uma rede social *peer-to-peer*. Com isso, também será possível avaliar o impacto na agregação de recursos compartilhados, uma vez que esta atividade é dependente do tamanho da rede social formada por um conjunto de *peers*. Assim, separados por fatores de variabilidade, foram avaliados o tamanho médio das redes sociais formadas nos experimentos e a quantidade média de recursos compartilhados agregados. Estas avaliações podem ser visualizadas nos gráficos apresentados pela Figura 6.17, Figura 6.18, Figura 6.19 e Figura 6.20.

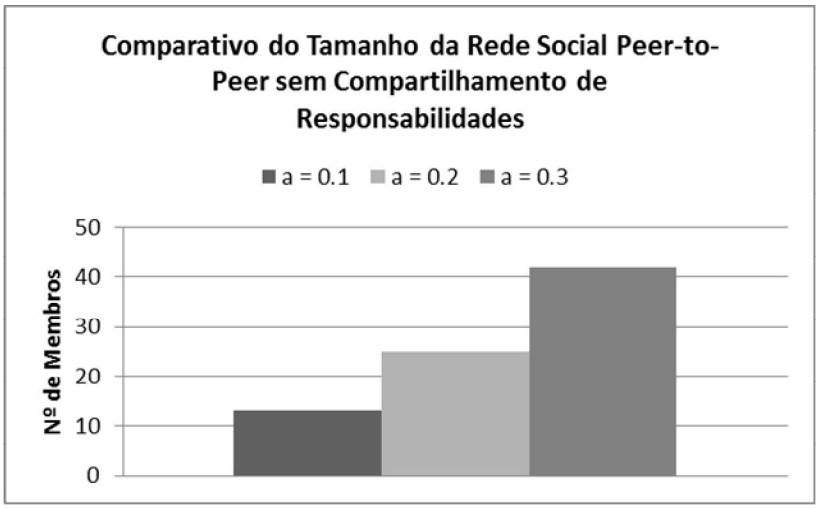


Figura 6.17: Comparativo do tamanho da rede social peer-to-peer sem compartilhamento de responsabilidades

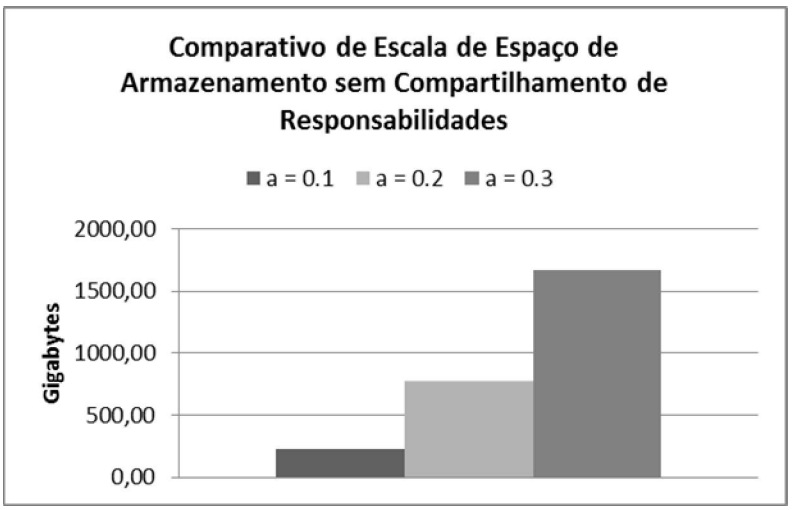


Figura 6.18: Comparativo de escala de espaço de armazenamento sem compartilhamento de responsabilidades

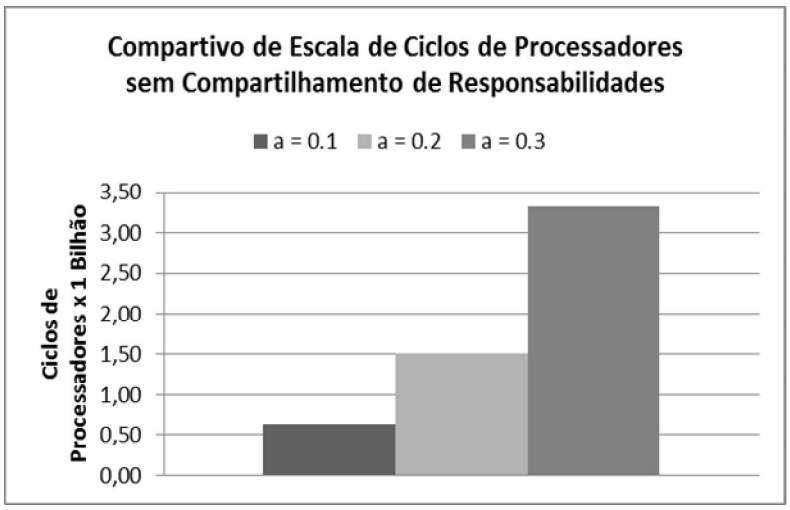


Figura 6.19: Comparativo de escala de ciclos de processadores sem compartilhamento de responsabilidades

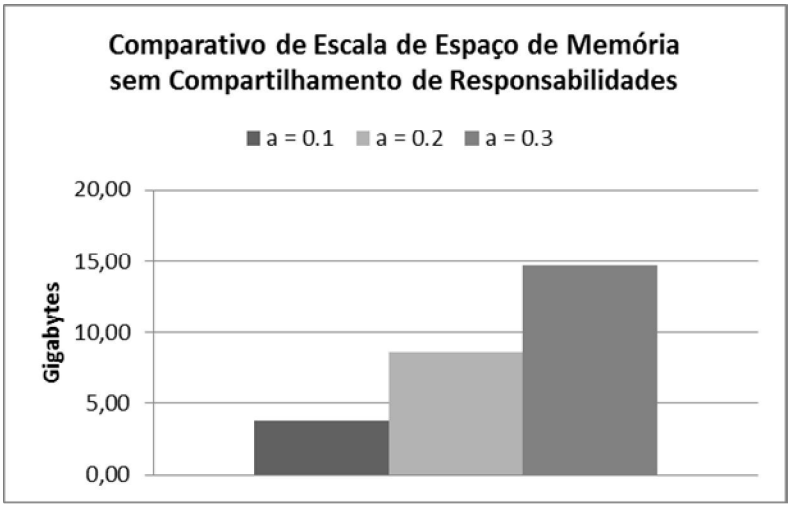


Figura 6.20: Comparativo de escala de espaço de memória sem compartilhamento de responsabilidades

Em uma rede *peer-to-peer* com grande instabilidade, utilizar uma abordagem onde um único *peer* tem a responsabilidade de publicar a oportunidade de colaboração, criar uma rede social *peer-to-peer* e admitir membros na rede social, não é recomendado. Como pode ser visto na Figura 6.17, Figura 6.18, Figura 6.19 e Figura 6.20, os resultados apresentados são muito baixos, se comparados com os demais cenários apresentados até aqui.

6.3.5 Cenário 5: Formação de uma Rede Social Peer-to-Peer em uma Rede Peer-to-Peer Instável, com Compartilhamento de Responsabilidade

No quinto cenário, o que se deseja avaliar é se o compartilhamento de responsabilidades entre membros de uma rede social *peer-to-peer* impacta nos processos de criação e expansão de uma rede social, bem como na agregação de recursos. Os resultados obtidos com o compartilhamento de responsabilidades em uma rede *peer-to-peer* instável é apresentado nos gráficos exibidos na Figura 6.21, Figura 6.22, Figura 6.23 e Figura 6.24.

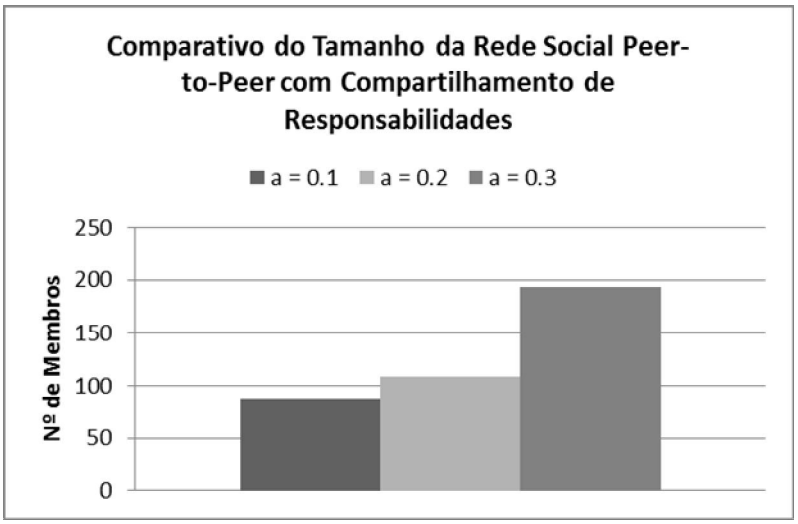


Figura 6.21: Comparativo do tamanho da rede social *peer-to-peer* com compartilhamento de responsabilidades

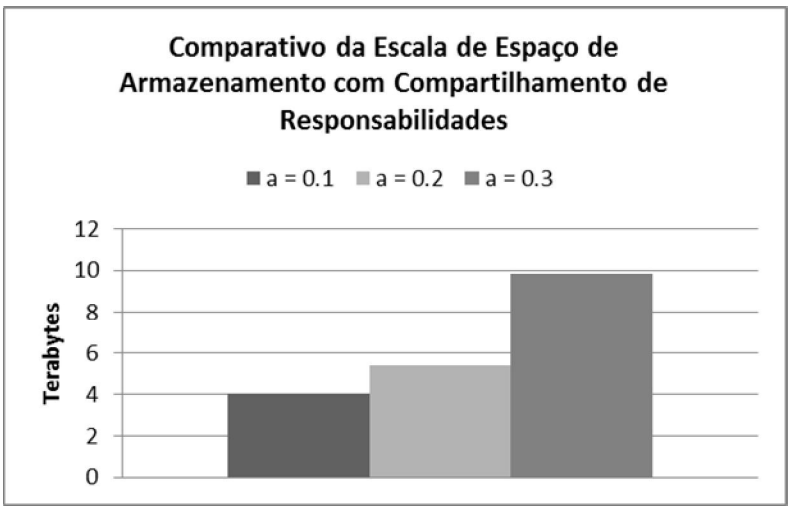


Figura 6.22: Comparativo da escala de espaço de armazenamento com compartilhamento de responsabilidades

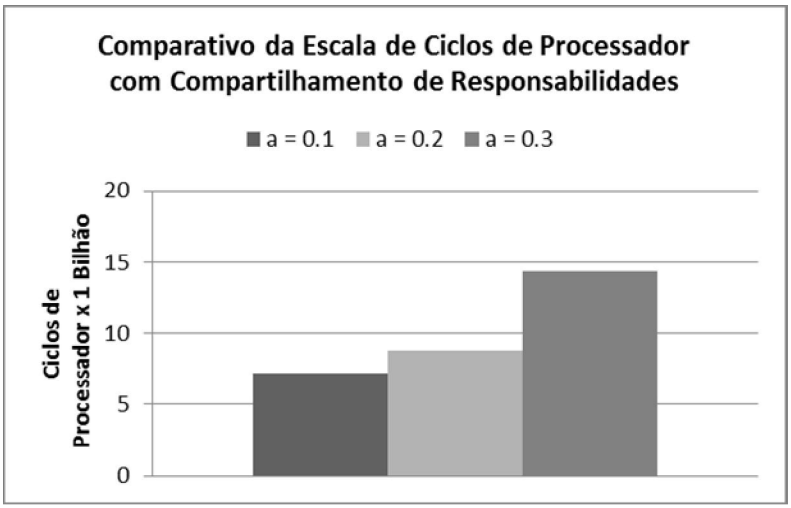


Figura 6.23: Comparativo da escala de ciclos de processador com compartilhamento de responsabilidades

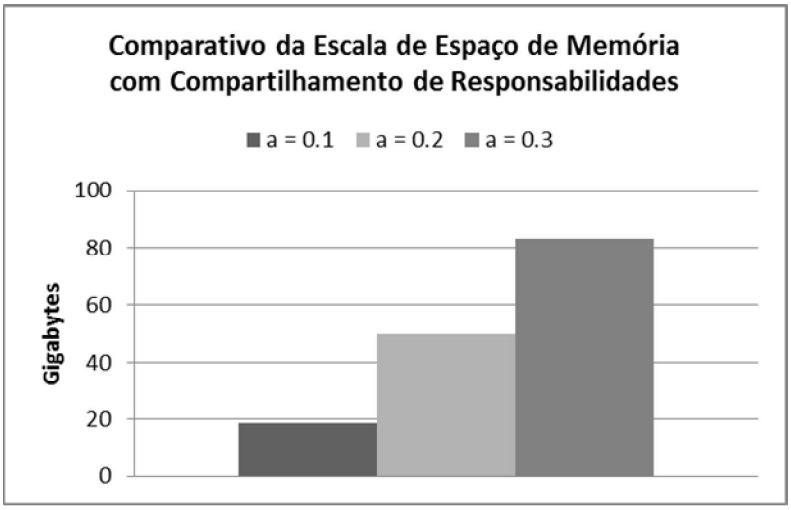


Figura 6.24: Comparativo da escala de espaço de memória com compartilhamento de responsabilidades

O compartilhamento das responsabilidades em uma rede social *peer-to-peer*, permitiu que, mesmo em condições com altas variações de tempos de desconexões de *peers*, fosse possível construir uma rede social com tamanho muito superior ao obtido anteriormente, quando se apresentou o Cenário 4. Além dos bons resultados obtidos sob altas variações nos tempos de desconexão dos *peers*, os experimentos em que a taxa de variabilidade foi igual a 0.3 apresentaram uma média de resultados muito próxima à média obtida em uma rede estável. Com estes dados, é possível concluir que redes sociais *peer-to-peer*, onde primeiramente um único *peer* possui todas as responsabilidades e, logo após, compartilha as responsabilidades de publicação de oportunidades de colaboração e admissão de membros na rede social, escalam mesmo em condições com variações nas conexões dos *peers* para com o sistema.

Outro ponto importante na avaliação da formação das redes sociais *peer-to-peer* é a análise do número de mensagens, consumo de banda e o número médio de saltos realizados pelas mensagens. Estes resultados foram obtidos por meio da média dos tamanhos de redes sociais *peer-to-peer* criadas durante as rodadas de simulação. Para avaliar a sobrecarga, foi considerado o número de mensagens enviadas desde a solicitação de admissão na rede social até as mensagens de atualização dos metadados de compartilhamentos de recursos da rede social *peer-to-peer*, onde cada uma das mensagens tem o tamanho de 256 bytes. A análise do consumo de banda foi realizada seguindo as configurações de simulação definidas na Seção 6.2.4. Por fim, o número médio de saltos foi calculado a partir da média dos saltos de todos os cenários.

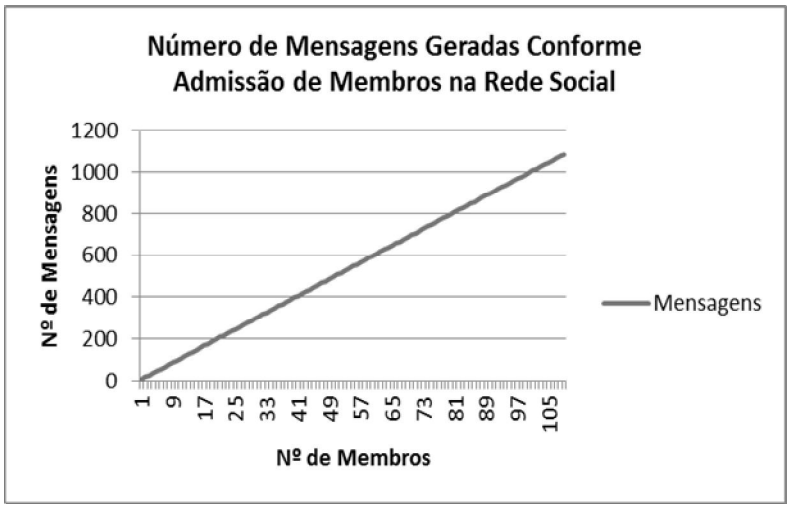


Figura 6.25: Número de mensagens geradas conforme admissão de membros na rede social

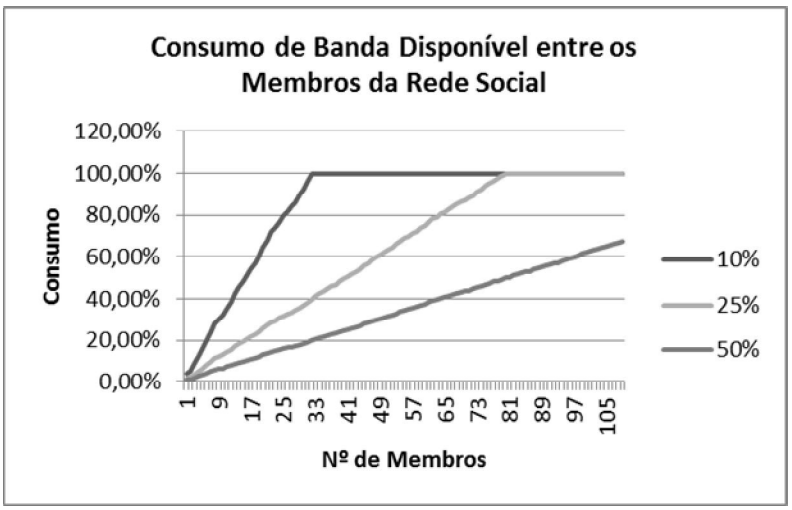


Figura 6.26: Consumo de banda disponível entre os membros da rede social *peer-to-peer*

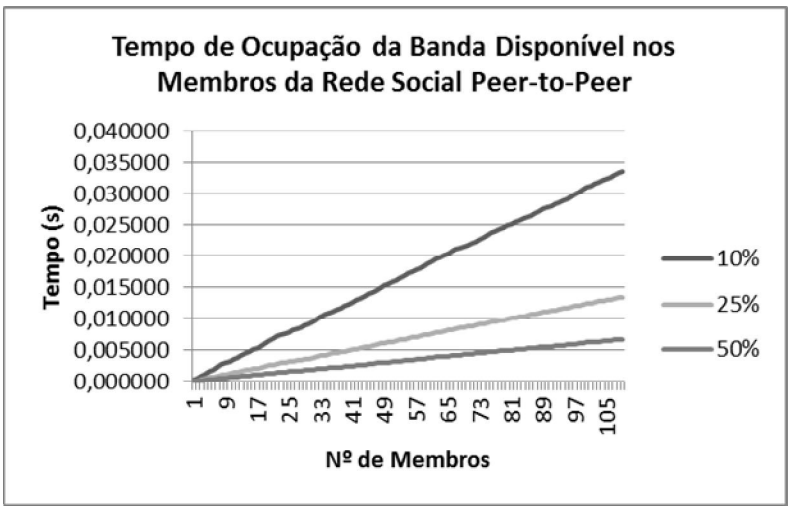


Figura 6.27: Tempo de ocupação da banda disponível nos membros da rede social *peer-to-peer*

Figura 6.28: Média de saltos realizados durantes a criação e expansão de uma rede social *peer-to-peer*

Fazendo uma análise do impacto das políticas relacionadas à formação e expansão da rede social *peer-to-peer*, incluindo aquelas para compartilhamento de recursos entre os membros da rede social, é possível concluir que o uso de redes sociais para compartilhamento de recursos no nível de hardware em ambientes de *e-Science* é viável. Como pode ser visto na Figura 6.25, o número de mensagens cresce de maneira linear à medida que os *peers* constituem uma rede social *peer-to-peer*. Além disso, o tempo em que a banda disponível é ocupada em 100%, como apresentado na Figura 6.26 e Figura 6.27, é muito pequeno. Uma rede social *peer-to-peer* com 110 membros, mesmo em um sistema distribuído com 10% da banda disponível, demanda menos de um segundo para ser totalmente atualizada. Outro motivo que torna o processo de formação e expansão de uma rede social *peer-to-peer* escalável é a média dos caminhos mínimos ou de saltos de mensagens enviadas de um membro para outro. Conforme apresentado na Figura 6.28, o número médio é de 4,9 saltos.

6.3.6 Cenário 6: Compartilhamento de Conteúdos

Tendo como base a média dos tamanhos de redes sociais *peer-to-peer*, obtidos durante as rodadas de simulação, foram realizados os experimentos para geração dos resultados do cenário de compartilhamento de conteúdos. Neste cenário, pretende-se avaliar o desempenho de uma rede social *peer-to-peer* no que diz respeito ao compartilhamento e transferência de grandes volumes de dados. Portanto, o cenário foi simulado com base em uma situação de stress na rede social *peer-to-peer*, onde inicialmente uma única fonte de para *download* é disponibilizada e, em seguida, cada um dos membros da rede social concomitantemente realizam o *download* do conteúdo compartilhado. À medida que os downloads são concluídos, o conteúdo se torna cada vez mais disponível nos membros da rede social, fazendo com que um número maior de partes de conteúdo seja disponibilizado. Isto deve permitir que o tempo de transferência se torne menor. Os resultados relacionados aos tempos de transferência de conteúdo entre os membros de uma rede social *peer-to-peer* podem ser visualizados nos gráficos exibidos pela Figura 6.29, Figura 6.30 e Figura 6.31, de acordo com as disponibilidades de banda do sistema distribuído.

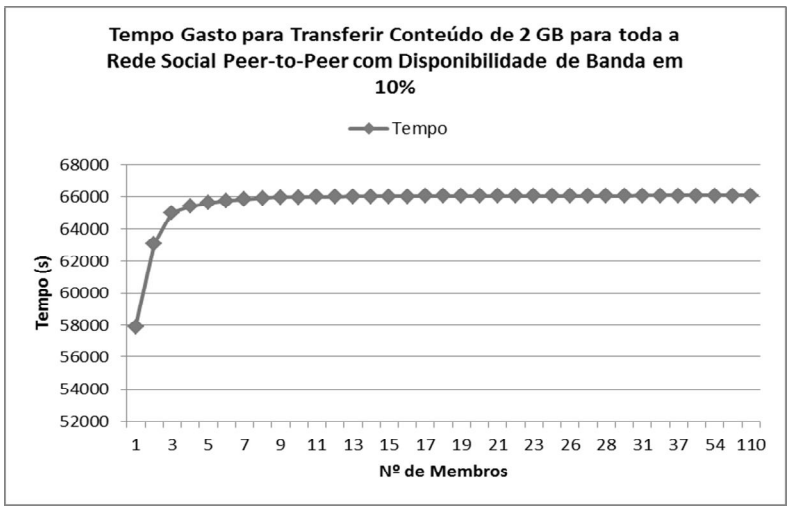


Figura 6.29: Tempo de transferência de conteúdo de 2 GB para toda a rede social *peer-to-peer* com disponibilidade de banda em 10 %

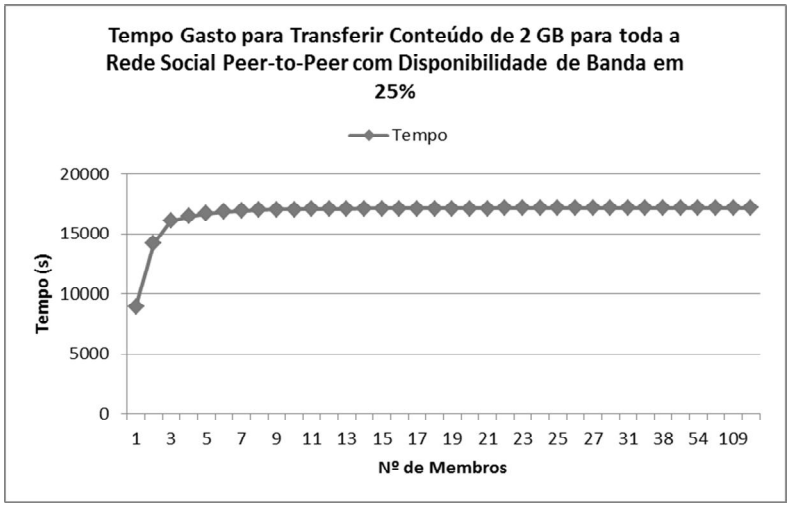


Figura 6.30: Tempo de transferência de conteúdo de 2 GB para toda a rede social *peer-to-peer* com disponibilidade de banda em 25 %

Figura 6.31:Tempo de transferência de conteúdo de 2 GB para toda a rede social peer-to-peer com disponibilidade de banda em 50 %

Como pode ser visto na Figura 6.29 e Figura 6.30, o tempo para se obter uma segunda fonte é bem alto, pois até este instante só existe uma única fonte de download para todos os demais membros da rede social *peer-to-peer*. No entanto, com duas fontes para *download*, já é possível notar uma ganho de tempo entre o surgimento da segunda e da terceira fonte. Na Figura 6.31, a curva de tempos de transferência apresenta-se de maneira diferente, devido uma maior disponibilidade de banda no sistema. Mesmo com baixa disponibilidade de banda, o algoritmo de transferência apresenta um bom comportamento, uma vez que os tempos tendem a estabilizar à medida que novas fontes de *download* surgem. Com maior disponibilidade de banda entre os membros de uma rede social *peer-to-peer,* os tempos de transferência formam uma curva que tende a apresentar um comportamento linear.

Outro ponto interessante, apresentado pelos algoritmos de transferência de conteúdos compartilhados, são as curvas de intervalos de conclusão de downloads. Tais curvas podem ser vistas nos gráficos apresentados pela Figura 6.32, Figura 6.33 e Figura 6.34.

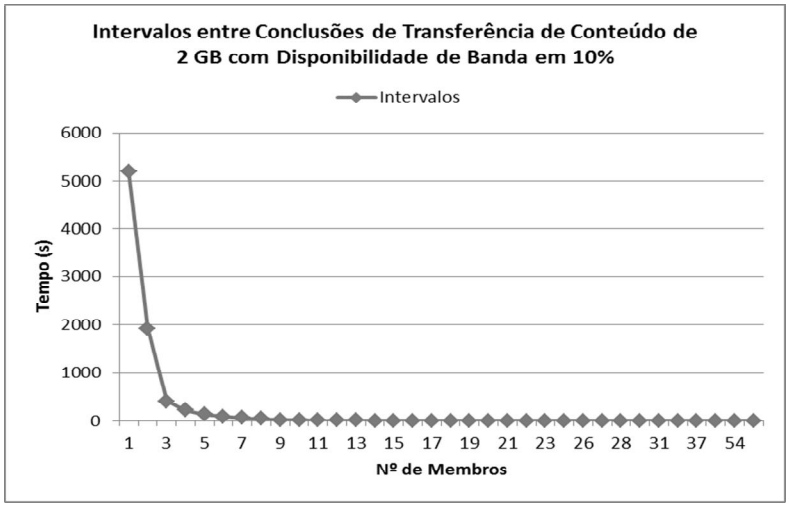


Figura 6.32: Intervalos entre conclusões de transferência de conteúdo de 2GB com disponibilidade de 10%

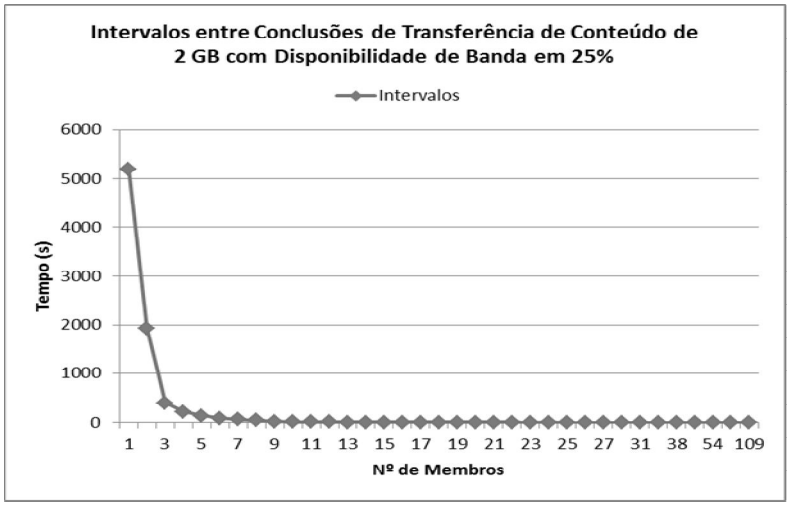


Figura 6.33: Intervalos entre Conclusões de Transferência de Conteúdo de 2 GB com Disponibilidade de Banda em 25%

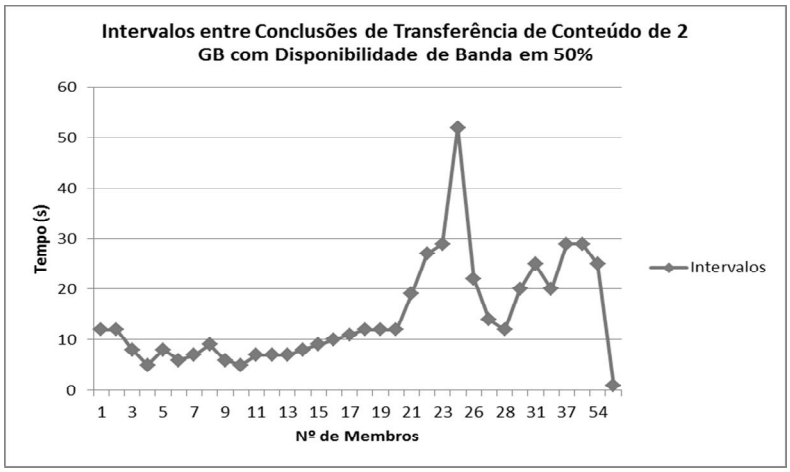


Figura 6.34: Intervalos entre Conclusões de Transferência de Conteúdo de 2 GB com Disponibilidade de Banda em 50%

O comportamento do algoritmo de transferência de conteúdos, no que diz respeito aos tempos de conclusão e intervalos entre términos de *downloads*, é causado pelo aumento da disponibilidade de partes dos conteúdos compartilhados. Uma vez que as fontes para *downloads* aumentam, o algoritmo tende a balancear a carga de transferência de conteúdo, a fim de aumentar a taxa de vazão de partes por segundo. As taxas de vazão de acordo com as disponibilidades de banda dos membros de uma rede social *peer-to-peer* podem ser visualizadas nos gráficos apresentados pela Figura 6.35, Figura 6.36 e Figura 6.37. O gráfico apresentado pela Figura 6.34 apresenta um pico que representa o momento em que aumenta o número de requerentes solicitando o conteúdo compartilhado. Isto faz com que a disputa por partes de conteúdo aumente, fazendo com que os intervalos entre conclusões de transferência de conteúdo aumente.

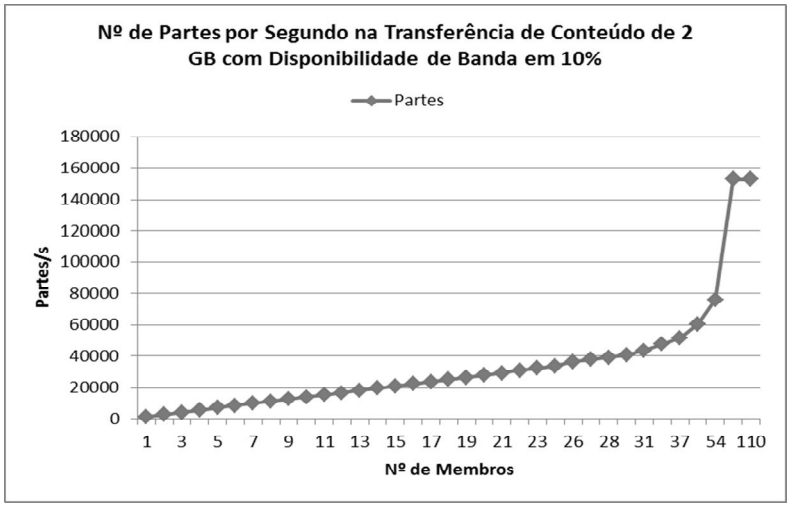


Figura 6.35: Número de partes por segundo na transferência de conteúdo de 2 GB com disponibilidade de banda em 10%

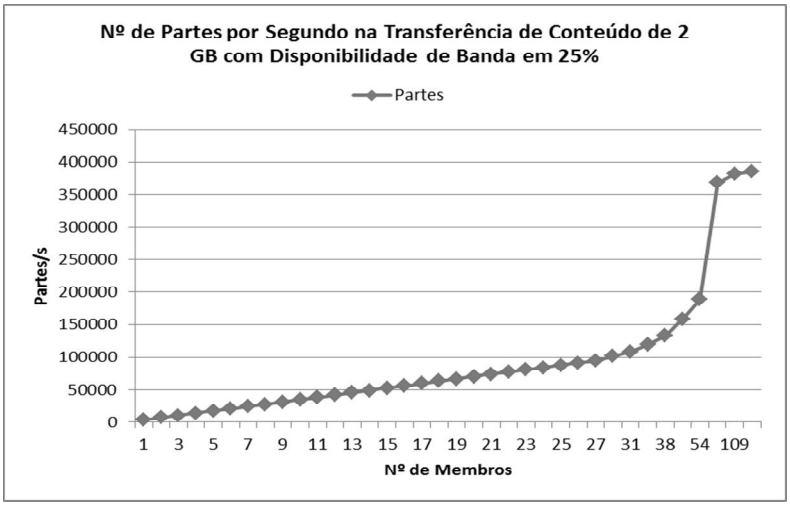


Figura 6.36: Número de partes por segundo na transferência de conteúdo de 2GB com disponibilidade de banda em 25%

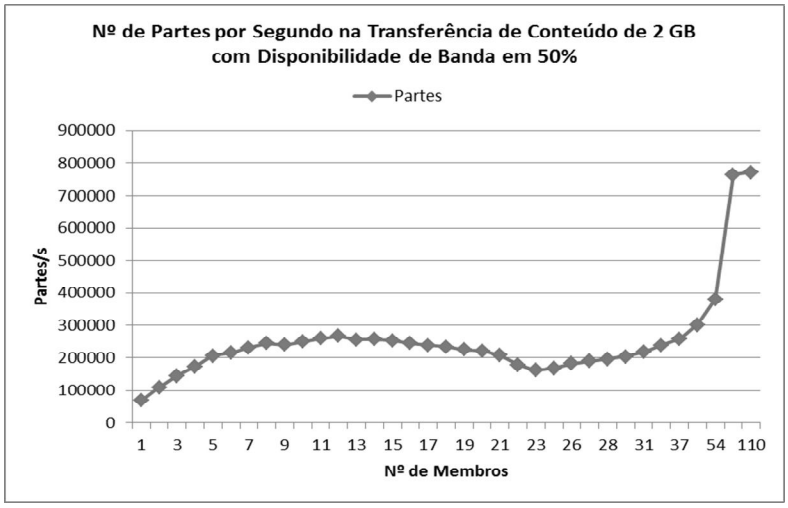
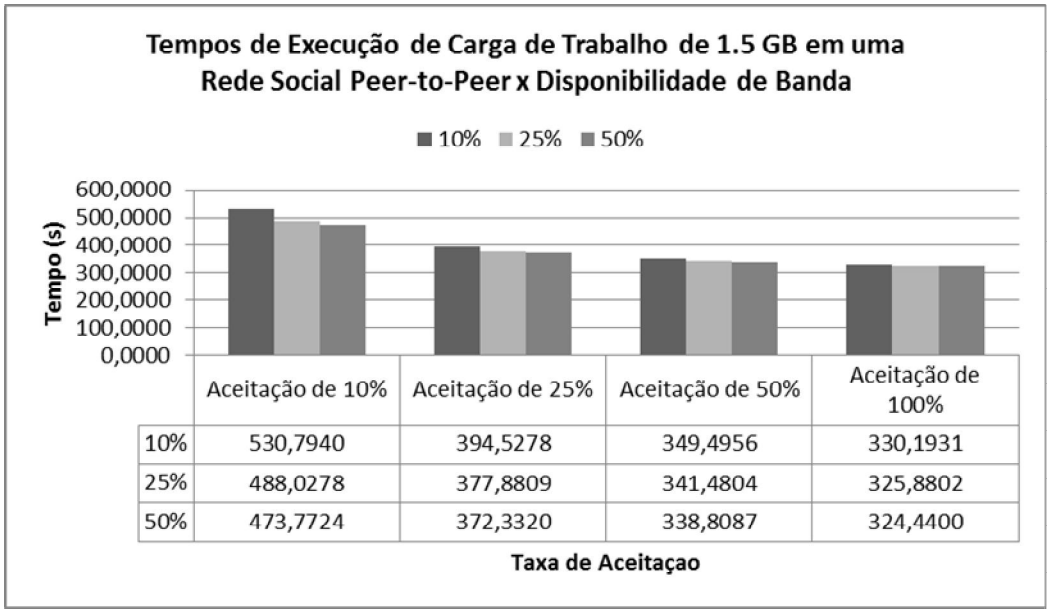


Figura 6.37: Número de partes por segundo na transferência de conteúdo de 2GB com disponibilidade de banda em 50%

Com em todos os resultados apresentados nesta seção, é possível concluir que redes sociais podem ser utilizadas como ambiente de armazenamento de conteúdos compartilhados. Além disto, também é possível concluir que a velocidade com que um conteúdo é transferido de um membro a outro em uma rede social *peer-to-peer* pode ser aumentada se o conteúdo compartilhado estiver armazenado de maneira colaborativa entre os membros da rede social *peer-to-peer*.

6.3.7 Cenário 7: Execução de Tarefas

Seguindo a mesma configuração de rede social *peer-to-peer* apresentada na Seção 6.3.6, foram realizados experimentos para avaliar o possível desempenho de uma rede social *peer-to-peer* como ambiente computacional para execução distribuída de tarefas, sobre os compartilhamentos de recursos computacionais disponíveis na rede social *peer-to-peer*. Nestes experimentos, foram consideradas quatro situações no que diz respeito à colaboração dos membros para a execução de tarefas em seus *peers.* Assim, em uma rede social *peer-to-peer* com 110 membros, foram consideradas situações em que 10%, 25%, 50% e 100% dos membros aceitam a execução de tarefas sobre seus compartilhamentos de recursos. Além destas situações, foi considerada a disponibilidade de banda, de modo que em cada uma das situações seja possível visualizar o impacto nos tempos de processamento de uma carga de trabalho. Como mencionado anteriormente, a banda de comunicação do sistema distribuído varia em 10%, 25% e 50% de sua capacidade total.



	Aceitação de 10%	Aceitação de 25%	Aceitação de 50%	Aceitação de 100%
10%	530,7940	394,5278	349,4956	330,1931
25%	488,0278	377,8809	341,4804	325,8802
50%	473,7724	372,3320	338,8087	324,4400

Figura 6.38: Tempos de execução de carga de trabalho de 1.5 GB em um rede social *peer-to-peer*

Com o objetivo de obter um ponto de referência no que tange o desempenho de uma rede social *peer-to-peer*, foram investigados trabalhos que fornecem informações de tempo de execução de aplicações de carga divisível sobre recursos computacionais heterogêneos. A partir de abstrações, semelhantes às realizadas em (YANG e CASANOVA, 2003; ASSIS *et al.* 2006), foi possível chegar a resultados próximos ao apresentados nestes trabalhos. No entanto, os resultados obtidos após as simulações de execução de tarefas não estão associados ao desempenho do algoritmo de escalonamento de tarefas. Na verdade, a proximidade de resultados se deu pela capacidade da infraestrutura computacional construída a partir da rede *peer-to-peer*, conforme apresentado na Seção 6.2.4. Além disto, não foi considerada a variação dos recursos computacionais cedidos pelos membros da rede social *peer-to-peer*, pois, uma vez que os recursos foram cedidos, parte-se do princípio que seus cedentes irão colaborar para a execução das tarefas.

Observando o gráfico apresentado pela Figura 6.38, percebe-se que os tempos de execução diminuem à medida que uma quantidade maior de recursos é ofertada, quando membros de uma rede social *peer-to-peer* aceitam ceder recursos para processamento de tarefas. Apesar de diminuírem, os tempos de execução tendem a estabilizar à medida que a oferta de recursos aumenta, pois número de partes de carga de trabalho por *peer* diminui até que chegue a uma parte por *peer*.

6.4 CONSIDERAÇÕES FINAIS

Neste capítulo, foi primeiramente avaliado o algoritmo para estruturação de redes *peer-to-peer*, dada a importância do mesmo no que diz respeito ao impacto do mesmo sobre a infraestrutura de comunicação utilizada para interconectar computadores em ambientes computacionais de *e-Science* de larga escala. Logo após, foram avaliadas cada uma das políticas de computação distribuída baseadas em redes sociais propostas neste estudo. A avaliação destas políticas teve como objetivo principal obter resultados que pudessem apresentar os prós e contras do uso de redes sociais *peer-to-peer* como uma alternativa metodológica para compartilhamento de recursos em ambientes de *e-Science*. Assim, é possível concluir que, por meio de uma infraestrutura computacional dinâmica como é uma rede *peer-to-peer* não estruturada, é possível formar redes sociais *peer-to-peer* com finalidades similares as de uma nuvem computacional (BOSS, et al. 2007). Como pode ser visto, é possível agregar uma grande quantidade de recursos compartilhados de hardware. Um ponto negativo, com relação ao uso de redes sociais *peer-to-peer* para compartilhamento de recursos em ambientes de *e-Science*, é a dependência da colaboração que deve haver entre os membros de uma rede social. Caso isto não ocorra, o uso de redes sociais *peer-to-peer* para a finalidade proposta neste trabalho fica comprometido. Mesmo que usuários tenham interesses em comum e agreguem uma grande quantidade de recursos compartilhados, isto não é suficiente para haver colaboração, pois os membros de uma rede social *peer-to-peer* devem realmente ser capazes de colaborar. Neste caso, de nada adianta formar redes sociais *peer-to-peer* baseadas em interesses relacionados a *e-Science*, se os membros são somente entusiastas em *e-Science*. Para haver colaboração e consequentemente o uso dos recursos compartilhados em uma rede social *peer-to-peer*, é necessário que uma rede social seja formada por especialistas cujos interesses estão relacionados aos conceitos definidos em uma oportunidade de colaboração. Como este trabalho está focado ao domínio de *e-Science*, esta restrição deve ser levada em consideração. Para outros domínios, como *e-Gov*, *e-Commerce*, *e-Learning*, entre outros, devem ser realizados outros estudos, aplicando a metodologia apresentada nesta dissertação de mestrado, de modo que seja possível identificar vantagens e desvantagens sobre o uso de compartilhamento de recursos em ambientes de computação distribuída baseados em redes sociais.

7 CONSIDERAÇÕES FINAIS

Este capítulo apresenta as principais contribuições sobre o estudo do uso de redes sociais para o compartilhamento de recursos em redes sociais *peer-to-peer*, bem como possíveis direcionamentos para futuras pesquisas a serem desenvolvidas. Assim, na Seção 7.1 serão descritas as contribuições trazidas por este trabalho. Na Seção 7.2, são apresentados os trabalhos futuros.

7.1 PRINCIPAIS CONTRIBUIÇÕES

Esta dissertação apresentou um estudo sobre o uso de redes sociais *peer-to-peer* para compartilhamento de recursos em ambientes de *e-Science.* O objetivo foi atingido por meio de políticas de computação distribuída baseadas em redes sociais, executadas sobre uma infraestrutura computacional formada por uma rede *peer-to-peer* não estruturada. A partir destas políticas, foi possível criar redes sociais *peer-to-peer*, agregar recursos compartilhados pelos membros da rede social, compartilhar conteúdo, armazenar conteúdos compartilhados de maneira colaborativa e executar aplicações de cargas de trabalho divisíveis.

Nesse sentido, as principais contribuições são:

1. Políticas de computação distribuída baseadas em redes sociais para compartilhamento de recursos. Estas políticas permitem a criação de redes sociais sobre uma infraestrutura computacional formada por uma rede *peer-to-peer* não estruturada. Além disso, elas possibilitam a formação de um ambiente de computacional cuja escala aumenta à medida que novos membros são admitidos na rede social. O ambiente computacional é formado a partir da agregação dos recursos computacionais compartilhados pelos membros de uma rede social *peer-to-peer*. Isto permite que novos recursos sejam adicionados, sem que o ambiente seja reconfigurado manualmente, ou seja, a capacidade computacional é aumentada de maneira transparente e sem a intervenção humana no que tange a configuração do ambiente computacional. Sobre o agregado de recursos computacionais, as políticas oferecem meios para que conteúdos sejam compartilhados e armazenados de maneira colaborativa, permitindo que

informações comuns aos membros da rede social estejam sempre disponíveis, a não ser que todos os membros da rede social se mantenham sempre desconectados. Por fim, as políticas ainda oferecem meios de uma aplicação de carga de trabalho divisível ser executada sobre os compartilhamentos de tempos de ocupação de processadores e espaços de memória.

2. Proposta de uma arquitetura para implementação de um *middleware* baseado nas políticas de computação distribuída baseadas em redes sociais. A arquitetura é responsável por disponibilizar interfaces e componentes que permitam a criação de aplicações de computação distribuída, de modo que estas possam tirar proveito da infraestrura computacional criada a partir dos compartilhamentos realizados por usuários, no momento da formação e, posteriormente, na evolução de uma rede social *peer-to-peer*.
3. Resultados experimentais de desempenho tanto dos algoritmos *peer-to-peer* quanto das políticas de computação distribuída baseadas em rede sociais. Os experimentos realizados com o simulador de redes mostraram que é possível, por meio de redes sociais *peer-to-peer*, criar ambientes computacionais distribuídos de grande escala. Para *e-Science*, isto é de grande relevância, pois as atividades nesta área necessitam de uma grande quantidade de recursos computacionais, de modo que estes possam ser utilizados para armazenar grandes montantes de dados e computar intensivamente estes dados.
4. Implementação de um simulador de redes *peer-to-peer*, onde algoritmos de redes *peer-to-peer* podem ser testados antes de serem utilizados na implementação de uma aplicação ou *middleware* de computação distribuída. Além disso, o simulador também oferece suporte a prototipação e simulação de políticas de computação distribuída de maneira fácil, pois não é necessário investir uma grande quantidade de tempo para aprender e usar o conjunto de componentes de software existentes no simulador.

Apesar de a abordagem proposta apresentar uma grande flexibilidade no que tange o aumento da escala de recursos para ambientes computacionais de *e-Science*, esta é frágil com relação ao modelo utilizado para compor a infraestrutura de computação, pois redes *peer-to-peer* normalmente são instáveis. No entanto, em ambientes de *e-Science*, pesquisadores se comprometem uns com os outros, quando todos buscam um objetivo comum, diferente dos usuários de sistemas *peer-to-peer* para compartilhamento de arquivos. Em sistemas de compartilhamento de arquivos, usuários tendem a ter um comportamento egoísta: os mesmos

realizam o *download* dos arquivos compartilhados de seu interesse e, posteriormente, se desconectam do sistema. O retorno ao sistema de compartilhamento de arquivos só se dá quando os usuários desejam novamente realizar o *download* de outro arquivo compartilhado no qual estejam interessados. A maneira de se manter uma rede social *peer-to-peer* com alta disponibilidade de seus recursos e conteúdos compartilhados é por meio da colaboração intensa de seus membros.

Durante a elaboração deste trabalho, obtivemos três publicações (GONÇALVES *et al.* 2007a; GONÇALVES *et al.* 2007b; GONÇALVES, OLIVEIRA e BRAGANHOLO 2009). São elas:

- *An Architectural Model for Applications Based on Mobile Services*, aceito para publicação na *International Multi-Conference on Computing in the Global Information Technology*, realizada em março de 2007. Este artigo apresenta uma proposta preliminar de uma arquitetura de software orientada a serviços;
- *A Software Architecture for the Provisioning of Mobile Services in Peer-to-Peer Environments*, publicado no *International Conference on Internet and Web Applications and Services*, realizada em maio de 2007. Este artigo apresenta uma evolução do modelo arquitetural publicado anteriormente;
- Compartilhamento de Dados e Recursos Computacionais de Armazenamento em Redes P2P Sociais, publicado no *e-Science Workshop*, realizado em outubro de 2009. Este artigo apresenta a proposta de uso de redes sociais *peer-to-peer* para compartilhamento e armazenamento colaborativo de conteúdos.

Após a apresentação de todo o estudo sobre o uso de redes sociais *peer-to-peer* para o compartilhamento de recursos em ambientes de *e-Science*, pretende-se trabalhar em um artigo apresentando todos os resultados deste estudo.

Devido à complexidade das políticas de computação distribuída baseada em redes sociais e limitações de tempo, os esforços foram direcionados para o desenvolvimento de um simulador de redes *peer-to-peer* cujo objetivo é oferecer mecanismos para a prototipação e simulação de ambientes distribuídos baseados em redes sociais *peer-to-peer*. Assim, a implementação e otimização das políticas apresentadas nesta dissertação ficaram como trabalhos futuros. É importante ressaltar, no entanto, que o simulador construído neste trabalho é capaz de simular todas as políticas definidas nesta dissertação.

7.2 TRABALHOS FUTUROS

Até o término deste trabalho, não foi encontrada nenhuma abordagem similar à que foi proposta aqui. Por isto, conclui-se que o uso de redes *peer-to-peer* sociais para compartilhamento de recursos em ambientes de *e-Science* é inédito. Apesar de existirem trabalhos relacionados a compartilhamento de recursos computacionais em redes *peer-to-peer*, armazenamento colaborativo baseado em redes sociais e serviços móveis para computação de tarefas, ainda não foram direcionados esforços na direção proposta nesta dissertação. Por isto, os trabalhos futuros serão realizados ao longo de diferentes etapas, que são apresentadas nas próximas subseções.

7.2.1 Plataforma de Computação *Peer-to-Peer* Orientada a Objetos Móveis

Esta fase compreenderá o desenvolvimento de uma plataforma de computação *peer-to-peer* orientada a serviços móveis. São três as motivações para este trabalho futuro:

1. Desenvolver aplicações distribuídas é muito complexo, pois o desenvolvedor precisa ter um grande conhecimento, tanto teórico quanto prático, no que diz respeito a tipos de redes *peer-to-peer* e seus respectivos protocolos;
2. Falta de plataformas para desenvolvimento de software baseado em computação *peer-to-peer*, tendo o JXTA (JXTA 2010) como a mais conhecida;
3. Mesmo existindo plataformas como o JXTA, estas impõem todo um *overhead* de conhecimento sobre sua estrutura de componentes de software. Muitas vezes, quando é necessário implementar funcionalidades não oferecidas pela plataforma, o custo para adaptar a plataforma a novos requisitos pode ser alto em termos de tempo e esforço de trabalho;

Por estes motivos, é proposto o desenvolvimento de uma plataforma de computação *peer-to-peer*, que além de oferecer todos os mecanismos para provisão de computação *peer-to-peer*, também é orientada a objetos móveis. Por isto, serviços de aplicações podem ser executados de maneira distribuída em computadores, onde tais serviços não foram configurados. Com isto, funcionalidades de aplicações que precisam lidar com o processamento intensivo de dados podem ser desenvolvidas utilizando serviços móveis. Isto deve permitir que os objetos móveis executem sobre um conjunto de compartilhamentos de tempos de processamento e espaços de memória, sem haver intervenção humana no processo de configuração do ambiente computacional e execução dos serviços de aplicações.

Para viabilizar a proposta de desenvolvimento dessa plataforma, o modelo arquitetural de software apresentado no Capítulo 4 servirá como especificação de base para o projeto e desenvolvimento da plataforma. Como mencionado anteriormente, o modelo arquitetural proposto segue o padrão arquitetural *microkernel*, a fim de possibilitar a inversão de dependência no que diz respeito aos serviços básicos e serviços de aplicações. Entre estes dois tipos de serviços não deve existir acoplamento forte na configuração das dependências entre um e outro. Por este motivo, cada camada do *microkernel* e seus respectivos componentes deverá oferecer interfaces bem definidas, a fim de possibilitar a adaptação da plataforma a requisitos relacionados aos ambientes e atividades de *e-Science*.

7.2.2 Políticas de Computação Distribuída Baseadas em Redes Sociais como Serviços de Plataforma

Uma vez implementada a plataforma de computação *peer-to-peer* orientada a serviços móveis, o principal objetivo desta etapa é introduzir as políticas de computação distribuída baseadas em redes sociais na forma de serviços básicos de plataforma. Portanto, ficarão previstas para esta etapa as seguintes atividades:

1. Implementação de serviços de registro de usuários no sistema distribuído;
2. Implementação de serviços de identificação e autorização de usuários no sistema distribuído;
3. Implementação das políticas de computação distribuída baseadas em redes sociais.
4. Execução das políticas em um ambiente computacional real, por meio de protótipos de aplicações desenvolvidas sobre a plataforma de computação.

7.2.3 Refinamento das Políticas de Computação Distribuída Baseada em Redes Sociais

Após a coleta dos resultados obtidos durante a execução em um ambiente computacional real, o principal objetivo desta etapa é avaliar o desempenho das políticas de computação distribuída baseadas em redes sociais. De antemão, propomos estudos em duas frentes:

- **Sistemas de recomendação –** estudos e propostas nesta área melhorariam os mecanismos de criação de redes sociais e admissão de membros nestas redes. Com isto, novos algoritmos de combinação social podem ser desenvolvidos, com o objetivo de trazer maior robustez no processo de análise de

compatibilidade entre restrições de oportunidades de colaboração e os perfis de usuários do sistema distribuído;

- **Balanceamento de carga –** estudos e propostas nesta área colaborariam para a melhoria da qualidade de serviço da plataforma. A partir disto, algoritmos e estratégias para balanceamento de carga podem ser propostos para otimizar a obtenção de grandes volumes de dados compartilhados em uma rede social *peer-to-peer* e a divisão das cargas de trabalho processadas sobre compartilhamentos de ciclos de processadores e espaços de memória.

REFERÊNCIAS

AKBARINIA, R. ; PACITTI, E. ; VALDURIEZ, P. Data concurrency in replicated DHTs. In: ACM INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA. 2007. Beijing. **Proceedings ...** New York: ACM, 2007. p. 211-222.

_____. Design and implementation of APPA. In: BALDONI. R. et al. **Global data management.** Amsterdan: IO Press, 2006. p. 98-123.

AL KISWANY, S. et al. Are P2P data-dissemination techiniques viable in today's data Intensive scientific colaborations. In: KERMARREC, A. M. ; BOUGE, L. ; PRIOL, T. (Ed.). **Euro-Par 2007 parallel processing.** Berlin: Springer-Verlag, 2007. p. 404-414. (Lecture Notes in Computer Science, 4641).

ASSIS, L. et al. Uma heurística de particionamento de carga divisível para grids computacionais". In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 24., 2006. Curitiba, **Anais ...** Curitib:, SBC, 2006.

BLAST: Basic local alignment search tool. 2010. Disponível em: http://blast.ncbi.nlm.nih.gov/ , Acesso em: 14 jun. 2010.

BOSS, G. et al. **Cloud computing**. 2007. Disponível em: grid.lzu.edu.cn/resource/article_download.jsp?id=33. Acesso em: 22 dez. o de 2009.

BUSCHMANN, F. et al. **Pattern-oriented software architecture***:* a system of pattern. Chichester: John Wiey & Sons, 2001. v. 1

CARCHIOLO, V. et al. Emerging structures of P2P networks induced by social relationships. **Computer Communications,** Newton**,** v. 31, n. 3, p. 620-628. Feb. 2008:

CHAPPEL, D. **A Short introdution to cloud platforms**: an enterprise-oriented view. 2009. Disponível em: http://www.davidchappell.com/CloudPlatforms--Chappell.pdf. Acesso em: 02 jan. 2009.

DAVID, P. A. **Towards a cyberinfrastructure for enhanced scientific collaboration: providing its 'soft' foundations may be the hardest part**. Oxford: Oxford Internet Institute, 2004. (Research Report).

DEEPAK, A. ; CRUPI, J. ; MALKS, D. **Core J2EE Patterns: melhores práticas e estratégias de design**. 2. ed. Rio de Janeiro: Campus, 2004.

DRUSCHEL, P. ; RWOSTROW, A. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM INTERNATIONAL CONFERENCE ON DISTRIBUTED SYSTEMS PLATFORMS. 2001. Heidelberg. **Proceedings ...** London: Springer-Verlag, 2001.

EL-KHATIB, K. et al. Personal and service mobility in ubiquitous computing environments. **Wireless Communications and Mobile Computing**, Chichester, v. 4, n. 6, p. 595-607, Sept. 2004.

FAROOQ, U. et al. Designing for e-science: requirements gathering for collaboration in CiteSeer". **International Journal of Human-Computer Studies**, London, v.67, n. 4, p. 297-312, Apr. 2009..

FASTA **Fasta tools for sequency similarity protein or nucleotide databases**. 2010. Disponível em: http://www.ebi.ac.uk/Tools/fasta/index.html. Acesso em 14 jun. 2010.

GAMMA, E. et al. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2001.

GILLESPIE, D. S. ; GLISSON, C. Quantitative methods in mocial work: state of the art. **Google Books**. 1992. Disponível em: http://books.google.com.br/books?id=hhK8IfVljFoC&printsec=frontcover&source=gbs_navlinks_s#v=onepage&q=&f=false. Acesso em 06 de jan 2010.

GNUTELLA. **The gnutella protocol specification v0.4**. 2009. Disponível em: http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf. Acesso em: 22 dez. 2009.

GONÇALVES, F. ; OLIVEIRA, C. E. T. ; BRAGANHOLO, V. Compartilhamento de dados e recursos computacionais de armazenamento em redes P2P". In: E-SCIENCE WORKSHOP. 2009, Fortaleza,. **Anais ...** Fortaleza: SBC/ACM, 2009. p. 41-48.

GONÇALVES, F. et al. A Software architecture for the provisioning of mobile services in peer-to-peer environments. In: INTERNATIONAL CONFERENCE ON INTERNET AND WEB APPLICATIONS AND SERVICES, 2., . Morne, Mauritius, 2007. **Proceedings ...** Piscataway: IEEE, 2007. p. 9.

_____. An Architectural model for applications based on mobile services. In: International Multi-Conference on Computing in the Global Information Technology-Challanges for the Next Generation of IT & C-, 2., 2007, Guadaloupe. **Proceedings ...** Guadaloupe: IARIA, 2007. p. 2. *ICCGI 2007.*

HANDOREAN, R. et al. Context aware session management for services in ad hoc networks. In: IEEE INTERNATIONAL CONFERENCE ON SERVICES. COMPUTING, 2005, Orlando. **Proceedings ...** Piscataway, 2005. p. 113-120.

HUANG, J-J. ; CHANG, S-C. ; HU, S-J Searching for answers via social networks". In: IEEE CONSUMER COMMUNICATIONS AND NETWORKING CONFERENCE. 2008, Las Vegas. **Proceedings ...** Piscataway, 2008. p. 289-293.

JEE. **Java EE**. 2010. Disponível em: http://pt.wikipedia.org/wiki/Java_EE. Acesso em: 07 mai. 2010.

JINI. **Jini.org**. 2009. Disponível em: http://www.jini.org. Acesso em: 27 dez. 2009.

JXTA. **JXTA**. 2010. Disponível em: https://jxta.dev.java.net. Acesso em: 07 mai. 2010.

LI, J. ; DABEK, F. **F2F**: reliable storage in open networks. In: INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS, 5*.,* 2006. Santa Barbara, **Proceedings ...** Santa Barbara, 2006.

LIU, L. ; ANTONOPOULOS, N. ; MACKIN, S. Social peer-to-peer for resource discovery. In: EUROMICRO INTERNATIONAL CONFERENCE ON PARALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING, 15*.,* 2007. Naples, **Proceedings …** Naples: LCAR, 2007. p. 459-466.

LO, V. ; ZHOU, D. Clustering on the fly: resource discovery in a cycle shar-ing peer-to-peer systems. In: IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID*.* 2004, Chicago. **Proceedings …** Piscatawary: IEEE, 2004. p. 66-73.

MASCOLO, C. ; CAPRA, L. ; EMMERICH, W. Mobile computing middleware. In: Gregori, E ; Anastasi, G.; Basagni, S. (Ed.) **Advanced Lectures on Networking**, Berlin: Springer-Verlag, 2002. p. 506-510. (Lecture Notes in Computer Science, 2497).

MATTOSO, M. et al. Gerenciando experimentos científicos em larga escala. In: SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE, 35., 2008, Belém. **Anais ...** *SEMISH.* Belém: SBC, 2008. p. 121-135.

MILOJICIC, D. S. et al. **Peer-to-peer computing**. Palo Alto: HP Laboratories, 2002. (Research Report, HPL-2002-57).

NANDY, S. ; CARTER, L. ; FERRANTE, F. GUARD: gossip used for autonomous resource detection. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 19., 2005, Denver. **Proceedings ...** Los Alamitos: IEEE/ACM, 2005. p. 58.

OGASAWARA, E., et al. A P2P approach to many tasks computing for scientific workflows. In: VECPAR´10. INTERNATIONAL MEETING HIGH PERFORMANCE COMPUTING FOR COMPUTATIONAL SCIENCE, 9., 2009. Berkeley, **Proceedings …** Berkeley: LBNL, 2009.

OLIVEIRA, M. I. S. **OurBackup**: uma solução P2P de backup baseada em redes sociais. 2007. Tese (Mestrado em Ciência da Computação) – Centro de Engenharia Elétrica e Informática, Universidade Federal de Campina Grande, Campina Grande, Paraíba, 2007.

PAPAZOGLOU, M. P. ; GEORGAKOPOULOS, D.. Service oriented computing. **Communications of the ACM**, New York, v. 46, n. 10, p. 25-28, Oct. 2003:

PARASHAR, M. ; HARIRI, S. Autonomic computing: an overview. In: BANÊTTRE, J-P, et. al (Ed). **Unconventional Programming Paradigms**. Berlin: Springer-verlag, 2005. p. 257-269. (Lectures Notes in Conputer Science, Vol. 3566).

PARETO. **Pareto distribution**. 2010. Disponível em: http://wapedia.mobi/en/Pareto_distribution. Acesso em: 07 mai. 2010.

POISSON. **Poisson distribution**. 2010. Disponível em: http://wapedia.mobi/en/Poisson_distribution. Acesso em: 07 mai. 2010.

PRIBERAM. **Paradigma.** 2010. Disponível em: http://www.priberam.pt/dlpo/default.aspx?pal=PARADIGMA. Acesso em: 14 jun. 2010.

RATNASAMY, S. et al. A Scalable content addressable network. In: CONFERENCE ON APPLICATIONS, TECHNOLOGIES, ARCHITECTURES AND PROTOCOL FOR COMPUTER COMMUNICATIONS. 2002, San Diego. **Proceedings ...** New York: ACM, 2001. p. 161-172.

RIVA, O. et al. Mobile services: context-aware service migration in ad hoc networks. **IEEE Transactions on mobile Computing**, New York, v. 6, n. 12, p. 1-16, Dec. 2007.

SCHIMIDT, D. et al. **Pattern-oriented software architecture**: patterns for concurrent and networked objects. Chichester: John Wiley & Sons, 2000. v. 2.

SEDMIDUBSKY, J. et al. Adaptive approximate similarity searching through metric social networks. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 24. 2008, Cancun. **Proceedings ...** Los Alamitos: IEEE, 2008. p. 1424-1426.

SEMENOV, A. **Evolution of peer-to-peer algorithms**: past, present and future. Helsinki: Helsinki University of Technology, 2005. (Techinal Report, HUT T-110.551).

SIMONTON, E. ; CHOI, K. B. ; SEIDEL, S. Using gossip for dynamic resource discovery. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, 2006, Columbus. **Proceedings ...** Los Vaqueros Circle: IEEE, 2006. p. 319-328.

SIT, E. ; MORRIS, R. Security considerations for peer-to-peer distributed hash tables. In: DRUSCHEL, P. ; KAASHOEK, F. ROWSTRON, A. (Ed). **Peer-to-Peer Systems.** Berlin: Springer-Verlag, 2002. p. 261-269. (Lecture Notes in Computer Science, 2429).

SONNENWALD, D. H. Scientific collaboration: a synthesis of challenges and strategies. **Annual Review of Information Science and Technology**, White Plains, v. 41, n. 1, p. 643-681, 2007.

STOICA, I. et al. Chord: a scalale peer-to-peer lookup service for internet applications. In: CONFERENCE ON APPLICATIONS, TECHNOLOGIES, ARCHITECTURES AND PROTOCOL FOR COMPUTER COMMUNICATIONS, 2001. San Diego. **Proceedings ...**, New York: ACM, 2001. p. 149-160.

STUTZBACH, D. ; REJAIE. R. Understanding churn in peer-to-peer networks. In: CONFERENCE ON INTERNET MEASUREMENT, 2006, Rio de Janeiro, **Proceedings ...** New York: ACM SIGCOMM, 2006. p. 189-202.

TANENBAUM, A. S. ; VAN STEEN, M. **Sistemas distribuídos:** princípios e paradigmas. 2. ed. São Paulo: Pearson Prentice Hall, 2007.

WCG. **World Community Grid.** 2009. Disponível em: http://www.worldcommunitygrid.org/. Acesso em 20 dez. 2009.

WEIBULL. **Weibull distribution**. 2010. Disponível em: http://wapedia.mobi/en/Weibull_distribution. Acesso em 07 mai. 2010.

YANG, S. J. H. et al. Social network supports in knowledge sharing. In: INTERNATIONAL CONFERENCE ON INNOVATIVE COMPUTING, INFORMATIO AND CONTROL 2., 2007, Kumamoto. **Proceedings ...** , Piscataway: IEEE, 2007. p. 148-148.

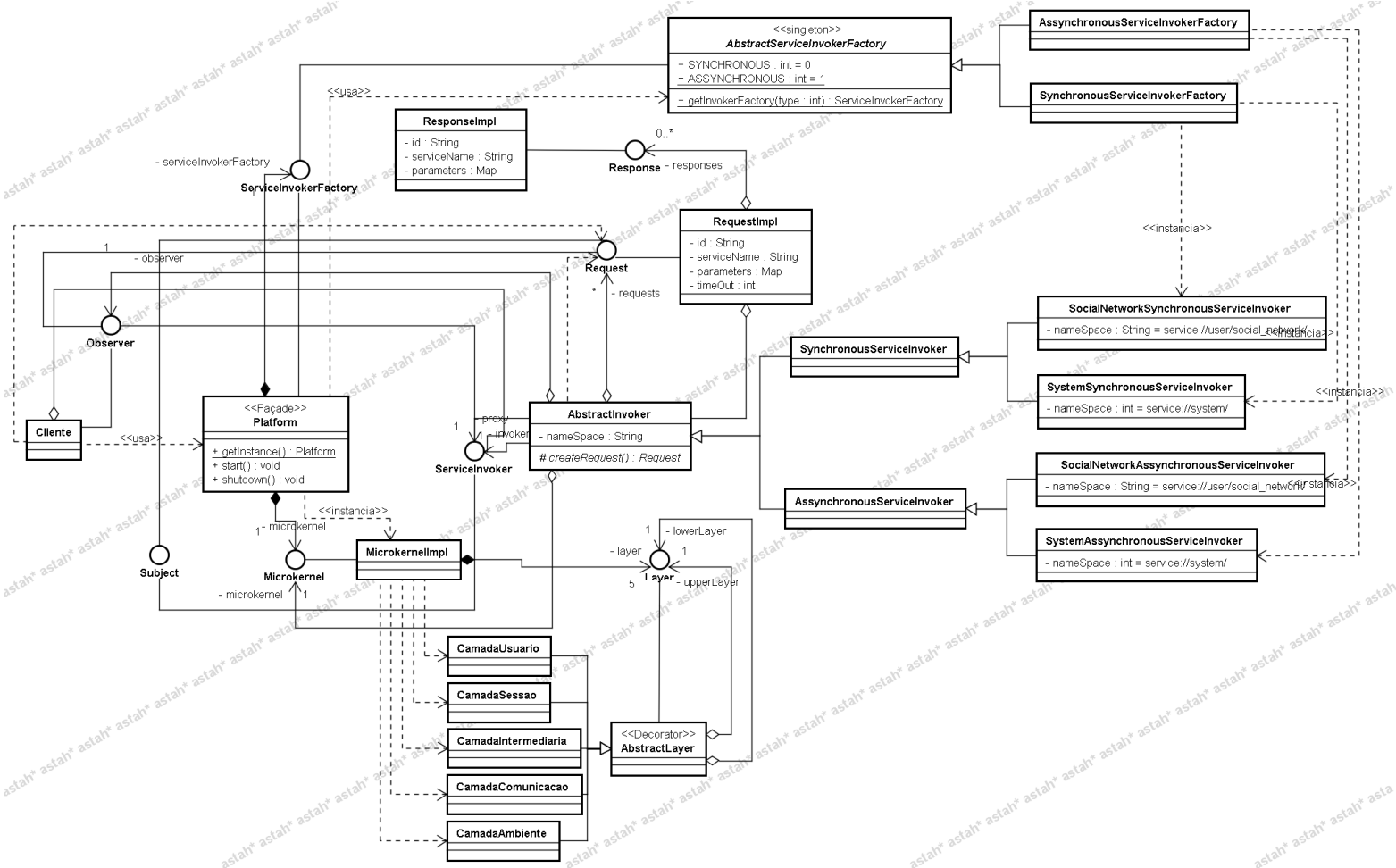
YANG, Y. ; CASANOVA, H. RUMR: Robust scheduling for divisible workloads. In: IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 12., 2003. Seattle. **Proceedings ...** , Los Alamitos: IEEE, 2003. p. 114-114.

ZHAO, B. Y. ; KUBIATOWICZ, J. E. ; JOSEPH, A. **Tapestry**: an infrastructure for fault-tolerance wide-area location and routing. Berkeley: University of California at Berkley, Computer Science Department, 2002. (Technical. Report,. UCB/CSD-01-1141).

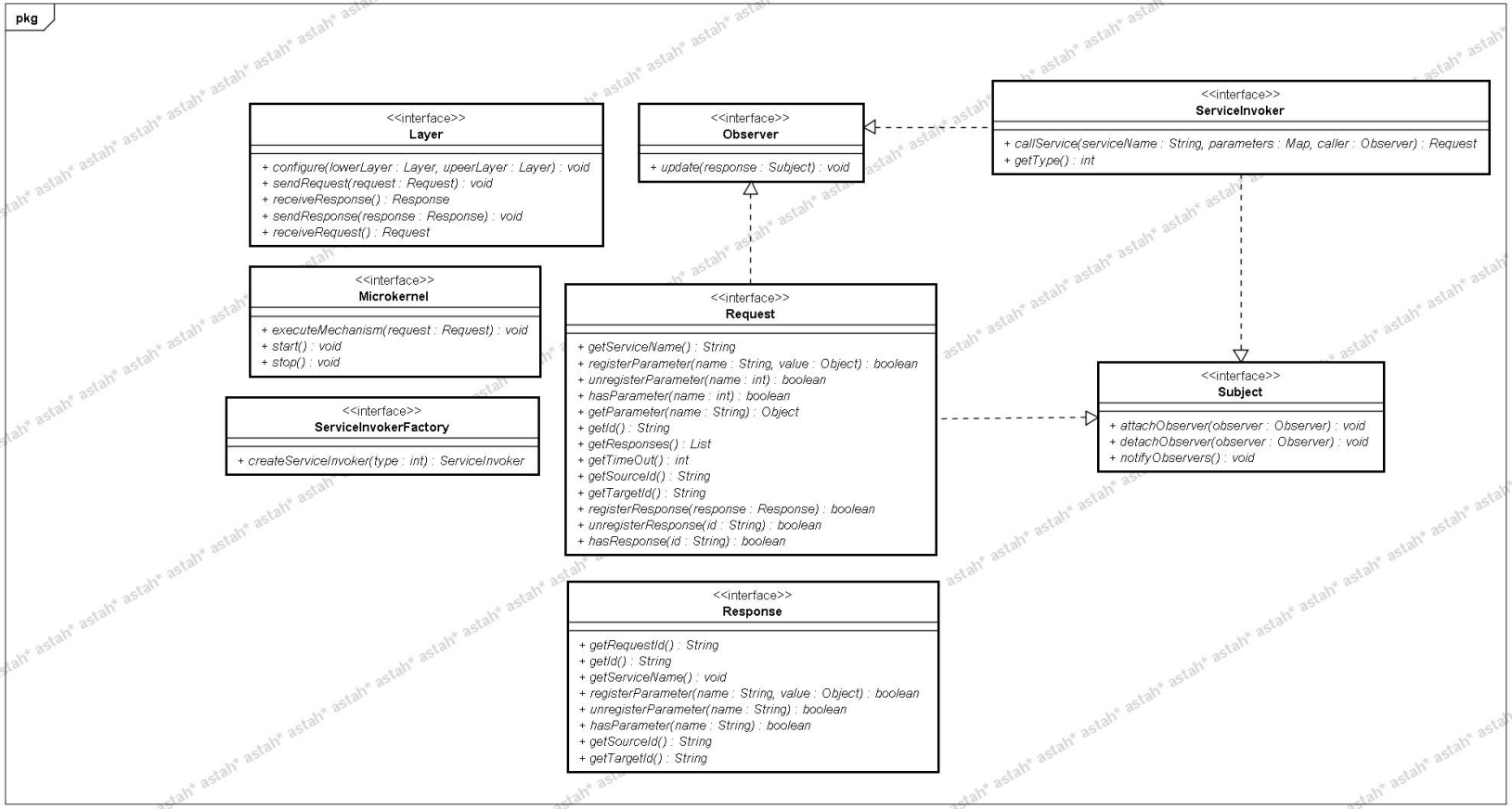
ZHAO, Y. ; RAICU, I ; FOSTER, I. Scientific workflow systems for 21st century. new bottle or new wine? IEEE CONGRESS ON SERVICES, 2008, Honolulu. **Proceedings ...** Los Alamitos: IEEE, 2008. Part. 1. p. 467-471.

ZLIB. **zLib Home Site**. 2010. Disponível em: http://www.zlib.net/. Acesso em: 07 mai. 2010.

APÊNDICE A – DIAGRAMA DE CLASSES DO MICROKERNEL



APÊNDICE B – INTERFACES DO MICROKERNEL



APÊNDICE C – ALGORITMOS DE REDE *PEER-TO-PEER*

Neste apêndice, serão apresentados os pseudocódigos dos algoritmos de rede *peer-to-peer* propostos no capítulo 5. Seguindo a organização de como foram propostos, as próximas seções detalharão o pseudocódigo de cada um dos algoritmos.

C.1 ESTRUTURAÇÃO DA REDE *PEER-TO-PEER*

Para que os *peers* possam conectar-se uns aos outros, a fim de forma uma rede *peer-to-peer* não estruturada, o algoritmo para estruturação da rede *peer-to-peer* é dividido em duas partes. A primeira é representada pela função *conectar*, que tem a responsabilidade de iniciar o processo de conexão de um *peer* com a rede *peer-to-peer*. Por fim, a segunda parte consiste no pseudocódigo relacionado ao tratamento de mensagens *JOIN*.

- **Solicitação de Conexão com a Rede *Peer-to-Peer*:**

```
conectar(peer)
inicio
   redePeerToPeer := peer.redePeerToPeer;
   para cada superpeer em redePeerToPeer.peers faça
      idMsg    := gerarIdMensagem();
      saltos   := 1;
      msg      := criarMensagem(idMsg, JOIN, peer.id, superpeer.id, saltos);
      registrarIdPeer(msg.caminho, superpeer.id);
      registrarParametro(msg.parametros, 'enderecos', peer.enderecos);
      registrarParametro(msg.parametros, 'tipoPeer', peer.tipo);
      endereco := obterEnderecoParaComunicacaoSincrona(superpeer.enderecos);
      enviarMsg(msg, endereco);
      msgResposta := receber();
      se msgResposta.rotulo = JOINED então
      inicio
         vizinho := criarVizinho(superpeer.id, superpeer.tipo, superpeer.enderecos);
         registrarVizinho(vizinho.tipo, vizinho);
         peer.relogio = msgResposta.relogio;
         se (peer.tipo = 0 e
```

```
            qtdeVizinhos(superpeer.tipo, peer.vizinhos) = redePeerToPeer.maxConexoesEntreSuperpeer) então
               pare;
          senão
            se peer.tipo = 1 e
               qtdeVizinhos(superpeer.tipo, peer.vizinhos) = redePeerToPeer.maxConexoesEntrePeerSimplesSuperpeers então
                        pare;
             fim;
         fimse;
         se msgResposta.rotulo = JOIN_ERROR então
            continue;
         fimse;
      fimpara;
      peer.conectado := qtdeVizinhos(superpeer.tipo, peer.vizinhos) > 0;
      se peer.conectador = Verdadeito então
         descobrirPeers(peer);
      fimse;
   fim;
```

- **Tratamento de Mensagens *JOIN* Realizada pelos *Superpeers*:**

```
para cada mensagem em peer.mensagensRecebidas faça
   ...
   se mensagem.rotulo = JOIN então
     inicio
        tipoPeer  := obterValorParametro(mensagem.parametros, 'tipoPeer');
        idMsg     := gerarIdMensagem();
        saltos    := 1;
        enderecos := obterValorParametro(mensagem.parametros, 'tipoPeer');
        endereco  := obterEnderecoParaComunicacaoSincrona(enderecos);

        se (tipoPeer = 0 e qtdeVizinhos(tipoPeer, peer.vizinhos) = maxConexoesEntreSuperpeers) ou
           (tipoPeer = 1 e qtdeVizinhos(tipoPeer, peer.vizinhos) = maxConexoesEntreSuperpeersPeersSimples) então
           inicio
              msgResposta = criarMensagem(idMsg, JOIN_ERROR, mensagem.destino, mensagem.origem, saltos);
              registrarIdPeer(msgResposta.caminho);
              enviarMsg(msgResposta, endereco);
              sair;
           fim;
        fimse;

        vizinho = criarVizinho(mesagem.origem, tipoPeer, enderecos);
        registrarVizinho(vizinho.tipo, vizinho);

        msgResposta = criarMensagem(idMsg, JOINED, mensagem.destino, mensagem.origem, saltos);
        registrarIdPeer(msgResposta.caminho);
        enviarMsg(msgResposta, endereco);
```

```
      fim;
    fimse;
    ...
fimpara;
```

C.2 DESCOBERTA DE PEERS

Por se tratar de um algoritmo distribuído, partes da descoberta de *peers* são executadas pelos vários *peers* conectados à rede *peer-to-peer*. Assim, cada tipo de *peer* tem uma responsabilidade durante o processo de descoberta de *peers*. Por isto, os pseudocódigos desta seção foram separados, conforme os tipos de *peers* utilizados para este trabalho, a saber:

- ***Peers* Simples:**
  - **Solicitação de Descoberta de *Peers*:**

```
descobrirPeer(peer)
inicio
      se qtdeVizinhos(0, peer.vizinhos) = 0 então
            sair;
      fimse;
      redePeerToPeer := peer.redePeerToPeer
      idMsg          := gerarIdMensagem();
      saltos         := redePeerToPeer.maxSaltos;
      vizinhos       := obterVizinhosSuperpeers(0, peer.vizinhos);
      para cada vizinho em vizinhos faça
            msg      := criarMensagem(idMsg, PING, peer.origem, vizinho.idPeer, saltos)
            registrarCaminho(msg.caminho, peer.id);
            registrarParametro(msg.parametros, 'tipoPeer', peer.tipo);
            endereco := obterEnderecoParaComunicacaoAssincrona(vizinho.enderecos);
            enviar(msg, endereco);
      fimpara;
fim;
```

  - **Tratamento de Mensagens *PING* e *PONG***:

```
para cada mensagem em simplepeer.mensagensRecebidas faça
   ...
   se mensagem.rotulo = PONG entao
      inicio
         caminhoVolta := obterValorParametro(mensagem.parametros, 'caminhoVolta');
         mensagem.saltos := mensagem.saltos - 1;

         se (não éVizinho(superpeer.vizinhos, caminhoVolta[tamanho(caminhoVolta)-1])) ou
```

```
            (superpeer.id = caminhoVolta[tamanho(caminhoVolta)-1]) entao
            sair;
         fimse;

         vizinho := obterVizinho(superpeer.vizinhos, caminhoVolta[tamanho(caminhoVolta)-1]))
         rota    := criarRota(caminhoVolta[0], caminhoVolta, tamanho(caminhoVolta));
         registrarRota(vizinho.rotas, rota);
      fim;
   fimse;

   se mensagem.rotulo = PING então
      inicio
         mensagem.saltos := mensagem.saltos - 1;
         idMsg           := criarIdMensagem();
         saltos                  := tamanho(mensagem.caminho);
         caminhoVolta       := criarLista();
         adicionarItem(caminhoVolta, superpeer.id);
         msgPong = criarMensagem(idMsg, PONG, mensagem.destino, mensagem.origem, saltos);
         msgPong.caminho := mensagem.caminho;
         registrarParametro(msgPong.parametros, 'caminhoVolta', caminhoVolta);
         vizinho = obterVizinho(superpeer.vizinhos, mensagem.origem);
         endereco        := obterEnderecoParaComunicacaoAssincrona(vizinho.enderecos);
         enviar(msgPong, endereco);

         se não éVizinho(superpeer.vizinhos, mensagem.caminho[0]) então
            inicio
               rota = criarRota(mensagem.caminho[0], mensagem.caminho, tamanho(mensagem.caminho));
               registrarRota(vizinho.rotas,rota);
            fim;
         fimse;
      fim;
   fimse;
   ...
fimpara;
```

- ***Superpeers*:**
  - **Tratamento de Mensagens *PING* e *PONG*:**

```
para cada mensagem em superpeer.mensagensRecebidas faça
   ...
   se mensagem.rotulo = PING então
      inicio
         se mensagem.id existe em superpeer.cache então
            sair;
         fimse;
         adicionarItem(superpeer.cache, mensagem.id);
         mensagem.saltos := mensagem.saltos - 1;
         idMsg           := criarIdMensagem();
```

```
      saltos                    := tamanho(mensagem.caminho);
      caminhoVolta       := criarLista();
      adicionarItem(caminhoVolta, superpeer.id);
      msgPong = criarMensagem(idMsg, PONG, mensagem.destino, mensagem.origem, saltos);
      msgPong.caminho := mensagem.caminho;
      registrarParametro(msgPong.parametros, 'caminhoVolta', caminhoVolta);
      vizinho = obterVizinho(superpeer.vizinhos, mensagem.origem);
      endereco        := obterEnderecoParaComunicacaoAssincrona(vizinho.enderecos);
      enviarMsg(msgPong, endereco);

      se não éVizinho(superpeer.vizinhos, mensagem.caminho[0]) então
         inicio
            rota = criarRota(mensagem.caminho[0], mensagem.caminho, tamanho(mensagem.caminho));
            registrarRota(vizinho.rotas,rota);
         fim;
      fimse;

      se mensagem.saltos = 0 então
         sair;
      fimse;

      vizinhos := obterVizinhos(0, superpeer.vizinhos) + obterVizinhos(1, superpeer.vizinhos);
      para cada vizinho em vizinhos faça
         se vizinho.id existe em mensagem.caminho então
            continue;
         fimse;
         msgPing := criarMensagem(mensagem.id, mensagem.rotulo, peer.id, vizinho.id, mensagem.saltos);
         msgPing.caminho := mensagem.caminho;
         registrarIdPeer(msgPing.caminho, superpeer.id);
         registrarParametro(msgPing.parametros, 'tipoPeer', superpeer.tipo);
         endereço := obterEnderecoParaComunicacaoAssincrona(vizinho.enderecos);
         enviarMsg(msgPing, endereco);
    fimpara;
  fim;
fimse;

se mensagem.rotulo = PONG entao
   inicio
      caminhoVolta := obterValorParametro(mensagem.parametros, 'caminhoVolta');

      se (não éVizinho(superpeer.vizinhos, caminhoVolta[tamanho(caminhoVolta)-1])) ou
         (superpeer.id = caminhoVolta[tamanho(caminhoVolta)-1]) entao
         sair;
      fimse;

      vizinho := obterVizinho(superpeer.vizinhos, caminhoVolta[tamanho(caminhoVolta)-1]))
      rota    := criarRota(caminhoVolta[0], caminhoVolta, tamanho(caminhoVolta));
      registrarRota(vizinho.rotas, rota);
```

```
                removerIdPeer(mensagem.caminho, superpeer.id);

                se não éVizinho(mensagem.caminho[tamanho(mensagem.caminho)-1]) então
                   sair;
                fimse;

                msgPong := criarMensagem(mensagem.id, mensagem.rotulo, mensagem.saltos - 1);
                msgPong.caminho := mensagem.caminho;
                adicionarItem(caminhoVolta, superpeer.id);
                registrarParametro(msgPong.parametros, 'caminhoVolta', caminhoVolta);

                vizinho := obterVizinho(superpeers.vizinhos, mensagem.caminho[tamanho(mensagem.caminho)-1]);

                endereco := obterEnderecoParaComunicacaoAssincrona(vizinho.enderecos);
                enviarMsg(msgPong, endereco);

             fim;
          fimse;
          ...
   fimpara;
```

C.3 ROTEAMENTO DE MENSAGENS

Em um sistema distribuído descentralizado, como uma rede *peer-to-peer* não estruturada, o mecanismo de troca de mensagens entre os *peers* deve ser capaz de garantir a entrega de uma mensagem, quando um *peer* deseja se comunicar com um ou mais *peers*. Assim, cada um dos *peers*, sendo estes *peers* simples ou *superpeers*, são capazes de enviar mensagens para outros *peers* por meio de uma função *enviarMsg*. Quando esta função não é capaz de enviar uma mensagem diretamente para o seu destino, outra função é executada para que a entrega da mensagem seja garantida. Esta outra função é conhecida como *rotear* e, por meio desta, mensagens são passadas de *peer* em *peer* até que seus destinos as recebam. De maneira genérica, o algoritmo de envio de mensagens é o mesmo para todos os tipos de *peers* apresentados neste trabalho. Este algoritmo é apresentado como segue:

```
enviarMsg(peer, mensagem, tipoEnvio)
   inicio
        se não éVizinho(peer.vizinhos, mensagem.destino) entao
           rotear(peer, mensagem);
           sair;
        fimse;
        vizinho := obterVizinho(peer.vizinhos, mensagem.destino);
```

```
    endereco := Nulo;
    se tipoEnvio = SINCRONO então
       endereco := obterEnderecoParaComunicacaoSincrona(vizinho.enderecos);
    senão
       se tipoEnvio = ASSINCRONO então
          endereco := obterEnderecoParaComunicacaoAssincrona(vizinho.enderecos);
       fimse;
    fimse;
    enviar(mensagem, endereco);
fim;
```

Conforme o pseudocódigo acima, a função *rotear* é invocada, quando o destino de uma mensagem não pertence à lista de vizinhos de um *peer* que está executando *enviarMsg*. No entanto, para cada tipo de *peer*, ou seja, *peer* simples e *superpeer*, a função *rotear* e o tratamento de mensagens *ROUTE* são definidos de maneira diferente, conforme segue:

- ***Peer* Simples:**
  - **Roteamento de Mensagens:**

```
rotear(peerSimples, mensagem)
inicio
   vizinhos := obterVizinhos(0, peerSimples.vizinhos);
   rotas := criarLista();
   para cada vizinho em vizinhos faça
      se existeRota(vizinho.rotas, mensage.destino) então
         inicio
            rts := obterRotas(vizinhos.rotas, mensagem.destino);
            rt := rts[0];
            para cada rota em rts faça
               se rota.custo < rt.custo então
                  rt := rota;
               fimse;
            fimpara;
            adicionarItem(rotas, rt);
         fim;
      fimse;
   fimpara;

   se tamanho(rotas) > 0 então
      inicio
         rota := escolhaRotaAleatoriamente(rotas);
         vizinho := obterVizinho(rota.caminho[tamanho(rota.caminho)-1])
         idMsg := criarIdMensagem();
         msgRoute := criarMensagem(idMsg, ROUTE, peerSimples.id, vizinho.id, rota.custo);
         registrarParametro(msgRoute.parametros, 'mensagemPeerToPeer', mensagem);
         removerItem(rota.caminho, rota.caminho[tamanho(rota.caminho)-1])
```

```
            msgRoute.caminho := rota.caminho;
            endereco := obterEnderecoParaComunicacaoAssincrona(vizinho.enderecos);
            enviar(msgRoute, endereco);
         fim;
      fimse;
   fim;
```

- **Tratamento da Mensagem *ROUTE*:**

```
para cada mensagem em peersimples.mensagensRecebidas faça
      ...
   se mensagem.rotulo = ROUTE então
      se mensage.destino = peersimples.id então
         inicio
            mensagemPeerToPeer := obterValorParametro(mensagem.parametros, 'mensagemPeerToPeer');
            encaminharParaTratamento(mensagemPeerToPeer);
         fim;
      fimse;
   fimse;
      ...
fimpara;
```

- ***Superpeers*:**
  - **Roteamento de Mensagens:**

```
rotear(superpeer, mensagem)
inicio
   se mensagem.rotulo == ROUTE então
      inicio
         se mensagem.saltos = 0 então
            sair;
         fimse;
         mensagem.saltos := mensagem.saltos - 1;
         removerItem(mensagem.caminho, superpeer.id);
         se éVizinho(superpeer.vizinhos, mensagem.caminho[tamanho(mensagem.caminho) - 1]) então
            inicio
               vizinho = obterVizinho(superpeer.vizinhos, mensagem.caminho[tamanho(mensagem.caminho) - 1]);
               msgRoute = criarMensagem(mensagem.id, ROUTE, superpeer.id, vizinho.id, message.saltos);
               msgRoute.caminho = mensagem.caminho;
               msgRoute.parametros = mensagem.parametros
               endereco := obterEnderecoParaComunicacaoAssincrona(vizinho.enderecos);
               enviar(msgRoute, endereco);
            fim;
         senão
            inicio
               vizinhos := obterVizinhos(0, peerSimples.vizinhos) + obterVizinhos(1, peerSimples.vizinhos);
               rotas := criarLista();
               para cada vizinho em vizinhos faça
```

```
                se existeRota(vizinho.rotas, mensage.destino) então
                   inicio
                        rts := obterRotas(vizinhos.rotas, mensagem.destino);
                        rt := rts[0];
                        para cada rota em rts faça
                           se rota.custo < rt.custo então
                              rt := rota;
                           fimse;
                        fimpara;
                        adicionarItem(rotas, rt);
                    fim;
                fimse;
            fimpara;

            se tamanho(rotas) > 0 então
               inicio
                  rota := escolhaRotaAleatoriamente(rotas);
                  vizinho := obterVizinho(rota.caminho[tamanho(rota.caminho)-1])
                  idMsg := criarIdMensagem();
                  msgRoute := criarMensagem(idMsg, ROUTE, peerSimples.id, vizinho.id, rota.custo);
                  removerItem(rota.caminho, rota.caminho[tamanho(rota.caminho)-1])
                  msgRoute.caminho := rota.caminho;
                  msgRoute.parametros = mensagem.parametros
                  endereco := obterEnderecoParaComunicacaoAssincrona(vizinho.enderecos);
                  enviarMsg(msgRoute, endereco);
               fim;
            fimse;
         fim;
       fimse;
   fim;
senão
   inicio
       vizinhos := obterVizinhos(0, peerSimples.vizinhos) + obterVizinhos(1, peerSimples.vizinhos);
       rotas := criarLista();
       para cada vizinho em vizinhos faça
          se existeRota(vizinho.rotas, mensage.destino) então
              inicio
                 rts := obterRotas(vizinhos.rotas, mensagem.destino);
                 rt := rts[0];
                 para cada rota em rts faça
                    se rota.custo < rt.custo então
                       rt := rota;
                    fimse;
                 fimpara;
                 adicionarItem(rotas, rt);
              fim;
          fimse;
       fimpara;
```

```
            se tamanho(rotas) > 0 então
                inicio
                  rota := escolhaRotaAleatoriamente(rotas);
                  vizinho := obterVizinho(rota.caminho[tamanho(rota.caminho)-1])
                  idMsg := criarIdMensagem();
                  msgRoute := criarMensagem(idMsg, ROUTE, peerSimples.id, vizinho.id, rota.custo);
                  removerItem(rota.caminho, rota.caminho[tamanho(rota.caminho)-1])
                  msgRoute.caminho := rota.caminho;
                  registrarParametro(msgRoute.parametros, 'mensagemPeerToPeer', mensagem);
                  endereco := obterEnderecoParaComunicacaoAssincrona(vizinho.enderecos);
                  enviarMsg(msgRoute, endereco);
               fim;
            fimse;
         fim;
      fimse;
   fim;
```

- **Tratamento da Mensagem *ROUTE*:**

```
para cada mensagem em superpeer.mensagensRecebidas faça
      ...
   se mensagem.rotulo = ROUTE então
       se mensage.destino = superpeer.id então
          inicio
             mensagemPeerToPeer := obterValorParametro(mensagem.parametros, 'mensagemPeerToPeer');
             encaminharParaTratamento(mensagemPeerToPeer);
          fim;
       senão
          rotear(superpeer, mensagem);
       fimse;
   fimse;
      ...
fimpara;
```

C.4 DESCONEXÃO DE *PEERS*

Para controlar as desconexões ocasionadas pelos usuários, é necessário que haja um algoritmo que trate os detalhes sobre as notificações de saída e a remoção de informações de vizinhança naqueles *peers* que receberam uma notificação de saída.

- **Solicitação de Desconexão:**

```
desconectar(peer)
inicio
```

```
        vizinhos := obterVizinhos(0, peer.vizinhos) + obterVizinhos(1, peer.vizinhos);
        idMsg := criarIdMensagem();
        saltos := 1;
        para cada vizinho em vizinhos faça
                msg := criarMensagem(idMsg, DISCONNECT, peer.id, vizinho.id, saltos);
                registrarIdentificador(msg.caminho, peer.id);
                enviarMsg(peer, msg, SINCRONO);
        fimpara;
        limparTabelaVizinhos(peer.vizinhos);
fim;
```

- **Tratamento da Mensagem *DISCONNECT*:**

```
para cada mensagem em peer.mensagensRecebidas faça
        ...
        se mensagem.rotulo = DISCONNECT entao
        inicio
                se éVizinho(peer.vizinhos, mensagem.origem) então
                        removerVizinho(peer.vizinhos, mensagem.destino);
                fimse;
        fim;
        fimse;
        ...
fimpara;
```

C.5 DISSEMINAÇÃO DE INFORMAÇÃO NA REDE *PEER-TO-PEER*

Neste trabalho, para que os *peers* possam receber informações sobre oportunidades de colaboração, é necessária a utilização de um algoritmo para disseminação de informação. Portanto, o pseudocódigo deste algoritmo é apresentado de acordo com o tipo os dois tipos de *peers* utilizados para a construção da infraestrutura computacional distribuída. Assim, a apresentação dos pseudocódigos será dividida em duas partes, uma parte que é executada pelos *peers* simples e a outra pelos *superpeers.*

- *Peer* simples:
  - **Solicitação para Disseminação de Informação:**

```
disseminarInformacao(peer, mensagem)
inicio
        se tamanho(obterVizinhos(SUPER_PEER, peer.vizinhos)) = 0 então
                sair;
        fimse;
```

```
        redePeerToPeer := peer.redePeerToPeer;
        vizinho  := escolhaAleatoriamenteUmVizinho(obterVizinhos(SUPER_PEER, peer.vizinhos));
        idMsg := criarIdMensagem();
        saltos := redePeerToPeer.maxSaltos;
        msg := criarMensagem(idMsg, PUSH, peer.id, vizinho.id, saltos);
        registrarIdPeer(msg.caminho, peer.id);
        registrarParametro(msg.parametros, 'mensagemPeerToPeer', mensagem);

        enviarMsg(peer, msg, ASSINCRONO);
fim;
```

- **Tratamento da Mensagem *PUSH:***

```
para cada mensagem em peersimples.mensagensRecebidas faça
        ...
        se mensagem.rotulo = PUSH então
                se mensage.destino = peersimples.id então
                        inicio
                                mensagemPeerToPeer := obterValorParametro(mensagem.parametros, 'mensagemPeerToPeer');
                                encaminharParaTratamento(mensagemPeerToPeer);
                        fim;
                fimse;
        fimse;
        ...
fimpara;
```

- *Superpeers*:
  - **Tratamento da Mensagem *PUSH*:**

```
para cada mensagem em superpeer.mensagensRecebidas faça
        ...
        se mensagem.rotulo = PUSH então
        inicio
                mensagem.saltos := mensagem.saltos - 1;

                se mensagem.saltos = 0 então
                        sair;
                fimse;

                vizinhos := obterVizinhos(0, superpeer.vizinhos) + obterVizinhos(1, superpeer.vizinhos);
                para cada vizinho em vizinhos faça
                        se vizinho.id existe em mensagem.caminho então
                                continue;
                        fimse;
                        msgPing         := criarMensagem(mensagem.id, mensagem.rotulo, peer.id, vizinho.id, mensagem.saltos);
```

```
                    msgPush.caminho := mensagem.caminho;
                    registrarIdPeer(msgPing.caminho, superpeer.id);
                    enviarMsg(superpeer, msgPush, ASSINCRONO);
                fimpara;
        fim;
        fimse;
        ...
fimpara;
```

APÊNDICE D – POLÍTICAS DE REDES SOCIAIS *PEER-TO-PEER*

Neste apêndice, serão apresentados os pseudocódigos das políticas de redes sociais *peer-to-peer* propostas no capítulo 5. Seguindo a organização de como foram propostas, as próximas seções detalharão o pseudocódigo de cada uma das políticas.

D.1 FORMAÇÃO DE REDES SOCIAIS

Com base no texto apresentado na seção 5.3.1, esta seção apresenta o pseudocódigo de cada um dos algoritmos utilizados para compor a política de computação distribuída para formação de redes sociais sobre uma infraestrutura computacional *peer-to-peer*. Esta política é executada pelos *peers* dos usuários envolvidos na formação de uma rede social, tendo como base os seguintes passos:

1. **Publicação da Oportunidade de Colaboração:**

```
publicarOportunidade(peer, oportunidade)
inicio
      redePeerToPeer := peer.redePeerToPeer;
      idMsg  := gerarIdMensagem();
      saltos := redePeerToPeer.maxSaltos;
      msg := criarMensagem(idMsg, ADV_OPORTUNITY, peer.id, Nulo, saltos);
      registrarParametro(msg.parametros, 'oportunidade', oportunidade);
      registrarContexto(oportunidade.tipo, usuario.contextos, oportunidade);
      disseminarInformacao(peer, msg);
fim;
```

2. **Cálculo de Similaridade:**

```
calcularSimilaridade(oportunidade, perfilSocial)
inicio
      similaridade := 0;
      para cada restricao em oportunidade.restricaoInteresses faça
            interesse := obterInteresse(perfilSocial.interesses);
            pontos := 0;
```

```
                para cada incidencia em restricao.interesse.incidenciasPalavrasChaves faça
                        aux := obterIncidenciaPalavraChave(interesse.incidenciasPalavrasChaves, inicidendia.palavraChave.valor);
                        se aux.valor >= incidencia.valor então
                                pontos := pontos + 1;
                        fimse;
                fimpara;
                similaridade := (pontos / tamanho(restricao.interesse.incidenciasPalavrasChaves));
                se  similaridade >= restricao.limiteAceitacao então
                        notificarUsuarioSobreSimilaridade(restricao, similaridade)
                fimse;
        fimpara;
fim;
```

3. Aceitação das Oportunidades de Colaboração:

```
aceitarOportunidade(oportunidade, usuario)
inicio
      se existeContextoRedeSocialCriadoPor(usuario.contextos, oportunidade) então
      inicio
              contextoRedeSocialCriadoPor(oportunidade);
              sair;
      fim;
      fimse;

      requererCriacaoRedeSocial(usuario, oportunidade.usuario, oportunidade);
fim;
```

4. Criação da Rede Social *Peer-to-Peer*:

```
criarRedeSocial(usuarioRequerente, usuario, oportunidade)
inicio
      se não existeContextoRedeSocialCriadoPor(usuario.contextos, oportunidade) então
      inicio
              idRedeSocial := criarIdRedeSocial();
              redeSocial := criarRedeSocial(idRedeSocial, oportunidade);
              redeSocial.versao := usuario.peer.relogio;
              registrarMembro(redeSocial.usuarios, usuario);
              registrarMembro(redeSocial.usuarios, usuarioRequerente);
              registrarContexto(redeSocial.tipo, usuario.contextos, redeSocial);
              requererAdicaoRedeSocial(usuario, usuarioRequerente, redeSocial);

      fim;
      senão
      inicio
              op := obterContexto(oportunidade.tipo, oportunidade.id);
              redeSocial := op.redeSocial;
              requererAdicaoRedeSocial(usuario, usuarioRequerente, redeSocial);
              registrarMembro(redeSocial.usuarios, usuarioRequerente);
              para cada membro em redeSocial.usuarios faça
```

```
                        requererAdicaoMembroRedeSocial(usuario, membro, redeSocial, usuarioRequerente);
                fimpara;
        fim
        fimse;
fim;
```

5. Adição de Rede Social no Usuário Requerente da Criação de uma Rede Social *Peer-to-Peer*:

```
adicionarRedeSocial(usuarioRequerente, usuario, redeSocial)
inicio
        se não existeContextoRedeSocialCriadoPor(usuario.contextos, oportunidade) então
                registrarContexto(redeSocial.tipo, usuario.contextos, redeSocial);
        fimse;
fim;
```

6. Atualização da Rede Social *Peer-toPpeer* por Meio da Adição de um Novo Membro:

```
adicionarMembroRedeSocial(usuarioRequerente, usuario, redeSocial, novoMembro)
inicio
        se não existeMembroEmRedeSocial(redeSocial.usuarios, novoMembro.id) então
                registrarMembro(redeSocial.membros, novoMembro);
        fimse;
fim;
```

D.2 COMPARTILHAMENTO DE RESPONSABILIDADES

Com base no texto apresentado na seção 5.3.2, esta seção apresenta o pseudocódigo de cada um dos algoritmos utilizados para compor a política de computação distribuída para o compartilhamento de responsabilidades em uma rede social *peer-to-peer*. Esta política é executada pelos *peers* dos usuários no compartilhamento de responsabilidades, tendo como base os seguintes passos:

1. Envio de Convite para Compartilhamento de Oportunidade:

```
convidarParaCompartilhamentoOportunidade(usuario, oportunidade)
inicio
        redeSocial := oportunidade.redeSocial;
        para cada membro em redeSocial.usuarios faça
                se membro não existe em redeSocial.usuarios então
                        requererAceiteConviteCompartilhamentoOportunidade(usuario, membro, oportunidade);
                fimse;
        fimpara;
fim;
```

2. Aceitação do Convite de Compartilhamento de Oportunidade:

```
aceitarConviteCompartilhamentoOportunidade(usuarioRequerente, usuario, oportunidade)
inicio
      se existeContexto(oportunidade.tipo, usuario.contextos, oportunidade.id) então
            sair;
      fimse;
      registrarMembro(oportunidade.usuarios, usuario);
      registrarContexto(oportunidade.tipo, usuario.contextos, oportunidade);
      para cada membro em oportunidade.usuarios faça
            se membro.id <> usuario.id então
                  requererAdicaoMembroOportunidade(usuario, usuarioRequerente, oportunidade);
            fimse;
      fimpara;
      registrarInformacaoParaDisseminacao(oportunidade);
fim;
```

3. Atualização da Lista de Membros de uma Oportunidade Compartilhada

```
aceitarConviteCompartilhamentoOportunidade(usuarioRequerente, usuario, oportunidade)
inicio
      se existeContexto(oportunidade.tipo, usuario.contextos, oportunidade.id) então
            sair;
      fimse;
      registrarMembro(oportunidade.usuarios, usuario);
      registrarContexto(oportunidade.tipo, usuario.contextos, oportunidade);
      para cada membro em oportunidade.usuarios faça
            se membro.id <> usuario.id então
                  requererAdicaoMembroOportunidade(usuario, usuarioRequerente, oportunidade);
            fimse;
      fimpara;
      registrarInformacaoParaDisseminacao(oportunidade);
fim;
```

D.3 COMPARTILHAMENTO DE HARDWARE

Com base no texto apresentado na seção 5.3.3, esta seção apresenta o pseudocódigo de cada um dos algoritmos utilizados para compor a política de computação distribuída para o compartilhamento de hardware em uma rede social *peer-to-peer*. Esta política é executada pelos *peers* dos usuários envolvidos no compartilhamento de hardware, tendo como base os seguintes passos:

1. Envio do Compartilhamento para Atualização do Conjunto de Compartilhamentos da Rede Social *Peer-to-Peer*:

```
compartilharHardware(usuario, redeSocial, compartilhamento)
inicio
      registrarCompartilhamentoHardaware(redeSocial.compartilhamentosHandware, compartilhamento)
```

```
      para cada membro em redeSocial.usuarios faça
            requererAdicaoCompartilhamentoHardwareRedeSocial(usuario, membro, redeSocial, compartilhamento);
      fimpara;
fim;
```

2. Atualização do Conjunto de Compartilhamentos de Hardware no *Peer* de um Membro de uma Rede Social *Peer-to-Peer:*

```
adicionarCompartilhamentoHardwareRedeSocial(usuarioRequerente, usuario, redeSocial, compartilhamento)
inicio
      se não existeContexto(redeSocial.tipo, usuario.contextos, redeSocial.id) então
            sair;
      fimse;
      redeSoc := obterContexto(redeSocial.tipo, usuario.contextos, redeSocial.id);
      registrarCompartilhamentoHardaware(redeSocial.compartilhamentosHandware, compartilhamento);
fim;
```

D.4 COMPARTILHAMENTO DE CONTEÚDOS

Com base no texto apresentado na seção 5.3.4, esta seção apresenta o pseudocódigo de cada um dos algoritmos utilizados para compor a política de computação distribuída para o compartilhamento de conteúdo em uma rede social *peer-to-peer*. Esta política é executada pelos *peers* dos usuários envolvidos no compartilhamento de conteúdos, tendo como base os seguintes passos:

1. **Solicitação de espaço de armazenamento de conteúdo:**

```
armazenarConteudoRedeSocial(usuario, redeSocial, conteudo)
inicio
      registrarUsuario(conteudo.usuarios, usuario);
      para cada compartilhamentoHardware em redeSocial.compartilhamentoHardware faça
            se compartilhamentoHardware.capacidade >= conteudo.tamanho entao
            inicio
                  requererEspacoArmazenamentoConteudo(usuario, compartilhamentoHardware.peer.usuario, compartilhamento,
conteudo);
            fim
      fimpara;
fim;
```

2. **Alocação de espaço para armazenamento em compartilhamento de disco:**

```
alocarEspacoArmazenamentoConteuto(usuarioRequerente, usuario, compartilhamento, conteudo)
inicio
      se não existeCompartilhamentoUsuario(compartilhamento.tipo, usuario.peer.compartilhamentosHardware, compartilhamento) entao
            sair;
      fimse;
```

```
    cessaoUsoEspaco := criarCessaoUsoEspaco(compartilhamento, conteudo);
    cessaoUsoEspaco.tempoEspera := obterComfiguracaoTempoEsperaCessaoUsoEspaco();
    compart := obterCompartilhamentoHardwareUsuario(compartilhamento.tipo, usuario.peer.compartilhamentosHardware,
compartilhamento.id);
    registrarCessaoUsoEspaco(compart.cessoesUsoEspaco, cessaoUsoEspaco);

    para cada membro em redeSocial.usuarios faça
           requererAdicaoCessaoUsoCompartilhamentoHardware(usuario, membro, compartilhamento, cessaUsoEspaco);
    fimpara;

    iniciarTransferenciaConteudo(usuario.peer, conteudo, cessaoUsoEspaco);
fim;
```

3. Atualização de compartilhamento de disco com espaço cedido para armazenamento de conteúdo:

```
adicionarCessaoUsoCompartilhamentoHardware(usuarioRequerente, usuario, compartilhamento, cessaoUsoEspaco)
inicio
    se não existeContexto(compartilhamento.redeSocial.tipo, usuario.contextos, compartilhamento.redeSocial.id) então
           sair;
    fimse;

    redeSocial := obterContexto(compartilhamento.redeSocial.tipo, usuario.contextos, compartilhamento.redeSocial.id);

    se não existeCompartilhamentoHardwareRedeSocial(compartilhamento.tipo, redeSocial.compartilhamentosHardware,
compartilhamento) entao
           sair;
    fimse;

    compart := obterCompartilhamentoHardwareRedeSocial(compartilhamento.tipo, redeSocial.compartilhamentosHardware,
compartilhamento.id);
    registrarCessaoUsoEspaco(compart.cessoesUsoEspaco, cessaoUsoEspaco);
end;
```

4. Início transferência de conteúdo:

```
iniciarTransferenciaConteudo(peer, conteudo, cessaoUsoEspaco)
inicio
    tmnParte := obterConfiguracaoTmnParte();
    numeroPartes := calcNumPartes(conteudo.tamanho, tmnParte);
    partesPorMembro := calcPartesPrpt(conteudo.tamanho, tmnParte, tamanho(conteudo.usuarios));

    distribuirPartesConteudo(peer, conteudo, tmnParte, numeroPartes, partesPorMembro, cessaoUsoEspaco);
fim;
```

5. Distribuição de partes do conteúdo

```
distribuirPartesConteudo(peer, conteudo, tmnParte, numeroPartes, partesPorMembro, cessaoUsoEspaco)
inicio
    se não existirThreadsTransferencia(peer.threadsTransferencia, conteudo.id)
```

```
    inicio
            idParte := 0;
            para cada usuario em conteudo.usuarios faça
                    partes := criarConjunto();
                    para de i := 1 até partesPorMembro faça
                            parte := conteudo.partesConteudo[idParte];
                            se parte.transferido entao
                                    continue;
                            fimse;
                            adicionarItem(partes, conteudo.partesConteudo[idParte]);
                            idParte := idParte + 1;
                    fimpara;
                    thread := criarThread(peer, conteudo, usuario, partes);
                    registrarThread(peer.threadsTransferencia, thread);
                    iniciar(thread);
            fimpara;
    fim;
    senão
    inicio
            removerThreadsTransferencia(peer.threadsTransferencia, conteudo.id);
            distribuirPartesConteudo(peer, conteudo, tmnParte, numeroPartes, partesPorMembro)
    fim;
    fimse;

fim;
```

6. Transferência partes conteúdo

```
transferirPartesConteudo(thread)
inicio
    i := 1;
    enquando i < tamanho(thread.partesConteudo) faça
            requererParteConteudo(thread.peer.usuario, thread.responsavelConteudo, thread.partesConteudo[i]);
            parteArquivo := esperarPelaChegadaDeParteArquivo(obterConfiguracaoTempoEsperaResposta());
            se parteArquivo = Nulo entao
                    para i := 1 até obterNumeroTentativasReevioRequisicao() faça
                            requererParteConteudo(thread.peer.usuario, thread.responsavelConteudo, thread.partesConteudo[i]);
                            parteArquivo := esperarPelaChegadaDeParteArquivo(obterConfiguracaoTempoEsperaResposta());
                            se parteArquivo <> Nulo então
                                    parar;
                            fimse;
                    fimpara;
                    se parteArquivo = Nulo entao
                    inicio
                            tmnParte := obterConfiguracaoTmnParte();
                            numeroPartes := calcNumPartes(conteudo.tamanho, tmnParte);
                            partesPorMembro := calcPartesPrpt(conteudo.tamanho, tmnParte, tamanho(conteudo.usuarios));
                            distribuirPartesConteudo(thread.peer, thread.conteudo, tmnParte, numeroPartes, partesPorMembro)
                    fim;
```

```
                fimse;
            fimse;
            se parteArquivo <> Nulo então
            inicio
                conteudo.partesConteudo[thread.partesConteudo[i].id].baixado := Verdadeiro;
                removerItem(thread.partesConteudo[i]);
                thread.cessaoUsoEspaco.utilizado := Verdadeiro;
                thread.cessaoUsoEspaco.esperaPorUso := 0;
                redeSocial := thread.cessaoUsoEspaco.compartilhamento.redeSocial
                para cada membro em redeSocial.usuarios faça
                    requererAtualizacaoCessaoUsoCompartilhamento(thread.peer.usuario, membro,

thread.cessaoUsoEspaco.compartilhamento, thread.cessaoUsoEspaco);
                fimpara;
            fim;
            fimse;

    fimenquanto;
fim;
```

7. Atualização da informação de contexto da rede social *peer-to-peer*

```
atualizarCessaoUsoCompartilhamentoHardware(usuarioRequerente, usuario, compartilhamento, cessaoUsoEspaco)
inicio
    se não existeContexto(compartilhamento.redeSocial.tipo, usuario.contextos, compartilhamento.redeSocial.id) então
        sair;
    fimse;

    redeSocial := obterContexto(compartilhamento.redeSocial.tipo, usuario.contextos, compartilhamento.redeSocial.id);

    se não existeCompartilhamentoHardwareRedeSocial(compartilhamento.tipo, redeSocial.compartilhamentosHardware,
compartilhamento) entao
        sair;
    fimse;

    compart := obterCompartilhamentoHardwareRedeSocial(compartilhamento.tipo, redeSocial.compartilhamentosHardware,
compartilhamento.id);
    removerCessaoUsoEspaco(compart.cessoesUsoEspaco, cessaoUsoEspaco.id);
    registrarCessaoUsoEspaco(compart.cessoesUsoEspaco, cessaoUsoEspaco);
end;
```

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA
CCMN - Bloco C - Cidade Universitária - Ilha do Fundão
Rio de Janeiro - RJ CEP: 21941-916
www.ppgi.ufrj.br