Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

p2pBIOFOCO: Um Framework Peer-to-Peer Para Processamento Distribuído do BLAST

Edward de Oliveira Ribeiro

Monografia apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientadora
Profª. Drª. Maria Emília Machado Telles Walter

Brasília
2006

Universidade de Brasília – UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenador: Prof. Dr. Li Weigang

Banca examinadora composta por:

Profª. Drª. Maria Emília Machado Telles Walter (Orientadora) – CIC/UnB
Prof. Dr. Mário A. R. Dantas – CIC/UFSC
Prof. Dr. Marcos M. C. da Costa – UCB/Embrapa

CIP – Catalogação Internacional na Publicação

Edward de Oliveira Ribeiro.

p2pBIOFOCO: Um Framework Peer-to-Peer Para Processamento Distribuído do BLAST/ Edward de Oliveira Ribeiro. Brasília : UnB, 2006. 92 p. : il. ; 29,5 cm.

Tese (Mestre) – Universidade de Brasília, Brasília, 2006.

1. Processamento Distribuído, 2. Bioinformática, 3. Peer-to-Peer, 4. JXTA, 5. BLAST

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro – Asa Norte
CEP 70910–900
Brasília – DF – Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

p2pBIOFOCO: Um Framework Peer-to-Peer Para Processamento Distribuído do BLAST

Edward de Oliveira Ribeiro

Monografia apresentada como requisito parcial
para conclusão do Mestrado em Informática

Profª. Drª. Maria Emília Machado Telles Walter (Orientadora)
CIC/UnB

Prof. Dr. Mário A. R. Dantas
CIC/UFSC

Prof. Dr. Marcos M. C. da Costa
UCB/Embrapa

Prof. Dr. Li Weigang
Coordenador do Mestrado em Informática

Brasília, 27 de Março de 2006

Dedicatória

Dedico este trabalho a memória de meu pai, Antônio Xavier Ribeiro, que de forma direta e indireta tornou tudo possível.

> *"True knowledge exists in knowing that you know nothing. And in knowing that you know nothing, that makes you the smartest of all."*
> **Socrates**

Agradecimentos

Em primeiro lugar, gostaria de agradecer a minha orientadora, profa. Dra. Maria Emília Machado Telles Walter, por sua paciência e confiança em mim. Sua competência, inteligência, foco, pragmatismo, e cordialidade me guiaram e inspiraram durante os dois anos de mestrado. Serei eternamente grato por sua confiança na possibilidade de realização deste trabalho. E outro muito obrigado aos professores Drs. Marcos Mota (UCB) e Mário Dantas (UFSC), que gentilmente aceitaram participar da minha banca de mestrado e cujas observações me permitiram melhorar a qualidade desta dissertação.

Gostaria de agradecer também aos professores Drs. Georgios Pappas (UCB) e Alba Cristina M. Melo (UnB), e aos funcionários, pesquisadores, professores e alunos da UnB, UCB e Embrapa que me deram a oportunidade de trabalhar em seus laboratórios e usar os recursos computacionais necessários para que eu pudesse realizar os experimentos que constam nesta dissertação.

Um agradecimento especial a minha família - Madalena, Edineide, Ediane, e Maria Luíza - por acreditar na importância deste trabalho e na conquista que representou para mim, e um grande abraço aos meus colegas de mestrado José Nélson e Lorena, Alexandre Gomes, Roberto Cantanhede, Marcelo Nardelli, José Geraldo, Bueno, Daniela, Marcelo Sousa, Tânia, Roberta, Caetano e Hederson pelas muitas horas de discussão, desabafo e resolução de problemas em grupo. E meus sinceros agradecimentos aos alunos de graduação Pedro Henrique, Gustavo Zerlotini, Irving Rocha, Victor Bortone e Zé Geraldo. Espero vê-los brilhar em breve na profissão que escolheram e gostaria de registrar que os gratos momentos de ajuda mútua e companheirismo que desfrutei com eles serão lembrados por muitos anos.

Gostaria de ressaltar que este trabalho não teria sido nem ao menos iniciado sem o apoio dos gerentes e colegas de trabalho do Superior Tribunal Militar (STM). Fontoura, Fábio, Ianne, Alexandre, Frederico, Luci, Antonella, Wilson, Helder, Samanta e Dílson, dentre outros colegas do CEINF, não somente foram compreensivos como também torceram por mim durante esses dois últimos anos. Graças a estas e outras demonstrações de carinho, tenho a plena certeza de que dificilmente serei capaz de encontrar ambiente tão amigável e compreensivo como este. Muito obrigado.

Finalmente, gostaria de agradecer a minha esposa Gicineide e meu filho Bruno, que suportaram a minha ausência diversas vezes. Aos dois eu dedico não somente esta dissertação, mas todo o meu carinho.

Resumo

Uma área promissora para o projeto e desenvolvimento de sistemas distribuídos tem sido a Bioinformática, um campo de pesquisa interdisciplinar que usa conhecimentos de Ciência da Computação, Matemática e Estatística para resolver problemas de Biologia Molecular.

Entretanto, apesar do amplo desenvolvimento e uso de tecnologias distribuídas no comércio, indústria e meio acadêmico, os sistemas distribuídos baseados no modelo *Peer-to-Peer* (P2P) ainda permanecem relativamente inexplorados no campo científico. Nesta dissertação, propomos uma nova arquitetura distribuída para a execução de aplicações em Bioinformática, particularmente o BLAST (*Basic Local Alignment Search Tool*), utilizando o modelo P2P. O BLAST é uma família de ferramentas que identifica a similaridade entre seqüências de DNA ou RNA fornecidas pelo usuário e seqüências existentes em bancos de dados de aminoácidos e nucleotídeos. Neste trabalho, projetamos e desenvolvemos um *framework*, baseado na plataforma P2P JXTA, para distribuir o processamento do BLAST entre dois ou mais domínios remotos utilizando um algoritmo de escalonamento de tarefas do tipo "alternância circular"(*round robin*) em uma rede privada virtual. O sistema conta ainda com um mecanismo de presença para anunciar o estado (ativo/inativo) dos *Peers*, e a flexibilidade de adicionar e remover serviços de forma dinâmica, isto é, sem a necessidade de reiniciar a aplicação. Os resultados do processamento do BLAST foram armazenados em um diretório FTP através de uma conexão segura.

O banco de dados utilizado pelo BLAST foi o *nr*, o maior banco de dados de nucleotídeos disponível no National Center for Biotechnology Information (NCBI). Analisamos os ganhos reais de execução de arquivos contendo seqüências de DNA em 10 máquinas, distribuídas entre três domínios remotos, de forma a verificar a aplicabilidade da abordagem P2P em um ambiente de testes real, e o impacto que as limitações de memória RAM de cada máquina exerce sobre o tempo de execução total do sistema. Os bons resultados obtidos motivam novas melhorias no modelo atual, como inclusão de novos algoritmos de escalonamento de tarefas ou mecanismos de tolerância a falhas, além do uso desta arquitetura em projetos reais de Bioinformática.

Palavras-chave: Processamento Distribuído, Bioinformática, Peer-to-Peer, JXTA, BLAST

Abstract

A rewarding area for the project and design of distributed systems has been Bioinformatics, an interdisciplinary research field that uses knowledge from Computer Science, Mathematics and Statistics to solve problems in Molecular Biology.

Nevertheless, in spite of the development and use of distributed technologies in business, industry and academia, distributed systems based on the Peer-to-Peer (P2P) model are still relatively unexplored in the scientific field. In this dissertation, we propose a new distributed architecture to the execution of Bioinformatics applications, particularly the BLAST (Basic Local Alignment Search Tool), using a P2P computing model. The BLAST is a suite of tools that verify the similarity between DNA or RNA sequences issued by the user and the sequences stored in nucleotides and aminoacids databases. In this work, we designed and developed a framework, based on JXTA P2P platform, to distribute BLAST processing among two or more remote sites according to a round robin task-scheduling algorithm in a virtual private network. The system has also a presence mechanism to advertise the status of the Peers (online/offline), and the flexibility to dynamically add or remove services, that is, without restarting the application. The results of the BLAST processing were stored in a FTP directory through a secure connection.

The database used by BLAST was nr, the largest nucleotide database available at the National Center for Biotechnology Information (NCBI). We analyzed the real gains of the execution of DNA sequence files in 10 machines, distributed among three remote sites, to verify the applicability of the P2P approach in a real testbed environment, and the impact that RAM memory limitations of each machine has over the total execution time of the system. The good results obtained motivate us new improvements in the current model, like the inclusion of new task scheduling algorithms or fault tolerance mechanisms, and the use of this architecture in real Bioinformatics projects.

Keywords: Distributed Systems, Bioinformatics, Peer-to-Peer, JXTA, BLAST

Sumário

Lista de Figuras

Capítulo 1

Introdução

A descoberta da estrutura em dupla hélice da molécula de DNA por Watson e Crick [3], em 1953, configurou-se como um marco importante para a Biologia Moderna, que desde então teve um grande desenvolvimento, culminando com um período da Biologia Molecular denominado *Era Genômica da Biologia*. Graças aos avanços provenientes da união entre a Biologia Molecular e as ciências exatas, notadamente a Ciência da Computação, as *proteínas* e *ácidos nucléicos*, estruturas que tornam possível a vida, passaram a ser melhor estudados e seus interrelacionamentos puderam ser melhor compreendidos.

Outro marco igualmente importante ocorreu com o *Projeto Genoma Humano* [4], coordenado pelo Departamento de Energia dos EUA e finalizado em 2003. Este projeto durou 13 anos e envolveu laboratórios de várias partes do mundo com o objetivo principal de sequenciar totalmente o DNA humano. Devido a magnitude deste projeto, genomas de outros organismos mais simples tais como bactérias, moscas e ratos também foram seqüenciados com o objetivo secundário de aperfeiçoar as técnicas e ferramentas de seqüenciamento a serem empregadas no seqüenciamento do genoma humano. Além dos objetivos citados, este projeto buscou:

- armazenar as informações do projeto em bancos de dados;
- aperfeiçoar as ferramentas de análise de dados biológicos;
- transferir as tecnologias do projeto para o setor privado;
- discutir questões éticas, sociais e legais originárias do projeto;

A partir do *Projeto Genoma Humano* seguiram-se outros esforços de seqüenciamento de genoma em várias partes do mundo, inclusive no Brasil. Dentre os laboratórios e institutos que se tornaram referência neste tipo de projeto destacam-se o Sanger Institute [5], o National Center for Biotechnology Information (NCBI) [6], o European Molecular Biology Laboratory (EMBL) [7], e o DNA Data Bank of Japan (DDBJ) [8]. Os dados provenientes dos projetos de seqüenciamento de genoma ao redor do mundo alimentam continuamente os bancos de dados de DNA e proteínas das instituições citadas, e em inúmeras outras instituições públicas e privadas ao redor do mundo.

Os bancos de dados de seqüências biológicas e as ferramentas associadas para pesquisá-los tornaram-se vitais na Biologia Molecular moderna. Tais bancos funcionam como repositórios de seqüências, e permitem a recuperação de qualquer seqüência de DNA, RNA ou proteína por um título ou número de acesso. A Figura 1, por exemplo, mostra uma entrada típica existente no GenBank, um banco de dados público de nucleotídeos e anotações biológicas, construído e distribuído pelo NCBI. Além disso, esses arquivos são acessíveis via Internet, e existe uma grande coleção de softwares para busca, visualização e manipulação de seqüências. Entretanto, um aspecto mais importante que esta função de armazenagem e transferência de dados é o fato destes bancos de dados agirem como "geradores de novos conhecimentos", pois permitem recuperações de seqüências baseadas em similaridades entre duas ou mais seqüências. Graças a isto, seqüências novas podem ser comparadas com seqüências catalogadas e de funcionalidade conhecida, o que permite inferir a função de genes e proteínas recém descobertos.

```
LOCUS       MMU22421                2235 bp    DNA     linear   ROD 23-MAR-1995
DEFINITION  Mus musculus obesity protein (ob) gene, complete cds.
ACCESSION   U22421
VERSION     U22421.1  GI:726296
KEYWORDS    .
SOURCE      Mus musculus (house mouse)
  ORGANISM  Mus musculus
            Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
            Mammalia; Eutheria; Euarchontoglires; Glires; Rodentia;
            Sciurognathi; Muridae; Murinae; Mus.
REFERENCE   1  (bases 1 to 2235)
  AUTHORS   Chehab,F.F. and Lim,M.E.
  TITLE     Genomic organization and sequence of the mouse obesity gene
  JOURNAL   Unpublished (1995)
REFERENCE   2  (bases 1 to 2235)
  AUTHORS   Chehab,F.F. and Lim,M.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (09-MAR-1995) Farid F. Chehab, Laboratory Medicine,
            University of California, San Francisco, 505 Parnassus Avenue, San
            Francisco, CA 94143-0134, USA
FEATURES             Location/Qualifiers
     source          1..2235
                     /organism="Mus musculus"
                     /mol_type="genomic DNA"
                     /strain="C57BL/6J"
                     /db_xref="taxon:10090"
                     /chromosome="6"
     gene            join(1..144,1876..2235)
                     /gene="ob"
     CDS             join(1..144,1876..2235)
ORIGIN
        1 atgtgctgga gacccctgtg tcggttcctg tggctttggt cctatctgtc ttatgttcaa
       61 gcagtgccta tccagaaagt ccaggatgac accaaaaccc tcatcaagac cattgtcacc
                                         ....
     2221 agccctgaat gctga
//
```

Figura 1.1: Uma entrada típica no banco de dados GenBank

O uso de bancos de dados de seqüências cresce rapidamente assim como o tamanho dos próprios bancos de dados. A Figura 1.2 mostra o crescimento do banco de dados GenBank entre os anos de 1982 a 2005. Um grande esforço foi

empregado no sentido de integrar vários bancos de dados para permitir o compartilhamento de informação e processamento automático de consultas complexas que necessitam de interação entre várias bases de dados.

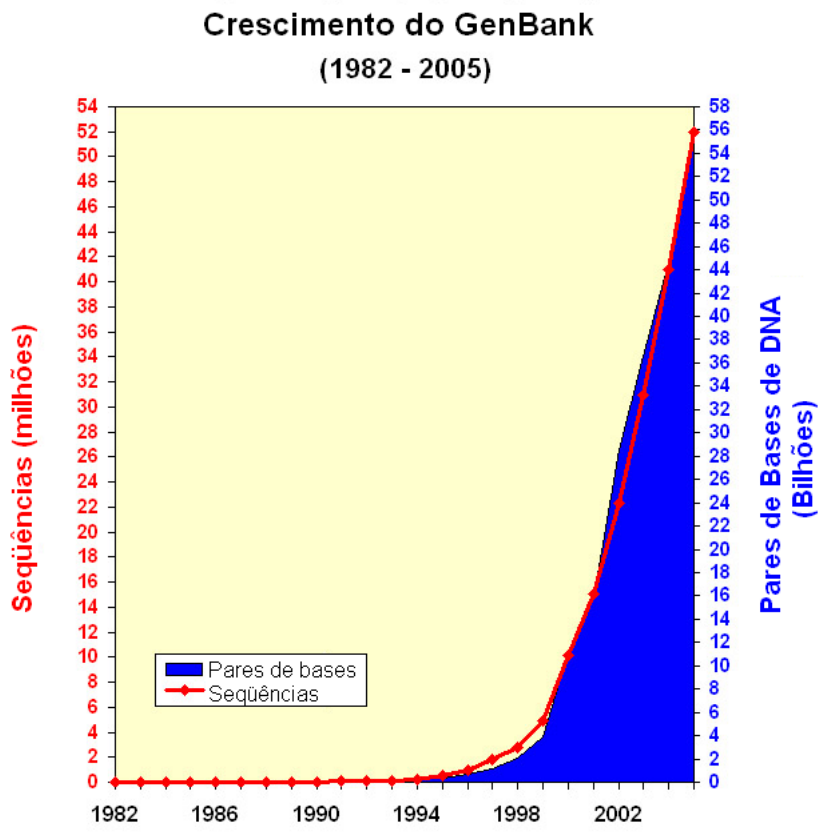

Figura 1.2: Crescimento do Banco de Dados GenBank

No Brasil, o seqüenciamento de genoma teve seu primeiro grande impulso com o projeto de seqüenciamento do genoma do fitopatógeno *Xylella fastidiosa* [9], responsável pela "praga do amarelinho"que ataca plantações de laranja, principalmente no estado de São Paulo, ocasionando grandes prejuízos financeiros. Este projeto foi patrocinado pela Fundação de Amparo a Pesquisa do Estado de São Paulo (FAPESP) e contou com pesquisadores de várias instituições de ensino e pesquisa brasileiros. O sucesso deste projeto permitiu que outros projetos de seqüenciamento pudessem ser iniciados. Este projeto teve excelente projeção internacional com a publicação dos resultados na renomada revista Nature [9], e permitiu que projetos similares pudessem ser desenvolvidos no país. A importância deste tipo de iniciativa é evidente, pois graças a tais projetos o país tem a chance de gerar conhecimento em uma área estratégica, criar independência científica em relação aos países desenvolvidos, e colocar as ciências biológicas a serviço da solução de problemas próprios do Brasil.

Entre os projetos que se seguiram ao da *Xylella fastidiosa* temos o mapeamento do genoma da cana-de-açúcar, do câncer humano (em colaboração com o Instituto Ludwig para Pesquisa do Câncer), do café, dentre outros. Em âmbito

nacional, tivemos o *Projeto Genoma Brasileiro* [10], criado pelo CNPQ em dezembro de 1999, e que conta com a participação de 25 grupos de pesquisa nacionais. O primeiro organismo escolhido para ser seqüenciado por este consórcio foi o *Chromobacterium violaceum* (uma bactéria capaz de sintetizar compostos antibióticos e anti-tumorais). Este projeto de seqüenciamento terminou em 2001, e o consórcio atualmente trabalha no seqüenciamento do genoma do *Mycoplasma synoviae*, um agente causador de doenças endêmicas em aves. Tivemos também o projeto *Genolyptus*, responsável pelo seqüenciamento do eucalipto (Fundo Verde-Amarelo/MCT).

Em âmbito regional temos a criação de redes regionais de seqüenciamento de genoma, dentre as quais:

- Rede Genoma do Estado de Minas Gerais (*Schistosoma mansoni*);
- Rede Genoma Nordeste (*Leishmania chagasi*);
- Rede Genômica do Estado da Bahia e São Paulo (*Crinipellis perniciosa*);
- Rede Genoma do Consórcio do Instituto de Biologia Molecular do Paraná, FIOCRUZ e Universidade de Mogi das Cruzes (*Trypanossoma cruzi*);
- Programa Genoma do Estado do Paraná (*Herbaspirillum seropedicae*);
- Rede Genoma do Rio de Janeiro (*Gluconacetobacter diazotrophicus*);
- Rede Sul de Análise de Genomas e Biologia Estrutural (*Mycoplasma hyopneumoniae*);
- Rede Genoma Centro-Oeste (*Paracoccidioides brasiliensis*)

Em 2001, três instituições do Centro-Oeste - Universidade de Brasília (UnB), Universidade Católica de Brasília (UCB) e Embrapa Recursos Genéticos e Biotecnologia - uniram esforços na criação de uma rede de Bioinformática denominada BIOFOCO (Rede de Pesquisa em Bioinformática do Centro-Oeste) [11]. O objetivo da rede BIOFOCO é criar uma rede de pesquisa e desenvolvimento para apoiar projetos em Biologia Molecular através do desenvolvimento de ferramentas e sistemas, capacitação de pesquisadores, e integração entre os pesquisadores em Bioinformática da região Centro-Oeste. Este projeto encontra-se atualmente na fase 2, a partir do qual tenta-se prover ferramentas distribuídas para pesquisa e análise de dados genômicos e proteômicos. Esta dissertação de mestrado insere-se no escopo da fase 2 desta rede de pesquisa.

Um projeto de seqüenciamento divide-se basicamente em três fases, a saber:

1. Submissão - as seqüências de DNA/proteína obtidas através de seqüenciadores automáticos são enviadas para processamento. São executados os programas *Phred*, *phd2fasta*, e *Crossmatch* nos arquivos de seqüência;
2. Montagem - As seqüências são processadas pelo programa CAP3, que agrupada algumas seqüências em *contigs* enquanto outras são mantidas em *siglets*;

3. Anotação - são executados os programas BLAST, FASTA e Interpro para extrair informações significativas das seqüências montadas;

A Bioinformática, que foi responsável pelo sucesso dos projetos citados anteriormente, é uma área de pesquisa inter-disciplinar que usa conhecimentos de Ciência da computação, Matemática e Estatística para resolver problemas de Biologia Molecular. O objetivo é desenvolver e implantar ferramentas que auxiliem os biólogos em projetos de seqüenciamento de genomas e proteomas. Conforme aponta Roos [12], o grande desafio da Bioinformática tem sido gerar, armazenar, analisar a grande quantidade de dados biológicos que suas ferramentas tem permitido obter.

1.1 Motivação

Conforme aponta Roos [12], um aspecto fundamental da pesquisa em Bioinformática concentra-se no desenvolvimento de bancos de dados: como integrar e pesquisar eficientemente dados de (por exemplo) seqüências genômicas de DNA e estrutura de proteínas. O segundo foco envolve algoritmos de reconhecimento de padrões para áreas como montagem de seqüências de proteínas e ácidos nucléicos, alinhamento de seqüências em comparações de similaridade e reconstrução filogenética. Estes dois focos dependem enormemente do acesso a dados provenientes de fontes diversas, e na capacidade de integrar, transformar e reproduzir tais dados em novos formatos.

Graças aos avanços na tecnologia de hardware e ao uso amplamente difundido das redes de computadores, a computação distribuída tem tido um papel crescente na solução de problemas complexos e na execução de aplicações que requerem grande quantidade de recursos computacionais (ciclos de CPU, memória RAM, armazenamento, etc.). Tais tecnologias impulsionam o desenvolvimento de sistemas capazes de processar e controlar um volume crescente de dados remotos. Desta forma, a convergência destas duas tendências, a computação distribuída e a Bioinformática, é um passo natural, sendo a World Wide Web [13] uma ferramenta indispensável a pesquisadores em biologia de todo o mundo nos dias atuais.

1.2 Objetivos

Apesar do amplo desenvolvimento e uso de tecnologias distribuídas no comércio, indústria e meio acadêmico, sistemas distribuídos baseados no modelo *Peer-to-Peer* (P2P) ainda permanecem relativamente inexplorados no campo científico. Os objetivos desta dissertação são:

- propor uma arquitetura distribuída para a execução de aplicações em Bioinformática utilizando o modelo *Peer-to-Peer* (P2P);
- desenvolver um *framework* baseado no modelo proposto, e que faz uso da plataforma JXTA para implementar as função P2P de mais baixo nível;

- usar o BLAST como a primeira ferramenta a ser implantada e testada no *framework*;
- interligar três instituições, Universidade de Brasília (UnB), Universidade Católica de Brasília (UCB), e Embrapa Recursos Genéticos e Biotecnologia em uma rede P2P privada para submissão de seqüências biológicas;
- realizar testes em um ambiente de testes real, utilizando arquivos provenientes de um projeto de seqüenciamento de genoma;
- analisar a viabilidade da arquitetura e o ganho de performance advindo de seu uso;

Capítulo 2

BLAST

2.1 Biologia Molecular

Conforme visto em Setubal e Meidanis [14], os principais responsáveis pelos processos bioquímicos que viabilizam a vida são dois tipos de moléculas, **proteínas** e **ácidos nucléicos**. Basicamente, as proteínas estão envolvidas na constituição dos seres vivos (aceleração de reações químicas ou constituição tecidos, por exemplo) enquanto os ácidos nucléicos são responsáveis por codificar a informação necessária para a produção dos mais variados tipos de proteínas.

Símbolo	Abreviação	Aminoácido
A	Ala	Alanina
C	Cys	Cistina
D	Asp	Ácido Aspártico
E	Glu	Glutamato
F	Phe	Fenilalanina
G	Gly	Glicina
H	His	Histidina
I	Ile	Isoleucina
K	Lys	Lysina
L	Leu	Leucina
M	Met	Metionina
N	Asn	Asparagina
P	Pro	Prolina
Q	Gln	Glutamina
R	Arg	Arginina
S	Ser	Serina
T	Thr	Treonina
V	Val	Valina
W	Trp	Triptofano
Y	Tyr	Tirosina

Tabela 2.1: Aminoácidos usualmente encontrados em proteínas

Uma proteína é uma cadeia de moléculas simples denominadas **aminoácidos**. Na natureza encontramos cerca de 20 aminoácidos diferentes (Tabela 2.1). As proteínas podem ter diferentes *conformações*, isto é, complexas orientações tridimensionais que estão intimamente relacionadas a sua função. De fato, as proteínas possuem quatro estruturas: a Estrutura Primária é a seqüência de aminoácidos que compõem a proteína, a Estrutura Secundária mostra a orientação tridimensional de partes da proteína, a Estrutura Terciária descreve a orientação tridimensional da proteína como um todo e, finalmente, a Estrutura Quaternária especifica

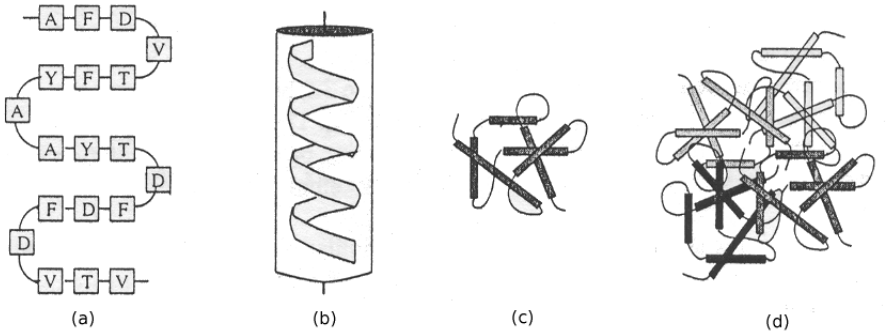

Figura 2.1: Estruturas Primária (a), Secundária (b), Terciária (c) e Quaternária (d) de proteínas

a orientação tridimensional de um grupo de proteínas (Figura 2.1).

Determinar a estrutura tridimensional de proteínas, e por conseguinte sua função, tem sido alvo de intensa pesquisa nos últimos anos, inclusive com o emprego de técnicas provenientes da geometria computacional e da robótica [15]. Entretanto, uma forma automática e eficiente de se realizar esta tarefa ainda permanece um problema de pesquisa em aberto.

Os ácidos nucléicos, por sua vez, podem ser de dois tipos: ácido ribonucléico (RNA) e ácido dexóciribonucléico (DNA). A molécula de DNA é uma cadeia dupla de moléculas mais simples, batizadas de *strand*. A estrutura básica, denominada **nucleotídeo**, é composta de repetições da mesma unidade básica: uma molécula de açucar chamada 2' deoxyribose ligada a um resíduo de fósfato e uma base. Existem 4 tipos de bases: Adenina (A), Citosina (C), Guanina (G) e Timina (T). Apesar de serem diferentes, os termos base e nucleotídeo são usadas de forma intercambiável e, desta forma, referimo-nos a uma molécula de DNA como tendo 200 bases, ou 200 nucleotídeos.

Cada célula de um organismo possui um pequeno conjunto de moléculas de DNA bem longas denominadas cromossomos. As moléculas de DNA podem conter milhões de nucleotídeos. As cadeias são duplas, pois cada base em uma cadeia liga-se a outra base em oura cadeia. A Adenina liga-se com a Timina, e a Citosina liga-se com a Guanina. Estes pares de bases são conhecidos como *pares de bases de Watson e Crick*, abreviados por **bp** (base pair). Além disso, cada proteína é codificada em uma única porção do DNA. Esta porção é chamada **gene**. O processo de obter a seqüência de pares de bases no DNA é denominada seqüenciamento.

2.2 A ferramenta BLAST

Dentre as ferramentas de Bioinformática atualmente disponíveis, a família de aplicações BLAST [16] - *Basic Local Alignment Search Tool* - tem uma posição de destaque na comunidade de Bioinformática. BLAST realiza pesquisas que retornam a similaridade entre uma seqüência de consulta e um banco de dados contendo seqüências catalogadas e com funcionalidade conhecida.

O BLAST tem como saída um conjunto de alinhamentos entre a seqüência de consulta e as seqüências presentes no banco de dados. Cada alinhamento retor-

nado por uma busca no BLAST recebe um *e-score* e uma medida de significância estatística, chamado *expectation value* (*e-value*), que permite julgar sua qualidade. É possível especificar um valor, ou intervalo de valores, para o *e-value* para limitar os alinhamentos retornados. As similaridades encontradas entre a nova seqüência, usada como consulta, e as seqüências presentes no banco dados podem ajudar a descobrir a funcionalidade da seqüência recém descoberta.

Para as pesquisas no BLAST tanto a seqüência de consulta quanto a base de dados podem ser compostas por aminoácidos ou nucleotídeos. Uma vez que os aminoácidos podem ser obtidos através da tradução de códons (triplas) de nucleotídeos em um aminoácido então é possível realizar as comparações entre seqüências de nucleotídeos e seqüências de aminoácidos, pois o BLAST permite a tradução em tempo de execução. A Tabela 2.2 lista as possíveis combinações entre tipos de seqüências de consulta e tipos de seqüências presentes no banco de dados, assim como as respectivas ferramentas BLAST usadas para estas comparações.

Tabela 2.2: Tipos de consultas BLAST

Ferramenta	**Tipo de Consulta**	**Tipo de Banco de Dados**	**Tradução**
blastn	Nucleotídeo	Nucleotídeo	Nenhuma
tblastn	Aminoácido	Nucleotídeo	Banco de Dados
blastx	Nucleotídeo	Aminoácido	Consulta
blastp	Aminoácido	Aminoácido	Nenhuma
tblastx	Nucleotídeo	Nucleotídeo	Consulta e Banco de Dados

Em geral, os centros de pesquisa disponibilizam interfaces baseadas na Web para a submissão de seqüências de consulta, configuração dos critérios de busca e execução remota da ferramenta como é o caso do NCI BLAST (Figura 2.2), por exemplo. Embora a Web proporcione um ambiente amigável, flexível e portável para a execução de sistemas remotos, a demanda por uma ferramenta popular como o BLAST faz com que o tráfego em sites como o do NCBI [6] torne seu uso inviável em horários de pico. Este gargalo se deve ao grande número de consultas simultâneas, vindas de inúmeros laboratórios ao redor do mundo, e ao crescimento exponencial das próprias bases de dados genéticas, que passa a exigir mais tempo de processamento da ferramenta.

O GenBank [17] possui mais de 56 bilhões de bases nucléicas extraídas de 52 milhões de seqüências individuais. Genomas completos representam uma porção crescente deste banco, pois cerca de 50 dos mais de 180 genomas microbiais completos existentes foram adicionados no ano de 2004. Além disso, o GenBank possui mais de 165.000 espécies catalogadas e novas espécies são adicionadas a taxa de mais de 2.000 por mês. Segundo Benson e co-autores [17], as buscas por similaridades de seqüências utilizando o BLAST representam o tipo mais freqüente e básico de análise realizado nos dados do GenBank.

Portanto, é fácil verificar que além de ser prontamente acessível e intuitivo, torna-se cada vez mais importante tornar o BLAST e outras ferramentas de Bioinformática escaláveis para um uso cada vez mais intenso e geograficamente distribuído. Esta necessidade tem motivado diversas propostas de sistemas paralelos e distribuídos para a execução do BLAST.

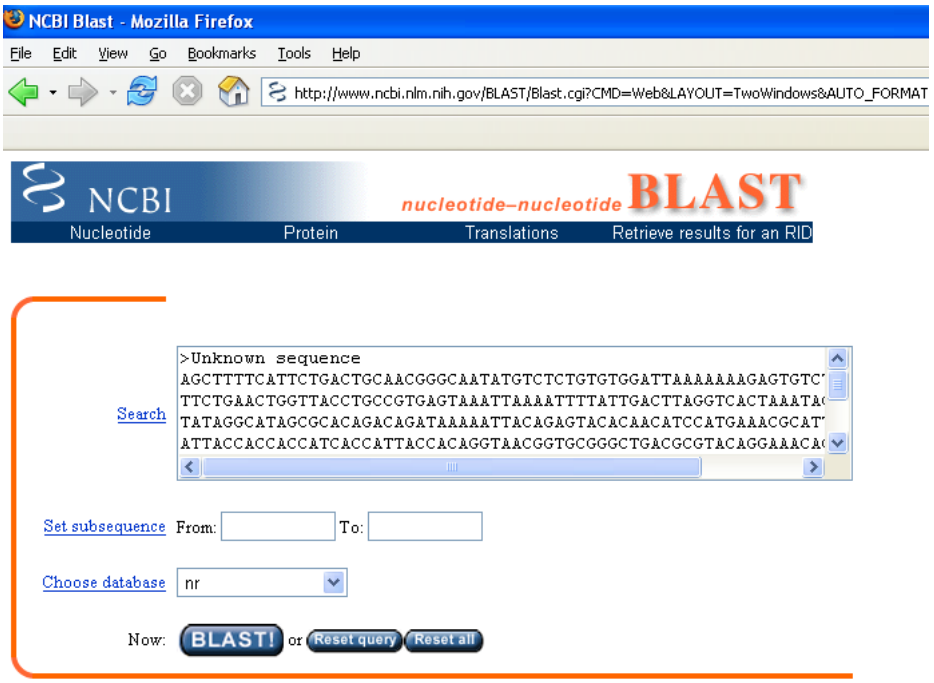

Figura 2.2: Exemplo de interface WEB para BLAST

2.3 Formato FASTA

A pesquisa e manipulação de seqüências de aminoácidos ou proteínas exige o uso de arquivos de seqüência em formatos pré-definidos. Dentre os vários formatos de arquivo existentes, o FASTA é o mais amplamente reconhecido pelas ferramentas de Bioinformática, inclusive o BLAST.

O formato FASTA começa com uma linha que contém informações sobre os dados presentes, seguido pelas linhas contendo os dados da seqüêncian propriamente ditos. A linha de descrição é diferenciada dos dados da seqüência por um sinal de maior-que (">"), que está presente na primeira coluna. A Figura 2.3 mostra um trecho de arquivo em formato FASTA.

```
>gi|532319|pir|TVFV2E|TVFV2E envelope protein
ELRLRYCAPAGFALLKCNDADYDGFKTNCSNVSVVHCTNLMNTTVTTGLLLNGSYSENRT
QIWQKHRTSNDSALILLNKHYNLTVTCKRPGNKTVLPVTIMAGLVFHSQKYNLRLRQAWC
HFPSNWKGAWKEVKEEIVNLPKERYRGTNDPKRIFFQRQWGDPETANLWFNCHGEFFYCK
MDWFLNYLNNLTVDADHNECKNTSGTKSGNKRAPGPCVQRTYVACHIRSVIIWLETISKK
TYAPPREGHLECTSTVTGMTVELNYIPKNRTNVTLSPQIESIWAAELDRYKLVEITPIGF
APTEVRRYTGGHERQKRVPFVXXXXXXXXXXXXXXXXXXXXXVQSQHLLAGILQQQKNL
LAAVEAQQQMLKLTIWGVK
```

Figura 2.3: Trecho de arquivo em formato FASTA

2.4 BLAST Paralelo e Distribuído

O fato do BLAST ser computacionalmente intenso e intrinsicamente paralelizável fez com que surgissem várias abordagens para distribuir sua execução entre nós computacionais [1, 18]. É possível perceber que, em geral, estas soluções fazem uso de *clusters* computacionais.

Um cluster pode ser definido como um sistema distribuído onde um conjunto de máquinas, em uma mesma rede local (LAN), trabalha de maneira conjunta para solucionar algum problema que geralmente envolve grande tempo de processamento. O processamento é então dividido entre as máquinas, que trabalham como se fossem um sistema único. Nas Seções 2.4.1, 2.4.2 e 2.4.3, descrevemos três exemplos típicos de aplicações distribuídas do BLAST. Estes exemplos não são exaustivos, visto que vários outros projetos tem utilizado versões paralelas ou distribuídas do BLAST.

2.4.1 BeoBLAST

BeoBLAST [1] é um sistema distribuído BLAST criado pelo grupo de Bioinformática do Fox Chase Cancer Center [19]. A arquitetura do sistema BeoBLAST é mostrada na Figura 2.4 [1]. Este sistema, escrito em Perl e executado sobre máquinas Linux, distribui as tarefas BLAST entre os nós de um cluster Beowulf [20]. A principal vantagem deste sistema, além de balancear a carga entre as máquinas de um cluster, é permitir um conjunto maior de opções de pesquisa no BLAST. O sistema possui uma interface Web que permite ao usuário especificar buscas simultâneas, isto é, o usuário pode selecionar vários bancos de dados, especificar múltiplas consultas, e submeter tais parâmetros pela Web. Um *script* Perl então distribui a carga de forma balanceada entre os nós de um cluster mediante o uso da ferramenta *GNU Queue*. O fato de ser escrito como um conjunto de scripts na linguagem de programação *Perl*, e fazer uso de sistemas de código fonte aberto (Linux, *GNU Queue*), permite a portabilidade do sistema para outras arquiteturas.

Os bancos de dados de nucleotídeos e aminoácidos foram distribuídos no cluster usando o sistema de arquivos virtual paralelo (PVFS) implementado no *kernel* para minimizar o uso de disco. Entretanto, segundo os autores, BeoBLAST pode ser implementado com bancos de dados locais em cada nó ou ainda em bancos de dados montados em sistema de arquivos NFS. Por fim, BeoBLAST usa o código de retorno do BLAST para determinar se uma tarefa executou com sucesso ou re-enviá-la para execução, caso contrário.

A Tabela 2.3 resume os experimentos presentes em Grant e co-autores [1]. As medições foram realizadas com seqüências idênticas de 594 aminoácidos executando em um sistema BeoBLAST com 8 nós. Apesar da quantidade limitada de experimentos realizados, podemos perceber um ganho de tempo a medida que o trabalho é dividido entre as máquinas do cluster.

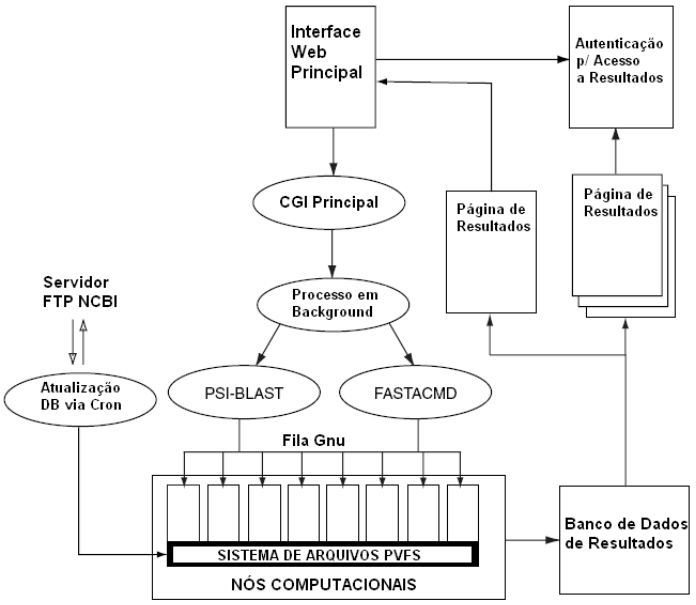

Figura 2.4: Arquitetura do BeoBLAST [1]

Tabela 2.3: BeoBLAST - Tempos de execução

Número de seqüências	Tempo (minutos)
10	2
40	12

2.4.2 mpiBLAST

O mpiBLAST é uma solução de código fonte aberto que segmenta e distribui um banco de dados BLAST entre os nós de um cluster de tal forma que cada nó realize buscas em uma porção única do banco de dados.

Segundo Darling e co-autores [18], a segmentação do banco de dados no BLAST oferece duas vantagens principais em relação aos algoritmos BLAST paralelos. Primeiramente, a segmentação do banco de dados elimina a alta sobrecarga que advém de repetidas chamadas a E/S, pois o tamanho dos bancos de dados biológicos são bem maiores que a memória principal dos computadores, forçando a paginação em disco. Além disso, a segmentação da base de dados permite que cada nó realize as buscas em uma porção menor do banco, reduzindo ou até mesmo eliminando E/S excessiva, e fazendo com que as buscas em um nó exibam uma aceleração (*speedup*) super-linear. Em segundo lugar, a segmentação do banco de dados no BLAST não produz intercomunicação pesada entre os nós, o que permite que os mesmos continuem alcançando aceleração (*speedup*) super-linear mesmo com centenas de nós fazendo parte do cluster.

O mpiBLAST usa a biblioteca de envio de mensagens *Message Passing Interface* (MPI) [21] para implementar a segmentação do banco, e permite que ele trabalhe com diversas arquiteturas de sistemas. O mpiBLAST foi projetado para ser executado em *clusters* com software de escalonamento de tarefas como PBS

(*Portable Batch System*). Nestes ambientes, ele se adapta a mudanças de recursos através da redistribuição dinâmica de fragmentos do banco de dados.

O algoritmo do mpiBLAST consiste de dois passos básicos. Em primeiro lugar, o banco de dados é segmentado em fragmentos pequenos de tamanho aproximadamente igual através do uso de comandos específicos do aplicativo **formatdb**, que é distribuído junto com outras ferramentas do BLAST, e os segmentos são armazenados em um dispositivo de armazenamento compartilhado (NFS, por exemplo).

Em segundo lugar, as consultas mpiBLAST são acionadas em cada nó. Caso um nó ainda não possua o fragmento que deverá utilizar, ele copia este fragmento do dispositivo compartilhado. A determinação de quais nós serão responsáveis por quais fragmentos é determinado por um algoritmo heurístico que minimiza o número de cópias de fragmentos durante cada busca.

Ao iniciar, cada *processo trabalhador* informa ao *processo mestre* quais fragmentos de bancos de dados ele possui atualmente em seu armazenamento local. Logo em seguida, o *processo mestre* lê as seqüências de entrada e as envia para todos processos no grupo de comunicação. Assim que o envio das seqüências de entrada termina, cada *processo trabalhador* reporta ao *processo mestre* que ele se encontra desocupado. Ao receber esta mensagem, o *mestre* associa a cada *trabalhador* desocupado um fragmento de banco de dados para que ele realize uma busca ou copie tal fragmento para seu armazenamento local. O *trabalhador* copia ou realiza a busca no fragmento informado e reporta ao mestre que ele encontra-se desocupado quando termina a operação. O processo se repete até que todos os fragmentos de banco de dados tenham sido pesquisados.

O *processo mestre* utiliza um algoritmo *guloso* (*greedy*) para associar fragmentos a *processos trabalhadores*. Primeiramente, se um *trabalhador* desocupado tem algum fragmento não pesquisado e que nenhum outro *trabalhador* possui então o *trabalhador* fica encarregado de realizar buscas neste fragmento único. Se o *trabalhador* não possui fragmento único, o *trabalhador* fica encarregado de fragmentos não pesquisados e existentes no menor número de outros trabalhadores. Finalmente, se um *trabalhador* desocupado possui somente fragmentos pesquisados, ele é instruído a copiar fragmentos não pesquisados existentes no menor número de trabalhadores.

Assim que cada *trabalhador* termina sua busca por fragmentos, ele reporta os resultados para o mestre. O *mestre* unifica os resultados de cada *trabalhador* e os ordena de acordo com seu escore. Uma vez que todos os resultados tenham sido recebidos, eles são gravados em um arquivo de saída especificado pelo usuário. Os resultados unificados podem ser gerados em vários formatos: XML, HTML, ASCII e ASN.1.

2.4.3 GridBLAST

Conforme aponta Foster [22], o termo *Grade* denota uma infraestrutura distribuida para engenharia e ciências avançadas através do compartilhamento coordenado de recursos e solução de problemas em um organizações virtuais dinâmicas e multi-institucionais. Este compartilhamento envolve não somente a troca de arquivos, mas também acesso direto a computadores, software, dados, instrumentos

científicos, dentre outros recursos. Este compartilhamento é altamente controlado, com provedores e consumidores de recursos definindo claramente e cuidadosamente o que será compartilhado, a quem é permitido compartilhar, e sob quais condições este compartilhamento irá ocorrer. Um conjunto de indivíduos e/ou organizações definidos por tais regras de compartilhamento é denominado *Organização Virtual*.

GridBLAST [23] é uma aplicação em *Grade* para execução do BLAST baseada no Globus Toolkit [24, 22]. Esta aplicação basicamente distribui as consultas (*queries*) para diferentes nós da *Grade*. Uma vez tendo chegado a um nó da *Grade*, as consultas são repassadas a máquinas individuais ou clusters para execução do BLAST. A arquitetura do GridBLAST pode ser vista na Figura 2.5.

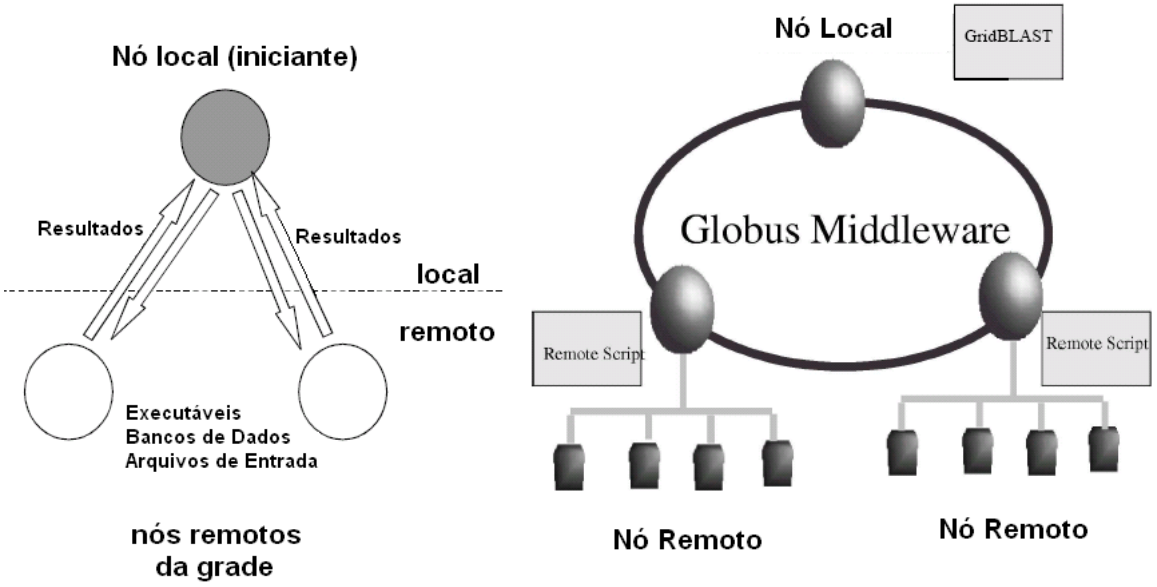

Figura 2.5: Arquitetura do GridBLAST

Uma característica interessante do GridBLAST é que sua execução não pressupõe que os nós remotos da grade possuem os programas, bases de dados ou arquivos de entrada necessários à execução do BLAST previamente instalados. A aplicação consiste basicamente de dois *scripts*: um sendo executado no nó local (iniciante) e o outro em cada um dos nós remotos.

O arquivo de script no nó iniciante realiza um escalonamento estático preliminar das consultas nos diferentes nós utilizando escalonadores estáticos ou dinâmicos. Após as consultas terem sido escalonadas, os arquivos executáveis, arquivos de consulta, e o banco de dados são comprimidos utilizando-se as ferramentas **tar** e **gz**. O script iniciante então aciona um script remoto em cada um dos nós remotos utilizando o comando ***globusrun*** provido pelo Globus Toolkit. O comando *globusrun* é utilizado para submeter tarefas para recursos no Globus. Além disso, este comando também aciona o servidor **GASS**, que é utilizado para prover acesso a arquivos remotos e para redirecionar fluxos de saída padrão no nó local.

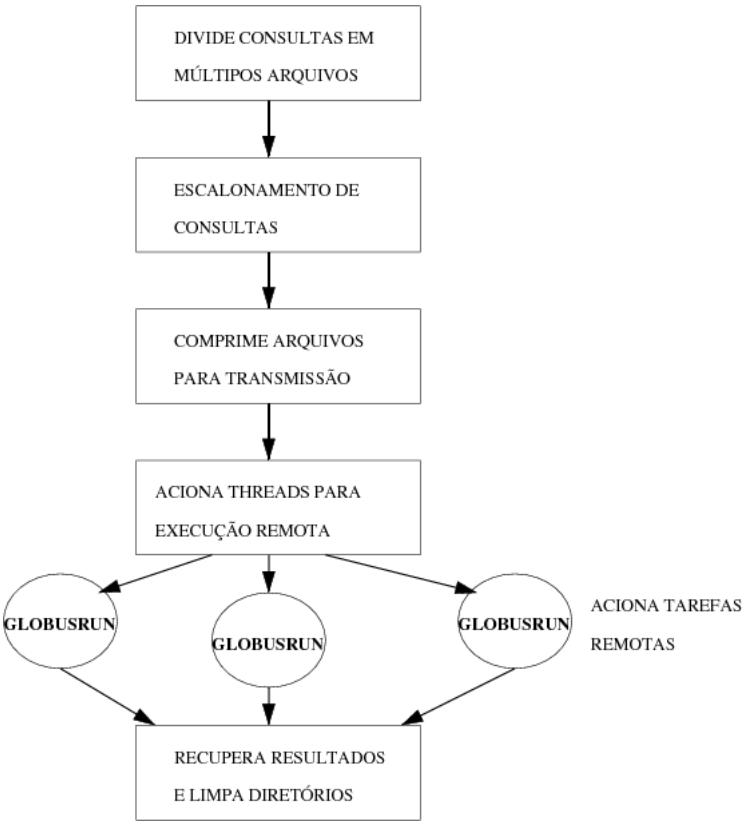

Figura 2.6: Diagrama de Execução GridBLAST (1)

O segundo script, iniciado nos nós remotos pelo comando *globusrun*, conecta-se ao servidor GASS no nó local. Uma vez estabelecida as conexões, o script remoto inicia a transferência dos arquivos necessários (utilizando o comando ***globus-url-copy***) e, quando completado, configura o ambiente e executa o BLAST em uma única máquina ou em um cluster. A medida que cada nó termina sua quota de consultas, os resultados são comprimidos utilizando **tar** e **gz** e copiados de volta para o nó local. O segundo script então limpa os diretórios temporários criados no nó remoto e finaliza sua execução. O script principal descomprime os resultados obtidos de todos os nós remotos e os agrupa em um único diretório. Então ele limpa todos os diretórios temporários e arquivos criados durante a execução da aplicação e finaliza sua execução. O fluxo de controle do primeiro script está ilustrado na Figura 2.6, enquanto o fluxo de execução do segundo script é ilustrado na Figura 2.7

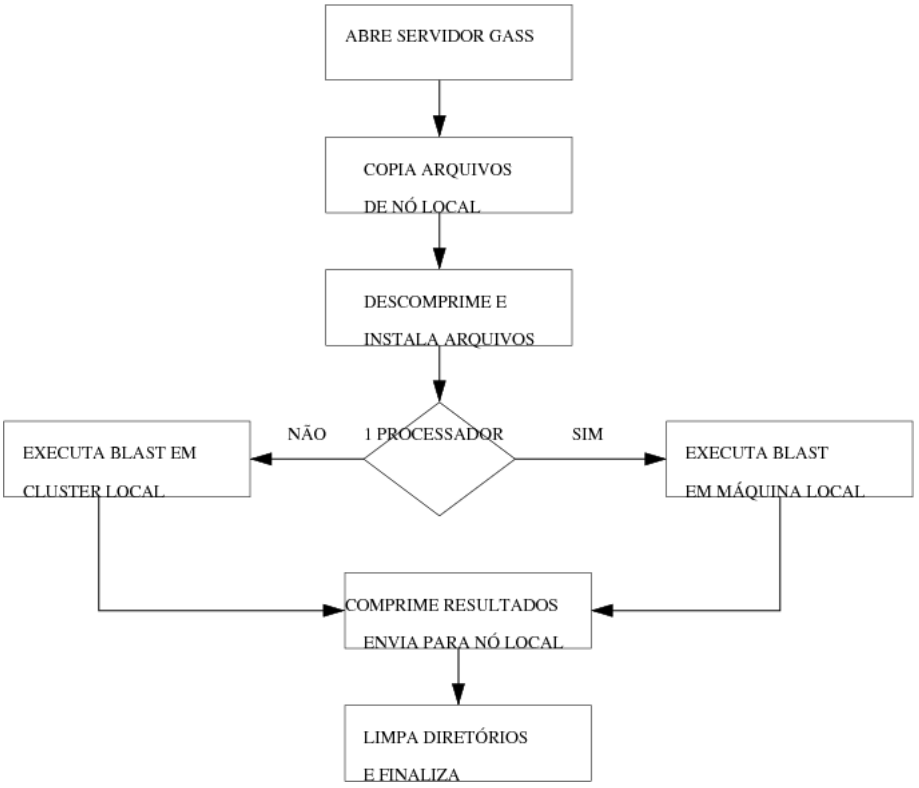

Figura 2.7: Diagrama de Execução GridBLAST (2)

Capítulo 3

Sistemas Peer-to-Peer

Durante as últimas duas décadas, a popularização das redes de computadores, particularmente aquelas conectadas à Internet, fez com que surgisse uma demanda crescente por sistemas distribuídos tanto no meio acadêmico quanto no setor industrial. As inovações trazidas por tais sistemas permitiram uma economia de tempo e recursos, e trouxeram eficiência e eficácia para instituições públicas e privadas ao redor do mundo.

Coulouris e co-autores [25] definem um sistema distribuído como *"um sistema no qual os componentes de hardware e software que fazem parte de computadores conectados às redes de comunicação trocam informações e coordenam atividades unicamente através da transferência de mensagens."*

A principal motivação por trás do uso de sistemas distribuídos é o compartilhamento de recursos (hardware, software, instrumentação eletrônica, serviços) remotamente e, em alguns casos, de forma transparente ao usuário final. Graças às redes de computadores, um astrônomo nos EUA pode controlar remotamente um telescópio localizado a milhares de quilômetros de distância como mostram Cecil e co-autores [26], por exemplo. Ainda, um instituto de pesquisa pode realizar, em questão de horas, complexas simulações bioquímicas, que levariam semanas para serem executadas em um super-computador, graças à distribuição da computação entre centenas de máquinas, espalhadas pela Internet, conforme Shirts e Pande [27].

Neste capítulo, descrevemos dois modelos de arquitetura utilizados em sistemas distribuídos: o clássico modelo cliente-servidor, descrito na seção 3.1, e o modelo *Peer-to-Peer*, apresentado na seção 3.4.

3.1 Sistemas Cliente-Servidor

Os sistemas distribuídos existentes usualmente seguem o modelo cliente-servidor (Figura 3.1). Em um modelo cliente-servidor existe uma divisão clara de tarefas desempenhadas pelas máquinas da rede em um determinado instante. As máquinas servidoras são responsáveis por prover serviços e recursos a vários clientes de forma simultânea. Um servidor de FTP, por exemplo, é responsável por armazenar e disponibilizar arquivos para várias máquinas clientes. As máquinas clientes, por sua vez, são responsáveis pela requisição de tais serviços, além de

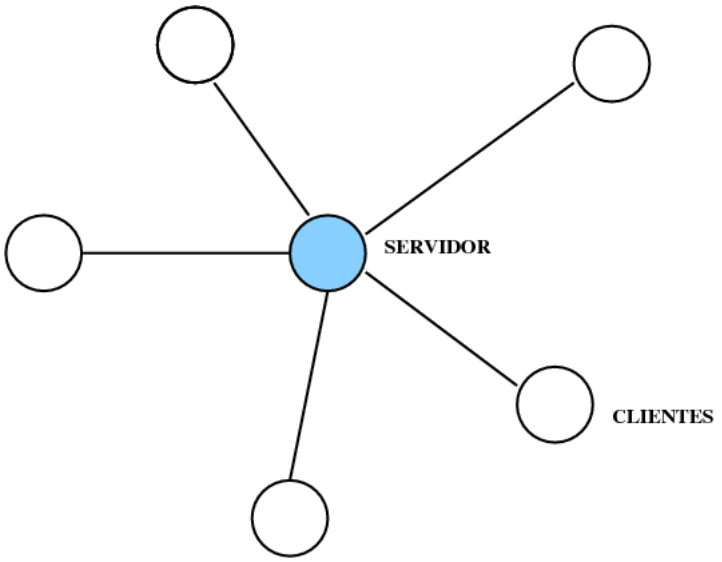

Figura 3.1: Modelo Cliente-Servidor

iniciarem o procedimento de comunicação. No caso de clientes de FTP teríamos várias máquinas enviando e recebendo arquivos do servidor de forma concorrente.

Atualmente, a World Wide Web [25, 13] é o exemplo mais bem sucedido de sistema distribuído que segue o modelo cliente-servidor, pois os navegadores Web (browsers) atuam como clientes, requisitando páginas e serviços Web, enquanto os servidores Web atuam como provedores destas páginas e serviços. Na Figura 3.2, a máquina A, um navegador Web, atua como cliente ao requisitar à máquina B - um servidor Web - páginas, que além de campos fixos contêm dados oriundos de um Sistema Gerenciador de Banco de Dados (SGBD), que é representado pela máquina C. A comunicação inicial entre A e B é feita através de uma requisição HTTP [28]. Em seguida, a máquina B recupera os dados armazenados na máquina C, um servidor SGBD, utilizando o protocolo TCP/IP, e os retorna através de uma resposta HTTP ao cliente A. Nesta segunda etapa, a máquina B atua como cliente da máquina C. Vemos, portanto, que uma máquina pode desempenhar o papel de servidor e cliente ao longo de seu tempo de execução em um sistema, pois a máquina B (servidor Web) atua como cliente da máquina C (servidor SGBD) durante uma determinada operação de recuperação de dados, embora seu papel inicial tenha sido de servidora de páginas para a máquina A.

3.2 Sistema Peer-to-Peer (P2P)

A partir da década de 90, os sistemas *Peer-to-Peer* (P2P) apresentaram-se como mais uma alternativa distribuída para compartilhamento de recursos e computação. Estes sistemas foram inicialmente utilizados para a transferência de arquivos multimídia (áudio e vídeo), que ainda corresponde a grande parte de seu tráfego de dados, mas logo começaram a ser adotados pela comunidade acadêmica para solução de problemas complexos nos mais variados campos do conhecimento [27].

Conforme definem Stoica e co-autores [29], *"sistemas e aplicações Peer-to-Peer*

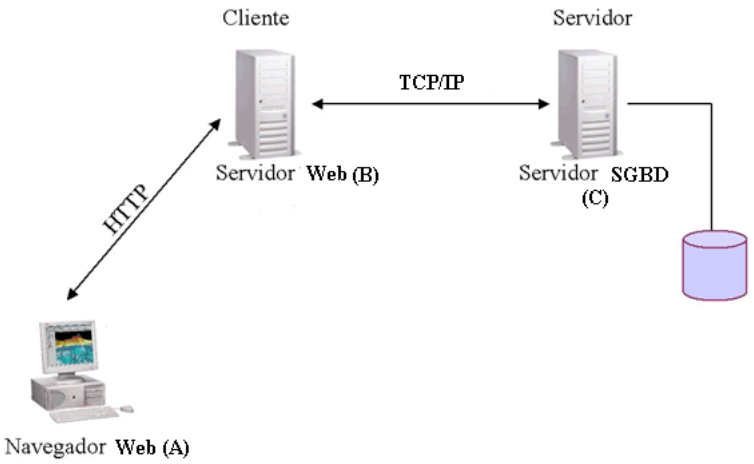

Figura 3.2: Exemplo de Arquitetura Cliente-Servidor

são sistemas distribuídos sem um controle centralizado ou organização hierárquica, no qual cada nó executa software de funcionalidade equivalente". Neste modelo, as máquinas desempenham *simultaneamente* o papel de clientes e servidores, isto é, todas as máquinas podem, a princípio, prover e consumir o mesmo serviço de forma simultânea não existindo uma diferenciação de papéis *a priori*. A Figura 3.3 mostra uma representação clássica de um sistema P2P onde cada um dos nós pode trocar dados com quaisquer outros nós.

Os exemplos de sistemas P2P mais amplamente difundidos na Internet são aqueles que permitem o compartilhamento de músicas e filmes como o Gnutella e BitTorrent. A difusão de material protegido por *copyright* gerou uma forte oposição aos sistemas P2P, o que tem prejudicado a adoção desta tecnologia por setores acadêmicos.

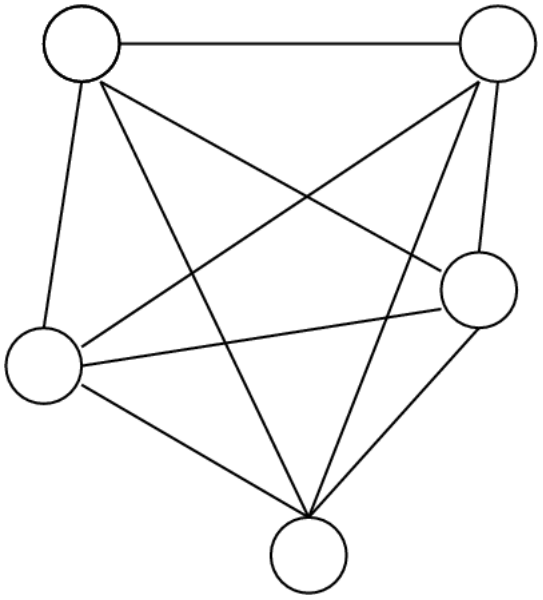

Figura 3.3: Modelo Peer-to-Peer

3.3 Taxonomias

Segundo Milojicic e co-autores [2], os sistemas computacionais podem ser basicamente divididos em *sistemas centralizados* - compostos por uma ou mais unidades processadoras - e *sistemas distribuídos* (Figura 3.4). Os sistemas distribuídos, por sua vez, podem ser classificados em modelo cliente-servidor e modelo *Peer-to-Peer*. O modelo cliente-servidor pode ser *plano*, onde todos os clientes se comunicam com um servidor somente (possivelmente utilizando replicação para aumentar a confiabilidade), ou *hierárquico*. No modelo hierárquico, os servidores de um nível agem como clientes para servidores de níveis mais altos, com o objetivo de aumentar a escalabilidade. Exemplos de modelos planos incluem tecnologias de middleware como CORBA e RMI. Exemplos de modelos hierárquicos incluem servidores DNS e sistemas de arquivos remotos.

O modelo *Peer-to-Peer* pode ser *puro* ou *híbrido*. No modelo puro não existe um servidor central. Todas as máquinas têm as mesmas características. No modelo híbrido, por sua vez, um ou mais servidores são utilizados para se obter informações, tais como a identidade de um *Peer*, as informações armazenadas por ele, ou como o *Peer* deve se autenticar na rede. Uma vez obtida a informação, a comunicação passa a ser realizada de forma direta entre os *Peers*.

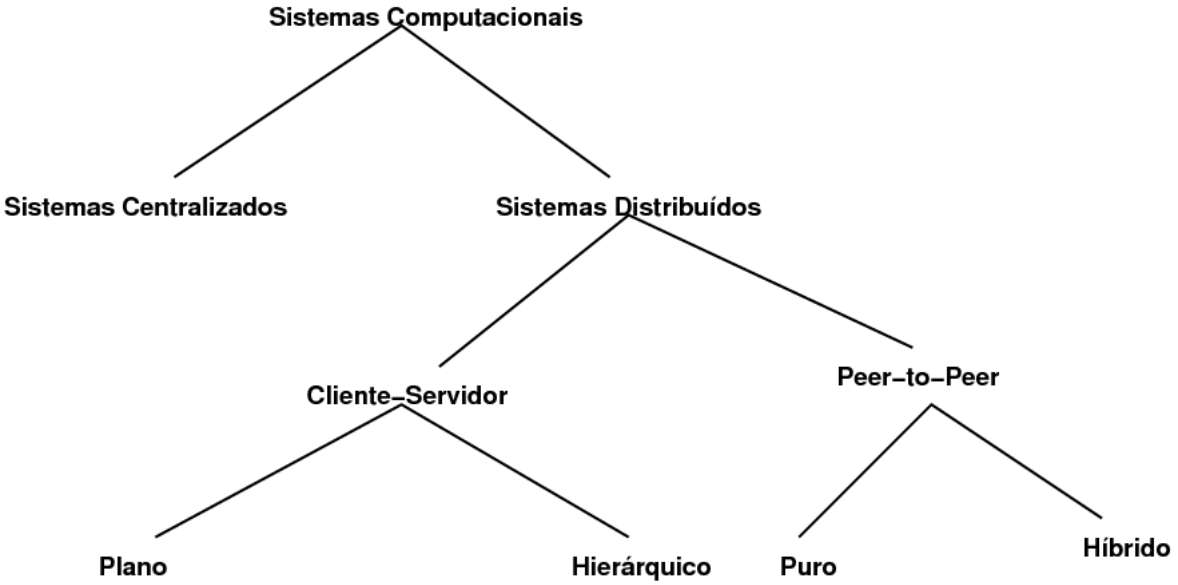

Figura 3.4: Taxonomia de Sistemas Distribuídos, Milojicic e co-autores [2]

Conforme observam Milojicic e co-autores [2] os diversos sistemas P2P existentes podem ainda ser classificados conforme a taxonomia mostrada na Figura 3.6. A computação distribuída quebra problemas de grande complexidade em partes menores e os distribui entre as máquinas ociosas. Folding@Home [30, 31], que calcula a estrutura 3D de proteínas de forma distribuída, é um exemplo de aplicação P2P que segue este paradigma e obteve bastante sucesso na solução de problemas em Bioinformática. Na Figura 3.5 vemos a quantidade de CPUs ativas trabalhando neste projeto.

O compartilhamento de arquivos, por sua vez, permite a troca de dados entre

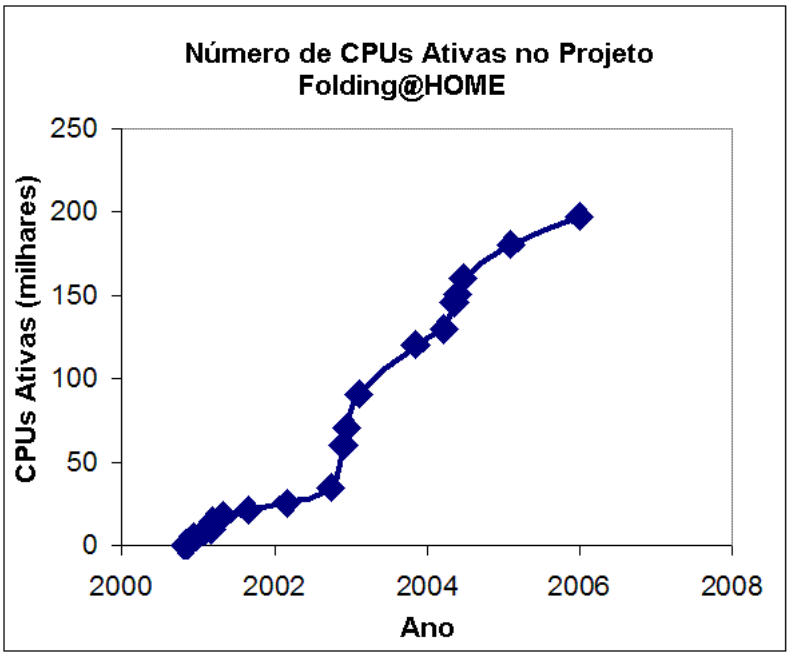

Figura 3.5: CPUs Ativas no Projeto Folding@HOME

os nós da rede tal como o fazem as aplicações Gnutella, Napster e BitTorrent. As aplicações que fazem uso de colaboração podem trocar mensagens e manipular dados de forma conjunta e seus exemplos mais famosos são as aplicações de mensagem instantânea como ICQ e MSN Messenger. Finalmente, a plataforma fornece uma infra-estrutura para a construção de aplicações P2P, que pode ser exemplificada pela plataforma JXTA, utilizada neste trabalho. Ressaltamos que um sistema P2P usualmente desempenha mais de uma das tarefas citadas como compartilhamento de arquivos e troca de mensagens, por exemplo.

Infelizmente, cada novo sistema P2P projetado implementa um conjunto distinto de protocolos de rede incompatíveis com os demais existentes. Além disso, tais implementações são, em geral, fortemente dependentes de um sistema operacional ou linguagem de programação. A plataforma JXTA (Capítulo 4) busca resolver este problema ao tentar padronizar uma plataforma portável para computação P2P.

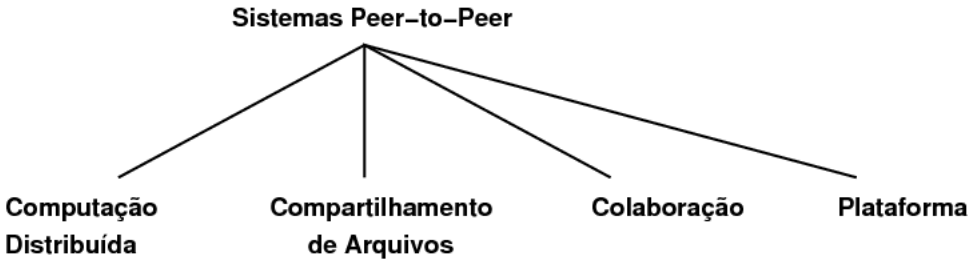

Figura 3.6: Taxonomia de Sistemas Peer-to-Peer

3.4 Tipos de Sistemas P2P

Um aspecto central de qualquer sistema P2P é a topologia das redes P2P, isto é, a organização dos *Peers* que compõem o sistema. Até o presente momento não existe consenso acerca da classificação estrutural dos sistemas P2P. Mesmo assim os sistemas P2P podem ser basicamente classificados de acordo com a organização de seus nós, isto é, a rede virtual criada pela aplicação. De acordo com Landers [32], podemos classificar os sistemas P2P em:

1. Sistemas Híbridos
2. Sistemas Não-Estruturados
3. Sistemas Super-Nó
4. Sistemas Estruturados

É importante salientar que apesar da existência de uma ordem cronológica no surgimento das arquiteturas P2P, existem atualmente na Internet sistemas P2P com as mais variadas arquiteturas, e não existe até o momento um tipo de arquitetura que seja superior às demais em termos de desempenho e eficiência. Tratam-se de abordagens diferentes com vantagens e desvantagens inerentes à forma de organização e troca de dados. A escolha de um determinado tipo de arquitetura P2P dependerá fortemente das necessidades da aplicação e dos recursos a serem disponibilizados pelo sistema.

3.4.1 Sistemas Híbridos

Os sistemas híbridos, ou sistemas de primeira geração, foram os primeiros a mover a computação para fora dos grandes servidores e em direção aos computadores pessoais dos usuários (*Peers de fronteira*). Sistemas como Napster e Folding@Home [30, 31] são exemplos clássicos deste tipo de aplicação. Para solucionar o problema da organização estes sistemas introduzem um servidor central para coordenar os *Peers*. Os sistemas são chamados de *híbridos* porque combinam o modelo cliente-servidor clássico com a natureza massivamente distribuída dos sistemas P2P.

Em um sistema híbrido, cada *Peer* se registra com o servidor no momento da conexão com a rede. Deste modo, o servidor terá sempre uma visão muito precisa sobre os *Peers* atualmente presentes na rede, isto é, um mecanismo presença eficiente. Devido à existência do servidor, os *Peers* raramente se comunicam diretamente uns com os outros (no Folding@Home eles nuncam fazem isso), e somente o fazem se forem identificados pelo servidor. No projeto Folding@Home, por exemplo, o servidor atribui a cada *Peer* registrado uma "unidade de trabalho"(dados protéicos brutos, não analisados), mantendo o registro de dados não analisados, em processo de análise, e dados finalizados. O resultado da computação realizada por cada *Peer* é enviada para o servidor à medida que é finalizado.

A interação é esquematizada na Figura 3.7 onde o *Peer* A consulta o servidor C sobre os possíveis dententores de algum dado ou serviço. O servidor C retorna

o endereço do *Peer* B **(1)**. Em seguida, o *Peer* A entra em contato direto com o *Peer* B **(2)**, e estes dois *Peers* passam a trocar informações de forma direta e descentralizada.

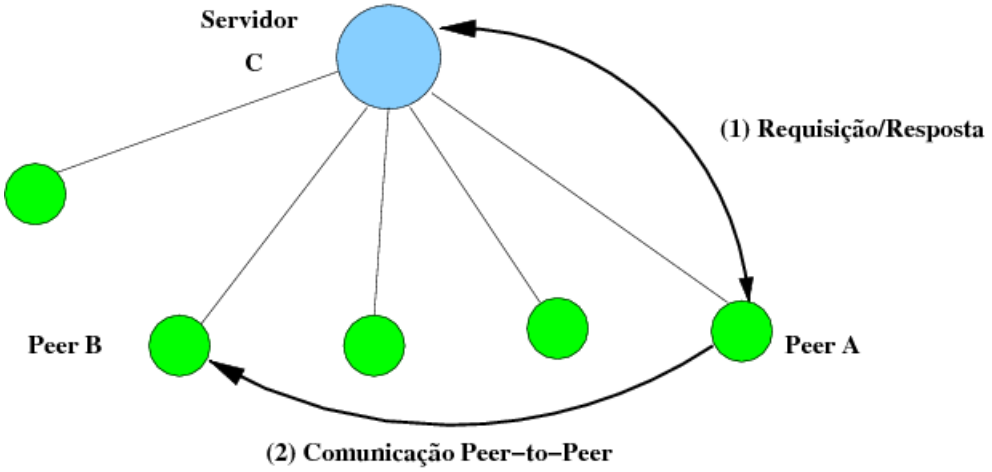

Figura 3.7: Rede Híbrida

A principal vantagem deste tipo de sistema é a visão consistente acerca dos *Peers* que o servidor mantém. Este controle permite que os *Peers* possam ser coordenados de forma mais eficiente, e as buscas têm maiores chances de sucesso. O principal problema da abordagem híbrida é o servidor central, que potencialmente representa um ponto único de falha e um gargalo para o sistema. À medida que a rede P2P cresce torna-se necessário inserir mais de um servidor para coordenar os *Peers*, como fez a rede Napster e, por conseguinte, adiciona mais complexidade ao sistema. Se o(s) servidor(es) vier(em) a falhar, os *Peers* não são capazes de se auto-organizar e ficam impossibilitados de trocar informações. Em outras palavras, a rede P2P deixa de existir.

Alguns criticam a real caracterização dos sistemas citados, pois acreditam que tais sistemas não podem ser qualificados como sistemas P2P, uma vez que representariam uma variação da arquitetura cliente-servidor. De fato, em termos gerais tais sistemas poderiam ser enquadrados no modelo cliente-servidor, mas o fato de moverem as tarefas computacionais para os computadores pessoais de usuários e permitirem, em maior ou menor grau, a interação direta entre máquinas faz com que desempenhem atividades tipicamente P2P.

3.4.2 Sistemas Não-Estruturados

Os sistemas P2P de segunda geração, denominados sistemas não-estruturados, não possuem qualquer controle centralizado e não organizam a arquitetura da rede P2P. Em um sistema não-estruturado, os *Peers* estão conectados entre si de forma aleatória e de forma que seus nós acabam por formar um grafo randômico.

Ao contrário dos sistemas híbridos, os sistemas não-estruturados não possuem uma visão global ou consistente dos *Peers* existentes na rede, pois cada *Peer* conhece apenas um número limitado de "vizinhos", isto é, um subconjunto de *Peer* aos quais tenha se conectado ao entrar na rede ou que tenha descoberto

durante o tempo de vida. O *Peer* que deseja descobrir um recurso envia uma mensagem contendo a consulta para cada um de seus vizinhos conhecidos.

Cada mensagem possui um *Time to Live (TTL)*, um número que indica o tempo de vida da mensagem, isto é, a quantidade máxima de máquinas pelas quais a mensagem deve passar antes de ser descartada. Ao chegar a cada *Peer* o TTL é subtraido de uma unidade e ao chegar a zero a mensagem é descartada. Este mecanismo evita que as mensagens fiquem trafegando indefinidamente pela rede e inundem as redes com mensagens, mas a estratégia de TTL cria um horizonte de propapação para as mensagens. Este horizonte de propagação pode fazer com que recursos existentes na rede P2P não venham a ser descobertos.

Na Figura 3.8, o *Peer* A envia uma mensagem em broadcast com TTL igual a três para seus *Peers* vizinhos (B e G) em busca de algum recurso. Os *Peers* B e G, propagam esta mensagem para seus *Peers* vizinhos, e a cada nó percorrido o TTL da mensagem é decrementado de uma unidade. Neste exemplo, o recurso estava presente no *Peer* D, que entra em contato direto com o *Peer* A para repassar o recurso ou ativar algum serviço. Entretanto, lembramos que se o recurso estivesse presente somente no *Peer* F, a consulta não iria retornar resultado apesar da existência do recurso na rede. Encontrar um valor de TTL que permita encontrar o maior número possível de recursos sem inundar a rede de mensagens tem sido o grande desafio de sistemas que implementam esta estratégia.

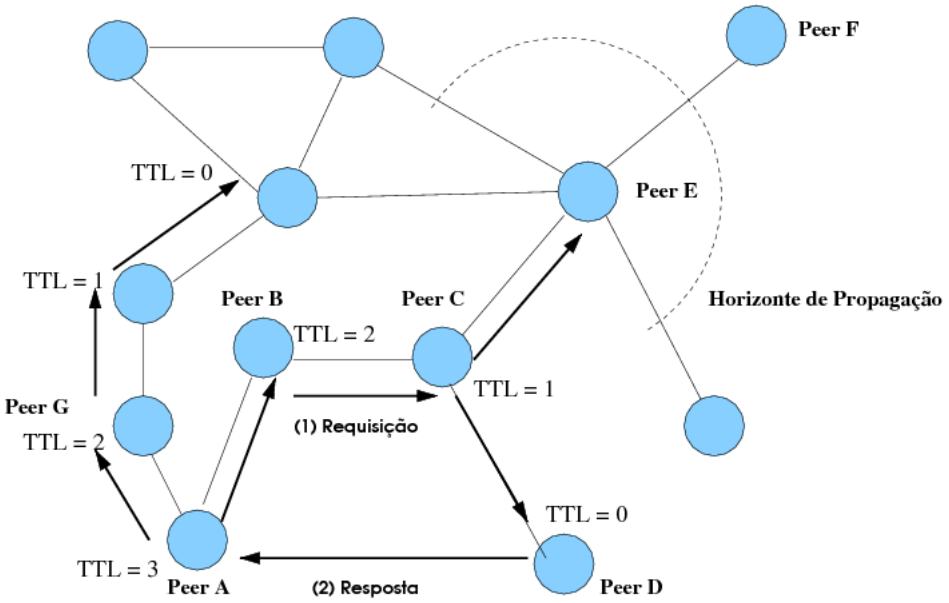

Figura 3.8: Rede Não-Estruturada

A grande desvantagem deste tipo de arquitetura é a falta de escalabilidade, pois à medida que a quantidade de *Peers* aumenta, as mensagens irão atingir apenas uma pequena parte da rede. Esta rede torna-se eficiente à medida que os recursos mais procurados (*hot spots*) estão replicados em quantidade suficiente na rede. A vantagem deste tipo de rede é a descentralização que elimina pontos únicos de falhas e torna a rede resistente a ataques maliciosos. O Gnutella é o exemplo mais conhecido deste tipo de rede.

3.4.3 Sistemas Super-Nó

Os sistemas P2P do tipo Super-Nó podem ser considerados um tipo especial de sistema não-estruturado. Mas devido a algumas peculiaridades, e pelo fato de terem surgido depois, são considerados sistemas de terceira geração e possuem uma classificação própria. Os sistemas Kazaa e Skype são exemplos de sistemas com este tipo de arquitetura.

A principal idéia por trás das sistemas Super-Nó é dividir os *Peers* em dois tipos: *Normais (edge)* e *Super-Nó (super node)*. Os *Peers* normais estão conectados a *super-nós* bem conhecidos, que possuem uma quantidade maior de recursos (CPU, RAM, conectividade de rede, etc). Somente uma pequena parcela dos *Peers* presente no sistema são *super-nós*, os quais formam uma rede não-estruturada, como demonstram os *Peers* A, B, C e D da Figura 3.9. Os *Peers normais* estarão necessariamente conectados a um super-nó (*Peers* E1 até E8) e poderão se comunicar diretamente entre si ou através do *Super-Nó*.

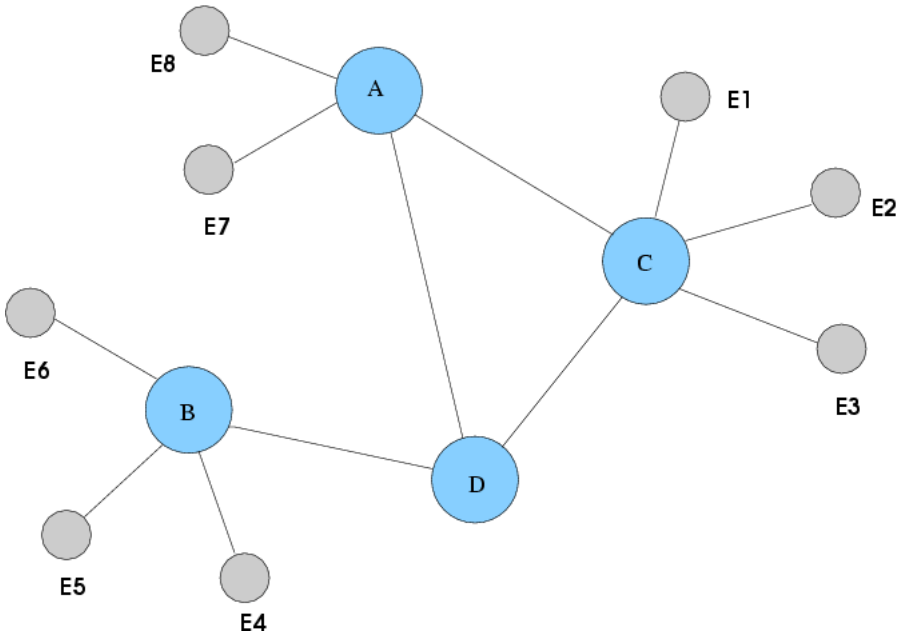

Figura 3.9: Rede Super-Nó

3.4.4 Sistemas Estruturados

Sistemas Estruturados são outro tipo de sistema P2P de terceira geração, que evoluiram em paralelo com os sistemas *Super-Nó*. Os sistemas nesta arquitetura são *organizadas pelo conteúdo*. Em outras palavras, os *Peers* na rede estruturada formam grupos (clusters), de acordo com o conteúdo que possuam em comum. Isto permite o desenvolvimento de algoritmos mais inteligentes para o distribuição de conteúdo, roteamento de mensagens e busca de recursos na rede P2P. Os exemplos de sistemas estruturados são Kademlia [33], Pastry [34], Chord [29] e JXTA [35].

A rede estruturada é organizada mediante a criação de uma "rede virtual"em cima de uma rede P2P básica. Nesta rede virtual, a proximidade entre *Peers* é determinada por alguma métrica geralmente derivada do conteúdo provido pelos *Peers*. Desta forma, dois *Peers* estarão tão mais próximos quanto estiverem os conteúdos disponibilizados por ambos. Tais sistemas têm sido objeto de um amplo estudo da comunidade acadêmica, pois têm permitido a criação de algoritmos eficientes para busca e distribuição de conteúdo na rede P2P.

Entretanto, a principal desvantagem deste tipo de rede é o sobre-custo (*overhead*) computacional necessário para manter a consistência da rede virtual, principalmente em redes com Peers que entram e saem com freqüência (*churn rate*). Além disso, *Peers* que sejam vizinhos na rede estruturada podem estar a uma longa distância física um do outro, o que acarreta um custo de tráfego de mensagem.

3.5 Busca e Roteamento

A *Tabela Hash Distribuída (DHT)* [29] é uma estrutura de dados que oferece uma interface distribuída para o armazenamento e recuperação de dados distribuídos em redes P2P. Esta estrutura de dados tem sido extensivamente utilizada como substrato para construção de sistemas P2P devido a sua escalabilidade e performance na localização de dados, pois as mensagens podem ser roteadas entre os nós de uma rede P2P com um número mínimo de pulos (*hops*) entre *Peers*.

A cada nó ou dado é associado a uma chave, e a DHT faz o mapeamento entre uma chave e o nó responsável por aquela chave, i.e., busca (*lookup*).

Existem inúmeras implementações de DHTs na literatura [29, 34, 33, 36, 37], sendo a grande maioria em caráter experimental, isto é, existente apenas em simuladores. Tais trabalhos buscam derivar algoritmos de roteamento eficientes - tipicamente da ordem de $O(logN)$, onde N é a quantidade de nós - , ou garantir a consistência e escalabilidade da DHT a medida que os *Peers* entram e saem aleatoriamente da rede. Por outro lado, apesar de diferenças específicas de projeto, as DHTs seguem alguns princípios básicos de operação que serão detalhados a partir do protocolo Chord [29], que serviu de base para o projeto de várias outras DHTs.

Chord faz o mapeamento de chaves a nós através do uso de *hash consistente*. O hash consistente permite que a distribuição das chaves entre os nós seja o mais balanceada possível. Além disso, existe grande probabilidade que com a entrada ou saída de um nó da rede apenas uma fração de $O(1/N)$ das chaves precise ser movida entre nós.

A escalabilidade é garantida pois cada nó só precisa ter conhecimento acerca de um subconjunto de N para fins de roteamento. Em uma rede com N nós, cada nó mantém informação sobre $O(logN)$ outros nós, e uma operação de busca requer $O(log^2N)$ mensagens.

A cada nó e chave é atribuído um identificador de m-bits usando hashing SHA-1. O identificador do nó pode ser obtido através do hashing do endereço IP enquanto o identificador da chave é obtido com o hashing da chave, por exemplo.

Os identificadores são ordenados em sentido horário em um *círculo identifi-*

cador (*Anel Chord*) variando de 0 a $2^m - 1$. Uma chave k é associada ao primeiro nó cujo identificador seja maior ou igual ao identificador da chave k. Este nó é chamado de nó sucessor da chave k e denotado por *sucessor(k)*, conforme podemos ver no exemplo da Figura 3.10.

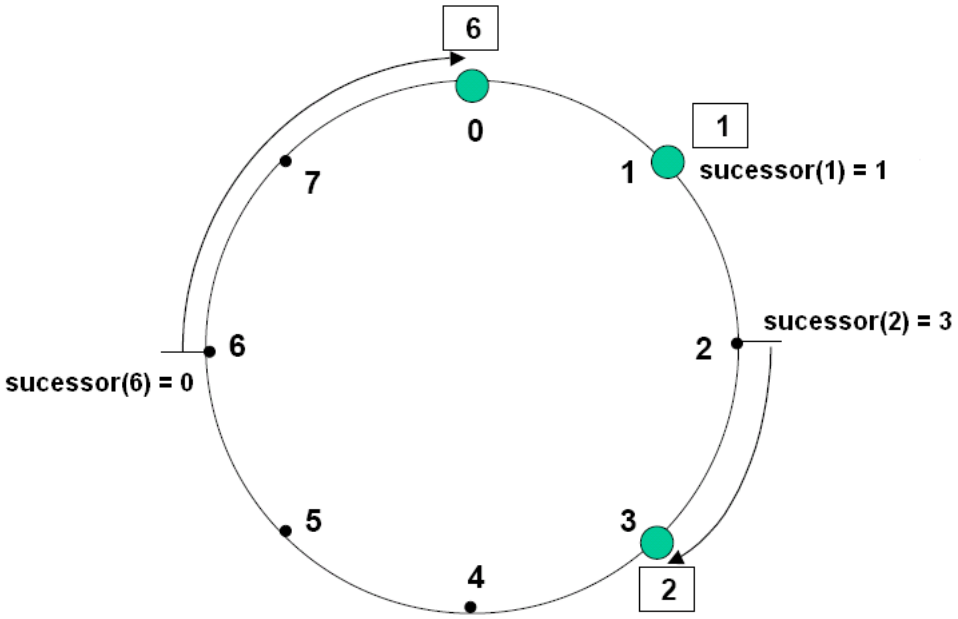

Figura 3.10: Anel Chord com m = 6

É importante ressaltar que quando um nó n entra na rede algumas chaves previamentes associados ao sucessor de n devem ser associadas a n. De forma análoga, quando n sai da rede, as chaves anteriormente associadas a n precisam ser associadas ao sucessor de n.

Para acelerar o processo de roteamento e evitar que uma consulta tenha que percorrer todos os nós de um círculo identificador, cada *Peer* mantém uma tabela de roteamento denominada *finger table*. Seja m o número de bits do identificador de chaves/nós, logo cada nó manterá no máximo m entradas em sua finger table. A i-ésima entrada na tabela referente ao nó n contém a identidade do *primeiro* nó, s, que sucede n por pelo menos 2^{i-1} no círculo identificador, isto é, $s = sucessor((n + 2^{i-1}) \ mod \ 2^m)$, onde $1 \leq i \leq m$. Chamamos o nó s o i-ésimo finger do nó n, e o denotamos por $n.finger[i].node$. Cada entrada na *finger table* inclui tanto o identificador do nó quanto seu endereço IP e porta para acesso, e a Tabela 3.1 mostra outros campos presentes em cada entrada k da *finger table* de um determinado nó n.

A Figura 3.11 mostra um anel Chord com $m = 3$, três nós 0,1,3 e suas respectivas *finger tables*. Maiores detalhes sobre a implementação do Chord e algoritmos de percorrimento do círculo podem ser encontrados em [29].

Tabela 3.1: Campos presentes em cada registro k da finger table

finger[k].start	$(n + 2^{k-1})\ mod\ 2^m, 1 \leq k \leq m$
finger[k].interval	[finger[k].start, finger[k+1].start)
finger[k].node	primeiro nó $\geq$ a n.finger[k].start
sucessor	finger[1].node
predecessor	o nó anterior no círculo

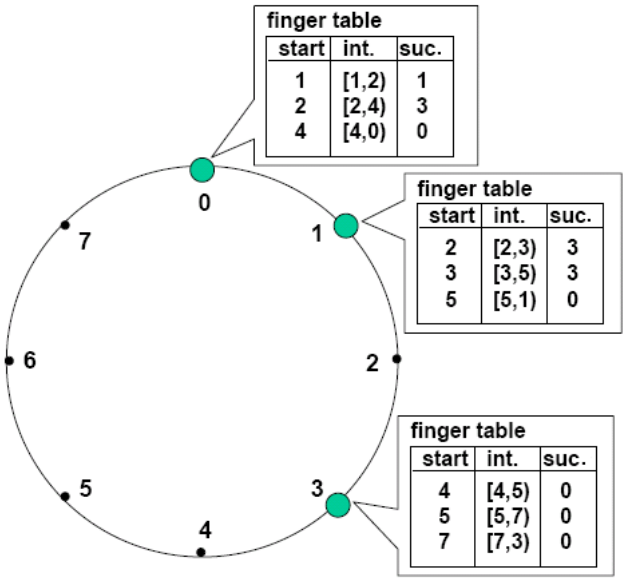

Figura 3.11: Anel Chord para m=3 com finger tables

3.6 Presença em Sistemas P2P

A *presença* desempenha um papel cada vez mais importante nos sistemas distribuidos e, particularmente, em sistemas *Peer-to-Peer*. Em tais sistemas os recursos (CPU, espaço em disco, largura de banda, ...) e usuários estão distribuidos, e os mecanismos de presença permitem que cada entidade tome conhecimento acerca do estado atual das outras entidades presentes na rede.

Presença é definida como o estado no qual se encontra o usuário, aplicação ou hardware. Esta informação é disponibilizada para o resto do sistema de tal forma que possa ser vista e utilizada por outras entidades. O uso de *presença* provê informação contextual que pode ser vantajosa para a utilização eficaz de um sistema.

Geralmente a *presença* é utilizada para capturar quando um usuário está ativo/inativo (*on-line/off-line*), mas pode ainda ser utilizada para obter outras informações tais como a localização física da entidade, contexto do ambiente computacional, atividades realizadas pela entidade, ou informações específicas da aplicação.

Tem-se despendido esforços para desenvolver modelos, formatos de dados e

protocolos que suportem mecanismos de presença na Internet. Um exemplo deste esforço é o eXtensible Messaging and Presence Protocol (XMPP) [38], que encontra-se sob processo de revisão pelo Internet Engineering Task Force (IETF). Embora sejam direcionado para a arquitetura *Peer-to-Peer*, este protocolo não abrange a ampla gama de arquiteturas P2P possíveis.

3.6.1 Tipos de Presença

A presença pode ser dividida em duas categorias: *presença de Peer* e *presença abstrata*.

1. ***Presença de Peer*** representa informação que está disponível sobre um *Peer* que faz parte da rede. Esta informação geralmente inclui se o *Peer* está ou não ativo, mas pode ainda incluir outras informações tais como o endereço IP de um *Peer* ou informações sobre a conectividade de rede.

2. ***Presença Abstrata*** representa informação que está disponível sobre as entidades que utilizam os *Peers*, ou informação sobre o ambiente operacional do *Peer*.

   A presença abstrata pode ainda ser dividida em:

   - *Presença de Usuário* representa informação sobre o usuário de um *Peer*. Geralmente diz se um usuário está ativo/inativo, ocupado ou indisponível. Embora o *Peer* esteja conectado à rede, o usuário pode optar por se registrar como estando inativo. Além disso, é perfeitamente possível que múltiplos usuários façam uso de um único *Peer* físico, i.e. compartilhem a mesma máquina. Desta forma é necessário prover um identificador único (ID) para cada um dos usuários.
   - *Presença de Recurso* representa informação que está disponível sobre os recursos que estão conectados aos *Peers*. Este recursos se dividem em duas categorias: *internos* e *externos*. Os recursos internos são aqueles que compõem a máquina, como CPU, espaço em disco, largura de banda, softwares, etc. Os recursos externos, por sua vez, são aqueles que independem de máquina específica e não desempenham papel importante para o funcionamento da mesma. Este é o caso de arquivos (áudio, vídeo, documentos, etc.) a serem compartilhados, por exemplo.
   - *Presença Física* representa informação sobre o ambiente físico do *Peer*. Esta informação diz respeito tanto a localização quanto informações sobre o navegador web atualmente utilizado, quais os arquivos o usuário está atualmente editando ou fluxos de áudio e vídeo sendo apresentados, por exemplo.

Embora sejam duas categorias distintas, *presença abstrata* pode ser considerada como uma extensão da *presença de Peer*. Entretanto, ressaltamos que o relacionamento entre os dois tipos de presença não será necessariamente um para um. Conforme citado anteriormente, N usuários (presença abstrata) podem

fazer uso de uma única máquina executando um sistema P2P (presença de Peer) e, neste caso, teríamos um relacionamento N para 1.

Ao implantar um mecanismo de presença em um sistema P2P, existem dois aspectos que devem ser observados. Em primeiro lugar, a informação de presença deve ser disseminada através do sistema e, em segundo lugar, o sistema deve reagir quando este informação for alterada. Estes dois aspectos são interdependentes e serão afetados pela escolha da arquitetura de rede P2P utilizada.

3.6.2 Eventos de Presença

Para que a *presença* seja incorporada a sistema *Peer-to-Peer* é necessário identificar os principais eventos geradores de novos estados de presença. Estes eventos irão ativar a disseminação ou atualização de informação de presença na rede. São estes os eventos:

1. Conexão - ocorre quando a entidade detentora de informação de presença conecta-se à rede P2P. O estado muda de inativo para ativo.

2. Desconexão - ocorre quando a entidade provendo detentora de informação de presença intencionalmente se desconecta da rede.

3. Falha de conexão - ocorre quando um *Peer* é acidentalmente desconectado da rede P2P devido a problemas na máquina ou falha na corrente elétrica, por exemplo. Neste caso o *Peer* não terá como informar à rede acerca de sua desconexão e, portanto, devem ser elaborados mecanismos que permitam a outras partes interessadas tomarem conhecimento da falha.

4. Mudança de Estado de Presença - ocorre quando a entidade responsável pela informação de presença muda a informação a ser publicada. Por exemplo, o usuário pode mudar o estado de ocupado para livre.

Além de tratar os eventos expostos anteriormente, um mecanismo de presença precisa possuir duas funcionalidades para seu correto funcionamento:

- Conhecimento - quando uma entidade se conecta à rede P2P ela precisa ser informada sobre o estado de presença das outras entidades presentes na rede. Essencialmente isto significa dar à entidade que acabou de entrar a informação mais atualizada possível sobre a presença de outras entidades na rede P2P.

- Atualização de Conhecimento - quando uma entidade muda seu estado é necessário atualizar esta informação para todas as outras entidades interessadas.

3.6.3 Presença x Arquitetura de Rede

Conforme apontam Walkerdine e co-autores [39], os mecanismos de presença são fortemente influenciados pela escolha da arquitetura de rede P2P. Para que possamos escolher o mecanismo de presença mais adequado para um determinado

sistema *Peer-to-Peer* é necessário conhecer a arquitetura da rede criada por este sistema. Tendo por base na divisão proposta na seção 3.4 teremos as seguintes soluções de projeto para tratar os eventos e estados de um mecanismo de presença:

1. Sistemas Não-Estruturados

   Este tipo de rede apresenta as maiores facilidades para se obter presença devido à existência de um servidor central que indexa ou coordena os *Peers*. Este servidor pode então ser utilizado para capturar e publicar a informação de presença, e como todos os *Peers* estarão conectados a ele de forma direta o servidor terá uma visão consistente e atualizada da presença. Obviamente o lado negativo é o Peer único de falha, pois se o servidor falhar o mecanismo de presença estará completamente comprometido. O mecanismo de presença baseado em arquiteturas híbridas tem sido o mais utilizado por sistemas que implementam presença, incluindo as aplicações de mensagens instântaneas como ICQ e MSN, assim como compartilhamento de arquivos.

2. Sistemas Não-Estruturados

   Este tipo de rede não possui um coordenador central, logo para implementar a presença cada *Peer* deve conhecer os outros *Peers* da rede ou pelo menos aqueles que o interessam. A partir daí cada *Peer* pode manter o controle e ter uma visão consistente acerca dos estados de presença dos outros *Peers* na rede. A principal desvantagem deste tipo de arquitetura diz respeito a escalabilidade porque um *Peer* precisa propagar sua informação de presença para todos os outros *Peers* da rede. Quanto mais *Peers* são adicionados à rede, maior a sobrecarga nas comunicações. Entretanto, se a rede permanecer em uma escala pequena então este tipo de rede oferece a melhor base para a implantação de presença.

3. Sistemas Super-Nós e Estruturados

   Estas duas arquiteturas oferecem os maiores desafios para implantação de presença devido ao fato das comunicações serem indiretas e pela falta de um coordenador central. Além disso, devido ao fato deste tipo de rede mudar constantemente pode ser difícil conseguir qualquer forma de atualização de presença em tempo real ou mesmo garantir que as mensagens de atualização serão recebidas por todos os *Peers* presentes na rede. Isto acaba por gerar inconsistência na informação de presença.

   A solução para estes problemas reside na criação de pequenos grupos de entidades. *Peers* presentes nestes grupos poderiam se comunicar de forma direta e haveria também suporte para comunicação inter-grupos. Seria também necessário adicionar algum grau de gerenciamento na rede P2P o que acabaria por distanciá-la do modelo P2P puro e colocá-la na categoria de sistema híbrido.

Capítulo 4

Plataforma JXTA

Em 2001, a multiplicidade de tecnologias *Peer-to-Peer* (P2P) existentes e incompatíveis entre si, com implementação fortemente dependente do sistema operacional ou do hardware, fez com que a empresa norte-americana Sun Microsystems Inc. [40] desse início ao projeto JXTA [35, 41, 42], uma plataforma P2P aberta para computação distribuída.

A plataforma JXTA é um projeto de código-fonte aberto cujo objetivo é criar um padrão que facilite o desenvolvimento de sistemas baseados no modelo P2P. Além disso, outro objetivo é possibilitar que uma aplicação baseada em JXTA seja executada não somente em computadores pessoais, mas em qualquer dispositivo eletrônico capaz de se conectar em rede (PDAs, celulares, mainframes, PCs).

Atualmente, a plataforma JXTA é a infra-estrutura mais madura e estável para o desenvolvimento de aplicações *Peer-to-Peer*, e conta com a colaboração cada vez maior de especialistas da indústria de software e instituições acadêmicas ao redor do mundo. Esta plataforma é definida como um conjunto de seis protocolos XML [43] (Tabela 4.1) que implementam as principais operações encontradas em sistemas P2P, tais como publicação e busca de recursos, auto-organização da rede, e troca de mensagens entre *Peers*. Cada um dos protocolos JXTA implementa funcionalidades distintas dos demais e um *Peer* não precisa necessariamente implementar todos os protocolos para ser um *Peer* JXTA, mas apenas aqueles protocolos necessários a sua operação. Além disso, esta padronização em formato XML faz com que o JXTA seja independente de sistema operacional, linguagem de programação e protocolo de transporte de rede.

Dentre as várias tentativas de implementação da plataforma JXTA, denominadas *bindings*, destacam-se a baseada em Java, **JXTA-J2SE**, que é a implementação de referência da plataforma e foi utilizada neste trabalho, e a implementação em C, **JXTA-C**, que embora não possua uma implementação completa tem sido ativamente desenvolvida e possui bons níveis de performance. Vale ressaltar que cada um dos protocolos mostrados na Tabela 4.1 é implementado mediante um serviço, desta forma temos o Resolver Service, Discovery Service, Rendezvous Service, Pipe Service e Peer Information Service e Endpoint Service.

Para entender os motivos que levaram ao surgimento da tecnologia JXTA é necessário contextualizar a situação atual das rede de computadores. Os avanços nas tecnologias de rede tornam cada vez mais difícil a conexão direta entre duas ou mais máquinas na Internet. Tecnologias como o DHCP [44], NAT [45] e

Protocolo	Função
Peer Information Protocol (PIP)	Obtenção de informações sobre *Peers*
Peer Resolver Protocol (PRP)	Envio/Recepção de consultas genéricas
Rendezvous Protocol (RVP)	Propagação de mensagens através de *Rendezvous*
Endpoint Routing Protocol (ERP)	Descoberta de rotas entre *Peers*
Peer Discovery Protocol (PDP)	Publicação e descoberta de *Anúncios* na rede
Pipe Biding Protocol (PBP)	Estabelecimento de canal de comunicação virtual

Tabela 4.1: Protocolos JXTA

Firewalls [46] permitem que as instituições exerçam um maior controle sobre suas comunicações internas e externas, além de garantir mais segurança e flexibilidade na administração de rede. Por outro lado, limitam o tipo de conexão possível entre as máquinas e o modelo de aplicação de rede passível de ser desenvolvido. Neste cenário as aplicações ficam limitadas à arquitetura definida pelos administradores de rede.

Além disso, o crescimento exponencial na quantidade de computadores conectados à Internet mostrou que a quantidade de endereços IP disponíveis era insuficiente para atender a nova demanda. O NAT (Network Addressing Protocol) [45] é um mecanismo criado para permitir que os computadores de uma rede privada possam se conectar à Internet fazendo uso de um único endereço IP externo. Os endereços IP da rede interna podem ser dinâmicos, e não podem ser acessados diretamente por um computador a partir da Internet. Toda vez que um computador da rede interna envia um pacote IP (datagrama) para a Internet, o NAT reescreve o cabeçalho do datagrama de forma a substituir o endereço de origem pelo número de IP externo válido. A resposta, se houver, segue o processo inverso e é traduzida para o endereço IP interno do computador que enviou originalmente o pacote, e é repassado para este.

Além de economizar os números IP externos, o NAT facilita o processo de administração de rede e permite uma maior segurança, pois as comunicações entre as máquinas internas e externas passam necessariamente pelo NAT, e não é possível a um computador externo iniciar conexões com quaisquer das máquinas internas.

Para permitir que as máquinas pudessem fazer parte de uma rede P2P independente de sua limitação para conexões diretas, o JXTA criou uma rede dinâmica virtual sobreposta a rede física existente (Figura 4.1) para permitir que *Peers* localizados em redes distintas - separados por firewalls e sistemas NATs, ou que façam uso de protocolos de rede diferentes -, sejam agrupados e possam trocar informações de forma transparente e flexível. Múltiplas redes virtuais podem ser criadas e dinamicamente mapeadas em um única rede física através do uso de duas estratégias básicas.

Em primeiro lugar, existe uma separação entre o endereço físico da máquina, que pode ser um endereço IP dinâmico, e o identificador do *Peer*, um número que identifica de forma única aquele *Peer* na rede JXTA. De fato, não somente os *Peers*, mas quaisquer outros recursos tais como dados, por exemplo, são identificados desta forma. O mapeamento entre o identificador e o endereço IP do

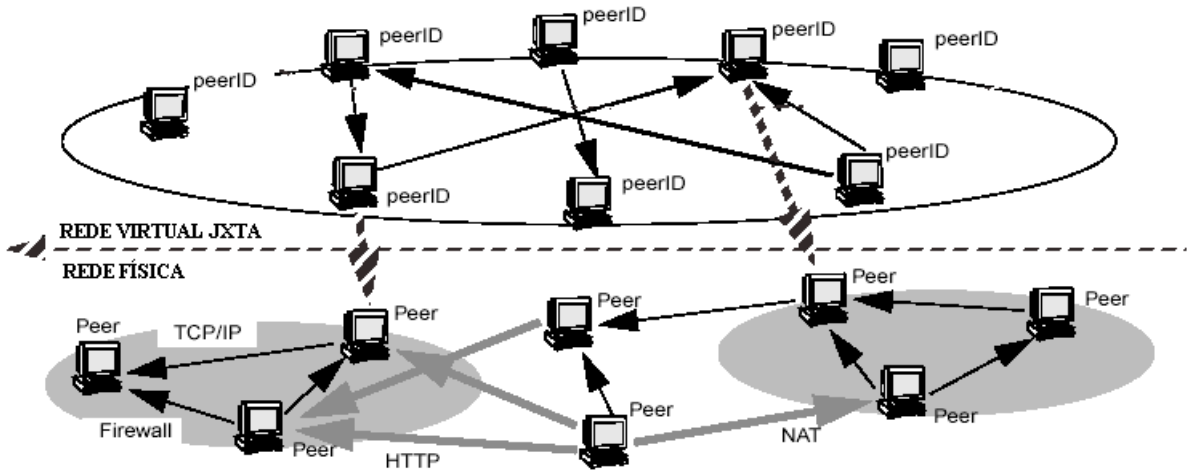

Figura 4.1: Rede Virtual JXTA

recurso passa a ser feito de forma dinâmica. Em segundo lugar, para garantir que *Peers* escondidos por um firewall ou separados geograficamente possam se comunicar são utilizados *super-nós*, denominados *Rendezvous* e *Relays*, que permitem fazer o roteamento de mensagens entre redes físicas distintas.

4.1 Componentes de uma rede JXTA

As implementações do JXTA são construídas em torno de cinco abstrações (entidades): Identificadores (IDs), *Peers* (Peers), *grupos* (PeerGroups), *anúncios* (Advertisements) e pipes.

4.1.1 Identificador (ID)

O ID é um endereçamento lógico e independente de localização física que identifica de forma única *todos* os recursos de uma rede JXTA (*Peer*, *grupo*, *anúncios*, serviços, etc.). A implementação de referência do JXTA utiliza a função de hash SHA-1 para gerar um número aleatório de 160 bits que representa e endereça os recursos na rede JXTA. Temos abaixo um exemplo de número utilizado como PeerID pela plataforma JXTA.

```
urn:596162616461626l4A78746150325033DC3E2FD07B96417F8669EE87CB4DC70303
```

A separação entre identificação e localização física do *Peer* permite que um computador troque seu endereço físico (endereço IP) inúmeras vezes via DHCP, por exemplo, mas permaneça com o mesmo peerID. Ainda, um *Peer* pode possuir várias interfaces de rede (ethernet, wireless, etc.) mapeadas para um único peerID. Isto ocorre graças ao conceito de *endpoint*. Um *endpoint* identifica de forma única todos os endereços físicos de rede e protocolos (TCP, HTTP) disponíveis para se acessar um dado *Peer*. Desta forma, ao receber um *anúncio de endpoint* de um *Peer* B, o *Peer* A pode selecionar o modo mais eficiente de se comunicar com o *Peer* B dentre a lista de endereços e protocolos de rede existentes no *anúncio de endpoint* de B.

4.1.2 *Peers*

Peers são entidades com algum poder de processamento e conectividade de rede. Não são sinônimos de máquinas, pois uma determinada máquina pode conter vários *Peers*. Na plataforma JXTA, os *Peers* devem ser configurados antes de serem executados pela primeira vez. Esta configuração define o nome do *Peer*, cria um ID que o identifica (PeerID), e permite informar endereços IP e portas a serem utilizados para comunicação com outros *Peers*. Além disso, o usuário pode configurar inicialmente um conjunto inicial de *Rendezvous* e *Relays* que o *Peer* irá utilizar para se conectar a rede JXTA (Figura 4.2).

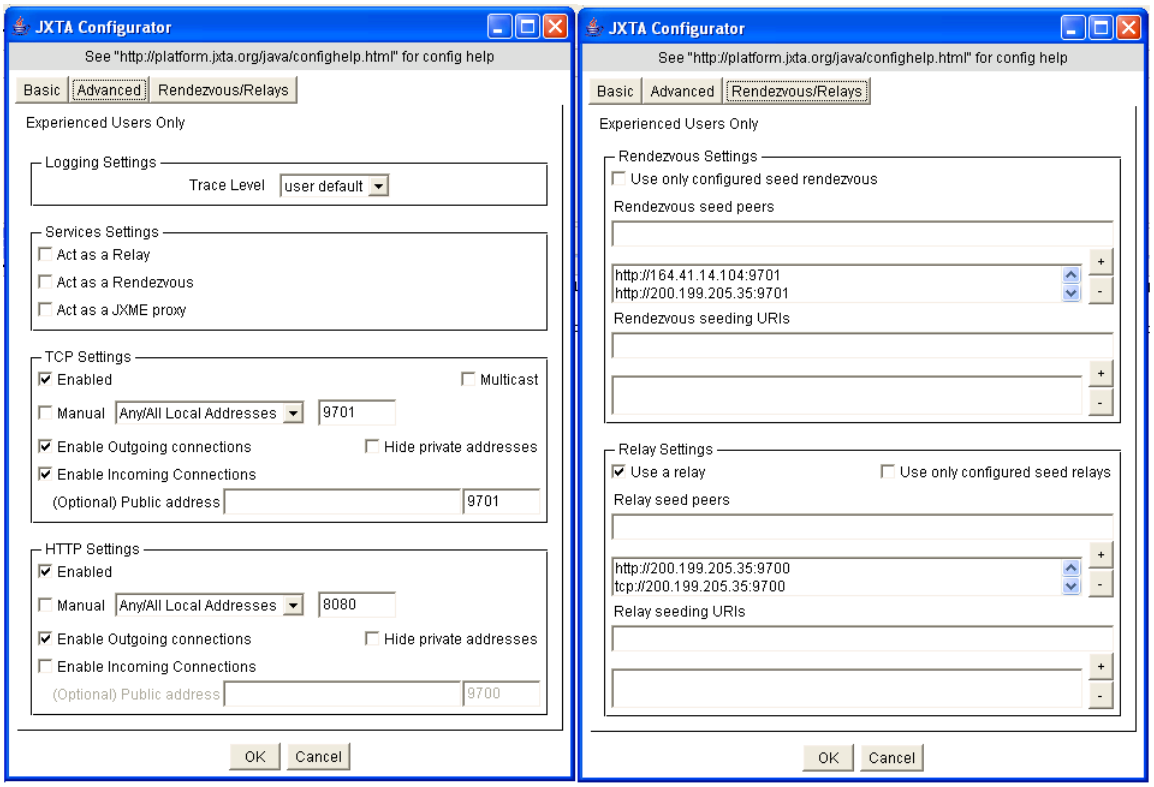

Figura 4.2: Telas de Configuração do JXTA

Apesar de ser conveniente, esta interface requer conhecimentos acerca da rede física que usuários menos experimentes podem desconhecer, e esta foi uma das razões que impediu a ampla adoção da plataforma JXTA como solução para aplicações P2P em larga escala. Por esta razão, atualmente existe outra solução de configuração, entitulada *ext:config*, que permite a configuração da plataforma JXTA a partir de um arquivo XML definido pelo projetista da aplicação.

Ao configurar um *Peer* JXTA, as informações de configuração são salvas em um arquivo denominado *PlatformConfig*, que por sua vez é armazenado no diretório de execução do JXTA (.jxta, na execução padrão). O diretório de execução contém ainda um diretório denominado *cm* onde são armazenados os *anúncios* criados ou recebidos da rede JXTA. Cada *Peer* JXTA deve dispor de um diretório de execução próprio.

4.1.3 Grupos (*peergroups*)

Grupos permitem aos *Peers* se auto-organizarem dinamicamente em domínios virtuais. Os *grupos* agregam *Peers* com interesses em comum tais como a execução distribuída de algum serviço ou o compartilhamento de dados de forma segura, por exemplo. Todo *Peer* é parte integrante de pelo menos um *grupo*, mas vale ressaltar que um *Peer* pode fazer, e geralmente é, parte de mais de um *grupo* ao mesmo tempo.

Existem três motivações básicas para a criação de *grupos*:

1. Criar domínios seguros para a troca de conteúdo. Mecanismos de autenticação centralizada ou distribuída podem ser impostos de forma a criar regiões lógicas protegidas contra acesso não autorizado. Os *grupos* criam um escopo de interesse, pois membros de um determinado *grupo* supostamente compartilham algum interesse em comum, como o compartilhamento de dados ou a execução de um serviço de forma distribuída.

2. Criar um escopo para a comunicação. Um número potencialmente grande de *Peers* que podem fazer parte de uma rede JXTA exige algum mecanismo que mantenha a propagação de mensagens e consultas na rede escalável. O *grupo* atua como este mecanismo, pois cada mensagem ou consulta deve necessariamente trafegar dentro de algum *grupo*. Além disso, não são permitidas as trocas de mensagens *inter-grupos*.

3. Criar um ambiente de monitoramento. O ambiente de *grupo* permite o monitoramento de *Peers* com vistas a medição de desempenho ou segurança.

Ao ser iniciado, todo *Peer* JXTA instancia e torna-se automaticamente membro de dois *grupos*: **NetPeerGroup** e **WorldPeerGroup** [47]. Os *grupos* em JXTA formam uma hierárquia *pai-filho* sendo o NetPeerGroup um subgrupo de WorldPeerGroup, que é o *grupo* inicial. Apesar do nome, WorldPeerGroup não tem por objetivo interligar todos os *Peers* existentes na Internet, mas prover mecanismos de comunicação mínimos para a plataforma JXTA, tais como protocolos de transporte, comunicação e *multicasting* em rede local. Cada *grupo* possui um conjunto de serviços (*PeerGroup Services*). Estes serviços permitem realizar as operações comumente encontradas em redes JXTA tais como descoberta de *anúncios*, comunicação através de pipes ou propagação de mensagens em *Rendezvous*. O NetPeerGroup serve ao propósito de implementar serviços básicos da plataforma JXTA, a saber:

1. Discovery Service : O serviço de descoberta é utilizado pelos *Peers* para realizar buscas por recursos do *grupo* tais como *Peers*, *subgrupos*, *pipes*, e serviços.

2. Membership Service : O serviço de acesso é usado pelos *Peers* para permitir ou negar acesso de *Peers* ao *grupo*. A plataforma JXTA não impõem algoritmos ou protocolos específicos de autenticação e acesso, logo cabe ao projetista da aplicação permitir o livre acesso de *Peers* ao *grupo*, ou criar mecanismos centralizados ou distribuídos de autenticação e acesso.

3. Pipe Service: o serviço de pipe é usado para criar, destruir conexões através de *pipes* entre os membros do *grupo*.

4. Resolver Service: o serviço de resolução permite realizar consultas genéricas e coletar respostas a estas consultas.

Na Figura 4.3 temos um exemplo hipotético da organização de *grupos* em uma rede JXTA. Conforme dito anteriormente, um *Peer* pode fazer parte de mais de um *grupo* ao mesmo e, graças a isto, podem existir *Peers* que façam parte de **FileTransferGroup** e **CafeGroup**, ou ainda *Peers* que façam parte de todos os *Grupos* mostrados. Inicialmente o NetPeerGroup conecta seus *Peers* a rede pública, mas a configuração do NetPeerGroup pode ser reescrita para se construir uma rede privada semelhante aquela utiliza neste trabalho.

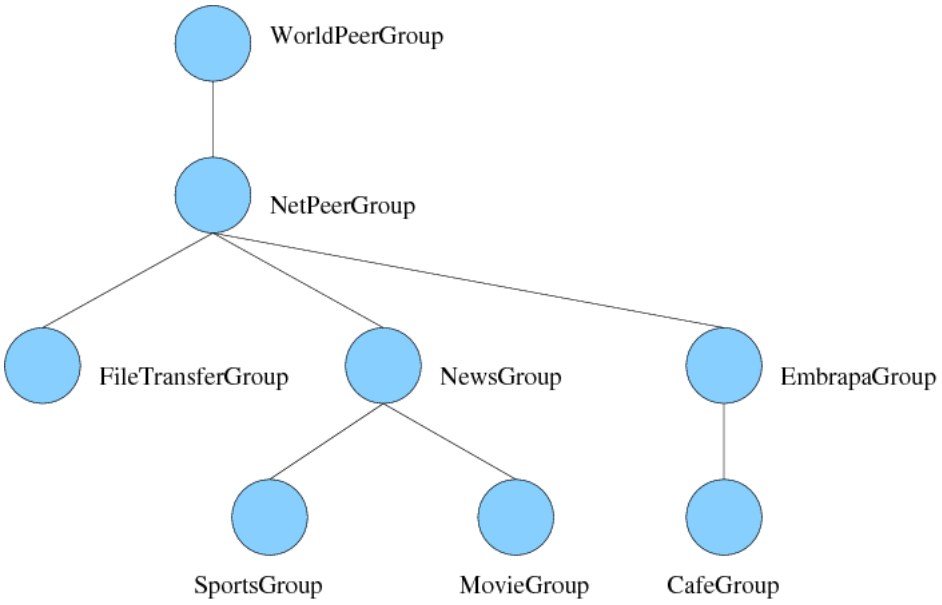

Figura 4.3: Hierárquia de *Grupos* na Rede JXTA

4.1.4 Anúncios (advertisements)

Todos os recursos de uma rede JXTA são representados por *anúncios*. *Anúncios* são documentos XML que são utilizados para descrever *Peers*, *grupos*, *pipes*, serviços, rotas, conteúdo, *Rendezvous*, *Relays*, *endpoints*, e mecanismo de transporte. Portanto, a publicação e descoberta de quaisquer recursos na rede JXTA é simplicada como a publicação, e descoberta de arquivos XML denominados *anúncios*. É possível criar *anúncios* customizados para as necessidades da aplicação, conforme realizado neste trabalho. A Figura 4.4 detalha um *anúncio de grupo*. A *tag* `GID` identifica de forma única o *grupo*, `Name` descreve o nome, e `Desc` é uma tag opcional que especifica alguma informação adicional acerca do *grupo*.

```
<?xml version=''1.0''?>
<!DOCTYPE jxta:PGA>
<jxta:PGA xmlns:jxta=''http://jxta.org''>
    <GID>urn:jxta:jxta-NetGroup</GID>
    <MSID>urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010206</MSID>
    <Name>NetPeerGroup</Name>
    <Desc>NetPeerGroup by default</Desc>
</jxta:PGA>
```

Figura 4.4: *Anúncio de grupo (PeerGroup Advertisement)*

4.1.5 *Pipes*

Pipes são canais de comunicação virtual que possibilitam a comunicação e troca de dados entre dois ou mais *Peers*. A princípio um *pipe* não pertence a nenhum *Peer* em especial, e o *pipe* liga dois ou mais *endpoints* de forma dinâmica, isto é, em tempo de execução. O fato de ser unidirecional faz com que o *pipe* tenha dois *Peers* de conexão: **pipe de entrada** (input pipe) e **pipe de saída** (output pipe). Uma mensagem sempre é enviada de um *pipe de saída* para um *pipe de entrada* de forma unidirecional, asíncrona e não confiável.

Os pipes podem ser classificados em *Unicast* e *Propagate pipes*. O unicast pipe é o modelo tradicional de comunicação com *pipes*. Muito embora os pipes sejam implementados utilizando-se o protocolo TCP, que é confiável e orientado a conexão, os pipes não assumem este protocolo como base de sua implementação. Por esta razão, uma mensagem que trafega pelo pipe não terá garantia alguma de chegar a seu destino (não confiável) e muito menos será orientado a conexão.

- Unicast Pipe - Os pipes unicast permite uma comunicação um-pra-um entre dois *Peers*. Uma mensagem é enviada de um pipe de saída em direção a um pipe de entrada de forma unidirecional (Figura 4.5).

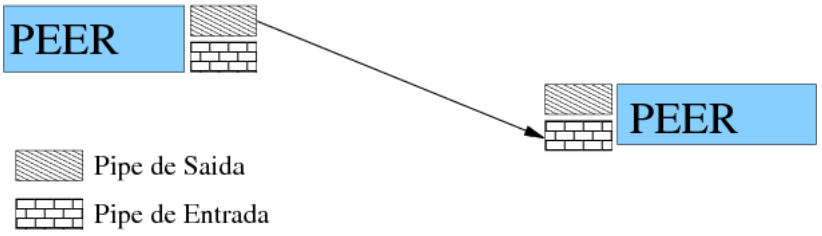

Figura 4.5: Unicast Pipe

- Propagate Pipe - Os pipes propagate permitem uma comunicação um-para-muitos entre *Peers*. Uma mensagem é enviada a partir de um pipe de saída em direção a um ou mais pipes de entrada (*multicasting*). De forma semelhante ao modelo *unicast*, este *pipe* não é confiável e permite a comunicação em um sentido apenas (Figura 4.6).

O mecanismo de comunicação entre dois *Peers* divide-se em quatro etapas:

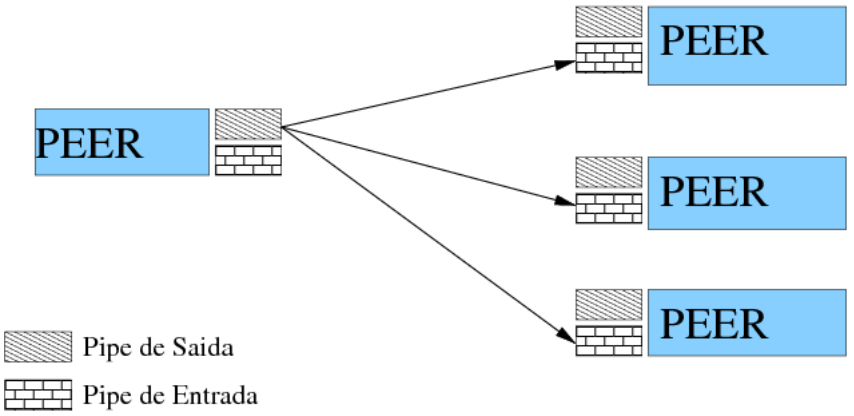

Figura 4.6: Propagate Pipe

1. Criação ou descoberta do *pipe* - Cada pipe é descrito por um *anúncio de pipe* (Figura 4.7), que o identifica de forma única na rede JXTA. Uma vez criado o *anúncio de pipe*, este deve ser publicado na rede para que outros *Peers* sejam capazes de descobrí-lo e utilizá-lo para estabelecer conexões. Um *pipe* pode ser bem conhecido, isto é, ter sido criado anteriormente e salvo em arquivo texto, que é importado em tempo de execução, ou ser criado mediante um identificador definido previamente. Neste caso, não se faz necessário publicar e descobrir o *pipe*, o que pode acelerar consideravelmente a execução de transferências de dados.

```
<!DOCTYPE jxta:PipeAdvertisement>
<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
<Id>
  urn:jxta:uuid-59616261646162614A757874614D504725184FBC4E5D498
  AA0919F662E40028B04
</Id>
<Type>
  JxtaUnicast
</Type>
<Name>
  PipeExample
</Name>
</jxta:PipeAdvertisement>
```

Figura 4.7: *Pipe Advertisement*

2. Resolução dos *endpoints* do *pipe* - Ao iniciar o processo de conexão, o *pipe* deve ter sua entrada e saída mapeados para *endpoints* de dois ou mais *Peers*. Este processo faz uso do *Pipe Binding Protocol* para descobrir pelo menos um *Peer* que esteja escutando no *pipe de entrada*. Em caso de sucesso, ao *Peer* que iniciou a descoberta será associado o *pipe de saída* e a conexão é estabelecida. Se o *pipe* de entrada não for resolvido em tempo especificado pelo usuário então a plataforma Jxta sinaliza com uma mensagem de erro.

3. Transferência da mensagem - Uma ou mais mensagens são criadas e transferidas pelo *pipe*. Cada mensagem deve ter seu tamanho limitado em 64 KB na implementação atual, e para enviar mensagens maiores que este tamanho é necessário que a aplicação se responsabilize pelo processo de fragmentação do dado original na origem, envio das mensagens, e montagem dos fragmentos no destino. Se uma aplicação tentar enviar uma mensagem maior que 64 KB através de um *Rendezvous* ou *Relay*, este poderá vir a descartá-la. JxtaSocket é uma alternativa que permite o envio de mensagens de qualquer tamanho, pois este serviço se encarrega da partição das mensagens e adiciona uma camada de confiabilidade para fazer a confirmação de chegada das mensagens e ordenação das mesmas, além de oferecer uma interface de programação semelhante aos sockets comuns.

4. Fechamento do *pipe* - O pipe que foi utilizado é desalocado.

Mecanismos de comunicação segura, bidirecional e confiáveis foram implementados em cima de pipes unicast tradicionais. São exemplos destes mecanismos:

1. JxtaBidirectional - este serviço faz uso de dois *pipes unicast* para estabelecer uma comunicação bidirecional, mas neste caso o limite das mensagens ainda é 64 KB.

2. JxtaSocket - utiliza *pipes unicast* para criar uma interface Socket que permite a comunicação orientada a conexão e confiável, além de transferir mensagens de qualquer tamanho.

3. JxtaUnicastSecure - versão de *unicast pipe* que utiliza comunicações criptografadas.

4.2 Super-Nós

Os *Peers* de uma rede JXTA podem ser basicamente classificados em *peers de fronteira* (edge peers), *Rendezvous* ou *Relays*. Os *Rendezvous* e *Relays* são categorizados como *Super-Nós*, *Peers* que atuam como agregadores dos *Peers* de fronteira. Os *Peers* de fronteira recebem este nome por atuarem na "fronteira"da Internet, isto é, são máquinas não dedicadas, conectadas a redes de baixa velocidade ou cuja conexão se dá de forma transiente e/ou instável. Além da dinamicidade inerente às redes P2P, uma das grandes vantagens das redes P2P advém justamente da possibilidade de se utilizar o poder de processamento destas máquinas para resolver problemas complexos.

Rendezvous e *Relays*, por sua vez, fazem parte da categoria de *super-nós*. Estes *Peers* são responsáveis por tarefas específicas dentro de uma rede JXTA, que geralmente requerem acesso direto através da Internet, uma maior capacidade de processamento ou espaço de armazenamento.

Os *Rendezvous* (Figura 4.8) atuam como roteadores de mensagens e agregadores de *Peers* dentro da rede JXTA. São os *Rendezvous* que permitem que os *Peers* em sub-redes distintas façam parte do mesmo *grupo* e troquem mensagens. Para que um anúncio seja amplamente propagado pela rede JXTA é necessário

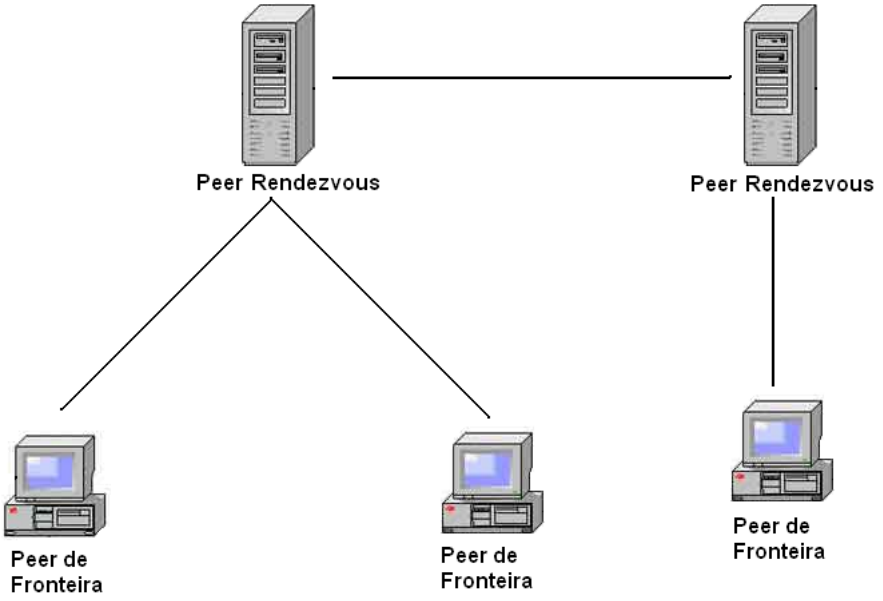

Figura 4.8: *Peer* Rendezvous

que cada *Peer* esteja conectado a um Rendezvous. Os *Rendezvous* são ainda responsáveis por armazenar em cache os *índices* dos *anúncios* publicados por outros *Peers* e permitem a localização de recursos publicados na rede JXTA utilizando algoritmos de propagação de mensagens. Em termos de implementação, são os *Rendezvous* que implementam o conceito de *grupo* na rede JXTA.

Os *Relays*, por sua vez, são *Peers* que atuam fora de qualquer Firewall ou NAT, isto é, são diretamente acessíveis por todos os outros *Peers*, e cujo propósito é permitir que *Peers* que estejam protegidos por firewalls possam fazer parte da rede JXTA. O *Relay* atua como um intermediário na comunicação entre os *Peers*, repassando as mensagens entre dois *Peers* conforme mostra a Figura 4.9.

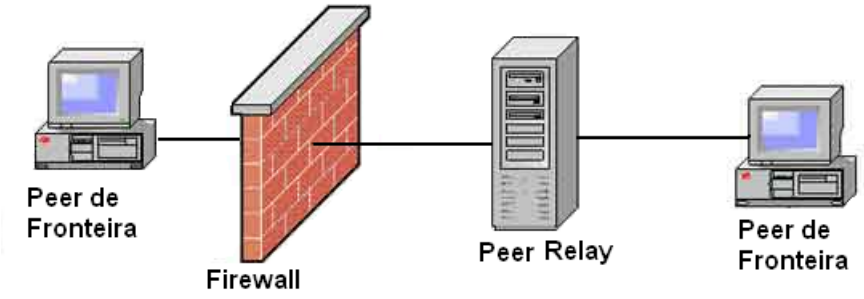

Figura 4.9: *Peer* Relay

Cada *Peer* possui uma lista de *Relays* preferenciais e decide, em tempo de execução, qual deles utilizar para se conectar com *Peers* fora do Firewall. Esta comunicação é feita utilizando-se um protocolo que permita atravessar o Firewall, usualmente o protocolo HTTP. Inicialmente, um *Peer* dentro do firewall envia uma mensagem para algum outro *Peer*, fora do Firewall, encapsulada em uma requisição HTTP. A partir de então, a cada intervalo de tempo pré-determinado,

o *Peer* dentro do Firewall envia uma requisição HTTP para o *Relay* consultando acerca de alguma mensagem direcionada a ele. Caso alguma mensagem esteja direcionada a este *Peer* então a mensagem será no corpo da resposta HTTP.

4.3 Protocolos

Conforme mencionado, a plataforma JXTA é composta por um conjunto de seis protocolos XML que padronizam as comunicações entre os *Peers*. Estes protocolos foram divididos em duas categorias: *Protocolos de Especificação Básica* e *Protocolos de Serviços Padrão* (Figura 4.10)

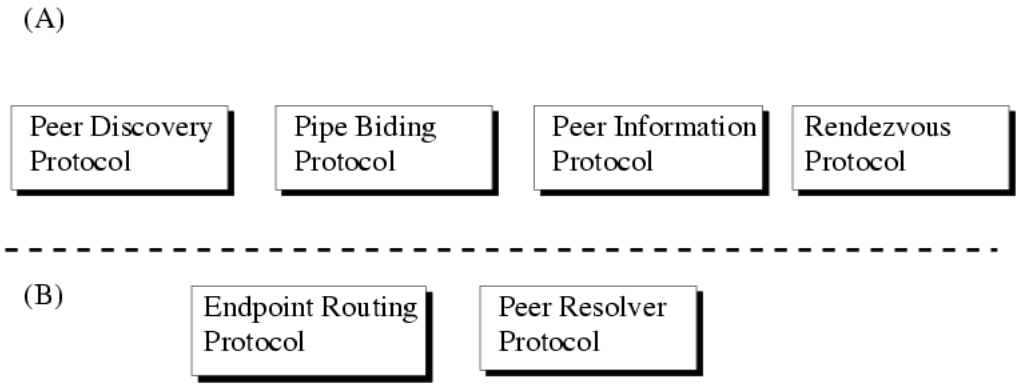

Figura 4.10: Protocolos JXTA. (A) Protocolos de Serviços Padrão e(B) Protocolos de Especificação Básica

Os Protocolos Especificação Básica são divididos em:

- Endpoint Routing Protocol (ERP) é um protocolo pelo qual o *Peer* pode descobrir uma rota (seqüência de *Peers* intermediários) usados para enviar uma mensagem de um *Peer* para outro. Neste caso um *Peer* A pode fazer uso de um ou mais *Peers* intermediários para enviar uma mensagem para um *Peer* B. O ERP é responsável por gerenciar e determinar a informação de roteamento. Vale ressaltar que esta informação é dinâmica, isto é, se uma rota não puder mais ser utilizada então a plataforma JXTA se encarregará de descobrir rotas alternativas para o envio da mensagem.

- Peer Resolver Protocol (PRP) é um protocolo que permite a um *Peer* enviar uma consulta genérica para um ou mais *Peers*, e receber uma (ou múltiplas) respostas para a consulta. Um *Peer* A envia uma consulta para os outros *Peers* de um determinado grupo. Esta consulta é endereçada a algum tratador (handler) específico. Este tratador é responsável por receber a consulta, interpretá-la e, possivelmente, enviar uma resposta. Se o *Peer* que recebe a consulta possui o tratador registrado, o PRP repassa a consulta para o tratador. Caso contrário, a mensagem é descartada.

Os Protocolos de Serviço Padrão, por sua vez, são divididos em:

- Rendezvous Protocol (RVP) é o protocolo que permite aos *Peers* propagar mensagens entre rendezvous. RVP é usado pelo PRP para propagar mensagens além de uma rede local.

- Peer Discovery Protocol (PDP) é o protocolo pelo qual um *Peer* publica seus anúncios, e descobre anúncios publicados por outros *Peers* na rede. PDP faz uso do PRP para envio e propagação de requisições de descoberta de anúncios.

- Peer Information Protocol (PIP) é o protocolo utilizado por um *Peer* para obter informações de estatus sobre outros *Peers*, tais como estado, tráfego, capacidades, etc. PIP usa PRP para enviar e receber requisições acerca de informações sobre *Peers*.

- Pipe Biding Protocol (PBP) é o protocolo que permite ao *Peer* estabelecer o canal de comunicação virtual, ou pipe, entre um ou mais *Peers*. O PBP é usado por um *Peer* para ligar dois ou mais pipe ends de uma conexão (input e output pipe) para um endpoint físico. PBP usa PRP para envio e recepção de requisições de resolução de pipes.

4.4 Busca e Roteamento em JXTA

Vimos que todas as operações de descoberta de recursos em JXTA são unificadas em um único processo de descoberta de um ou mais *anúncios*. Os protocolos do projeto JXTA não especificam como tais buscas devem ser realizadas, mas as implementações de referência provêem um mecanismo padrão de resolução baseado em *Super-Nós* do tipo *Rendezvous*.

Conforme visto anteriormente, *Rendezvous* são *Peers* bem conhecidos encarregados de armazenar índices de *anúncios* e localizar *anúncios*. Vale ressaltar que qualquer *Peer* pode se tornar um *Rendezvous* desde que possua os requisitos necessários tais como espaço em disco, autorização para tornar-se um *Rendezvous*, ou ser acessível a outros *Peers* na rede, por exemplo.

A partir da versão 2.0 da plataforma JXTA, os *Rendezvous* passaram a armazenar somente os índices dos *anúncios* publicados por seus *Peers*, pois isto economiza espaço em disco, torna a arquitetura de *Rendezvous* mais escalável, e evita o tratamento de problemas tais como a manutenção de *anúncios* desatualizados.

Para indexar seus *anúncios* no *Rendezvous*, um *Peer* de fronteira utiliza o serviço **Shared Resource Distributed Index (SRDI)**. Usando este serviço, os *Peers* de fronteira enviam índices de *anúncios* para seus Rendezvous ao publicar novos *anúncios*. Os índices podem ser enviados sincronamente quando um novo *anúncio* é publicado, ou assíncronamente pelo daemon SRDI que é acionado a intervalos regulares.

Na Figura 4.11 temos um exemplo de publicação e recuperação de *anúncios* através de SRDI. Neste exemplo, temos:

1. *Peers* A e B, dentre outros, enviam os índices de seus *anúncios* (ADV) para os rendevous RDV1 e RDV2, respectivamente;

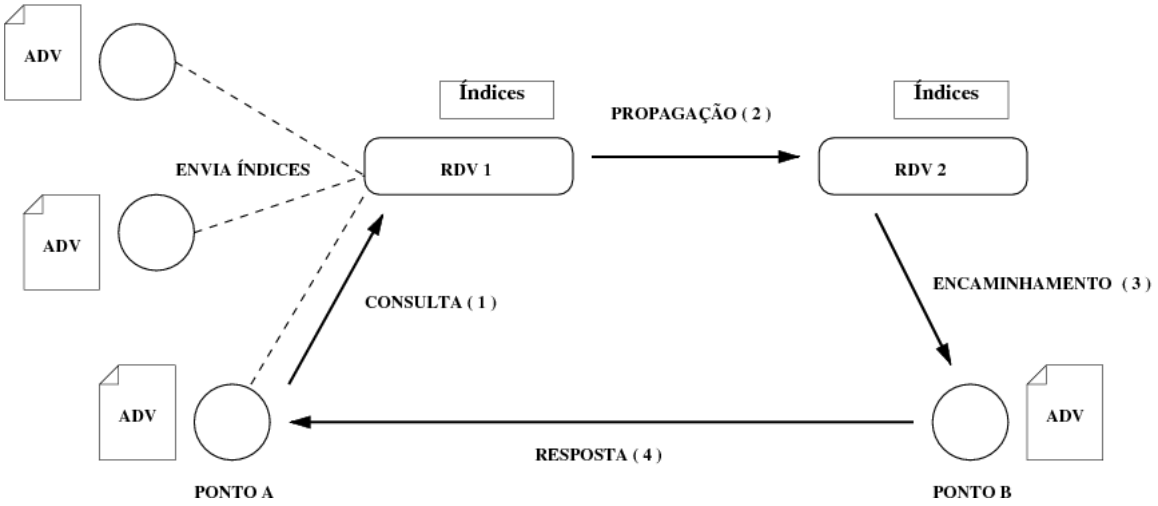

Figura 4.11: Publicação e Recuperação de *anúncios* através de SRDI

2. O *Peer* A envia uma consulta a RDV1, que verifica se o índice deste *anúncio* está presente em seu cache local (Passo 1);

3. Se RDV1 não encontra o índice então propaga a consulta para o próximo rendezvous, RDV2 (Passo 2);

4. RDV2 encontra o índice do *anúncio* procurado e encaminha a consulta para o *Peer* B, que é responsável pelo *anúncio*;

5. O *Peer* B recebe a consulta e envia o *anúncio* para o *Peer* A;

Outra importante melhoria em relação a versões anteriores da plataforma JXTA diz respeito a propagação de consultas do Passo 2, que somente são propagadas entre *Rendezvous*, com vistas a reduzir o tráfego de rede. Uma consulta somente é direcionada a um *Peer* de fronteira quando o *Peer* é responsável por um *anúncio* que está sendo procurado.

A próxima seção detalha o mecanismo de propagação de mensagens entre *Rendezvous*.

4.4.1 DHT e Rendezvous Walker

Os Rendezvous se auto-organizam em uma rede fracamente acoplada. Isto se deve ao fato de que em uma rede com constante entrada e saída de *Peers*, torna-se difícil manter uma visão consistente acerca de todos os *Peers* presentes em um dado instante sem impor limitações na quantidade de *Peers* em um *grupo* e sem a adoção de algum mecanismo centralizado de controle.

Em uma rede altamente flutuante e ambiente imprevisível, o custo de se manter um índice distribuído consistente pode superar as vantagens de se adotar tal índice. É possível que se gaste mais tempo atualizando índices que realizando consultas (*index trashing*). Nós podemos separar o custo de uma DHT entre manutenção de índice e consulta ao índice. A abordagem de DHT provê o mecanismo de consulta ao índice mais eficiente ($O(log(N))$), onde N é o número

de *Peers*. Entretanto, o custo de manutenção tipicamente cresce de forma exponencial a medida que a entrada/saída de *Peers* aumenta. Por outro lado, não ter uma DHT significa que operações exaustivas de busca precisam ser realizadas ocasionando tráfegos de rede ainda piores.

O projeto JXTA 2.0 propõe uma abordagem híbrida que combina o uso de uma DHT fracamente consistente com um *andarilho rendezvous de extensão limitada* (limited range rendezvous walker). *Rendezvous* não são obrigados a manter um índice consistente entre todos os *Peers*, originando o termo fracamente consistente. Se a entrada/saída de *Peers* é baixa então o RPV mantêm-se estável, e a DHT tende a atingir a consistência entre todos os *Peers* e, consequentemente, performance ótima.

Cada *Rendezvous* possui uma lista, ordenada por ID, de rendezvous conhecidos, chamada *Rendezvous Peer View* (RPV). Na figura 4.12 temos a RPV representada como uma tabela e como um círculo DHT.

Rendezvous podem ter RPVs inconsistentes de forma temporária ou permanente. No exemplo da Figura 4.13, a RPV do Rendezvous *R2* contém referências dos *Peers* R1, R2, R3, R5 e R6. O Rendezvous *R3*, por sua vez, contém uma RPV com referências aos *Peers* R3, R4 e R5. A RPV é montada de forma fracamente consistente, pois, devido à constante entrada e saída de *Peers* da rede JXTA, a manutenção de uma visão consistente tornaria o algoritmo lento e complexo. É utilizado um algoritmo para convergir a RPV entre todos os *Rendezvous*.

NOME	IDENTIFICADOR
R1	10223040
R2	15001877
R3	21954297
R4	50923430
R5	88000143
R6	90129321

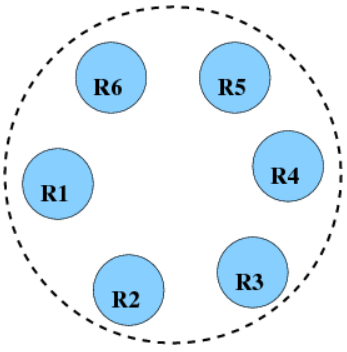

Figura 4.12: Rendezvous Peer View (RPV) representada como uma tabela e um círculo DHT, respectivamente.

Periodicamente, o *Rendezvous* seleciona um número aleatório de *Rendezvous* de sua RPV local, e envia a estes outra lista aleatória de *Rendezvous* conhecidos. *Rendezvous* também enviam um sinal de vida (*heartbeat*) para seus *Rendezvous* vizinhos (as posições +1 e -1 na RPV). *Rendezvous* atualizam e expurgam *Rendezvous* que não respondem de suas RPVs. Além disso, *Rendezvous* pode recuperar informação acerca de *Rendezvous* de um grupo de *Rendezvous* sementes (*seeding*). *Rendezvous* sementes permitem acelerar a convergência da RPV, a medida que todos os *Rendezvous* são pré-configurados com uma lista de *Rendezvous* sementes. Qualquer *Rendezvous* pode agir como *Rendezvous* semente. *Rendezvous* sementes são utilizados como o último recurso quando um *Rendezvous*

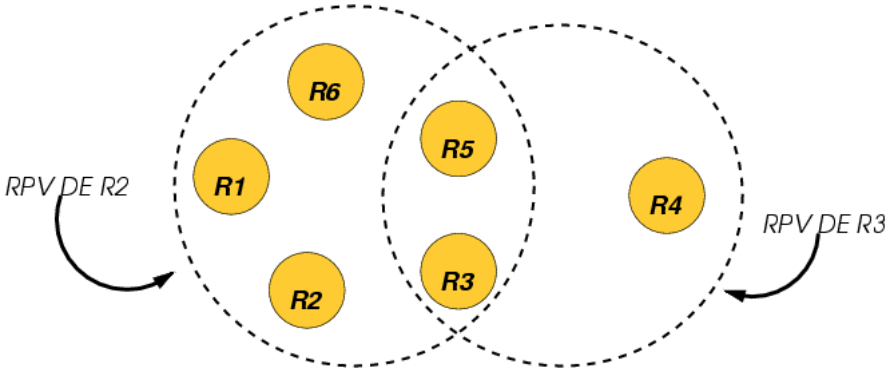

Figura 4.13: RPVs inconsistentes entre os rendezvous R2 e R3

não pode encontrar qualquer outro *Rendezvous*, e para colocar o *Rendezvous* inicialmente na rede. Uma vez feito isso, e após um *Rendezvous* ter aprendido o suficiente sobre outros *Rendezvous*, *Rendezvous* sementes são tratados como qualquer outro *Rendezvous*. Se uma rede de *Rendezvous* é relativamente estável, RPVs convergem rapidamente em todos os *Rendezvous* permitindo distribuição consistente de índices.

Peers de fronteira primeiramente checam em seu cache local por quaisquer *anúncios* que apontem para *Rendezvous* acessíveis. Os *Peers* ordenam os *Rendezvous* candidatos em um lote de cinco, e tentam conectar-se simultâneamente a estes *Rendezvous* até que uma conexão seja estabelecida. Um *Peer* de fronteira mantém apenas uma conexão ativa com um *Rendezvous*. Se uma conexão a *Rendezvous* não é estabelecida após um período configurável (40 segundos), o *Peer* de fronteira tenta procurar por *Rendezvous* através da propagação de requisições utilizando protocolos de transporte (IP multicast), ou através de um pipe propagate se o *Peer* de fronteira entrou em um *grupo*. Se após um período maior (30 segundos) nenhum *Rendezvous* tiver sido descoberto, o *Peer* de fronteira irá consultar um dos *Rendezvous* semente. Finalmente, se nenhum *Rendezvous* semente é alcançavel após um período configurável (5 minutos), o *Peer* de fronteira tentará tornar-se um *Rendezvous* (se tiver credenciais para isto). O *Rendezvous* pode voltar a ser um *Peer* de fronteira tão logo um *Rendezvous* seja descoberto. Mesmo tendo se tornado *Rendezvous*, *Peer* de fronteira continua a buscar por *Rendezvous* em background.

Particionamento da RPV pode ocorrer se um conjunto de *Rendezvous* não puder ser contactado. Entretanto, partições RPV irão se fundir tão logo cada partição alcançe um dos *Rendezvous* semente, ou um *Rendezvous* em comum exista em ambas as partições. Inconsistências serão resolvidas com o tempo, pois os *Rendezvous* continuam a trocar informações aleatórias.

O *Peer* P1 publica um novo *anúncio* em seu *Rendezvous* R2 via o previamente mencionado SRDI. Cada *anúncio* é indexado pelo SRDI usando chaves previamente definidas tais como o nome, ou ID do Anúncio. R2 usa a função DHT (H(adv1)) para mapear o índice para um *Rendezvous* em sua RPV local. A RPV contém os rendezvous de R1 a R6. Suponha que a função DHT retorne R5, então este índice será enviado para R5 (Figura 4.14). Para aumentar a probabilidade de recuperar o índice na falha de R5, o índice é replicado para os vizinhos de R5

na RPV (+1 e -1 na lista ordenada da RPV).

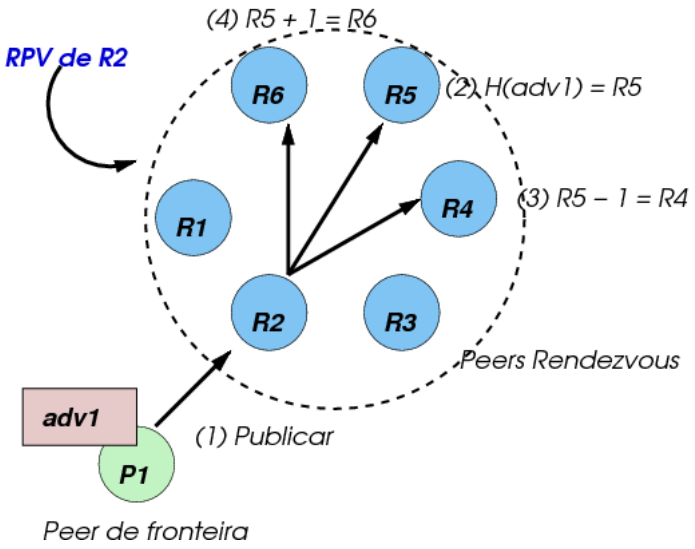

Figura 4.14: Publicação de um *anúncio* na rede JXTA

Em nosso exemplo, o índice é replicado em R4 e R6. Replicar o índice ao redor de R5 significa que estamos criando uma região lógica onde o índice pode ser localizado. Vamos assumir que um *Peer* de fronteira P2 está procurando pelo anúncio adv1 (Figura 4.15). P2 irá enviar uma consulta a seu *Rendezvous* R3. O serviço SRDI em R3 irá computar a função DHT (H(adv1)) usando a RPV local de R3. Se o RPV em R2 e R3 forem os mesmos, a DHT retornará o mesmo rendezvous R5. R3 irá encaminhar a requisição para R5 que irá encaminhar para P1 para que este responda a P2. Isto irá funcionar caso as RPV de R2 e R3 sejam as mesmas.

Suponha agora que R5 saia da rede, e R3 atualize sua RPV para refletir este fato (Figura 4.16). R3 irá descobrir que que R5 não está mais na rede através de uma atualização da RPV, ou caso ele tente contactá-lo para enviar a mensagem. Neste cenário, R3 tem uma nova RPV que contém *Rendezvous* de R1 a R5. R5 agora aponta para o *Rendezvous* R6 do RPV anterior. O desparecimento de um *Rendezvous* significa que a RPV deslocou-se de uma posição. Ao receber a requisição de P2, R3 irá computar a função DHT que irá retornar o *Rendezvous* R5. Caso o deslocamento da RPV esteja dentro da distância de replicação da DHT (+1,-1), é possível garantir que o índice será encontrado. A distância de replicação pode ser aumentada (+2 ou +3) para RPVs maiores.

Agora vamos analisar um caso mais caótico onde a RPV sofreu mudanças drásticas (Figura 4.17). O RPV é agora composta por 8 *Rendezvous* - R1 a R8. R7 corresponde ao R4 original. Quando R3 recebe uma consulta de P2 ele calcula a função DHT que mapeia o índice para R5. Uma vez que RPV mudou bastante, o índice não será encontrado em R5. Neste caso, um mecanismo alternativo é usado para "caminhar"(walk) a RPV de forma a continuar a busca. Um *andarilho de extensão limitada* (limited-range walker) é usado para caminhar

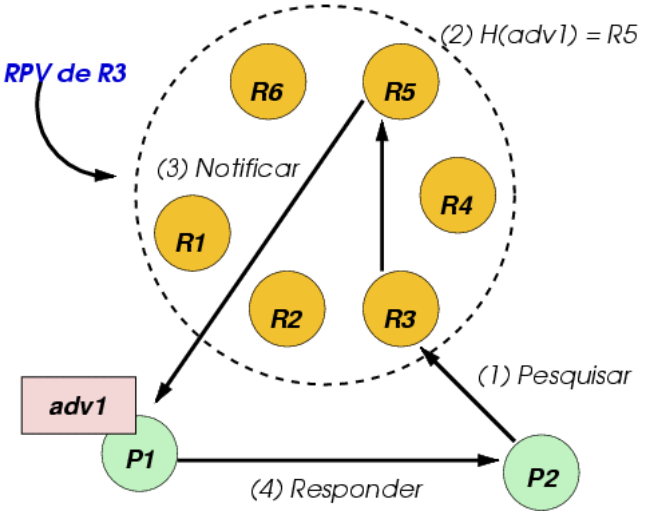

Figura 4.15: Busca de um *anúncio* na rede JXTA

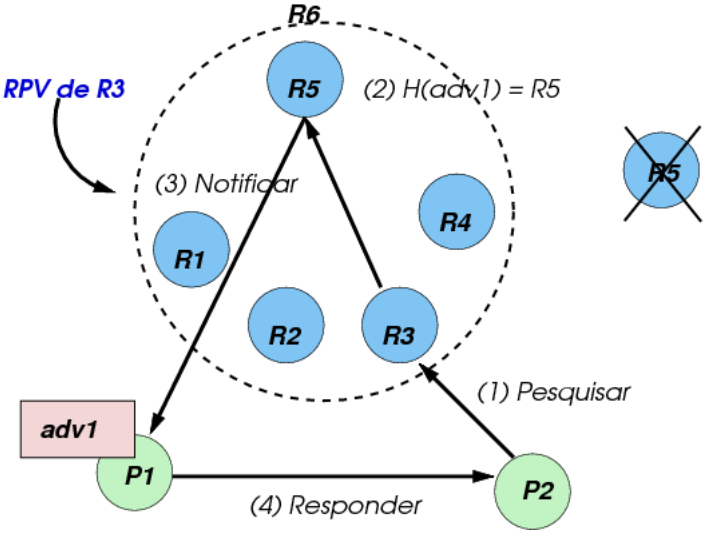

Figura 4.16: Busca de um *anúncio* em uma RPV ligeiramente modificada

a *Rendezvous* a partir da posição inicial calculada pela DHT. O *andarilho* irá proceder em ambas as direções (para cima e para baixo) (Figura 4.4.1) na direção de R6 e para baixo na direção de R4. O *andarilho de extensão limitada* pressupõe que existe uma grande probabilidade de descobrir o índice esteja em uma região devido ao seu esquema de replicação de vizinhos da DHT. Em nosso exemplo, R5 encaminha a requisição tanto para R4 quanto para R6. Um contador de pulos é usado para especificar a quantidade máxima de vezes que a requisição pode ser repassada. Se R4 não tiver o índice, ele irá encaminhar a requisição para o próximo *Peer* na direção de baixo de R3. De forma análoga, R6 irá encaminhar a requisição para a direção superior R7. Quando o índice é descoberto em R7, a consulta é encaminhada para P1 e a pesquisa é interrompida na direção superior. O caminho na direção inferior irá continuar até a quantidade máxima de pulos ser atingida, ou não existir mais *Rendezvous* naquela direção. Ao invés de continuar o caminhar até o contador de pulos ser atingido, uma requisição de confirmação de continuação pode ser enviada para P2 após uma porção do caminho ter sido percorrido para perguntar a P2 se a o caminhar deve continuar. P2 pode ter atualmente encontrado uma resposta do outro lado do caminho. P2 pode decidir continuar ou interromper o caminho.

Para minimizar a degeneração do índice quando a RPV aumenta ou diminui, ou está temporariamente inconsistente, a função hash utilizada tenta compensar as mudanças na RPV. A função DHT utilizada divide o espaço hash igualmente pelo número de rendezvous na RPV. Cada rendezvous recebe um segmento do espaço da função hash de acordo com sua posição na RPV. Se o número hash cai no meio do espaço hash então o rendezvous selecionado irá estar no meio da RPV. A função hash permite escalonar o espaço hash para o tamanho da RPV. Mesmo se a RPV mudar drasticamente entre o tempo que o índice foi inserido, e o tempo em que é consultada, existe a chance que as mudanças na RPV tenham sido tanto para a direita quanto para a esquerda de um dado rendezvous. A posição relativa de um rendezvous tende a permanecer aproximadamente a mesma a medida que a RPV evolui. Utilizar o ID do peer para ordenar a RPV permite uma distribuição uniforme através da RPV. Se uma RPV aumenta ou diminui, um *Rendezvous* que estava no meio da RPV tende a permanecer próximo ao meio.

4.5 Performance

Jan e co-autores [48] realizaram uma série de testes para avaliar a performance das duas principais implementações JXTA existentes: JXTA-J2SE e JXTA-C. Conforme apontam os autores, existem vários trabalhos na literatura que expõem a sobrecarga de comunicação introduzida pelo JXTA e dificuldade de uso da API. Parker e co-autores [49], por exemplo, mostraram que o interpretador XML utilizado na versão JXTA J2SE tem baixa performance, inferior a de interpretadores XML de código fonte aberto existentes, o que torna a plataforma inapropriada para a construção de aplicações de tempo crítico. Este estudo mostra que a adoção de analisadores XML otimizados terá um impacto significativo na performance.

Entretanto, a maioria dos estudos de performance realizados utilizam versões antigas (1.0 ou anteriores), e não otimizadas, da plataforma JXTA. Além disso,

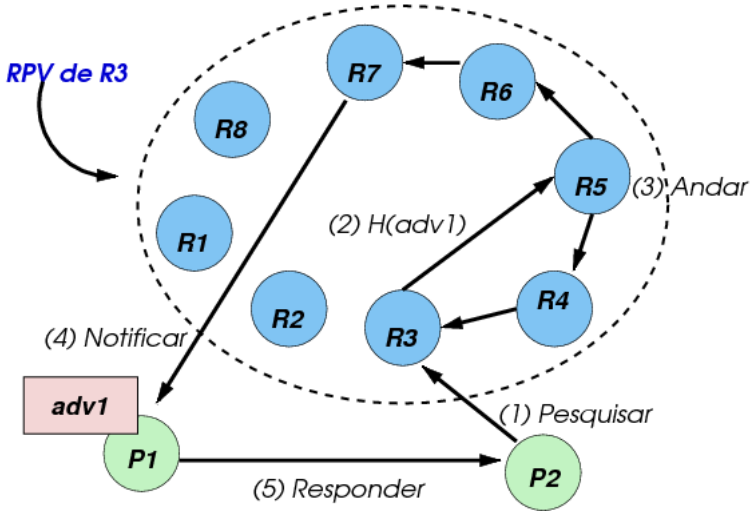

Figura 4.17: Busca de um *anúncio* em uma RPV severamente modificada

os estudos anteriores foram não conclusivos, e até mesmo inconsistentes entre si, o que torna difícil obter uma visão clara da performance das camadas de comunicação JXTA.

Para avaliar o custo de comunicações em JXTA foram realizados uma série de testes de taxa de transmissão bidirecionais (*ping pong tests*) entre dois *Peers* JXTA. Os testes foram realizados em uma rede local Ethernet, utilizando as camadas de comunicação disponíveis nas duas versões do JXTA, e variando parâmetros experimentais tais como tamanho da mensagem, tamanho do buffer, e opções da JVM.

Muito embora estes testes permitam expor os ganhos (ou perdas) de performance inerentes na comunicação direta entre dois *Peers*, vale ressaltar que a comunicação direta entre dois *Peers* constitui-se mais a exceção que a regra pois em geral existem intermediários na comunicação.

No trabalho de Jan e co-autores [48], foram analisados os seguintes mecanismos de comunicação JXTA:

1. Endpoint service
2. Pipe service
3. JxtaSockets

Conforme mostrado na Figura 4.18 cada camada de transporte é construida sobre a camada anterior, sendo o *endpoint service* a camada de mais baixo nível.

O teste de largura de banda bidirecional foi escolhido por ser uma métrica de performance básica, um teste freqüentemente utilizado para medida de performance de outros protocolos de rede, e por conta de sua habilidade para levantar informações sobre características importantes tais como largura de banda e latência. Este teste consiste na troca de mensagens idênticas entre dois *Peers*

e nos dois sentidos. Cada teste é composto de medidas sucessivas obtidas sobre uma série de tamanhos de mensagens (*payloads*) que variam de 1 byte até 16 MB. Todas as medidas são amostradas no nível de aplicação e são calculadas mediante cinco medidas subsequentes de tempo de envio-resposta de 100 mensagens consecutivas.

A eficiência do protocolo é outro fator que foi explorado na avaliação de performance dos protocolos JXTA. A eficiência do protocolo é definida como a média entre a quantidade de dados que um usuário deseja enviar e a quantidade total de dados que realmente foi enviada pelo protocolo. Portanto, qualquer dado adicional incluido na transmissão da da mensagem irá reduzir a eficiência do protocolo e pode reduzir a performance.

Este resultados são obtidos mediante a análise de mensagens entre *Peers* através do uso de dois analisadores de protocolo: tcpdump e ethereal.

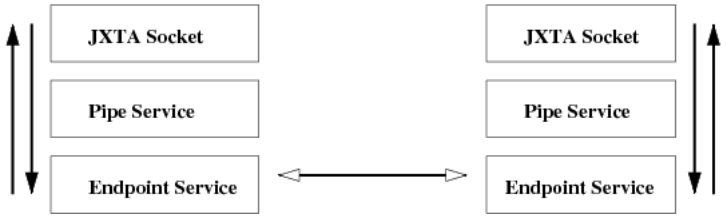

Figura 4.18: Protocolos de Comunicação JXTA

Os nós usados no benchmark consistem de máquinas com processadores Pentium IV 2.4GHz com 1 GB de RAM cada, executando a versão 2.4 do kernel Linux. Os testes foram realizados em uma rede Fast Ethernet (100 Mb/s). Foram executados testes usando JXTA-J2SE 2.2.1 e 2.3 no caso da análise das implementações Java. Foi utilizada a máquina virtual Java 1.4.2, executada com as opções -server -Xms256m -Xmx256m.

O gráfico 4.19 mostra a comparação entre os três mecanismos de comunicação do JXTA e a comunicação utilizando Java Sockets, enquanto a latência é mostrada na Tabela 4.2. O gráfico citado mostra que a diferença para enviar mensagens grandes é insignificante, mas que para mensagens menores esta diferença pode ser significativa devido a adição de mensagens XML por cada um dos protocolos JXTA. As curvas mostram que os dois mecanismos de transporte mais utilizados pelas aplicações JXTA (JxtaSockets e *Unicast pipe*) são capazes de alcancar uma vazão próxima da obtida com Java Sockets em uma rede Fast Ethernet a medida que aumenta o tamanho das mensagens. Portanto, a escolha de um protocolo XML é vantajosa por ser independente de plataforma, mas acarreta limitações de performance.

Entretanto, a tabela 4.2 mostra que estes dois mecanismos apresentam uma latência ruim quando comparados ao Java Socket. Isto se deve ao alto custo de processamento de grandes documentos XML incluidos em cada mensagem. Quando é utilizado o serviço de *endpoint* a diferença em relação ao Java Socket é diminuida, mas mesmo assim ainda permanece $400usec$ maior. Segundo os autores, esta diferença provavelmente refere-se a problemas relacionados a gerência de *threads* (escalonamento, criação e destruição) na plataforma JXTA, uma vez

Java socket	< 0.10 ms
Endpoint service	0.48 ms
Unicast pipe	1.22 ms
JXTA socket	1.76 ms

Tabela 4.2: Latência do JXTA

que os testes indicaram que a plataforma JXTA utiliza aproximadamente 35 *threads*, com picos de até 40 *threads*.

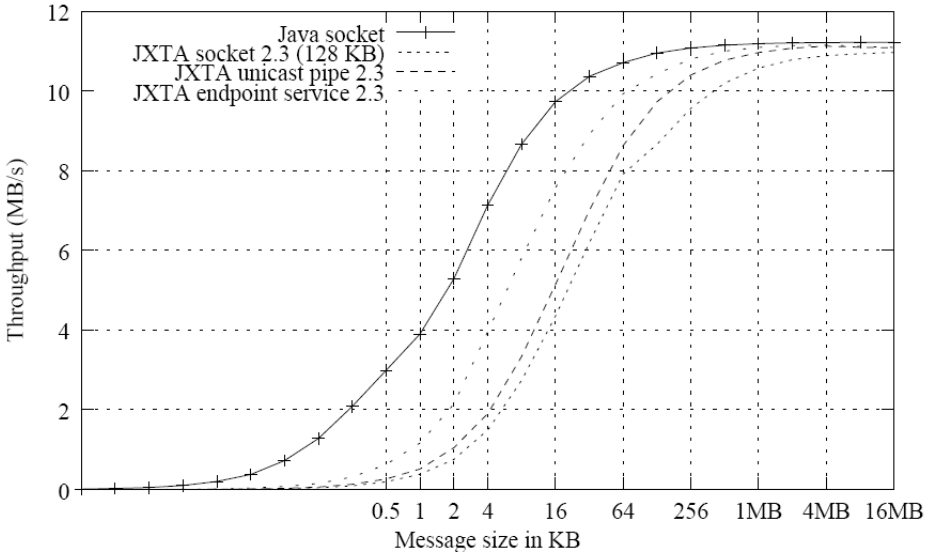

Figura 4.19: Vazão alcançada pelas camadas de comunicação JXTA e Java Sockets

4.6 Limitações do JXTA

Dentre as desvantagens e limitações da rede JXTA, está o uso de XML como base para os protocolos P2P. Apesar do formato XML dar grande flexibilidade e portabilidade ao JXTA, o uso deste formato de dados criou uma sobrecarga de processamento, pois a performance da plataforma JXTA está estreitamente vinculada à performance do interpretador XML utilizado.

Além disso, existe uma sobrecarga de rede, pois uma mensagem transmitida entre dois *Peers* geralmente é encapsulada em várias camadas de documentos XML. Isto faz com que as mensagens transmitidas na rede JXTA sejam grandes, ocasionando uma sobrecarga na rede de comunicação. A transmissão de uma série de mensagens pequenas pode inundar a rede, consumindo a largura de banda e recursos do *Rendezvous*.

Finalmente, some-se a isto a escassez de documentação. O JXTA ainda não dispõem de documentação consistente e abrangente. Isto dificulta a inclusão de novos colaboradores do projeto, ou mesmo a utilização da plataforma por programadores não habituados a programação *Peer-to-Peer*. Os livros sobre JXTA estão desatualizados e alguns aspectos de implementação importantes para o en-

tendimento do protocolo só podem ser apreendidos mediante consultas ao código fonte.

As desvantagens citadas fazem com que o JXTA não possa ser utilizado em aplicações de missão crítica como aplicações em tempo real, por exemplo, ou mesmo aquelas onde a performance seja um fator decisivo. Entretanto, vários estudos de performance tem sido feitos de forma a identificar os gargalos do sistema e as possibilidades de melhoria na arquitetura e nos protocolos. A implementação JXTA-C tem conseguido índices de performance bons para redes *Gigabit Ethernet* e *Myriad*. Um caminho promissor diz respeito ao uso de novos protocolos a nível de aplicação que permitem atravessar firewalls e sistemas NAT, e abrem a possibilidade de abolir o uso de *Relays* para estabelecer comunicação com *Peers* fora do firewall.

O protocolo, por sua vez, continua a evoluir e ser aperfeiçoado pela comunidade criada em torno da plataforma JXTA. Acreditamos que a medida que novas demandas de rede - a integração de uma gama maior de dispositivos móveis, por exemplo - sejam integradas à plataforma, revisões e aperfeiçoamentos dos protocolos poderão ser efetuados com mais segurança.

Capítulo 5

p2pBIOFOCO

O primeiro resultado concreto desta dissertação de mestrado foi o desenvolvimento de uma arquitetura distribuida, batizada de BioGridDF, escrita em Java e que fazia uso de filas de mensagens para a integração das três instituições. A arquitetura deste protótipo inicial é mostrada na Figura 5.1. Além disso, nesta primeira etapa optamos por testar as seqüências de DNA com a família de aplicações para anotação protéica denominada Interpro.

Ao submetermos um arquivo contendo um conjunto de seqüências de DNA, a interface gráfica acionava um módulo escalonador que particionava o arquivo de entrada em sub-arquivos, armazenava-os em um diretório NFS compartilhado por todas as instituições, e enviava um comando para a fila de mensagens. Este comando especificava qual domínio seria o responsável por recuperar um determinado sub-arquivo e processá-lo no Interpro. Os domínios, por sua vez, possuíam clientes que escutavam continuamente a fila de mensagens, e ao receberem uma mensagem endereçada a eles, tratavam de recuperar o arquivo de entrada especificado na mensagem, e distribuir o processamento entre máquinas locais através do escalonador de tarefas *Sun Grid Engine (SGE)* [50]. Os resultados deste processamento eram posteriormente armazenados no diretório NFS citado anteriormente.

Embora não tenha implementado mecanismos sofisticados de tolerância a falhas e monitoramento, este protótipo inicial nos permitiu vislumbrar os ganhos de performance e problemas a serem tratados durante o desenvolvimento e implantação de um sistema distribuído que integrasse as três instituições. Os detalhes de implementação e resultados obtidos com esta primeira infra-estrutura são apresentados em [51, 52], e no apêndice desta dissertação apresentamos um dos artigos publicados com o resultado deste primeiro trabalho.

Este sistema apresentava ainda uma interface gráfica baseada na WEB (Figura 5.2) que permitia a submissão dos arquivos de entrada em formato FASTA, e posterior visualização dos arquivos de saída em formato XML de forma intuitiva.

O fato de optarmos por uma arquitetura centralizada representou uma limitação severa na segurança e confiabilidade do sistema, pois além de sobrecarregarmos uma única máquina do sistema com a tarefa de coordenação dos domínios, esta representava um ponto único de falha. Some-se a isto os problemas decorrentas da impossibilidade de estabelecer algumas conexões com máquinas protegidas por firewalls, que acabaram por inviabilizar a inclusão de uma das instituições nos testes realizados.

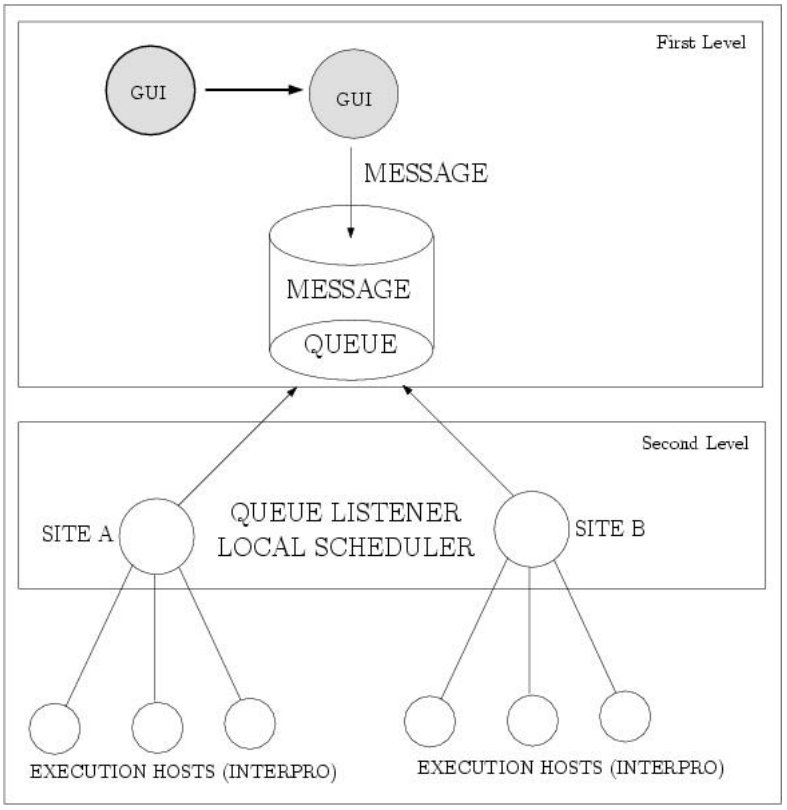

Figura 5.1: Arquitetura do Sistema BioGridDF

5.1 p2pBIOFOCO

A contribuição posterior deste trabalho foi o projeto, implementação e teste de um sistema distribuído para execução do BLAST baseado no modelo P2P. O BLAST foi escolhido por tratar-se de uma ferramenta amplamente utilizada em projetos de seqüenciamento de genomas. A linguagem de programação Java [53] foi escolhida para implementar este protótipo, pois a implementação mais estável de JXTA, denominada JXTA2SE, foi implementada nesta linguagem, que por sua vez é amplamente difundida e portável.

Como decisão inicial de projeto, desenvolvemos um sistema capaz de definir, carregar, exportar e executar serviços usando a infra-estrutura da plataforma JXTA. O objetivo é que além do BLAST, mas outras ferramentas de Bioinformática possam ser executadas remotamente através do sistema p2pBIOFOCO. Além disso, procurou-se desenvolver um sistema baseado em módulos fracamente acoplados com o intuito de permitir a fácil integração, ou mesmo remoção, de funcionalidades e componentes. Na Figura 5.3 podemos visualizar os módulos básicos que compõem o protótipo inicial do sistema p2pBIOFOCO, a saber:

- *Plataforma Peer-to-Peer* - Módulo que implementa a interface entre a aplicação e a plataforma JXTA;

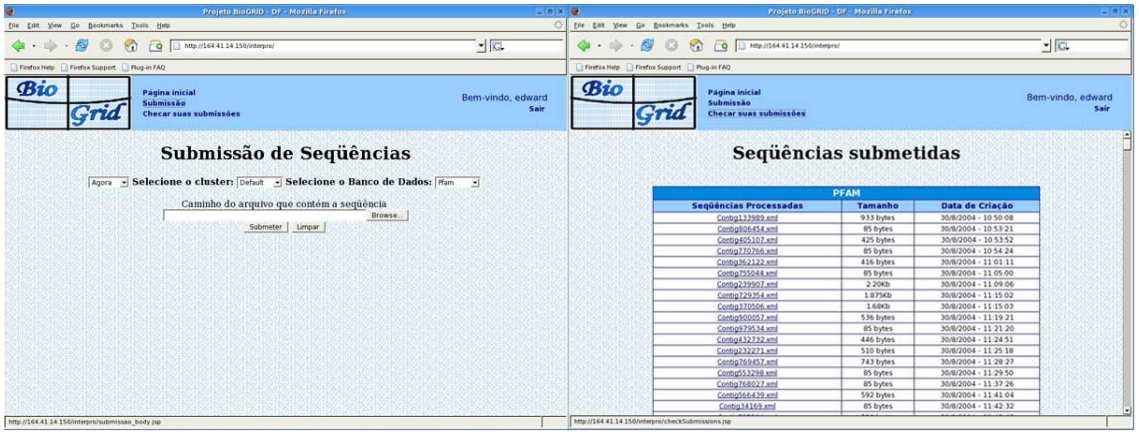

Figura 5.2: Interface WEB do sistema BioGridDF

- *Módulo de Consulta* - Módulo encarregado de criar, enviar, receber e processar consultas a serviços de Bioinformática na rede *Peer-to-Peer*;
- *Módulo de Presença* - Módulo encarregado de obter informações sobre o estado dos *Peers* na rede (ativo/inativo).
- *Módulo de Ativação Remota de Serviços* - Módulo encarregado de fazer chamadas a serviços remotos utilizando a infra-estrutura *Peer-to-Peer* através do envio de mensagens e transferência de arquivos de entrada e saída. Este módulo é ainda composto por um cliente de FTP e um banco de dados de serviços remotos e locais.
- *Interface Gráfica (GUI)* - Módulo opcional que provê uma interface interativa para a pesquisa e execução de serviços remotos, além da visualização dos *Peers* ativos.

A aplicação pode ser executada em dois modos: GUI e *daemon*. O modo *daemon*, que é padrão, permite a execução da aplicação em modo não interativo, e faz com que o p2pBIOFOCO atue apenas como um executor de serviços previamente carregados no início da aplicação e acionados remotamente. O modo GUI, por sua vez, fornece a interface gráfica para pesquisa e chamada a serviços remotos, além do mecanismo de presença que permite saber quais *Peers* estão atualmente na rede. Nas próximas seções detalhamos cada um dos módulos do p2pBIOFOCO.

5.2 Interface Gráfica (GUI)

Na Figura 5.4 temos a tela inicial do modo GUI, que é composta por uma caixa de texto e botão para realizar as buscas por serviços, e três painéis para informações. O painel esquerdo mostra os *Peers* atualmente presentes na rede, o painel direito apresenta os resultados da busca executada e, finalmente, o painel central exibe

Figura 5.3: Arquitetura do p2pBIOFOCO

os detalhes dos *Anúncios* de serviços, chamados **BioService**, recuperados. A pequena esfera verde no canto superior direito, que inicialmente é cinza, indica que o *Peer* em questão tornou-se um *Rendezvous* mediante comando interativo do usuário ou de forma automática, caso este *Peer* não tenha conseguido conectar-se a um *Rendezvous* após um tempo determinado (2 minutos, atualmente).

Conforme citado anteriormente, esta interface gráfica é opcional. Para que o p2pBIOFOCO seja acionado com a opção de interface gráfica é necessário que o usuário informe a opção "-gui"ao executar o arquivo de script *startup.sh*, que acompanha a distribuição do sistema, conforme mostrado abaixo. Caso não seja informado este parâmetro a aplicação p2pBIOFOCO será iniciada no modo *daemon*, isto é, um processo não gráfico e não interativo.

```
$ ./startup.sh ../carbona -gui
```

A Figura 5.4 mostra uma interface extremamente simples para a carga de serviços locais. Ao ser iniciada, o sistema p2pBIOFOCO carrega um ou mais serviços mediante a leitura de um arquivo XML. Através do botão mostrado na figura é possível, após a edição do arquivo XML, recarregar os serviços locais, remover serviços e carregar novos serviços sem necessitar reiniciar a aplicação.

5.3 Plataforma

O principal módulo do p2pBIOFOCO, denominado *Plataforma Peer-to-Peer*, é responsável pela interface entre a plataforma JXTA e os demais módulos do sistema. A criação deste módulo teve por objetivo evitar que mudanças posteriores na arquitetura da plataforma JXTA, ou até mesmo a adoção de outro *framework Peer-to-Peer*, exigissem mudanças significativas na arquitetura do sistema p2pBIOFOCO. Desta forma, os módulos existentes, e quaisquer outros módulos que venham a ser desenvolvidos, fazem chamadas a esta plataforma para realizar as tarefas comuns a sistemas *Peer-to-Peer*, tais como o publicação e a busca de recursos, localização de *Peers*, envio de mensagens entre *Peers* e compartilhamento de arquivos, por exemplo. Com vistas a garantir a privacidade e aumentar o desempenho, o sistema p2pBIOFOCO utiliza uma rede privada JXTA totalmente

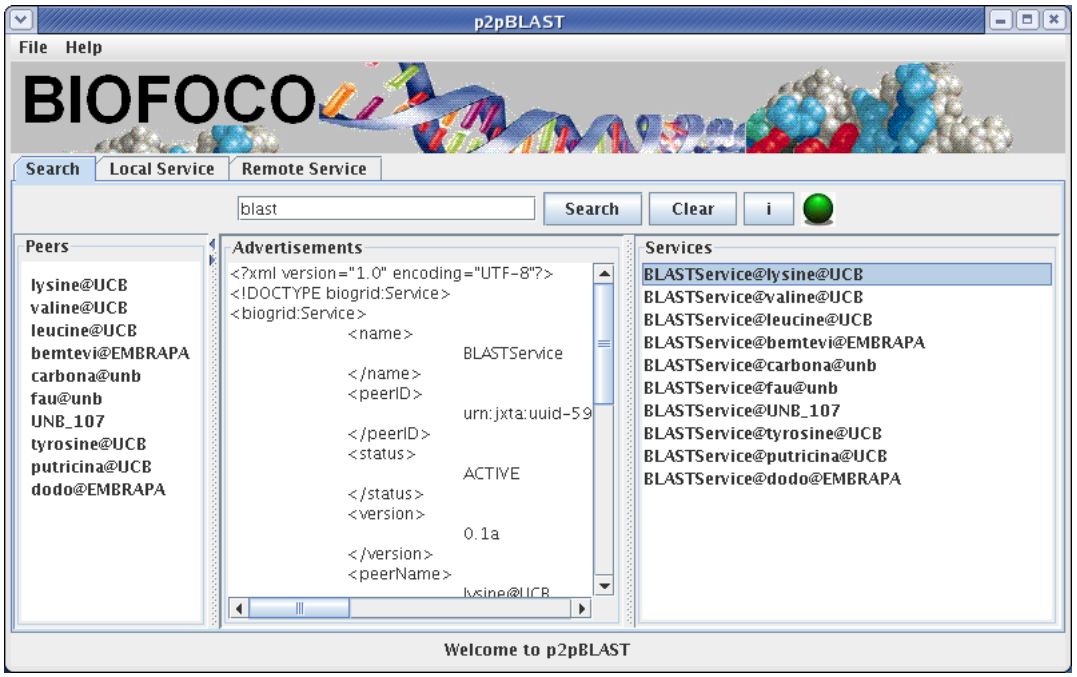

Figura 5.4: Interface Gráfica (GUI) da aplicação p2pBIOFOCO

isolada da rede pública. Isto é garantido pelo módulo *Plataforma Peer-to-Peer*, mediante a carga do arquivo *config.properties* ao iniciar a aplicação.

```
NetPeerGroupID=uuid-15B97053243A4D2BA57DA3D53638EF5D02
NetPeerGroupName=BioFocoGroup
NetPeerGroupDesc=BioFocoGroup
```

Figura 5.5: Arquivo *config.properties* usado pelo p2pBIOFOCO

O NetPeerGroupID, que identifica de forma única uma rede privada JXTA, foi criado uma única vez e de forma não automática, isto é, com o uso de um pequeno aplicativo Java especialmente desenvolvido para tal propósito. O módulo *Plataforma Peer-to-Peer* precisa carregar o arquivo *config.properties* 5.3 para poder se conectar a rede privada P2P criada para este projeto.

Além do arquivo citado, a *Plataforma Peer-to-Peer* faz uso de outro arquivo de configuração, *app.properties*, que é utilizado pelo sistema p2pBIOFOCO para carregar configurações iniciais do sistema tais como o diretório FTP (com credenciais de acesso) onde os arquivos de entrada e saída serão armazenados, por exemplo.

Finalmente, para que a plataforma JXTA seja executada é necessário informar um diretório de execução JXTA para cada *Peer*. Tal diretório contém o cache de *Anúncios* recebidos/enviados pela plataforma JXTA, o arquivo *PlatformConfig* com informações essencias acerca do *Peer* JXTA (PeerID, PeerName, portas de

Figura 5.6: GUI - Recarga de serviços locais

rede, etc.), e o arquivo *config.properties*, que cria a rede privada. Ao iniciar o módulo *Plataforma Peer-to-Peer*, este diretório de execução deve estar acessível à aplicação p2pBIOFOCO.

5.4 Módulo de Consulta

Neste sistema cada *Peer* é responsável por carregar e disponibilizar um conjunto de serviços para os demais *Peers* da rede P2P. Desta forma, tornou-se necessário que os serviços diponibilizados pelos *Peers* pudessem ser localizados na rede P2P. Cada serviço neste sistema é exportado para a rede JXTA mediante a publicação de um *Anúncio* customizado, conhecido como **BioService** (Figura 5.4). Este módulo é responsável pelas operações de publicação e localização de serviços.

Todos os serviços acessíveis através do sistema p2pBIOFOCO devem ser escritos em linguagem Java ou devem poder de ser ativados mediante uma classe adaptadora Java (*wrapper class*). Em ambos os casos, a classe deve implementar a *interface* Java *br.org.biofoco.p2p.services.Service*, que define como a chamada a serviços remotos no sistema p2pBIOFOCO deve ser feita. No caso particular de aplicações de Bioinformática como BLAST, que geralmente são escritas em Perl ou C/C++, a classe Java faz uma chamada ao Sistema Operacional para a execução do programa. Uma vez escritas as classes que implementam ou acionam serviços, o usuário deve escrever um arquivo XML de exportação de serviços locais, *services.xml*, e colocá-lo em um diretório acessível ao sistema p2pBIOFOCO. A Figura 5.4 mostra um exemplo de arquivo de exportação, que contém informações básicas sobre o serviço (nome do serviço e classe que implementa o serviço).

```
<?xml version="1.0" encoding="UTF-8" ?>
<services>
<service>
<name>BLASTService</name>
<class>br.org.biofoco.p2p.blast.BlastService</class>
</service>
<service>
<name>HelloWorldService</name>
<class>br.org.biofoco.p2p.services.impl.HelloWorldService</class>
</service>
</services>
```

Figura 5.7: O arquivo services.xml permite a exportação de serviços locais para a rede P2P

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE biogrid:Service>
<biogrid:Service>
<name> BLASTService </name>
<peerID> urn:jxta:uuid-596162616461626l4A7874615</peerID>
<status> ACTIVE </status>
<version> 0.1a </version>
<peerName> PeerOne </peerName>
<specification> BLAST Service under Linux </specification>
<url></url>
<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
<Id>urn:jxta:uuid-15B97053243A4D2BA57DA3D53638EF</Id>
<Type> JxtaUnicast </Type>
<Name> service-pipe </Name>
<Desc> </Desc>
</jxta:PipeAdvertisement>
</biogrid:Service>
```

Figura 5.8: Um *Anúncio* descrevendo um serviço do sistema p2pBIOFOCO

Ao iniciar o sistema p2pBIOFOCO, o arquivo *services.xml* é lido, os serviços descritos no arquivo são carregados em memória, e armazenados em um banco de dados de serviços locais através do módulo de *Ativação Remota de Serviços*. Logo em seguida, este módulo de *Ativação Remota* aciona o módulo de *Busca* que cria um *Anúncio* JXTA (Figura 5.4), customizado para descrever especificamente cada um dos serviços carregados, e publica estes *Anúncios* na rede *Peer-to-Peer* (Figura 5.9).

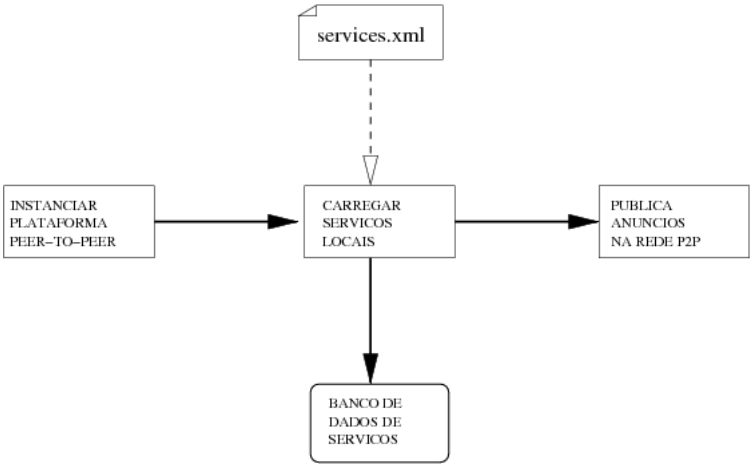

Figura 5.9: Publicação de Serviços no p2pBIOFOCO

Ressaltamos que cada *Anúncio* publicado contém informações adicionais sobre o *Peer* que provê o serviço, tais como o nome do *Peer* e o *Anúncio de Pipe* (*Pipe Advertisement*), que deve ser utilizado para quando for feita uma conexão com o *Peer* e para que seja possível acessar o serviço. O *Anúncio de Pipe* é o equivalente na arquitetura JXTA à tupla **(número IP, porta)**, presente na arquitetura **TCP/IP**. Teoricamente, poderiamos ter serviços distintos utilizando *pipes* distintos, mas atualmente todos os serviços utilizam o mesmo *pipe* para contactar os serviços de um dado *Peer*, e a mensagem de requisição enviada pelo canal de comunicação determina qual serviço deve ser ativado.

No projeto do sistema p2pBIOFOCO adotamos uma estratégia baseada no modelo clássico de publicação de *Anúncios* existente no JXTA. Ao iniciar a aplicação pela primeira vez, o *Módulo de Consulta* publica seus serviços locais carregados na rede P2P. Caso publicassemos os *Anúncios* uma única vez, aqueles serviços que não estivessem mais disponíveis continuariam a ser recuperados por outros *Peers* do sistema, e grande parte do tempo teria que ser gasto na tentativa de manter uma visão consistente dos serviços disponíveis, e eliminar *Anúncios* espúrios ainda distribuídos pela rede. Para evitar este tipo de problema, cada *Anúncio* de serviço é publicado com um tempo de vida máximo (1 hora atualmente). Decorrido este tempo o *Anúncio* é automaticamente removido do cache de cada *Peer* do sistema p2pBIOFOCO. Isto ainda obriga cada *Peer* a republicar seus *Anúncios* de serviços periodicamente, caso o serviço ainda esteja disponível naquele Peer. Neste modelo a busca por um serviço torna-se a busca por um *Anúncio* previamente publicado na rede, utilizando o serviço de descoberta padrão da plataforma JXTA, conforme especificado na Seção 4.4.

5.5 Módulo de Presença

Infelizmente, a plataforma JXTA não dispõe de um mecanismo de presença embutido. Por esta razão, as aplicações baseadas em JXTA que desejarem tal funcionalidade, cada vez mais importante em aplicações distribuídas, tem que implementar seu próprio mecanismo de presença. O módulo de presença do p2pBIOFOCO permite que um Peer anuncie sua disponibilidade na rede. Atualmente são permitidos somente os estados de ativo e inativo, mas no futuro pretendemos integrar estados adicionais (ocupado, não disponível). A presença faz uso de dois pipes multi-Peer (*propagate pipes*) e centra-se basicamente no papel do rendezvous para divulgar os anúncios de presença. Em outras palavras, trata-se de um serviço de presença semi-centralizado.

Ao iniciar o sistema p2pBIOFOCO, cada *Peer* pode escutar dois tipos de *pipes* multi-Peer: um ***pipe* de eventos de presença** e outro ***pipe* de lista de presença**. Ambos os *pipes* são bem conhecidos, isto é, para instanciá-los são utilizadas cadeias de caracteres previamente definidas pelo sistema p2pBIOFOCO. Os *pipes de eventos* são instanciados por todos os *Peers* no sistema, ao passo que os *pipes de lista de presença* são instanciados somente por aqueles *Peers* que são, ou tornaram-se, *Rendezvous*.

Assim que um *Peer* se conecta a um *Rendezvous*, ele publica periodicamente (5 minutos atualmente) um *Anúncio de presença* no *pipe de eventos*. Este anúncio é enviado para todos os outros *Peers* atualmente conectados. Cada *Peer* que recebe este *Anúncio de presença* atualiza a sua lista particular de *Peers* ativos no sistema (lista de presença). Se um dado *Peer* não se manifestar em tempo hábil, os outros *Peers* assumem que este *Peer* foi desconectado de forma não natural, e o retiram de suas listas de *Peers* ativos. Dentre os *Peers* que recebem o *Anúncio* de presença encontra-se o *Rendezvous* ao qual o *Peer* está conectado. Este *Rendezvous* também atualiza sua lista de presença, mas logo em seguida envia a lista para o *pipe de lista de presença*, que é escutado e publicado somente por *Rendezvous*. A intenção desta estratégia é fazer com que os *Rendezvous* possam sincronizar suas listas de presença.

Além disso, as listas de presença são trocadas entre os *Rendezvous* a intervalos regulares (10 minutos atualmente) para sincronizar e manter consistentes tais listas. Quando um *Peer* de fronteira conecta-se a um *Rendezvous*, ele recebe a lista de presença do *Rendezvous* para poder ter a visão correta acerca da presença atual dos *Peers* na rede.

Quando um *Peer* deseja se desconectar ele envia um *Anúncio* de presença contendo o status de inativo para o *pipe de eventos*. Os *Peers* que recebem tal evento retiram o *Peer* da lista e a operação prossegue de forma semelhante a entrada do *Peer* na rede.

Atualmente, este *Módulo de Presença* atua de forma totalmente independente dos demais módulos do sistema p2pBIOFOCO e tem servido basicamente para monitorar o estado dos Peers na rede durante a fase de preparação dos testes de execução do BLAST, visto que as máquinas tem conectividade transiente ou poderiam ter sido desligadas. Entretanto, acreditamos que este módulo terá um papel de destaque, visto que pode ser utilizado por algoritmos de alocação de tarefas para distribuir os trabalhos.

Este tipo de mecanismo de presença baseado em *Rendezvous* tem sido extensivamente utilizado em aplicações JXTA, pois a estrutura de Super-Nó é a que melhor se adapta para implementar presença. O único Peer falho desta abordagem diz respeito a desconexão repentina, geralmente ocasionada por falha no *Peer*. Uma vez que a conexão entre um *Peer* de fronteira e um Super-Nó é implementada mediante o uso de sinais enviados regularmente ao *Rendezvous*, um *Rendezvous* demora cerca de 20 minutos até perceber que os nós de fronteira não estão mais vivos. Por esta razão, optamos por enviar mensagens de sinal de vida (heartbeat) a cada 5 minutos para notificar que um dado *Peer* ainda encontra-se ativo.

5.6 Execução Remota

O *Módulo de Execução Remota* é o mais importante do sistema p2pBIOFOCO, pois permite a execução remota de serviços descobertos na rede *Peer-to-Peer*. Através deste módulo, o programa BLAST pode ser executado de forma distribuída, dividindo um arquivo de entrada em formato FASTA contendo as seqüências biológicas em sub-arquivos e enviando estes arquivos para execução em um dado número de *Peers*.

Este módulo mantêm um bancos de dados com informações sobre serviços locais, carregados no início da aplicação, e outro banco de dados sobre serviços remotos, que é alimentado mediante o resultado de pesquisas na rede P2P. Ao realizar uma busca, os resultados encontrados são armazenados no segundo banco de dados, sendo que cada serviço é automaticamente retirado do banco de dados quando seu tempo de expiração (1 hora atualmente) é alcançado, ou mediante comando interativo do usuário. Após selecionado o arquivo FASTA a ser enviado e os *Peers* de destino, o usuário interativamente submete o arquivo para processamento distribuído (Figura 5.12). Neste etapa, entra em ação o módulo responsável pelo parsing do arquivo FASTA. Este módulo divide o arquivo de acordo com o número de *Peers* encarregados do processamento, transfere os sub-arquivos para um servidor FTP,previamente especificado, e passa o controle para o módulo encarregado de fazer as chamadas remotas 5.10.

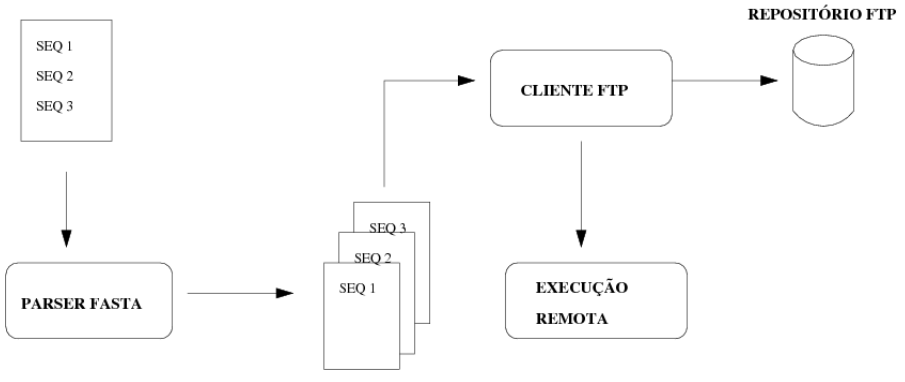

Figura 5.10: Etapa de particionamento de envio de arquivos de entrada

O módulo de chamadas remotas, por sua vez, estabelece conexões com cada

um dos *Peers*, utilizando o serviço de pipes bidirecionais (*JxtaBiDiPipes*), e envia um comando, em formato XML, contendo os argumentos de execução:

- o comando a ser acionado
- os parâmetros de execução do BLAST
- o endereço do repositório FTP
- conta de acesso do repositório FTP
- o nome da seqüência a ser processada

O *Peer* encarregado de processar os pedidos retorna uma confirmação de recebimento do pedido, contacta o repositório FTP, transfere os arquivos de entrada, e executa o serviço BLAST localmente. O processamento local de uma seqüência é repassado ao serviço responsável mediante o módulo de execução remota e, findo o processamento, cada um dos *Peers* conecta-se novamente ao repositório FTP para armazenamento dos resultados do processamento (Figura 5.11).

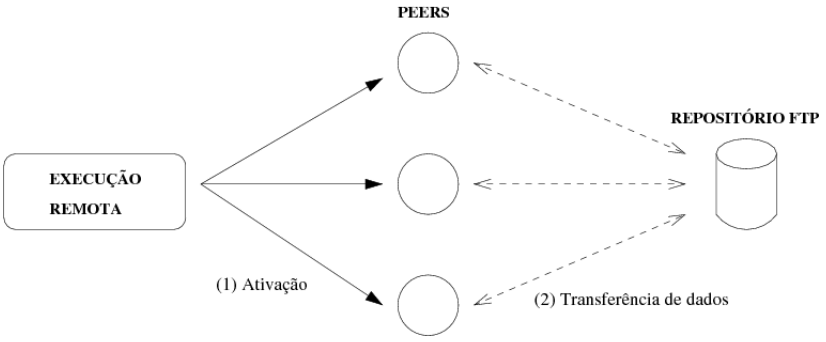

Figura 5.11: Ativação dos Peers no sistema p2pBIOFOCO

Para otimizar a transferência de arquivos foi integrado ao sistema uma implementação Java de código-fonte aberto do protocolo sftp. Optamos por utilizar FTP por ser um protocolo otimizado para a transferência de dados. Muito embora o uso atual deste protocolo represente uma falha de segurança, esperamos integrar um *framework* de autenticação e autorização baseado na API GSS para garantir maior privacidade.

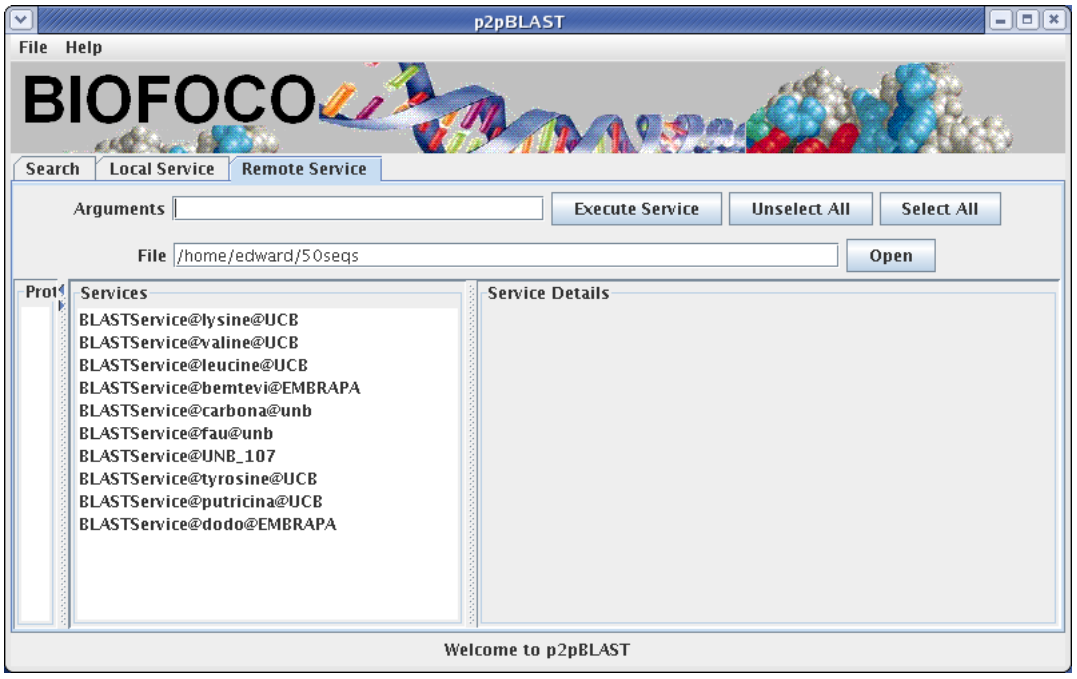

Figura 5.12: GUI - Tela de execução remota do p2pBIOFOCO

Capítulo 6

Experimentos

Para verificar a aplicabilidade e performance da arquitetura proposta, foram realizados dois experimentos envolvendo a execução distribuída do BLAST sobre o banco *nr*, e tendo como entrada seqüências de DNA provenientes do projeto Genoma Pb. Os experimentos envolveram três redes geograficamente separadas e localizadas na Universidade de Brasília (UnB), na Universidade Católica de Brasília (UCB) e na Embrapa Recursos Genéticos. Todas as redes possuem Firewalls instalados e limitações quanto a conexões que podem ser realizadas a partir da Internet. Por medidas de segurança e performance, foi construída uma rede privada JXTA na qual os experimentos foram realizados. Some-se a isto o fato do ambiente de hardware e software utilizado ser bastante heterogêneo, compreendendo máquinas com diferentes velocidades, capacidades de armazenamento e versões do Sistema Operacional Linux. A execução destes experimentos em um ambiente heterogêneo e distribuído nos permitiu ter uma visão mais precisa acerca dos requisitos e ganhos reais de execução distribuída de aplicações de Bioinformática.

6.1 Configuração

Todas as máquinas encarregadas de rodar o BLAST dispunham do sistema operacional Linux, BLAST versão 2.2.10, e do banco *nr*, um dos maiores bancos de proteínas e que ocupa cerca de 1.4 GB de espaço em disco, totalmente instalado e pré-formatado com a ferramenta **formatdb**, que acompanha a distribuição do BLAST. Os tempos de execução foram medidos tomando-se como início o tempo de armazenamento dos arquivos FASTA de entrada no repositório FTP, e como término o armazenamento dos arquivos de saída do BLAST neste mesmo repositório FTP. Desta forma, este cálculo inclui o tempo necessário para notificar cada *Peer* acerca da tarefa a ser realizada, a transferência dos arquivos de entrada para cada *Peer*, o processamento do BLAST, e a transferência dos arquivos de saída do *Peer* para o repositório FTP.

As Tabelas 6.2 a 6.8 detalham as características de hardware e software todas as máquinas utilizadas nos dois experimentos, e as classificam por tipos para facilitar a compreensão das configurações de rede utilizadas.

No primeiro experimento, foram utilizadas 12 máquinas distribuídas entre

Domínio	Experimento	Infra-estrutura
Universidade Católica de Brasília	5	1
Universidade de Brasília	5	1

Tabela 6.1: Máquinas utilizadas no Experimento 1

Característica	Especificação
Processador	Pentium 4
Clock	2.40 GHz
Cache	512 KB
RAM	1 GB
Swap	2 GB
HD	36 GB
Sistema Operacional	Linux 2.6
Java Virtual Machine	1.5
JXTA	2.5.6

Tabela 6.2: Máquinas utilizadas na UCB (Tipo 1)

Característica	Especificação
Processador	AMD Athlon
Clock	995 MHz
Cache	256 KB
RAM	239 MB
Swap	538 MB
HD	4.2 GB
Sistema Operacional	Linux 2.6
Java Virtual Machine	1.5
JXTA	2.5.6

Tabela 6.3: Máquinas utilizadas na UnB (Tipo 2)

os dois domínios, UCB e UnB, conectando duas instituições, sendo duas destas máquinas utilizadas de forma exclusiva na infra-estrutura de comunicação do sistema p2pBIOFOCO, e o restante das máquinas dispunha do sistema p2pBIOFOCO para rodar os experimentos propriamente ditos, conforme resumido na Tabela 6.1 e detalhado na Tabela 6.9. A representação gráfica da rede JXTA obtida com esta configuração (Figura 6.1) pode ser obtida mediante uma aplicação Java denominada **iView**.

As máquinas utilizadas na infra-estrutura do sistema foram exclusivamente reservadas para executar os *Rendezvous/Relays* JXTA de forma a permitir a interligação entre os dois domínios, a construção de uma rede privada JXTA, e a aceleração dos tempos de conexão dos *Peers* de fronteira. Entretanto, conforme visto na seção anterior, quaisquer *Peers* poderiam ser automaticamente promovidos a *Rendezvous*, e por vezes isto realmente ocorreu durante os experimentos quando os dois *Rendezvous* principais tornaram-se momentaneamente indisponíveis ou sobrecarregados, por exemplo.

Conforme citado anteriormente, na Figura 6.1 visualizamos a rede privada JXTA que foi implantada para a realização do primeiro experimento. Os dois quadriláteros representam os *Rendezvous* de infra-estrutura, que por sua vez delimitam as fronteiras dos dois domínios participantes do experimento, enquanto os círculos representam *Peers* de fronteira.

Um fato interessante que merece ser citado por ter ocorrido com certa freqüência

Característica	Especificação
Processador	AMD Athlon
Clock	1.3 GHz
Cache	256 KB
RAM	223 MB
Swap	522 MB
HD	4.2 GB
Sistema Operacional	Linux 2.6
Java Virtual Machine	1.5
JXTA	2.5.6

Tabela 6.4: Máquinas Utilizadas na UnB (Tipo 3)

Característica	Especificação
Processador	Pentium 4
Clock	1.7 GHz
Cache	256 KB
RAM	255 MB
Swap	1 GB
HD	24 GB
Sistema Operacional	Linux 2.6
Java Virtual Machine	1.5
JXTA	2.5.6

Tabela 6.5: Máquinas utilizadas na UnB (Tipo 4)

Característica	Especificação
Processador	Intel Xeon
Clock	3.0 GHz
Cache	2.0 MB
RAM	2.0 GB
Swap	2.0 GB
HD	97 GB
Sistema Operacional	Linux 2.6
Java Virtual Machine	1.5
JXTA	2.5.6

Tabela 6.6: Máquinas Utilizadas na Embrapa (Tipo 5)

Característica	Especificação
Processador	Pentium 4
Clock	1.7 GHz
Cache	256 KB
RAM	255 MB
Swap	1 GB
HD	24 GB
Sistema Operacional	Linux 2.6
Java Virtual Machine	1.5
JXTA	2.5.6

Tabela 6.7: Máquinas utilizadas na Embrapa (Tipo 6)

Característica	Especificação
Processador	Pentium III
Clock	700 MHz
Cache	256 KB
RAM	256 MB
Swap	522 MB
HD	11 GB
Sistema Operacional	Linux 2.6
Java Virtual Machine	1.5
JXTA	2.5.6

Tabela 6.8: Máquina utilizada na UCB (Tipo 7)

Instituição	Identificação	Quantidade	Função
Universidade de Brasília	Tipo 2	2	execução
Universidade de Brasília	Tipo 3	2	execução
Universidade de Brasília	Tipo 4	1	execução
Universidade de Brasília	Tipo 4	1	infra-estrutura
Universidade Católica de Brasília	Tipo 1	5	execução
Universidade Católica de Brasília	Tipo 7	1	infra-estrutura

Tabela 6.9: Especificação de Máquinas utilizadas no Experimento 1

diz respeito à um estado de inconsistência que acontece entre os *Rendezvous* quando um *Peer* aparentemente está conectado a dois *Rendezvous* ao mesmo tempo. Isto provavelmente ocorre quando um *Peer* consegue conectar-se no primeiro *Rendezvous*, perde a conexão, e conecta-se no segundo *Rendezvous*. Como cada *Rendezvous* leva cerca de 20 minutos para verificar que uma conexão de um *Peer* falhou, ou então quando tenta enviar uma mensagem para aquele *Peer*, então o sistema leva algum tempo para resolver esta inconsistência. Na ferramenta gráfica da Figura 6.1 o *Peer* passa a aparecer como se estivesse conectado a dois Rendezvous, o que não é possível, e este é o caso do *Peer* UNB_90.

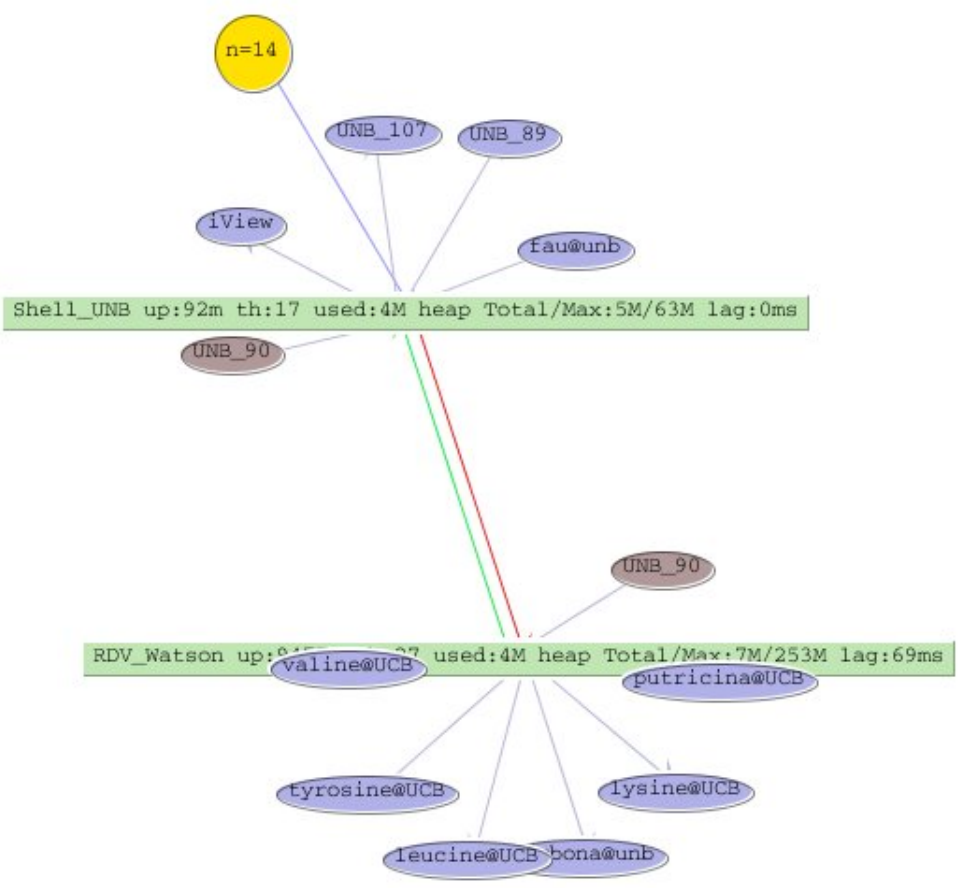

Figura 6.1: Rede Peer-to-Peer utilizada no primeiro experimento

No segundo experimento, foram utilizadas 13 máquinas distribuídas entre os três domínios, UCB, UnB, e Embrapa, sendo três destas máquinas utilizadas na infra-estrutura de comunicação do sistema p2pBIOFOCO, conectando as três instituições, e o restante das máquinas tiveram o sistema p2pBIOFOCO instalado para rodar os experimentos propriamente ditos, conforme resumido na Tabela 6.10 e detalhado na Tabela 6.11. A decisão de manter basicamente o mesmo número de máquinas de execução do experimento anterior deveu-se ao fato de que com a inclusão de três novas máquinas foi possível retirar as duas máquinas menos eficientes do conjunto de máquinas encarregadas de executar a aplicação

Domínio	Experimento	Infra-estrutura
Universidade de Brasília	3	1
Universidade Católica de Brasília	5	1
Embrapa Recursos Genéticos	2	1

Tabela 6.10: Máquinas utilizadas no Experimento 2

Instituição	Identificação	Quantidade	Função
Universidade de Brasília	Tipo 3	2	execução
Universidade de Brasília	Tipo 4	1	execução
Universidade de Brasília	Tipo 4	1	infra-estrutura
Universidade Católica de Brasília	Tipo 1	5	execução
Universidade Católica de Brasília	Tipo 7	1	infra-estrutura
Embrapa Recursos Genéticos	Tipo 5	1	infra-estrutura
Embrapa Recursos Genéticos	Tipo 6	2	execução

Tabela 6.11: Especificação de Máquinas utilizadas no Experimento 2

no primeiro experimento. A representação gráfica da rede JXTA obtida com esta configuração obtida com o software **iView** está na Figura 6.4.

6.2 Experimentos

Em todas as execuções do BLAST utilizamos o programa **blastall** configurado para rodar a versão **blastx**. Todas as seqüências de entrada foram provenientes do projeto Genoma Pb e foram devidamente armazenadas em arquivos de formato FASTA.

Na primeira parte dos experimentos, executamos o BLAST sobre o banco *nr* tendo como entrada arquivos FASTA com 50, 100, 200, 400, e 800 seqüências em uma única máquina da UCB (Tipo 1), cuja especificação é detalhada na Tabela 6.2. Os tempos de execução para 1 Máquina são mostrados na Tabela 6.12. O tempo de execução para 1.600 não pode ser verificado por limitações de tempo e recursos.

No segundo experimento, selecionamos 10 máquinas, distribuídas entre os dois domínios, UCB e UnB, para execução dos mesmos grupos de seqüências: 50, 100, 200, 400, 800. Os tempos de execução (em horas) obtidos estão sumarizados na Tabela 6.13.

Os dados de execução em uma máquina e no sistema P2P estão sumarizados no gráfico 6.3.

Finalmente, no terceiro experimento, selecionamos 10 máquinas, distribuídas entre os três domínios, UCB, UnB, e Embrapa, para execução dos mesmos grupos

Seqüências	Tempo (hs)
50	3:17
100	6:22
200	12:46
400	25:31
800	50:54

Tabela 6.12: Tempo de execução do BLAST em 1 máquina da UCB

Seqüências	Tempo (hs)
50	0:48
100	1:21
200	2:46
400	4:42
800	11:34

Tabela 6.13: Tempo de execução do BLAST no p2pBIOFOCO utilizando 10 máquinas entre duas instituições

Seqüências	Tempo (hs)
50	0:36
100	1:18
200	3:04
400	5:34
800	9:09

Tabela 6.14: Tempo de execução do BLAST no p2pBIOFOCO utilizando 10 máquinas entre três instituições

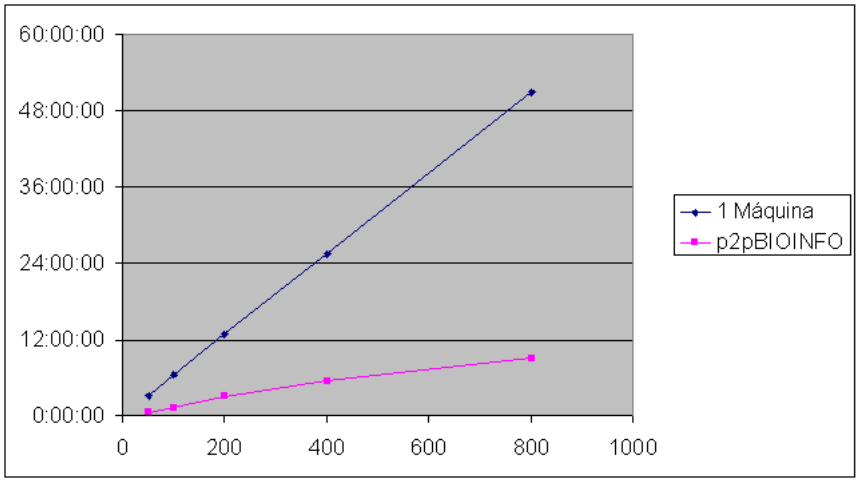

Figura 6.2: Gráfico de Execução em 1 máquina e no sistema p2pBIOFOCO em três instituições

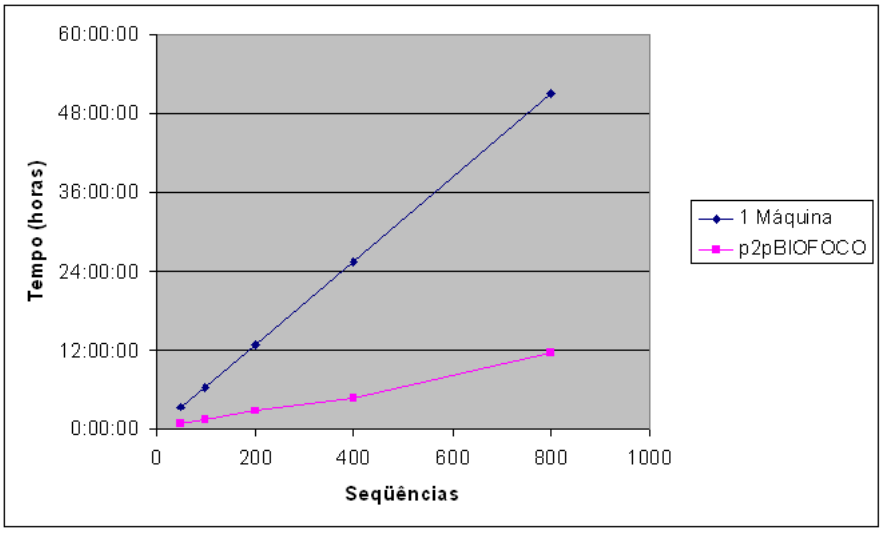

Figura 6.3: Gráfico de Execução em 1 máquina e no sistema p2pBIOFOCO em duas instituições

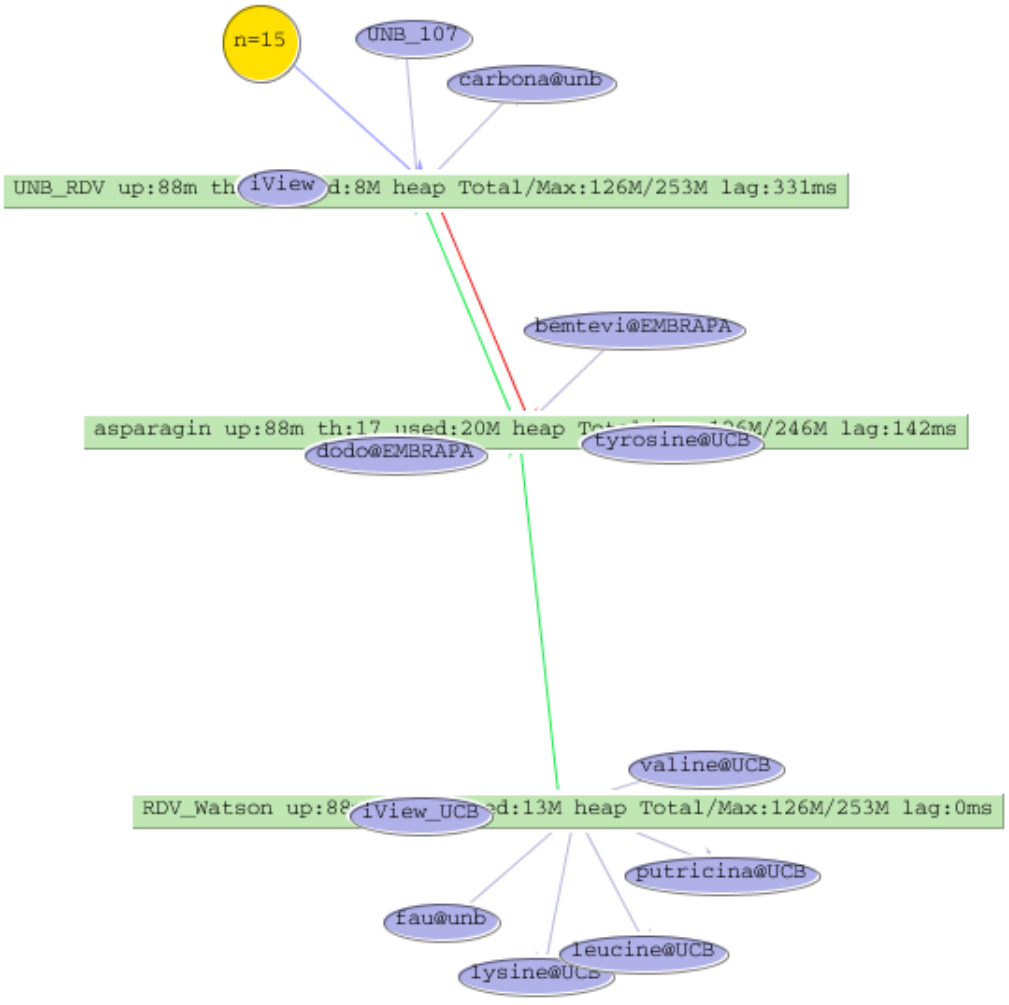

Figura 6.4: Rede Peer-to-Peer utilizada no segundo experimento

de seqüências: 50, 100, 200, 400, 800. Os tempos de execução (em horas) obtidos estão sumarizados na Tabela 6.14. Na figura 6.5, temos o gráfico que resume os três experimentos realizados durante este trabalho. Verificamos que houve um desempenho ligeiramente superior ao substituir as duas máquinas menos eficientes por máquinas melhores.

Em cada etapa dos dois experimentos, as seqüências foram divididas igualmente entre os dez *Peers* e atribuídas a cada um deles através de um algoritmo de *alternância circular (round robin)*.

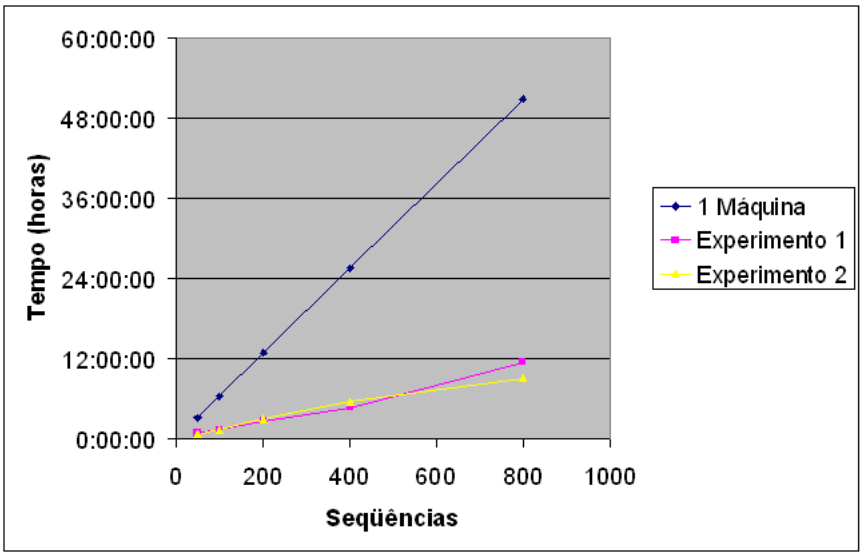

Figura 6.5: Grafico com tempos de execução nos três experimentos

6.3 Análise

A partir dos dados obtidos é claro o ganho de performance ao se executar os dados em ambiente distribuído. A perfomance em ambiente distribuído foi até 80% superior a de uma única máquina. Entretanto, tendo sido utilizado um conjunto de máquinas heterogêneo, houveram diferenças acentuadas entre os tempos de execução de cada máquina. Não houve nenhuma iniciativa no sentido de tirar proveito de máquinas mais poderosas existentes na rede P2P. Isto com certeza é algo a ser melhorado no futuro.

Observamos que algumas máquinas atuaram como gargalo do sistema, aumentando o tempo de execução total do sistema, ou seja, os tempos poderiam ter sido menores caso tivessemos tirado mais proveito das máquinas que dispunham de mais recursos computacionais (clock, memória RAM, etc). Uma solução imediata para este problema seria a adoção de algoritmos mais sofisticados de escalonamento de tarefas que distribuissem mais tarefas para máquinas que dispõem de mais recursos e menos tarefas para máquinas que possuem menos recursos.

Outro aspecto a ser destacado neste trabalho é o fato das comunicações P2P serem não confiáveis mesmo quando implementadas sobre um protocolo confiável e orientado a conexão como é o caso do TCP. Particularmente, o JXTA mostrou-se

rápido na publicação e recuperação de *anúncios* na rede e no envio de mensagens entre dois *Peers*, mas como a resolução do endereço de dois *Peers* é feito no momento anterior ao comunicação entre os mesmos, observamos que algumas vezes dois *Peers* não são capazes de se conectar para transmitir os dados. Acreditamos que isto se deve ao fato das comunicações em uma rede JXTA ocorrerem frequentemente de forma indireta, isto é, utilizando dois ou mais intermediários.

Um dos aspectos que acreditávamos poder impactar negativamente o desempenho do sistema mostrou-se na verdade bastante econômico. A utilização de memória por parte do sistema p2pBIOFOCO ficou em aproximadamente 20 MB, o que o torna relativamente leve para a execução em máquinas mais modestas. Ainda são necessários estudos sobre o tráfego de rede imposto pelas aplicações JXTA.

Capítulo 7

Conclusões e Trabalhos Futuros

Neste trabalho de dissertação realizamos as seguintes atividades:

- projetamos e implementamos um sistema Peer-to-Peer, denominado p2p-BIOINFO, para a execução distribuída de aplicações em Bioinformática;
- utilizamos a suíte de aplicações BLAST como primeiro programa a ser executado nesta arquitetura distribuída;
- montamos uma rede privada P2P que integra três instituições - UnB, UCB, e Embrapa.
- realizamos testes em ambiente real utilizando dados de seqüência provenientes do projeto Genoma Pb;
- analisamos os ganhos de performance advindos da execução distribuída do BLAST.

Os objetivos deste projeto de pesquisa foram alcançados de forma satisfatória e acreditamos ter contribuído com uma abordagem útil para o processamento de aplicações em Bioinformática. Os bons resultados advindos deste projeto de pesquisa nos motivam e abrem a possibilidade de integração de outras ferramentas de Bioinformática. Observamos que apesar do ganho de performance ainda restam questões relativas a confiabilidade e tolerância a falhas da abordagem P2P que precisam ser endereçadas. Na verdade, estes dois aspectos são os mais críticos ao tentar usar um sistema P2P para processamento distribuído em um ambiente controlado. Esperamos que esta arquitetura possa ser integrada em um ambiente de produção, isto é, em projetos de seqüênciamento de genoma/proteoma reais.

Esta dissertação abriu a possibilidade de novas trabalhos na área de sistemas distribuídos e *Bioinformática* dentre os quais destacamos:

Escalonamento de Tarefas

Conforme vimos no capítulo anterior, a performance deste sistema poderia ter sido melhor caso tivessemos adotado algoritmos de escalonamento de tarefas sofisticados. Algoritmos de escalonamento dinâmicos utilizados no sistema GridBLAST parecem os mais apropriados para adaptação ao sistema p2pBIOFOCO, mas soluções novas podem ser implementadas e integradas neste sistema, o que abre novas possibilidades para pesquisa futura nesta área.

Característica	**Ambiente *Grade***	**Ambiente P2P**
Origem	Comunidade Científica	Usuários da Internet
Usuários Simultâneos	Centenas	Centenas de milhares
Serviços	Diversos	Específicos
Arquitetura	Centralizada/Hierárquica	*Peer-to-Peer*
Infra-estrutura	Dedicada	Não Dedicada
Rede Virtual	Sim	Sim
Segurança	Sim	Não
Confiabilidade	Sim	Não
Qualidade de Serviço (QoS)	Sim	Não
Compartilhamento de Recursos	Sim	Sim

Tabela 7.1: Comparação entre os ambientes de *Grade* e P2P

Sistemas P2P e Computação em *Grade*

A medida que os sistemas em *Grade* [22, 24] crescem, aumentando a quantidade de recursos compartilhados, tornam-se necessários mecanismos flexíveis de auto-organização, gerenciamento e replicação de recursos. Sistemas P2P apresentam tais propriedades, pois são escalá veis, isto é, capazes de gerenciar milhões de recursos de forma descentralizada e suportam recursos voláteis. Por por esta razão, existe uma convergência natural entre sistemas em *Grade* e sistemas P2P [48, 54].

Neste cenário, as arquiteturas e algoritmos P2P atuariam como gerenciadores de recursos, permitindo a descoberta e replicação de recursos, e tolerância a falhas. Segundo Jan e co-autores [48] as duas formas básicas de integração P2P são: o uso de serviços em *Grade* como blocos de construção para implementação de serviços P2P, e o uso de bibliotecas P2P - executando sobre infra-estrutura de rede própria de grids - para a implementação de serviços de *Grade*.

O uso de protocolos e arquiteturas *Peer-to-Peer* para a construção de sistemas em *Grade* é uma área promissora, mas pouco explorada devido a sua complexidade. As tentativas mais conhecidas de integração da plataforma JXTA e o Globus (Cog Kit JXTA e JXTA-Grid project) encontram-se atualmente inativas, por exemplo. Isto se deve ao fato de que aplicações em *Grade* possuem importantes requisitos de performance que não são requisitos usuais de sistemas P2P tais como a transmissão eficiente de dados, por exemplo.

A Tabela 7 sumariza as semelhanças e diferenças, geralmente encontradas, entre os ambientes P2P e *Grade*. Podemos ver que a principal discrepância encontra-se nos critérios de segurança, confiabilidade e Qualidade de Serviço (QoS). Tais diferenças se devem ao fato de que os ambientes em *Grade* foram criados por comunidades dispostas a pagar o preço de montar e dar suporte a uma infra-estrutura dedicada com um mínimo de segurança tendo em vista os recursos sofisticados (instrumentação dedicada, supercomputadores) disponibilizados. A arquitetura P2P, por sua vez, é composta primordialmente por máquinas não dedicadas, com menor poder de processamento e conectividade transiente [54].

Por fim, acreditamos que as melhorias na plataforma JXTA tendem a permitir integrações efetivas entre os dois ambientes. A vantagem advinda da sinergia entre os ambientes P2P e *Grande* tende a beneficiar ambas as comunidades.

Adição de Novas Ferramentas
Ferramentas de Monitoramento

1. ***JXTASink*** é uma ferramenta que pode ser utilizada para medir a es-

calabilidade e identificar falhas de implementação em Rendezvous. Este programa emula o protocolo *endpoint* e permite simular uma rede de até 1.000 *Peers* em um único computador. Tais *Peers* podem ser utilizados para se conectar a *Rendezvous* reais e testar a carga máxima que tais *Rendezvous* são capazes de suportar, além de identificar gargalos de comunicação (Figura 7.1). Infelizmente, a implementação atual do *JXTASink* (disponível em http://www.jxta.org) encontra-se em estágio inicial, uma *prova de conceito*, e não pode ser utilizada neste trabalho. Uma nova implementação desta ferramenta será importante para o aperfeiçoamento da plataforma JXTA.

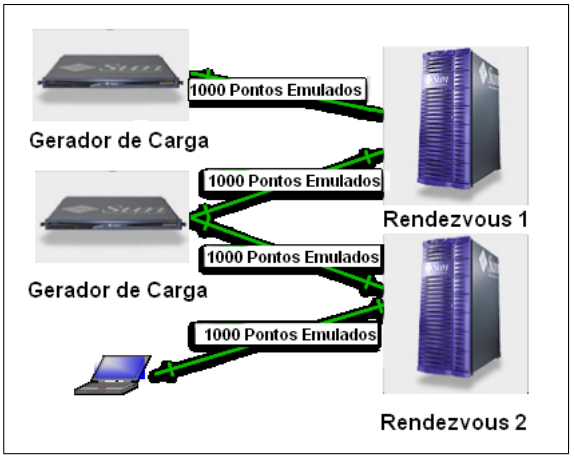

Figura 7.1: JXTASink - Geradores de carga emulam peers para testar dois Rendezvous

2. ***Bench JXTA*** é um projeto JXTA que também encontra-se inativo. Trata-se de um repositório para benchmarks, e programas de teste. Assim como JXTASink, a criação de uma suite de testes consistentes será de enorme ajuda para a comunidade que desenvolve ou dá suporte à plataforma JXTA.

3. ***JXTA Ethereal dissector*** é um plugin criado para interpretar mensagens JXTA.

Referências

[1] J. D. Grant, R. L. Dunbrack, F. J. Manion, and M. F. Ochs, "BeoBLAST: distributed BLAST and PSI-BLAST on a Beowulf cluster," *Bioinformatics*, vol. 18, pp. 765–766, May 2002.

[2] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-Peer Computing," tech. rep., HP Laboratories Palo Alto, March 2002.

[3] J. D. Watson and F. H. Crick, "Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid," *Nature*, vol. 171, pp. 737–738, 1953.

[4] A. P. Francis S. Collins, Michael Morgan, "The Human Genome Project: Lessons from Large-Scale Biology," *Science Magazine*, vol. 300, pp. 286–290, April 2003.

[5] Wellcome Trust Sanger Institute
http://www.sanger.ac.uk.

[6] NCBI – National Center for Biotechnology Information
http://www.ncbi.nlm.nih.gov.

[7] European Molecular Biology Laboratory
http://www.ebi.ac.uk/embl//.

[8] DNA Data Bank of Japan, 2005
http://www.ddbj.nig.ac.jp/.

[9] The Xylella fastidiosa Consortium, "The genome sequence of the plant pathogen Xylella fastidiosa," *Nature*, vol. 406, pp. 151–157, July 2000.

[10] Brazilian Genome - Virtual Institute of Genomic Research
http://www.brgene.lncc.br/.

[11] BIOFOCO - Centro Oeste Bioinformatics Network
http://www.biofoco.org/.

[12] D. S. Roos, "Bioinformatics-Trying to swim in a sea of data," *Science Magazine*, vol. 291, pp. 1260–1261, February 2001.

[13] W3C - World Wide Web Consortium
http://www.w3.org/.

[14] J. Setubal and J. Meidanis, *Introduction to Computational Molecular Biology*. Brooks Cole Publishing Company, 1 ed., 1997.

[15] M. Zhang and L. Kavraki, "Solving molecular inverse kinematics problems for protein folding and drug design," in *Currents in Computational Molecular Biology*, pp. 214–215, ACM Press, April 2002. Book includes short papers from The Sixth ACM International Conference on Research in Computational Biology (RECOM 2002), Washington, DC, 2002.

[16] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic Local Alignment Search Tool," *Journal of Molecular Biology*, vol. 215, pp. 403–410, May 1990.

[17] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler, "Genbank," *Nucleic Acids Research*, vol. 33 (Database issue), pp. D34–D38, 2005.

[18] A. Darling, L. Carey, and W. Feng, "The design, implementation, and evaluation of mpiBLAST," in *Proceedings of the ClusterWorld Conference and Expo, in conjunction with the 4th International Conference on Linux Clusters: The HPC Revolution*, vol. 2735 of *Lecture Notes in Computer Science*, Springer, 2003.

[19] Fox Chase Cancer Center, 2005
http://bioinformatics.fccc.edu/.

[20] D. Becker and P. Merkey, "The Beowulf Project." http://www.beowulf.org, April 2005.

[21] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard," Tech. Rep. UT-CS-94-230, 1994.

[22] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. Supercomputer Applications*, vol. 15(3), 2001.

[23] A. Krishnan, "GridBLAST: a Globus-based high-throughput implementation of BLAST in a Grid computing framework," *Concurrency and Computation: Practice and Experience*, vol. 17, pp. 1607–1623, June 2005.

[24] Globus
http://www.globus.org/.

[25] G. Colouris, J. Dollimore, and T. KindBerg, *Distributed Systems - Concepts and Design*. Pearson Addison Wesley, 3 ed., 2001.

[26] G. N. Cecil, A. Crain, and G. Schumacher, "Remote use of the SOAR 4.25m telescope with LabVIEW," *Advanced Global Communications Technologies for Astronomy II. Proceedings of the SPIE*, vol. 4845, pp. 72–79, November 2002.

[27] M. Shirts and V. S. Pande, "Screen Savers of the World Unite!," *Science*, vol. 290, pp. 1903–1904, December 2000.

[28] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1 – RFC 2616," June 1999.

[29] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer Lookup Service for Internet Applications," in *Proceedings of the 2001 ACM SIGCOMM Conference*, pp. 149–160, 2001.

[30] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande, "Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology," in *Computational Genomics* (R. Grant, ed.), Horizon Press, 2002.

[31] Folding@HOME Distributed Computing
http://folding.stanford.edu/.

[32] M. A. Landers, "An Overview of Peer-to-Peer Network Topologies," tech. rep., Technical University of Munich, April 2004.

[33] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric.," in *IPTPS First International Peer-to-Peer Systems Workshop*, pp. 53–65, 2002.

[34] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pp. 329–350, November 2001.

[35] Project JXTA – an open protocol for Peer-to-Peer
http://www.jxta.org.

[36] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content Addressable Network," in *Proceedings of ACM SIGCOMM 2001*, 2001.

[37] B. Y. Zhao, L. Huang, J. Stribling, A. D. J. S. C. Rhea, and J. D. Kubiatowicz, "Tapestry: A Resilient Global-scale Overlay for Service Deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, January 2004.

[38] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence." Internet Engineering Task Force: RFC 3921, October 2004.

[39] J. Walkerdine, L. Melville, and I. Sommerville, "Designing for Presence within P2P Systems," tech. rep., Computing Department, 2004.

[40] Sun Microsystems Inc.
http://www.sun.com/.

[41] Sun Microsystems, "JXTA v2.3.x: Java Programmer's Guide." White Paper, 2005.

[42] B. Traversat, A. Arora, and M. Abdelaziz, "Project JXTA 2.0: Super-Peer Virtual Network." White Paper, May 2003.

[43] W. S. M. E. R. Harold, *XML in a Nutshell*. O'Reilly Media, Inc., 3 ed., 2004.

[44] R. Droms, "Dynamic Host Configuration Protocol." RFC 1541, October 1993.

[45] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)." RFC 3022, January 2001.

[46] D. B. Chapman and E. D. Zwicky, *Building Internet Firewalls*. OŔeilly, 1 ed., 1995.

[47] JXTA v2.0 Protocols Specification http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.txt.

[48] M. Jan, G. Antoniu, P. Hatcher, and D. A. Noblet, "Performance evaluation of jxta communication layers," in *Proc. Workshop on Global and Peer-to-Peer Computing (GP2PC 2005)*, (Cardiff, UK), Held in conjunction with the 5th IEEE/ACM Int. Symp. on Cluster Computing and the Grid (CCGRID 2005), IEEE TFCC, May 2005.

[49] D. C. Parker, S. A. Collins, and D. C. Cleary, "Building near real-time p2p applications with jxta," in *4th International Scientific Workshop on Global and Peer-to-Peer Computing (GP2PC 2004), IEEE Computer Society*, (Chicago, USA), Held in conjunction with CCGRID 2004, April 2004.

[50] SGE–Sun Grid Engine http://gridengine.sunsource.net.

[51] E. Ribeiro, G. Zerlotini, I. Lopes, V. Ribeiro, A. C. M. A. de Melo, M. E. T. Walter, and M. Motta, "Rede BIOFOCO: A distributed computation of Interpro PFAM, PROSITE and ProDom for protein annotation," in *III Brazilian Workshop on Bioinformatics, WOB2004*, (Brasilia), pp. 65–72, 2004.

[52] E. Ribeiro, G. Zerlotini, I. Lopes, V. Ribeiro, A. C. M. A. de Melo, M. E. T. Walter, and M. Motta, "A distributed computation of Interpro for protein annotation," in *3rd International Information and Telecommunication Technologies Symposium, I2TS2004*, (São Carlos), December 2004.

[53] C. S. Horstmann and G. Cornell, *Core Java, Volume I - Fundamentals*. Sun Microsystems Press and Prentice Hall, 2 ed., 1999.

[54] I. Foster and A. Iamnitchi, "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing.," in *IPTPS*, vol. 2735 of *Lecture Notes in Computer Science*, pp. 118–128, Springer, 2003.