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

Escalonamento de Tarefas no Ambiente Peer-to-Peer do BIOFOCO III

José Carlos Tedesque

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Computação

Orientadora
Prof.ª Maria Emília Machado Telles Walter

Brasília
2010

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

Escalonamento de Tarefas no Ambiente Peer-to-Peer do BIOFOCO III

José Carlos Tedesque

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Computação

Prof.ª Maria Emília Machado Telles Walter (Orientadora)
CIC/UnB

Prof. Fábio Moreira Costa
Universidade Federal de Goiás

Prof. Roberto Coiti Togawa
Embrapa/Recursos Genéticos e Biotecnologia

Prof. Alba C. M. A. Mello
Coordenador do Mestrado em Computação

Brasília, 23 de setembro de 2010

Dedicatória

Dedico esta dissertação às três pessoas mais importantes da minha vida: minha espôsa Soraia e minha filhas, Érica e Luíza.

Agradecimentos

Meus agradecimentos são para todos aqueles que, de alguma forma, contribuíram para a realização deste trabalho. Em especial, para a minha orientadora, Professora Maria Emília, que pacientemente ajudou-me a tilhar os caminhos mais difíceis da pesquisa acadêmica. Faço, também, um agradecimento especial ao mestre Edward, pelas suas horas de esforço e de empenho, sempre prestativo na implementação e alteração do sistema implementado neste trabalho.

Resumo

A utilização de recursos computacionais (estações de trabalho e servidores) instalados em instituições de pesquisa e de ensino localizadas em Brasília permitiu construir em 2006 uma rede Peer-to-Peer, denominada p2pBIOFOCO, que ligava essas instituições. Essa rede é destinada ao processamento de aplicações de Bioinformática. Nesse sistema, dois problemas impactavam bastante a eficiência do sistema: a localização dos *hosts* e o escalonamento de tarefas. Para solucionar o primeiro problema, modificou-se o sistema incluindo-se um novo mecanismo de busca dos *peers*. Desse modo, foi implementada uma rede que utiliza o algoritmo de uma estrutura de armazenamento distribuída, a DHT (Distributed Hash Table) [102], adotando-se o protocolo Kademlia. Este trabalho visa solucionar o problema de escalonamento de tarefas no p2pBIOFOCO, implementando uma estratégia que permita incorporar ao p2pBIOFOCO o uso flexível de escalonadores. Mais especificamente, utilizamos o método WQR como escalonador. Além disso, incluímos um mecanismo de transferência eficiente de dados utilizando o método DP-RR, muito útil para aplicações de Bioinformática. Os experimentos realizados mostraram que o p2pBIOFOCO teve um melhor desempenho com a incorporação desses dois métodos, mostrando que ele pode ser usado para aumentar a capacidade de processamento tanto de anotação quanto de análises comparativas em projetos de sequenciamento de alto desempenho.

Palavras-chave: Bioinformática, *Peer-to-Peer*, Computação Distribuída, DHT, Kademlia, BLAST

Abstract

The use of computer resources (work stations and servers) installed in research and teaching institutions located in Brasilia allowed us to construct in 2006 a network Peer-to-Peer, called p2pBIOFOCO that linked that institutions. This network is intended for processing Bioinformatics applications. In this system, two problems have been impacted enough the efficiency of the system: the location of hosts and the scheduling of tasks. To solve the first problem, we changed the system including a new mechanism for searching peers. Thereby, it was implemented a network which uses the algorithm of a structure of distributed storage, DHT (Distributed Hash Table) [102], adopting the protocol Kademlia. This work seeks to solve the problem of scheduling of tasks in p2pBIOFOCO, implementing a strategy to incorporate the p2pBIOFOCO the flexible use of schedulers. More specifically, we used the method WQR as scheduler. In addition, we have included a efficient data transfer mechanism using the method DP-RR, very useful for Bioinformatics applications. The experiments showed that the p2pBIOFOCO had a better performance with the incorporation of the two methods, showing that it can be used to increase processing capacity both notation as comparative analyzes in sequencing projects of high performance.

Keywords: Bioinformatics, *Peer-to-Peer*, Distributed Computing, DHT, Kademlia, BLAST

Sumário

Lista de Figuras

Capítulo 1

Introdução

Um sistema distribuído, conforme definição de Coulouris e et al [31], é aquele onde os componentes de *hardware* e *software* estão localizados em computadores interconectados em rede e a comunicação e a coordenação entre esses componentes ocorre unicamente através de passagem de mensagens.

A evolução tecnológica do *hardware* e das redes de computadores faz com que os sistemas distribuídos sejam cada vez mais utilizados para resolver problemas de grande complexidade, como a manipulação de dados da ordem de petabytes [24, 37, 47].

Uma das principais motivações para a construção e uso de sistemas distribuídos é a necessidade de compartilhar recursos distribuídos geograficamente [34, 53, 62, 75]. Esses recursos vão desde dispositivos de *hardware* - instrumentação científica, armazenagem, ciclos de processamento (CPU) - até sistemas de *software* - bancos de dados e aplicativos [54]. Além disso, outras vantagens na utilização de um sistema distribuído são: melhor aproveitamento do potencial ocioso de processamento dos componentes do sistema, escalonamento inteligente de tarefas, diminuição dos custos de operação, tolerância a falhas, reutilização de serviços e escalabilidade.

Os sistemas distribuídos apresentam grande heterogeneidade quanto à arquitetura, propósito e escala de operação [31]. Neste contexto, métodos de escalonamento de tarefas são fundamentais para explorar as diferentes configurações e recursos de máquinas interconectadas em um sistema distribuído.

Em particular, sistemas *peer-to-peer* (P2P) [81, 102] são sistemas distribuídos descentralizados, utilizados, principalmente, em aplicações de compartilhamento de arquivos e distribuição de conteúdo utilizando a infra-estrutura pública da Internet. O uso científico de sistemas P2P [47, 98] tem se mostrado uma alternativa bastante promissora na área de sistemas distribuídos.

Por outro lado, a Bioinformática [27] é uma área que nasceu da necessidade dos biólogos interpretarem a enorme e crescente quantidade de dados que eram gerados em laboratório de Biologia Molecular, sobretudo daqueles envolvidos em pesquisa. O principal objetivo da Bioinformática é desenvolver modelos computacionais que apoiem pesquisas em Biologia Molecular, particularmente de projetos de sequenciamento de genomas. Trata-se de um campo interdisciplinar, no qual disciplinas diversas como a Biologia Molecular, a Química Orgânica e Algoritmos fazem parte das questões diárias que são tratadas nessa área.

Na Região Centro-Oeste do Brasil, vários projetos de sequenciamento de genomas estão em desenvolvimento. O Laboratório de Biologia Molecular da Universidade de Brasília (UnB) vem participando ativamente desses projetos desde o seu início em 2001. Dentre os projetos dos quais o Laboratório participou destacamos aqueles referentes ao estudo do genoma de fungos filamentosos (*Paracoccidioides Brasiliensis*), de bactérias (*Anaplasma Marginale*) e de plantas (*Paullinia Cupana*).

O Projeto Genoma Pb [35] teve como foco o fungo *Paracoccidioides brasiliensis* e teve como objetivo o mapeamento do genoma funcional e diferencial entre as formas de micélio e de levedura desse fungo. O *Paracoccidioides brasiliensis* é agente causador de micose endêmica (paracoccidioidomicose) de alta prevalência na América Latina. A patologia é grave e, embora o projeto tenha avançado no conhecimento sobre seus genes transcritos e outras características importantes, não se conhece completamente a biologia do fungo. A transmissão é feita através da inalação de esporos na forma miceliana, que ataca os pulmões, onde sua forma original é transformada em levedura, que é a forma patogênica humana.

O Projeto Genoma Anaplasma [36] estudou a anaplosme bovina, causada pela riquétsia *Anaplasma maginale*, que ocorre em áreas tropicais e subtropicais do mundo e resulta em prejuízos à exploração da atividade agropecuária. O sequenciamento e a categorização funcional do genoma da *A. marginale* possibilitou a descoberta das proteínas envolvidas nos processos biológicos da bactéria, particulamente aqueles relativos à membrana celular.

O Projeto Genoma Funcional e Genética Genômica da *Paullinia cupana* (Guaranazeiro) [41] teve como objetivo geral a geração e a análise de EST's (*expressed sequence tags*) para diferentes condições de cultivo e diferentes cultivares do guaranazeiro, clonando e sequenciando as regiões ativas do genoma, visando melhorar o cultivo do guaraná através dos resultados obtidos pela pesquisa genômica. O guaraná é uma das mais importantes plantas nativas da Região Amazônica. O grande potencial econômico e a importante participação no desenvolvimento sustentável

da região foram determinantes para desenvolver o projeto, tendo em vista que ele traria benefícios sociais, ambientais e econômicos para a exploração do guaraná.

O projeto BIOFOCO (Rede de Pesquisa e Desenvolvimento do Centro-Oeste) representa a união dos interesses de três instituições da Região Centro-Oeste Brasileira, a saber, UnB (Universidade de Brasília), UCB (Universidade Católica de Brasília) e Embrapa/Recursos Genéticos e Biotecnologia, para usar o conhecimento e os recursos dessas instituições com o objetivo de desenvolver pesquisas na área de Bioinformática.

O projeto BIOFOCO I [10] promoveu a integração das instituições através do compartilhamento de conhecimento e de sistemas.

No BIOFOCO II foi construída uma arquitetura distribuída baseada no conceito *peer-to-peer* para dar suporte aos trabalhos dos pesquisadores e professores dessas três instituições, consolidando a fase inicial de integração.

O projeto BIOFOCO III [11] dá continuidade aos trabalhos realizados nos projetos BIOFOCO I e II, contemplando as seguintes instituições: Universidade de Brasília (UnB/CIC e IB), Universidade Federal do Mato Grosso do Sul (UFMS/DCT), Universidade Federal de Goiás (UFG/Informática e Biologia, Campus de Goiânia, e Computação, Campus de Catalao) e Embrapa/Recursos Genéticos e Biotecnologia. A ênfase está no desenvolvimento de *software* para o uso nas análises genômicas em ambiente computacional distribuído.

As três áreas priorizadas no projeto são algoritmos (genômica comparativa, identificação de RNAs não-codificadores e redes metabólicas), sistemas distribuídos e *hardware* configurável.

O problema do escalonamento de tarefas no p2pBIOFOCO tem impacto relevante no seu desempenho. Além disso, há atualmente o problema de transferência de grandes volumes de dados, devido ao tamanho dos arquivo usados em bioinformática.

Neste contexto, este trabalho tem os seguintes objetivos:

- implementar um novo método de escalonamento de tarefas no p2pBIOFOCO, de forma que novos métodos possam ser incluídos facilmente e o sistema possa escolher o método mais eficiente dinamicamente;
- implementar no p2pBIOFOCO um método de transferência de grandes volumes de dados;
- realizar experimentos utilizando uma ferramenta de Bioinformática, o BLAST [2], com dados reais do fungo *P. brasiliensis*;

- comparar os resultados obtidos com o novo escalonador com os resultados do escalonador da versão anterior do p2pBIOFOCO;
- comparar os tempos de transferência de grande volume de dados (arquivo nr/GenBank), resultantes da implementação do método de transferência.

Este trabalho foi dividido da seguinte forma. No Capítulo 2, descrevemos alguns conceitos básicos de Biologia Molecular, necessários ao entendimento desta dissertação. No Capítulo 3, descrevemos métodos para o sequenciamento de genomas. No Capítulo 4, apresentamos conceitos básicos de escalonamento de tarefas e, em seguida, descrevemos trabalhos correlatos, principalmente aqueles ligados ao escalonamento de tarefas independentes em sistemas distribuídos. Desses, utilizamos um algoritmo que foi incluído no p2pBIOFOCO. A arquitetura atual do p2pBIOFOCO é apresentada no Capítulo 5. Os métodos de escalonamento de tarefas e de transferência de dados do p2pBIOFOCO e os detalhes das implementações são descritos no Capítulo 6. No capitulo 7, discutimos os resultados obtidos com a inclusão do escalonador, comparando-o com a primeira versão do p2pBIOFOCO e mostrando os resultados obtidos com a execução do método de transferência de dados. Finalmente, no Capítulo 8, concluímos este trabalho e propomos trabalhos futuros.

Capítulo 2

Conceitos Básicos de Biologia Molecular

Pesquisas em Biologia Molecular [97] têm como objetivo o entendimento da estrutura e da função das proteínas e dos ácidos nucleicos. Essas moléculas são centrais na química da vida, sendo as proteínas responsáveis pelas funções vitais realizadas por um organismo, enquanto os ácidos nucleicos codificam a informação necessária para a produção de proteínas e passam suas informações entre as sucessivas gerações.

Assim, neste capítulo descrevemos os conceitos básicos de Biologia Molecular e de Bioinformática necessários para a compreensão deste trabalho. Na Seção 2.1 discorremos sobre as células e seu processo de replicação. Na Seção 2.2 apresentamos um resumo sobre a composição das proteínas. Os ácidos nucleicos e os genes são apresentados nas seções 2.3 e 2.4, respectivamente. Na Seção 2.5, discutimos as principais ideias relacionadas ao Dogma Central da Bioinformática, ou o processo da síntese de proteínas.

2.1 Células

As células, no mesmo organismo ou em diferentes organismos, variam significativamente em forma, tamanho e função, compartilhando características que são essenciais para a vida dos seres vivos [27] (Figura 2.1). A célula é construída a partir de componentes moleculares, que podem ser identificados através de estruturas em três dimensões (3D) de várias formas.

Em seguida, fazemos um descrição sucinta das funções dos componentes de uma célula, identificados na Figura 2.1:

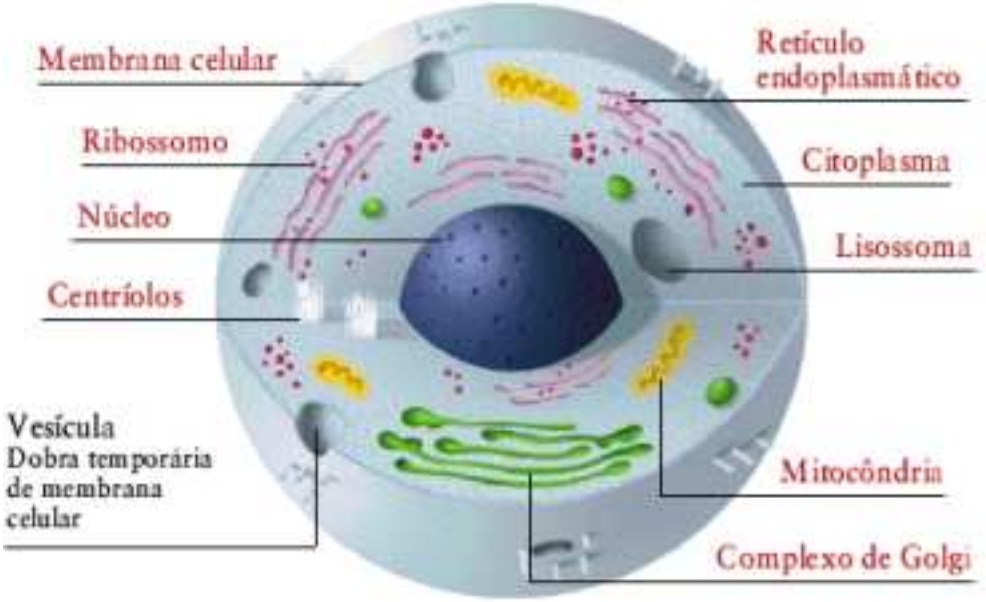

Figura 2.1: Uma célula e alguns de seus componentes principais.

- membrana celular: age como um filtro que controla o acesso de elementos exteriores e a saída de determinados tipos de moléculas;
- ribossomo: componente responsável pela síntese de proteínas;
- núcleo: sua principal função é controlar e mediar a replicação do DNA durante o ciclo celular;
- centríolo: estrutura cilíndrica encontrada aos pares nas células que desempenha papel fundamental na divisão celular;
- vesícula dobra temporária: transporta, armazena ou digere produtos ou resíduos celulares;
- retículo endoplasmático: transporte de proteínas;
- citoplasma: é o espaço celular existente entre a membrana plasmática e o núcleo da célula, no qual encontramos diversas estruturas celulares (organelas);
- lisossoma: digestão de organismos estranhos à composição da célula (por exemplo, bactérias) e de componentes que não são mais necessários à célula;
- mitocôndria: produção de ATP (adenosina trifosfato) e regulação do metabolismo celular;
- complexo de Golgi: síntese de um grande número de diferentes moléculas.

Quanto ao tamanho, os componenetes de uma célula podem variar de estruturas muito pequenas (em escala molecular) como, por exemplo, o DNA, até estruturas relativamente grandes (proteínas que constituem a membrana celular).

Os componentes da células são formados por vários tipos de moléculas. Os movimentos realizados pelas moléculas no interior da célula implicam em várias interações, que resultam em combinações para a formação de novas moléculas, com novas estruturas 3D e, consequentemente, novas formas espaciais. Em determinadas situações, a interação provoca a fragmentação da molécula em novos componentes. Esse movimento é consequência das interações das cargas dos átomos constituintes das moléculas. Essas interações ocorrem frequentemente relacionadas à forma e à localização das moléculas no interior da célula.

O crescimento da célula é um processo decorrente da disponibilidade de moléculas externas (nutrientes) que penetram pela membrana celular para o interior da célula, participando de interações com as moléculas existentes no ambiente intracelular. Desse modo, a capacidade de crescimento das moléculas depende da absorção de nutrientes externos.

As células reproduzem-se através da metabolização de um número suficiente de componentes moleculares, existentes no seu interior, para a produção de uma cópia da célula original com a capacidade de atuar independentemente. Quando as células se duplicam, pequenas alterações podem produzir mudanças na estrutura de seus componentes. Se tais mudanças acompanham o ciclo de vida da célula, podem ser incorporadas às futuras gerações dessas células.

2.2 Proteínas

Uma proteína é formada por uma cadeia de moléculas mais simples chamadas aminoácidos [97]. Cada aminoácido tem um átomo central de carbono, denominado de C_α. Um átomo de C_α é ligado a um átomo de higrogênio, a um grupo amina(NH_2), a uma carboxila ($COOH$) e a uma *cadeia lateral* (Figura 2.2). A cadeia lateral distingue um aminoácido de outro. As cadeias laterais podem ser tão simples como um átomo de hidrogênio presente no aminoácido alanina ou complexas como a presente no aminoácido triptofano (Figura 2.3).

Na natureza existem 20 diferentes tipos de aminoácidos, listados na tabela 2.1. Estes são os componentes mais comuns das proteínas, considerando que podem existir na composição de alguma proteína aminoácidos fora dessa classificação.

Na constituição da proteína, os aminoácidos são unidos de acordo com ligações peptídicas, o que caracteriza as proteínas como cadeias peptídicas (Figura 2.4). Na

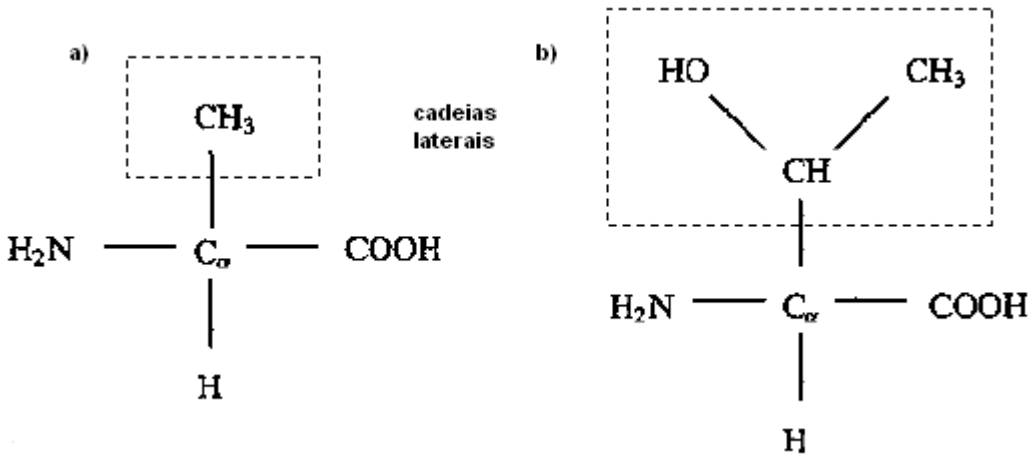

Figura 2.2: Duas moléculas de aminoácidos. a) Alanina. b) Triptofano.

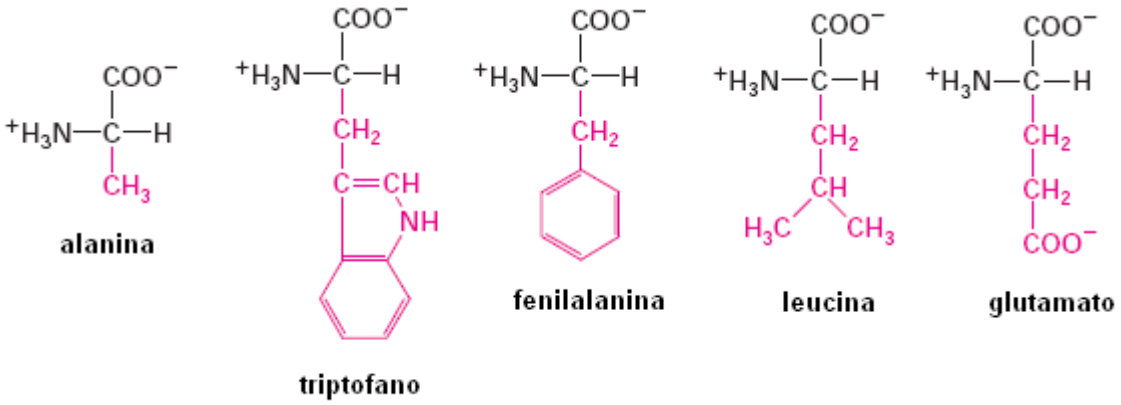

Figura 2.3: Exemplos de aminoácidos.

ligação peptídica, o átomo de carbono pertencente ao grupo carboxila do aminoácido A_i é ligado ao átomo de nitrogêncio do aminoácido A_{i+1} do grupo amina. Na realização dessa ligação, uma molécula de água é liberada em decorrênca da ligação dos átomos de oxigênio e hidrogênio do grupo carboxila com o hidrogênio do grupo amina. Então, o que realmente é produzido dentro de uma cadeia polipeptídica é um resíduo do aminoácido original. Proteínas comuns possuem cerca de 300 resíduos, com a existência de proteínas com menos de 100 (cem) resíduos até proteínas com 5.000 (cinco mil) resíduos.

A ligação peptídica confere a toda proteína uma cadeia principal, em razão de possuir repetições do bloco $-N-C\alpha-(CO)-$. Para todo átomo de C_α existe uma cadeia lateral. Como resultado da presença de um grupo amina em uma extremidade de uma cadeia principal e de um grupo carboxila na outra extremidade, pode-se distinguir ambos os finais de uma cadeia polipeptídica e, desse modo, associar a ela

Tabela 2.1: Os aminoácidos presentes na natureza

Nome	Símbolo	Abreviação
Glicina	Gly, Gli	G
Alanina	Ala	A
Leucina	Leu	L
Valina	Val	V
Isoleucina	Ile	I
Prolina	Pro	P
Fenilalanina	Phe ou Fen	F
Serina	Ser	S
Treonina	Thr, The	T
Cisteína	Cys, Cis	C
Tirosina	Tyr, Tir	Y
Asparagina	Asn	N
Glutamina	Glu	Q
Aspartato ou ácido aspártico	Asp	D
Glutamato ou ácido glutamático	Glu	E
Arginina	Arg	R
Lisina	Lys, Lis	K
Histidina	His	H
Triptofano	Trp, Tri	W
Metionina	Met	M

uma direção.

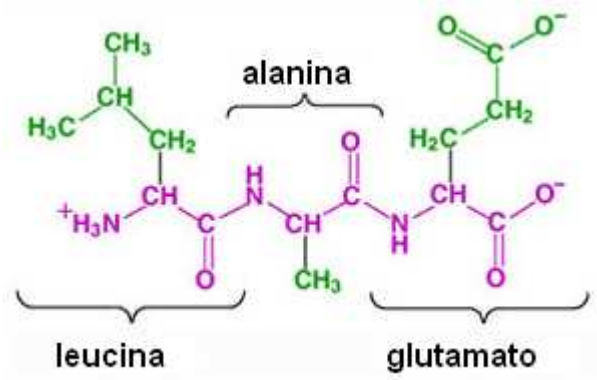

Figura 2.4: Cadeia peptídica com três aminoácidos.

As proteínas apresentam três tipos de estruturas espaciais - secundária, terciária e quaternária - além da estrutura primária formada pelos resíduos. As interações locais da cadeia principal formam a estrutura secundária da proteína, resultando em estruturas locais helicoidais. A estrutura terciária refere-se à formação de estruturas espaciais a partir das estruturas secundárias e, finalmente, na estrutura quaternária, visualiza-se a junção de diferentes grupos de proteínas (Figura 2.5).

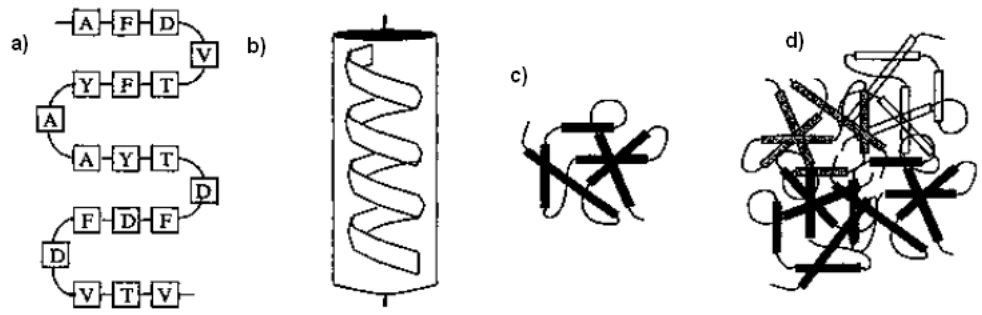

Figura 2.5: Estruturas Espaciais da Proteína: a) primária, b) secundária, c) terciária e d) quaternária [97].

2.3 Ácidos Nucleicos

Os organismos vivos possuem dois tipos de ácidos nucleicos: o DNA (ácido desoxirribonucleico) e o RNA (ácido ribonucleico). A estrutura molecular do DNA foi desvendada a partir do trabalho seminal de Watson e Crick [109], que descreveram o famoso formato de dupla hélice do DNA, formada por duas cadeias conforme a Figura 2.6. Cada cadeia é composta pela repetição de uma mesma unidade básica e por uma cadeia dupla, que gira perpendicularmente formando uma hélice em torno de um eixo.

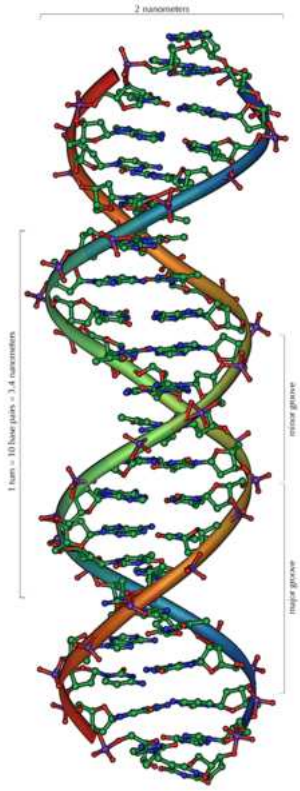

Figura 2.6: Estrutura Espacial do DNA: a dupla hélice.

Uma unidade básica de uma molécula de DNA é composta por moléculas de açúcar, fosfato e uma base, sendo essa composição chamada de **nucleotídeo** (Figura 2.7). As bases estão no interior da hélice enquanto os fosfatos estão localizados exteriormente. Existem quatro tipos de bases: **adenina (A), guanina (G), citosina (C) e timina (T)**. As bases A e G pertencem ao grupo de substâncias chamadas purinas, enquanto as bases C e T pertencem ao grupo das pirimidinas.

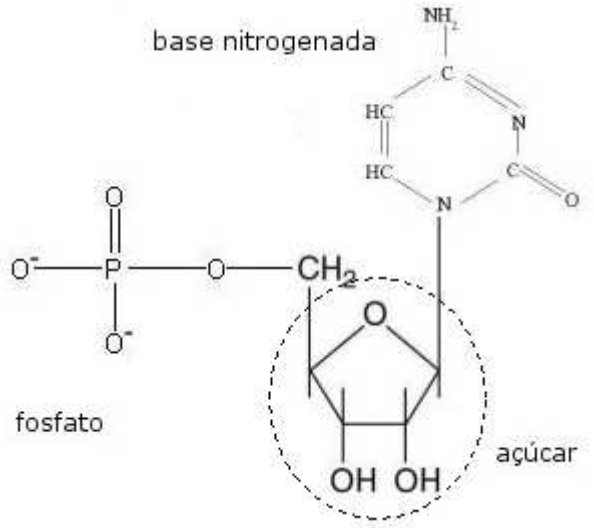

Figura 2.7: Nucleotideo

A formação de pares de bases entre as cadeias surge a partir da interação entre bases complementares, que estão sujeitas a determinadas reações químicas. Assim, as bases complementares do DNA são formadas pelos pares A≡T e C≡G. As cadeias possuem orientações diferentes em decorrência da ligação dos átomos de carbono de um açúcar (carbono 3') de uma cadeia com o de outro da outra cadeia (carbono 5').

Para avaliar o tamanho de um pedaço de DNA, foi estabelecida uma unidade de tamanho baseada no número de pares de bases de Watson-Crick, conhecida como **bp**. Dizemos, então, que um pedaço de DNA tem 100.000 bp. Em função da estrutura descrita, é possível inferir a sequência de bases de uma cadeia dada a sequência da outra. Tal operação é denominada de *complementação reversa*, constituindo-se no mecanismo básico para replicação das células (Figura 2.8).

Como os nucleotídeos diferem pelas bases, podemos representar cada cadeia pelas suas bases (Figuras 2.9 e 2.10). Esta representação é muito útil em computação, pois podemos tratar o DNA como cadeia de caracteres (*strings*).

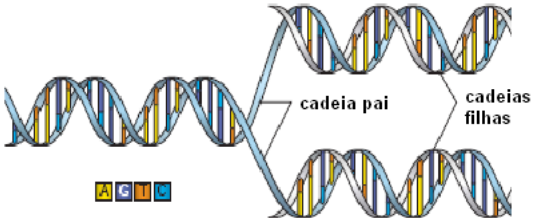

Figura 2.8: Exemplo de Cadeias Complementares

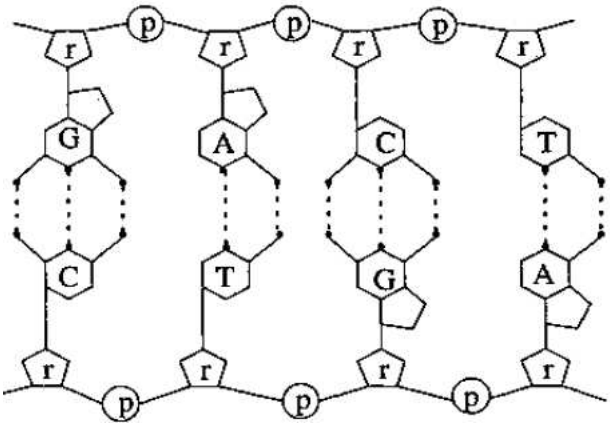

Figura 2.9: Esquema Molecular de uma Cadeia de DNA.

5′ ··· TACTGAA ··· 3′
3′ ··· ATGACTT ··· 5′

Figura 2.10: Cadeias podem ser representadas pelas suas bases.

2.4 Genes

As células de um organismo possuem longas moléculas de DNA chamadas de cromossomos. Certas regiões contínuas do DNA possuem informação codificada para construir proteínas, enquanto outras não possuem tal característica. Cada tipo diferente de proteína corresponde a uma região contígua do DNA. Tais regiões são denominadas *genes*.

Como uma proteína é uma cadeia de aminoácidos, para explicitá-la basta listar cada aminoácido que a forma. Cada aminoácido é formado por uma tripla de nucleotídeos (códons), que especifica cada aminoácido. O conjunto de cada tripla e seu respectivo aminoácido é chamado de *código genético*, mostrado na tabela 2.2.

Tabela 2.2: Os aminoácidos presentes na natureza

	U	**C**	**A**	**G**	
U	UUU-Phe	UCU-Ser	UAU-Tyr	UGU-Cys	U
	UUC-Phe	UCC-Ser	UAC-Tyr	UGC-Cys	C
	UUA-Leu	UCA-Ser	UAA-stop	UGA-stop	A
	UUG-Leu	UCG-Ser	UAG-stop	UGG-Trp	G
C	CUU-Leu	CCU-Pro	CAU-His	CGU-Arg	U
	CUC-Leu	CCC-Pro	CAC-His	CGC-Arg	C
	CUA-Leu	CCA-Pro	CAA-Gln	CGA-Arg	A
	CUG-Leu	CCG-Pro	CAG-Gln	CGG-Arg	G
A	AUU-Ile	ACU-Thr	AAU-Asn	AGU-Ser	U
	AUC-Ile	ACC-Thr	AAC-Asn	AGC-Ser	C
	AUA-Ile	ACA-Thr	AAA-Lys	AGA-Arg	A
	AUG-Met	ACG-Thr	AAG-Lys	AGG-Arg	G
G	GUU-Val	GCU-Ala	GAU-Asp	GGU-Gly	U
	GUC-Val	GCC-Ala	GAC-Asp	GGC-Gly	C
	GUA-Val	GCA-Ala	GAA-Glu	GGA-Gly	A
	GUG-Val	GCG-Ala	GAG-Glu	GGG-Gly	G

2.5 Dogma Central da Biologia Molecular

Os biólogos denominam de Dogma Central da Biologia Molecular o modelo pelo qual a partir do DNA são sintetizadas as proteínas. Conforme Crick [32], o dogma estabelece que a transferência de informação de um ácido nucleico para um outro ácido nucleico, ou de um ácido nucleico para uma proteína pode ser possível, mas a transferência de proteína para proteína, ou de proteína para ácido nucleico é impossível. *Informação*, neste caso, refere-se à sequência correta de bases presentes em um ácido nucleico ou de resíduos de aminoácidos presentes em um proteína. Na Figura 2.11 visualizamos uma esquematização do processo.

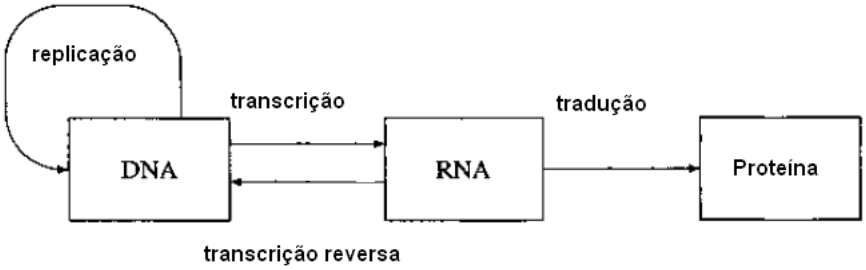

Figura 2.11: Dogma Central da Biologia Molecular

2.5.1 Replicação do DNA

A replicação do DNA é um processo intrincado, que envolve a ação coordenada de diferentes proteínas no ribossomo da célula. A dupla hélice de DNA sofre a ação dessas proteínas e divide-se em duas fitas, que serão os moldes para a construção de duas novas cadeias de DNA. Como metade da fita refere-se ao DNA original, a replicação é rotulada de replicação semi-conservativa (Figura 2.12).

A replicação do DNA é feita por diversas proteínas, cujas funções são descritas em seguida:

- *helicase*: proteína em forma de anel responsável pela divisão da dupla hélice em duas cadeias;
- *SSB (single strand binding)*: proteína que se liga às fitas de DNA produzidas na divisão, com o intutito de evitar o realinhamento, produzindo a mesma molécula de DNA;
- *primase*: enzima responsável pela síntese do *iniciador*[1];

[1]segmento da cadeia nova que permite a ação iniciadora da DNA polimerase

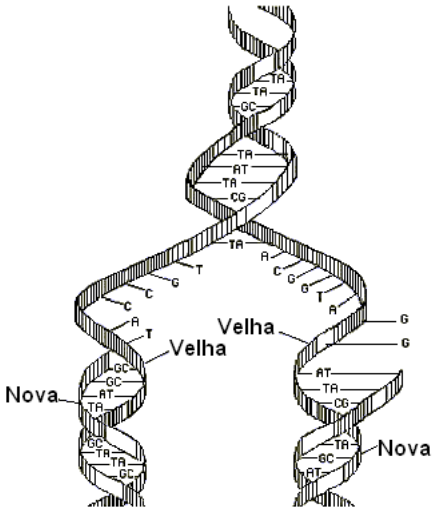

Figura 2.12: Replicação semi-conservativa do DNA

- *polimerase*: é responsável pela polimerização[2] das nova fitas de DNA, exercendo uma função corretora;
- *sliding clamp*: proteína que evita a dissociação da polimerase da nova fita de DNA;
- *RNase H*: enzima responsável pela retirada das bases iniciadoras de RNA's (*primers*) da replicação;
- *ligase*: é um tipo especial de enzima de ligação que promove a união das duas fitas de DNA, que estão separadas por ação das outras proteínas envolvidas no processo de replicação.

2.5.2 Transcrição

A transcrição (Figura 2.13) usa informações armazenadas no DNA. O mecanismo celular reconhece o início de um gene ou conjunto de genes por meio de uma região próxima ao gene, designada região iniciadora (*promoter*), que facilita o mecanismo de transcrição. O códon **AUG** (metionina) indica o início de um gene para os mecanismos celulares.

Tendo reconhecido o início do um gene ou um conjunto de genes, tem-se o começo da produção de uma cópia do gene por meio da formação de uma molécula de RNA, chamada de RNA mensageiro (mRNA), que terá a mesma sequência de uma das fitas da dupla hélice do DNA, mas com a substituição da base **U** pela base

[2]reação química em que moléculas menores se agrupam para formação de moléculas mais longas

T. O mRNA será utilizado por estruturas celulares chamadas ribossomos para a produção de proteína.

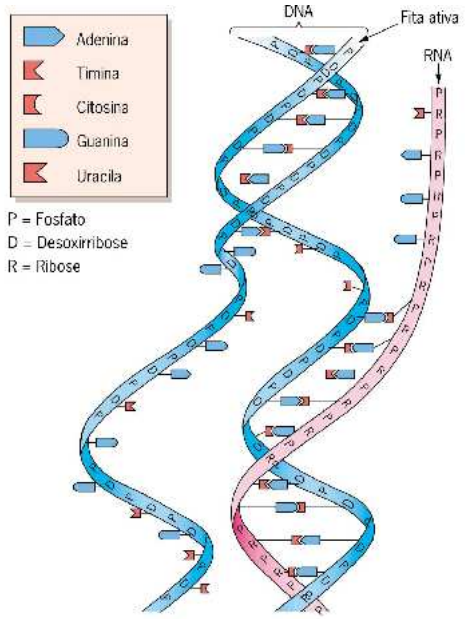

Figura 2.13: Processo de Transcrição

2.5.3 Tradução

Os ribossomos, assim como uma linha de montagem, usam a informação trazida pelo mRNA e outro tipo de molécula de RNA, chamada de RNA transportador (tRNA), para a produção de proteínas. Os tRNAs são as moléculas que de fato implementam o código genético em uma proteína. Eles fazem a ligação entre um códon e o aminoácido que esse códon codifica. Cada molécula de tRNA tem, de um lado, alta afinidade com determinado códon, e de outro lado, uma formação que facilita a ligação para o correspondente aminoácido.

Enquanto o mRNA trafega pelo interior do ribossomo, um tRNA encontra o códon correspondente ligando-o ao aminoácido. A posição tridimensional de todas as moléculas neste momento é tal que, enquanto um tRNA liga-se ao seu códon, seu aminoácido ajusta-se ao local próximo do aminoácido anterior na cadeia proteica. Desse modo, uma proteína é construída resíduo a resíduo. Finalmente, quando um códon rotulado como STOP aparece, nenhum tRNA é associado a ele e a síntese, então, termina.

2.6 Resumo do Capítulo

Neste catítulo discutimos os conceitos básicos de Biologia Molecular, particularmente as reações identificadas pelos biólogos no Dogma Central da Biologia Molecular. Este conhecimento permitiu desenvolver os projetos de sequenciamento de genoma, que usam partes do mRNA ou partes do próprio DNA como informações básicas. Além disso, como podemos representar o DNA como uma cadeia de caracteres, veremos, no capítulo seguinte, que algoritmos de comparação de padrões (*string matching*) são importantes para o reconhecimento e análise das moléculas de DNA e mRNA.

Capítulo 3

Métodos de Sequenciamento de Genomas e Bioinformática

Neste capítulo descrevemos os métodos existentes para o sequenciamento de genomas e os conceitos básicos inerentes à área de Bioinformática. Na Seção 3.1, apresentamos os métodos clássicos de sequenciamento, especialmente o método de Sanger. Na Seção 3.2, o texto é dedicado aos algoritmos de alinhamento e a ferramentas de Bioinformática, em particular o BLAST, ora utilizado neste trabalho. Para finalizar, na Seção3.3 fazemos uma breve discussão sobre o sequenciamento de alto desempenho.

3.1 Métodos Clássicos de Sequenciamento do DNA

Saccone [93] aponta duas técnicas básicas de sequenciamento: degradação química (Maxam e Gilbert) e a terminação da cadeia enzimática (Sanger et al). Os primeiros métodos de sequenciamento de proteínas foram desenvolvido por Sanger e Tuppy na década de 50, resultando no sequenciamento de várias famílias de proteínas. Na década de 60, Dayhoff et al foram os primeiros a construírem bases de dados das sequências, originando um banco de sequências proteicas conhecido como PIR (*Protein Information Resource*). Este banco de sequências foi organizado classificando as proteínas em famílias e superfamílias, com base no grau de similaridade das sequências.

As primeiras sequências de DNA foram coletadas por Walter Goad em uma base de dados do GenBank [50]. A entrada da sequência incluía o nome do arquivo e os arquivos de sequências de proteínas e do DNA. Esses arquivos eram expandidos para incluir mais informações sobre a sequência, tais como função, mutação, pro-

teínas codificadas e referências. Desde então, o número de entradas das sequências nas bases de dados do GenBank não para de crescer (Figura 3.1).

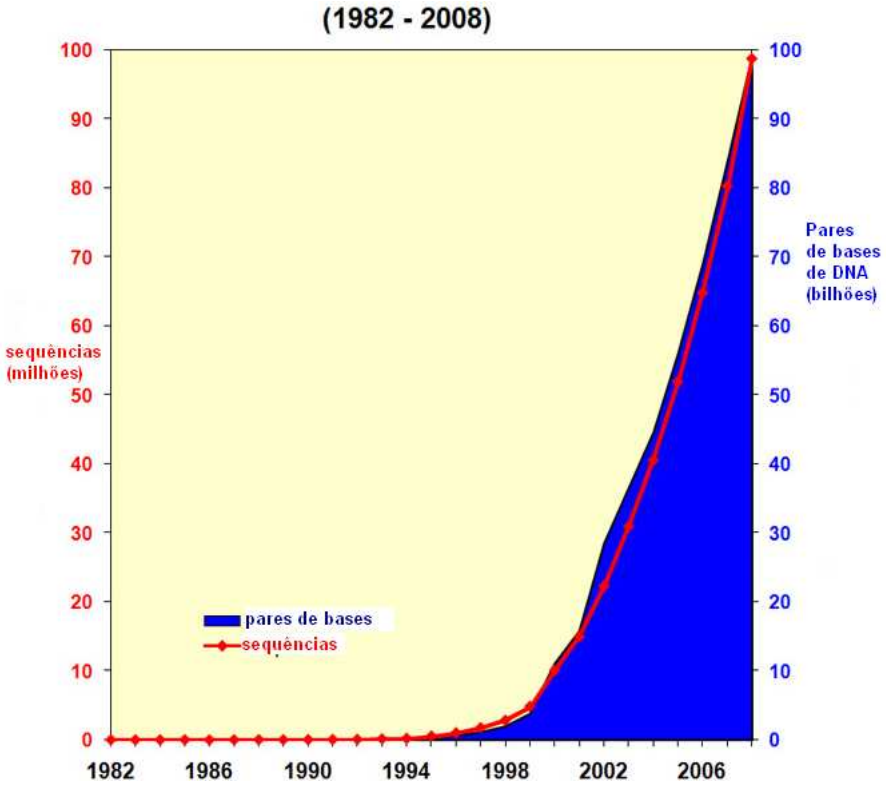

Figura 3.1: Gráfico de Crescimento do GenBank [49]

O desenvolvimento de páginas Web facilitou o acesso às bases de sequências. Como exemplo, pode-se citar o programa desenvolvido por D. Benson et al para o NCBI [8]. Sua execução baseia-se na procura rápida através de bases de dados indexadas pelas entradas de pesquisa do biólogo. Logo após, um programa chamado ENTREZ foi disponibilizado pelo NCBI. Sua principal característica é prover uma interface fácil de usar e com flexibilidade para a pesquisa das sequências nas bases de dados.

3.1.1 O Método de Sanger

O método desenvolvido por Fred Sanger forma a base do ciclo de reações automatizadas de sequenciamento, que foram utilizadas intensamente até o final do século 20 ([82], [23]). Corantes fluorescentes são adicionados às reações e um feixe de *laser* é disparado dentro da máquina que realiza o sequenciamento para analisar os fragmentos de DNA produzidos.

Para iniciar o método, toma-se uma amostra de DNA que se deseja sequenciar, denominado de DNA iniciador, complementar ao DNA que está sendo sequenciado,

uma enzima chamada DNA polimerase e quatro nucleotídeos. Adiciona-se aos elementos citados um segundo tipo de nucleotídio chamado dideoxinucleotídio [3], que pode ser reconhecido pelo sequenciador de DNA. Para iniciar a reação de sequenciamento, a mistura é aquecida e a amostra de DNA é separada em duas sequências complementares.

Então, submete-se a mistura a um resfriamento de maneira que o DNA iniciador encontre a sua sequência complementar na amostra. Finalmente, aplica-se uma pequena elevação de temperatura que permite que a enzima de polimerase ligue-se ao DNA e crie uma nova amostra. A sequência dessa nova amostra é complementar à amostra original. A aplicação da enzima não faz distinção entre dNTPs (deoxinucleotídeos trifosfatos) e didNTPs (dideoxinucleotídeos trifosfatos), e cada vez que um didNTP é incorporado a síntese do DNA para.

Considerando que bilhões de moléculas de DNA estão presentes em um experimento, a amostra pode ser terminada em qualquer posição. Como resultado tem-se a coleta de vários tamanhos diferentes de amostras de DNA. A mistura é, então, transferida para uma lâmina de gel poliacrilamida (gel de sequenciamento) que é colocada no sequenciador de DNA para eletroforese [4] e análise.

Em sequenciadores automáticos, os fragmentos de DNA são migrados de acordo com o tamanho e cada um é detectado pela aplicação de um raio de *laser* na base do gel. Em seguida, cada tipo de dideoxinucleotídeo emite uma luz colorida com determinado tamanho de onda, que é gravado como uma faixa colorida na imagem simulada do gel (Figura 3.2).

O programa do sequenciador automático, com base nos dados gerados, interpreta as informações e emite um gráfico de curvas coloridas com picos representando cada letra (A, C, G e T) na sequência analisada (as letras N representam que um determinado nucleotídio não foi reconhecido). Finalmente, na imagem simulada do gel, os fragmentos são identificados de cima para baixo, começando com os fragmentos de tamanho menor (fragmentos ordenados pelo tamanho). O sequenciador automático gera então gráficos, denominados de eletroferogramas (Figura 3.3).

3.1.2 *Pipelines* Computacionais - Método de Sanger

Alvarez [3] descreve três fases básicas em um projeto de sequenciamento: a submissão, a montagem e a anotação.

[3] nucleotídeo sintético no qual falta um grupo 3'-hidroxila, impedindo a ligação 3' - 5' necessária ao alongamento da cadeia de DNA

[4] processo pelo qual uma mistura complexa de proteínas é fracionada em diferentes bandas de proteínas arranjadas de acordo com sua massa molecular

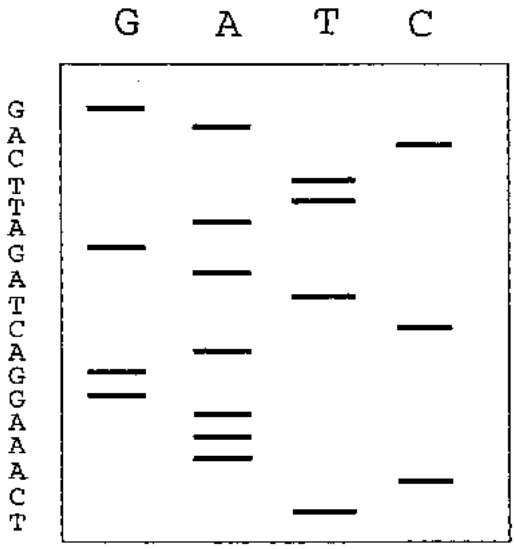

Figura 3.2: Visão esquemática do filme produzido pela gel eletroforese. Bases individuais de DNA podem ser identificadas em cada uma das quatro colunas, ficando as bases mais pesadas próximas ao lado esquerdo e as mais leves ao lado direito. Além disso, fragmentos mais curtos deixam suas marcas mais próximas ao topo, enquanto fragmentos mais longos efetuam marcas mais próximas à base [97].

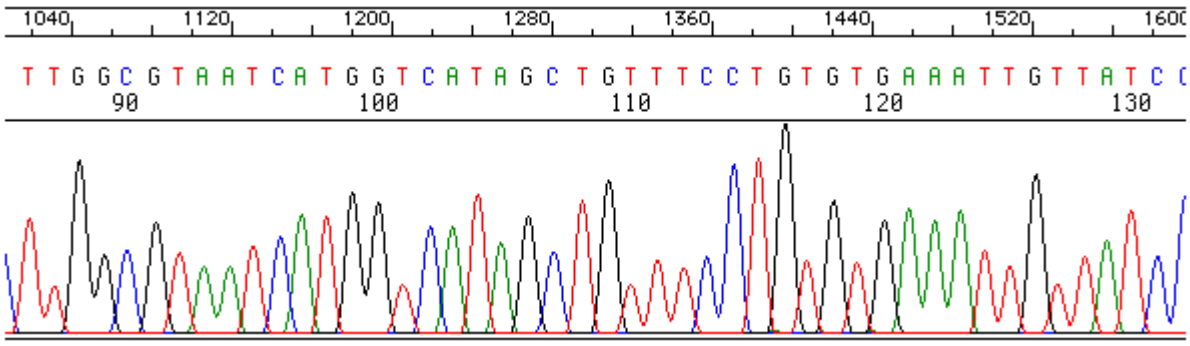

Figura 3.3: Gráfico gerado pelo sequenciador

Submissão

A submissão consiste do recebimento de dados gerados pelo processo de sequenciamento e do armazenamento desses dados em um formato particular para processamento em computadores. Com a utilização do método de Sanger, mencionado na seção anterior, os arquivos com os resultados do sequenciamento são processados pelo programa *Phred* [42], que traduz os dados presentes no arquivo em uma sequência de letras contendo as bases identificadas e as probabilidades de erros associadas a determinação de cada base. Em seguida, utiliza-se o programa *Phd2Fasta* [59] em cada arquivo processado pelo *Phred* para a criação de dois arquivos no formato FASTA: um contendo a sequência de bases nitrogenadas e outro

contendo os valores das probabilidades de erro.

As probabilidades de erro são determinantes para o projeto de sequenciamento, pois sequências com alta probabilidade de erro são desconsideradas para evitar a ocorrência de resultados incorretos nas próximas fases. Cada projeto, portanto, tem uma probabilidade de erro máxima, pela qual são filtradas as sequências que permanecerão sob análise. Assim, evita-se o consumo desnecessário de recursos computacionais e o consequente aumento dos custos do projeto.

Montagem

A etapa de montagem é definida pela utilização de programas que tentam ”unir” os fragmentos de DNA, obtidos durante o processo de sequenciamento, com o objetivo de recriar as sequências de DNA originais. As sequências de DNA obtidas pela ”união” de um ou mais fragmentos são chamadas de *contigs*, enquanto as que não puderam ser agrupadas são chamadas de *singlets*.

Neste trabalho, são utilizados bancos de dados contendo sequências de entrada no formato FASTA [43]. O formato é definido por uma linha que começa com o símbolo ”>” seguido por informações sobre os dados e pelas linhas contendo os dados da sequência propriamente dita (Figura 3.4):

```
>human_CFTR
ATATTTGAAAGCTGTGTCTGTAAACTGATGGCTAACAAAACTAGGATTTTGGTCACTTC
TAAAATGGAACATTTAAAGAAAGCTGACAAAATATTAATTTTGAATGAAGGTAGCAGCT
ATTTTTATGGGACATTTTCAGAACTCCAAAATCTACAGCCAGACTTTAGCTCAAAACTC
ATGGGATGTGATTCTTTCGACCAATTTAGTGCAGAAAGAAGAAATTCAATCCTAACTGA
GACCTTACACCGTTTCTCATTAGAAGGAGATGCTCCTGTCTCCTGGACAGAAACCAATC
TTTTAAACAGACTGGAGAGTTTGGGGAAAAAAGGAAGAATTCTATTCTCAATCCAATCA
ACTCTATACGAAAATTTTCCATTGTGCAAAAGACTCCCTTACAAATGAATGGCATCGAA
GAGGATTCTGATGAGCCTTTAGAGAGAAGGCTGTCCTTAGTACCAGATTCTGAGCAGGG
AGAGGCGATACTGCCTCGCATCAGCGTGATCAGCACTGGCCCCACGCTTCAGGCACGAA
GGAGGCAGTCTGTCCTGAACCTGATGACACACTCAGTTAACCAAGGTCAGAACATTCAC
CGAAAGACAACAGCATCCACACGAAAAGTGTCACTGGCCCCTCAGGCAAACTTGACTGA
ACTGGATATATATTCAAGAAGGTTATCTCAAGAAACTGGCTTGGAAATAAGTGAAGAAA
TTAACGAAGAAGACTTAAAGG
```

Figura 3.4: Sequência parcial do DNA de seres humanos para a fibrose cística (CFTR), em formato FASTA [18]

Programas de computadores são utilizados na etapa de montagem para a geração de arquivos no formato FASTA. Um arquivo contém todos os *singlets* identificados. Outro arquivo contém dados sobre a composição dos *contigs* e suas sequências, juntamente com informações gerais sobre como foi feita a montagem dos fragmentos do DNA.

Anotação

Finalmente, na fase de anotação, são estudadas as diversas funções biológicas do DNA sequenciado, podendo ser descobertos novos genes. Esta fase é subdividida em duas partes: a anotação automática e a anotação manual. Com a anotação automática, tenta-se inferir as funções biológicas das sequências de DNA comparando-as a banco de dados com sequências de funções conhecidas. Uma ferramenta de bioinformática muito utilizada nesta fase é o BLAST (vide seção 3.1.4). A anotação manual é o último passo do projeto de sequenciamento, no qual os biólogos podem confirmar, alterar ou recusar as informações geradas pela anotação automática. Experimentos importantes relacionados ao organismo objeto da pesquisa podem ser realizados com base nas informações obtidas pelas fases do *pipeline*.

3.2 Algoritmos de Alinhamento

3.2.1 Sobre Bioinformática

A Bioinformática é um ramo de pesquisa multidisciplinar que essencialmente usa ferramentas da Matemática e da Ciência da Computação para a solução de problemas em Biologia Molecular [73, 30]. Esse novo campo da ciência foi estabelecido de acordo com duas premissas que permitiram de fato a integração da Biologia Molecular com as técnicas da Computação: os dados utilizados pelos cientistas, principalmente os gerados pelas sequências de aminoácidos e nucleotídeos, tornaram-se *discretos* e surgiu, também, a necessidade de análise da gigantesca quantidade de informações geradas pelos vários projetos genômicos de larga escala ao redor do mundo [72, 106, 28, 44, 100].

Para a realização de trabalhos na Bioinformática são utilizados bancos de dados genômicos de diversos organismos [8] e cujos tamanhos vão desde poucos megabytes até terabytes. A existência de modelos computacionais eficientes para tratar esses enormes volumes de dados é, desse modo, fundamental para o desenvolvimento e o crescimento das pesquisas na área de Biologia Molecular.

De acordo com Luscombe [76] são três os objetivos da Bioinformática:

1. em sua forma mais simples, a Bioinformática emprega técnicas de organização dos dados que permitem aos pesquisadores o acesso às informações existentes, permitindo, também, a submissão de novos registros enquanto eles são produzidos;

2. o desenvolvimento de ferramentas e recursos que ajudem na análise dos dados. Por exemplo, uma sequência de proteína em particular pode ser comparada com sequências caracterizadas anteriormente, necessitando de programas que façam uma comparação biológica entre as sequências e não somente uma comparação textual. Duas ferramentas que servem como exemplo para esse tipo de pesquisa são o *Blast* e o *Exonerate*;

3. a utilização das ferramentas citadas para a interpretação dos resultados e para a atribuição de significado biológico aos dados coletados.

Na busca dos objetivos acima citados, é gerada uma enorme quantidade de dados, que devem ser armazenados e disponibilizados para novas pesquisas, instituindo um ciclo de crescimento exponencial no tamanho das estruturas que armazenam essas informações.

A análise de sequências é a tarefa mais comum executada na Bioinformática. Seu trabalho consiste em determinar quais partes de uma sequência (formada por nucleotídeo ou aminoácidos) são similares e quais partes são diferentes. O resultado dessa tarefa gera similaridades das sequências em estudo, sendo alta similaridade a indicação de similaridades funcionais e estruturais dos organismos estudados. Em contrapartida, sequências diferentes podem conter informações importantes com respeito a diversidade e evolução.

Muitos dos problemas originados no campo da Bioinformática não têm ainda complexidade conhecida, enquanto outros são problemas NP-difíceis. Desse modo, diversas heurísticas foram elaboradas para a obtenção de resultados sub-ótimos, com razoável grau de acurácia. A utilização de sistemas computacionais classificados como de processamento de alto desempenho (HPC) também é fundamental para os problemas tratados atualmente em Bioinformática [7].

3.2.2 Algoritmo Needleman-Wunsch

O algoritmo de Needleman e Wunsch foi construído com base na técnica de Programação Dinâmica para encontrar uma solução ótima para o alinhamento de duas sequências genômicas. Suas complexidades de tempo e espaço são quadráticas, O(n^2), tendo como como entrada o tamanho n das sequências. Boukerche e Magalhães [12] descrevem o algoritmo em duas partes: cálculo da matriz de similaridades entre as sequências e recuperação dos alinhamentos globais.

Cálculo da Matriz de Similaridades

O algoritmo recebe como entrada duas sequências *s* e *t*, de tamanhos $|s| = m$ e $|t| = n$. Existem, portanto, $m+1$ prefixos possíveis para a sequência *s* e $n+1$ prefixos para a sequência *t*. Uma matriz $A_{m+1,n+1}$ é construída, com a posição $A[i, j]$ contendo o valor de similaridade entre dois prefixos das sequências *s* e *t*, formalizada por $sim(s[1..i], t[1..j])$. Na Figura 3.5(a), vemos uma matriz de similaridade entre as sequências *s* = ATAGCT e *t* = GATATGCA.

Após a inserção dos *gaps* de penalidade na primeira linha e na primeira coluna da matriz, as outras entradas são calculadas conforme a equação 3.1. Os valores das posições $A[i, j]$, para $i, j > 0$, são definidos como $sim(s[1..i], t[1..j])$. Então, são computados os valores linha por linha, da esquerda para a direita, ou coluna por coluna, de cima para baixo. Em seguida, são traçadas setas para indicar de onde vem o valor máximo, de acordo com a equação 3.1. Claramente, as complexidades de tempo e espaço dessa parte do algoritmo são iguais a $O(mn)$, onde *m* e *n* são os tamanhos das duas sequências, com complexidade $O(n^2)$ para o caso particular onde as sequências tem o mesmo tamanho.

sim(s[1..i],t[1..j]) = max

$$\begin{cases} sim(s[1..i], t[1..j-1]) - 2, \\ sim(s[1..i-1], t[1..j-1]) + 1 \quad (\textit{if} s[i] = t[j]), \\ sim(s[1..i-1], t[1..j-1]) - 1 \quad (\textit{if} s[i] \neq t[j]), \\ sim(s[1..i-1], t[1..j]) - 2 \end{cases} \tag{3.1}$$

Recuperação dos Alinhamentos Locais

Após o cálculo da matriz de similaridades e aplicação da equação 3.1, inicia-se na posição $A[s + 1, t + 1]$ o caminho até a posição $A[0, 0]$. Uma seta partindo da posição $A[i, j]$ para a esquerda corresponde a um espaço na sequência *s* e um caractere na coluna $t[j]$. Se a seta apontar para cima, significa um caractere na sequência *s* e um espaço na sequência *t*. Uma seta indicando uma movimentação diagonal corresponde a comparação de um caractere da sequência *s* e um caractere da sequência *t*. Portanto, um alinhamento global ótimo é construído seguindo as setas da direita até a posição final à esquerda. Dessa maneira, pode-se encontrar diversos alinhamentos ótimos. As complexidades de tempo e espaço são equivalentes a $O(n)$.

No exemplo anterior, a Figura 3.5(b) mostra um exemplo de alinhamento global.

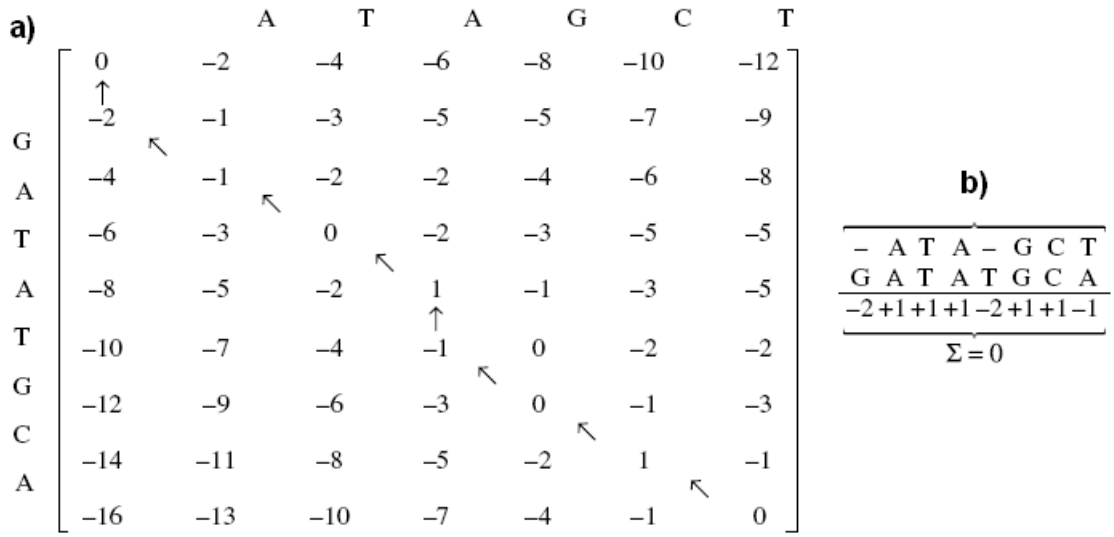

Figura 3.5: a) Matriz de Similaridade entre as Sequências. b) Alinhamento Global).

3.2.3 Algoritmo de Smith-Waterman

Os alinhamentos produzidos pelos métodos mencionados acima foram chamados de alinhamentos globais e coube a Smith e Waterman [101] o reconhecimento de que a maioria das regiões do DNA e das proteínas que são biologicamente significantes são formadas por subregiões que produzem bons alinhamentos, e não por alinhamentos globais. Esse tipo de alinhamento é conhecido como alinhamento local.

Os dois cientistas desenvolveram uma alteração no algoritmo de Needleman-Wunsch que permite a identificação de alinhamentos locais. A primeira alteração foi a exclusão de valores negativos, resultando na inclusão do valor 0 no conjunto de maximização da equação 3.1. Em seguida, a primeira linha e a primeira coluna foram preenchidas com o valor 0. Assim, lacunas introduzidas no início das sequências não são penalizadas e as diferenças entre as sequências são computadas de forma menos agressiva, em decorrência do valor mínimo ser igual a 0. Para a recuperação dos alinhamentos locais, procura-se o valor máximo mais à direita na matriz de similaridades e a partir dele inicia-se o caminho definido pelas setas até alcançar uma posição que não tem nehuma seta partindo dela ou uma posição cujo valor é igual a zero, conforme exemplo da Figura 3.6 (a). Na mesma figura, no item b, vemos um exemplo da alinhamento local.

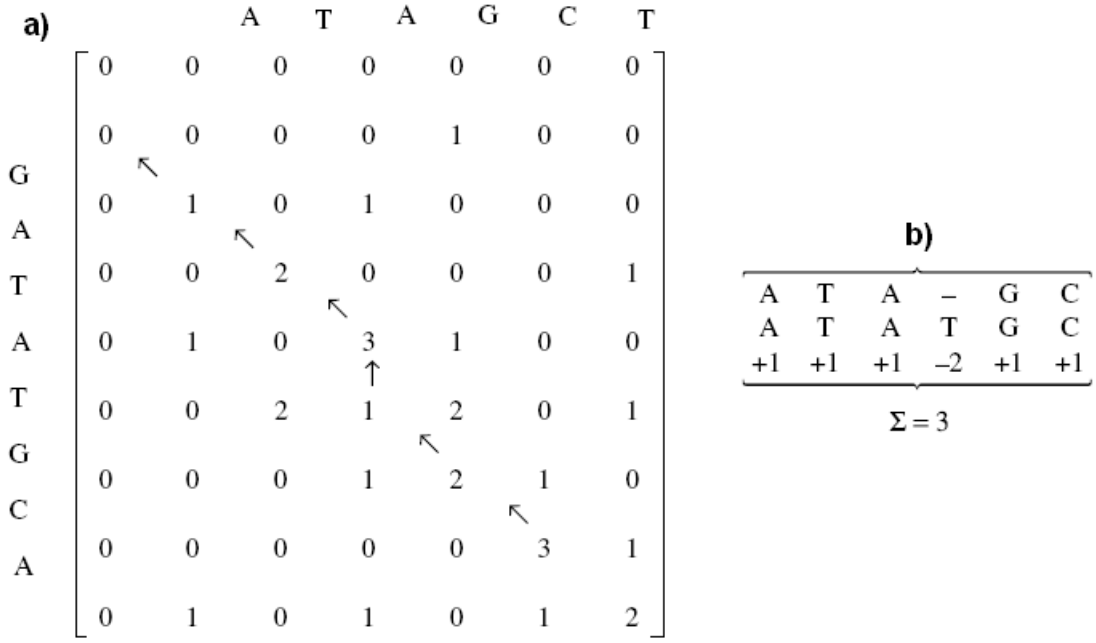

Figura 3.6: a) Matriz de Similaridade entre as Sequências. b) Alinhamento local.

3.2.4 Algoritmos para Alinhamentos Múltiplos

Além dos métodos que utilizam pares de sequência, outros processos tem sido desenvolvidos para o alinhamento de três ou mais sequências (sequências múltiplas). São métodos que demandam grande poder computacional (*computer-intensive*) e são construídos com base no alinhamento sequencial das sequências mais similares. Esses métodos são conhecidos como *alinhamentos múltiplos de sequências* (MSA) e expõem padrões de conservação de aminoácidos, dos quais o relacionamento distante pode ser mais corretamente detectado. Aplicar um método computacional exato a um algoritmo MSA é praticamente impossível e na prática são usadas heurísticas (algoritmos aproximados) para alinhar as sequências, maximizando suas similaridades [83].

Setubal e Meidanis [97] identificam a noção de alinhamento múltiplo como uma generalização do caso do alinhamento de pares de sequências. Assim, se $s_1, ..., s_k$ é um conjunto de sequências do mesmo alfabeto, o alinhamento múltiplo é obtido do conjunto pela inserção de espaços nas sequências, de tal modo a deixá-las do mesmo tamanho. As sequências extendidas são, então, colocadas em uma lista com os caracteres (ou espaços) de posições correspondentes na mesma coluna. Como requisito dessa composição, não podemos ter uma coluna preenchida somente com espaços. Na Figura 3.7 vemos o alinhamento múltiplo de quatro sequências de aminoácidos.

```
MQPILLL
MLR-LL-
MK-ILLL
MPPVLIL
```

Figura 3.7: Alinhamento múltiplo de quatro sequências de aminoácidos.

Como vimos anteriormente, os melhores valores de alinhamento (*scoring*) são encontrados com a utilização de Programação Dinâmica (algoritmos de Needleman-Wunsch e de Smith-Waterman). Devido ao alto custo computacional desta técnica, a maioria dos métodos de alinhamento múltiplo de sequência utiliza algoritmos heurísticos aproximados. Gondro e Kinghorn [57] citam o *método progressivo* [45] como o principal método utilizado hoje em dia para o alinhamento múltiplo de sequências. Este método é bastante rápido e direto mas tem a inconveniência de produzir facilmente um valor mínimo local. O *método exato* [74] é um método que produz melhores resultados que o método progressivo, mas seu custo computacional cresce exponencialmente, restringindo-se seu uso a um limite de dez sequências. Os métodos iterativos têm como base a produção de um alinhamento inicial para, em seguida, tentar melhorá-lo através de sucessivas iterações.

3.2.5 BLAST

O **BLAST (*Basic Local Alignment Search Tool*)** foi desenvolvido por S. Altschul et al (1990) e é um método extensivamente usado em projetos de sequenciamento de genomas. O processo inicia-se com a preparação de uma tabela com sequências curtas de cada sequência, determinando quais dessas palavras são mais significativas, de tal modo que sejam um bom indicador de similaridade e, então, restringe a pesquisa a essas sequências.

Tendo em vista que os aminoácidos podem ser obtidos através da tradução de códons (triplas) de nucleotídeos em um aminoácido, então é possível realizar as comparações entre sequências de nucleotídeos e sequências de aminoácidos, pois o BLAST permite a tradução em tempo de execução. Na Tabela 3.1, visualizamos as diferentes combinações de consultas de sequências e de tipos de sequências presentes no banco de dados, assim como as ferramentas BLAST utilizadas em cada tipo de pesquisa.

Existem versões do BLAST para pesquisa em bases de dados de ácidos nucleicos e proteínas. Boukerche e Magalhães [12] identificam três fases bem distintas na execução do BLAST: *seeding*, extensão e avaliação

Tabela 3.1: Consultas/Banco de Dados do 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

Seeding

Na fase *seeding*, o BLAST lê a sequência de pesquisa, aplica filtros de baixa complexidade e constrói uma *lookup table*[5]. Para isso, o BLAST usa o conceito de *palavras*, sendo cada palavra definida como um conjunto finito de letras de tamanho w, que aparece em uma determinada sequência. Por exemplo: a sequência TCACGA contém quatro palavras de tamanho três: TCA, CAC, ACG e CGA. Assim, o BLAST assume que os alinhamentos significantes tem palavras em comum.

As posições de todas as palavras compartilhadas (w-letras) entre duas sequências é determinada pelo reconhecimento exato do padrão. Essas posições são conhecidas como *palavras idênticas*. Somente regiões com palavras idênticas podem ser utilizadas no alinhamento.

Para casos onde alinhamentos significantes não contém palavras em comum, o conceito de ”vizinhança” é utilizado. A vizinhança de uma palavra inclui a própria palavra e outras palavras que possuem valor mínimo igual a T, quando comparada através de uma matriz de substituição. Se considerarmos, por exemplo, $T = 11$ e uma matriz de substituição PAM 200 (PAM-*Point Accepted Mutation*), as palavras RGD e KGD são vizinhas e o valor entre elas é 14.

Extensão

Na fase de extensão, as ”sementes” geradas na fase anterior devem ser extendidas para gerar um alinhamento. Isto é feito pela inspeção dos caracteres próximos à semente em ambas as direções, concatenando-os até um determinado *drop-off score* X. O *drop-off score* é definido como o valor que pode ser reduzido, considerando o último valor máximo. Assim, para uma semente A, o parâmetro X igual a três e uma pontuação de +1 para localizadas e -1 para não localizadas, o resultado é obtido da seguinte maneira:

ATGC GATA CTAGA

ATTC GATC GATGA

[5]estrutura de dados, usualmente um *array*

1212 3454 32123 <—- valor

0010 0001 23432 <—- *drop off score*

Assim, a extensão do primeiro *A* resulta no seguinte alinhamento:

ATGC GATA CT

ATTC GATC GA

Avaliação

Finalmente, na fase de avaliação, os alinhamentos gerados na fase de extensão são avaliados com remoção dos alinhamentos não significantes. Os alinhamento significantes, chamados pares HSP (*high score segment*), são aqueles valores maiores ou iguais a um determinado valor *S*. Adicionalmente, grupos consistentes de HSP são gerados, que incluem HSPs não sobrepostos presentes na mesma diagonal. Por fim, os grupos HSP são comparados com um parâmetro *E*, e somente os valores acima deste parâmetro são considerados.

3.3 Sequenciamento de Alto Desempenho

Após décadas de utilização do método de Sanger, novos métodos de sequenciamento foram elaborados para atingir o objetivo de redução dos custos dos projetos de sequenciamento, enquanto, paralelamente, busca-se o incremento na geração de dados genômicos.

Kato [69], em artigo de 2009, mostra o impacto da nova geração de sequenciadores, baseados em princípios diferentes do método de Sanger. Nas pesquisas científicas, como por exemplo, referentes à identificação de novos agentes infecciosos e no sequenciamento em larga escala de genomas cancerígenos.

Os novos sequenciadores podem produzir 100.000 vezes mais dados comparados com o mais sofisticados sequenciadores baseados no método de Sanger. A segunda geração de sequenciadores é baseada na análise paralela, enquanto a terceira geração utiliza o princípio de sequenciamento de uma única molécula em adição à análise paralela.

3.3.1 Sequenciadores

A segunda geração de sequenciadores é baseada no mesmo princípio: uso de produtos da PCR (*polymerase chain reaction*) de uma única molécula como modelo e o sequenciamento pela repetição das reações. Os principais sequenciadores dessa geração utilizam um dos três métodos abaixo para execução das reações:

1. *pirosequenciamento (454/Roche) [91]*: o sequenciamento do DNA é baseado no sequenciamento por síntese. A técnica possibilita a detecção em tempo real utilizando um sistema de enzimas em cascata, consistindo de quatro enzimas e substratos específicos, para produzir luz na formação do par de bases com o nucleotídeo complementar em uma amostra de DNA. Como resultado da incorporação do nucleotídeo, o pirofosfato (PPi) é liberado e, subsequentemente, convertido para ATP pela ATP sulfurilase, usada pela luciferase na geração de determinada quantia de luz. Essa luz é captada por um fotodiodo, um tubo fotomultiplicador ou uma câmera CCD (*charge-couple device*). Essa técnica tem as vantagens potenciais de ser precisa, flexível, sujeita ao processamento paralelo e de automação simples. Na Figura 3.8 temos um esquema desse método;

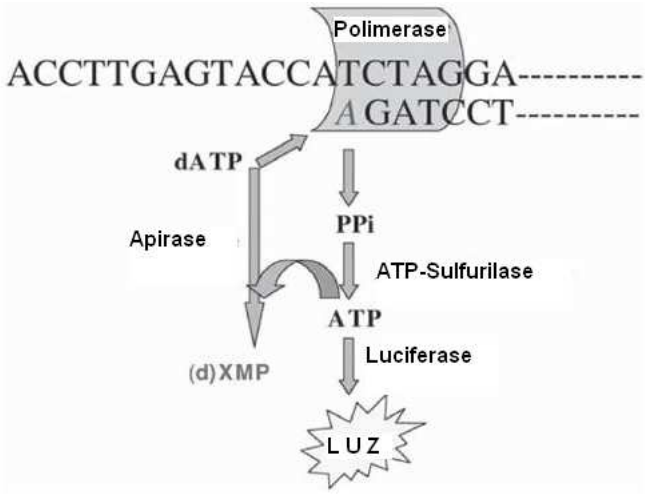

Figura 3.8: Esquema do progresso de uma reação enzimática no pirosequenciamento. A amostra de DNA com um *primer* híbrido e quatro enzimas envolvida no pirosequenciamento são adicionadas a uma célula da placa de titulação. Os quatro nucleotídeos, diferentes entre si, são adicionados e incorporados pelo uso das enzimas ATP sulfurilase e luciferase. Os nucleotídeos são continuamente degradados pela apirase, permitindo a adição de nucleotídeos novos.

2. *terminador reversível (Ilumina Solexa)*: com o uso de um terminador rotulado com uma determinada cor fluorescente, uma reação de extensão de uma simples base é executada. Após esse processo inicial, a cor fluorescente e o bloco formado são removidos quimicamente e a próxima reação de extensão é executada [69];

3. *sequenciamento por ligação (SOLiD)*: utiliza a enzima DNA ligase para identificar o nucleotídeo presente em uma determinada posição da sequência do DNA. A reação utiliza a propriedade discriminatória que a enzima DNA li-

gase possui com relação às bases. Duas bases adjacentes ao ponto de ligação são usadas para o sequenciamento. Um ciclo é constituído de ligações de oligonucleotídeos, divisão e remoção do produto excedente. Os ciclos são repetidos até que nenhum sinal fluorescente seja emitido [69].

Uma diferença básica entre os sequenciadores de alto desempenho e a tecnologia Sanger está relacionada ao tamanho e à quantidade de fragmentos gerados para o processo de montagem (*assembly*). Com fragmentos maiores pode-se obter maior exatidão, mas ao custo de utilização de algoritmos com tempo de execução quadrática ou mesmo exponencial.

No sequenciador 454 pode-se sequenciar 100 Mb de DNA em 8 horas com fragmentos do tamanho médio de 250 bp. O Illumina pode sequenciar 600 Mb de DNA por dia, com a utilização de fragmentos com o tamanho aproximado de 36 bp. Com o sistema SOLiD pode-se gerar 500 Mb de sequencias por dia com fragmentos de 35 bp de tamanho [104].

3.3.2 Pipelines para Sequenciamento de Alto Desempenho

Alvarez [3] enumera dois tipos genéricos de *pipelines* possíveis para os sequenciadores de alto desempenho.

O primeiro é constittuído de quatro fases: submissão, mapeamento, montagem e anotação. Este tipo de *pipeline* é indicado para sequenciadores que produzem sequências muito pequenas, que impedem a montagem direta de todos os fragmentos, como, por exemplo, o Illumina. Os dados das sequências são obtidos do sequenciador e armazenados. Depois, as sequências são alinhadas a um genoma de referência, formando grupos de sequências próximas. Os grupos são posteriormente montados obtendo os *singlets* e *contigs*. Finalmente, os *singlets* e *contigs* são anotados utilizando programas como o BLAST. Na Figura 3.9 vemos um esquema mostrando a interação entre as fases deste *pipeline*.

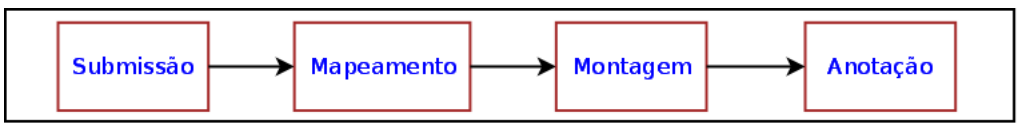

Figura 3.9: *Pipeline* COM a fase de mapeamento

O segundo *pipeline* possível possui as fases de submissão, montagem e anotação. É indicado quando sequências produzidas pelo sequenciador permitem a montagem dos fragmentos diretamente ou quando não há um genoma de referência para permitir a execução da fase de mapeamento. As sequências são obtidas

do sequenciador na fase de submissão e, a seguir, os fragmentos são montados com a consequente obtenção de *singlets* e *contigs*. Finalmente, na fase de anotação as funções dos *singlets* e *contigs* são inferidas por meio de programas como o BLAST. Na Figura 3.10 vemos a interação entre as fases do *pipeline*.

Figura 3.10: *Pipeline* SEM a fase de mapeamento

3.4 Resumo do Capítulo

Neste capítulo descrevemos alguns métodos e procedimentos utilizados em projetos de sequenciamento de genomas. O entendimento dos aspectos relativos ao trabalho computacional desses projetos serve como base para a compreensão das aplicações de bioinformática implementadas visando sistemas distribuídos. Em particular, para o p2pBIOFOCO, temos a execução da ferramenta BLAST, apresentada na Seção 3.2.5, em ambiente P2P.

No capítulo seguinte, estudaremos conceitos básicos de escalonamento de tarefas para a execução de um trabalho em ambientes distribuídos e faremos uma discussão do estado da arte em sistema distribuídos.

Capítulo 4

Escalonamento de Tarefas

Neste capítulo, apresentamos conceitos básicos de escalonamento de tarefas, que subsidiarão este trabalho. Na Seção 4.1, definimos e caracterizamos o escalonamento de acordo com vários autores. Na seção 4.2, mostramos como é classificado um problema de escalonamento. Na Seção 4.3, discutimos a complexidade computacional de problemas de escalonamento de tarefas. Na Seção 4.4, descrevemos inicialmente trabalhos encontrados na literatura relativos ao escalonamento de tarefas em *grid*, P2P e *grid*/P2P, e apresentamos esses trabalhos em uma tabela comparativa com as principais características dos métodos e sistemas estudados. A Seção 4.5 é utilizada para fazermos um detalhamento de um método de escalonamento que utiliza funções de minimização e maximização. Finalmente, na Seção 4.6 descrevemos um método de escalonamento com agrupamento de tarefas e transferência de dados.

4.1 Definições

Conforme Conway et al [29], o escalonamento refere-se a um plano para executar um trabalho ou alcançar algum objetivo, especificando a ordem e o tempo de cada parte desse trabalho. Várias atividades realizadas no dia-a-dia das pessoas estão sujeitas à execução de planos que conduzem à realização de um trabalho, à conquista de um objetivo, ou, no nosso caso, à solução de um problema de bioinformática. O escalonamento refere-se a um trabalho que será dividido em tarefas, que serão alocadas para recursos responsáveis pela execução dessas tarefas, de acordo com uma ordem particular. Como exemplo, citam-se as tarefas executadas em uma linha de produção de uma indústria, a utilização de pistas de pouso de um aeroporto, clientes de um banco na fila dos caixas, as tarefas domésticas de um dia

qualquer ou os programas executados em sistemas distribuídos. Um problema de escalonamento pode ser definido, então, a partir de quatro tipos de informação:

1. os trabalhos a serem processados;
2. o número e as características dos recursos que executarão os trabalhos;
3. as restrições e os requisitos que disciplinam a atribuição dos trabalhos aos recursos (tempo de execução, tempo de compilação, prioridade, tempo de início, tempo de término e etc), e;
4. os critérios pelos quais o escalonamento será avaliado.

Baseado nas informações mencionadas, Conway et al [29] criaram uma notação de quatro parâmetros, A/B/C/D, para representar problemas de escalonamento, sendo:

- A: para trabalhos dinâmicos, identifica a probabilidade de distribuição dos tempos entre as atribuições de trabalhos aos recursos. Para problemas estáticos, refere-se ao número de trabalhos;
- B: descreve o número de recursos (máquinas, homens, etc.) que serão utilizados para execução dos trabalhos;
- C: descreve um padrão do fluxo dos trabalhos nos recursos;
- D: descreve um critério pelo qual será avaliado o escalonamento.

Como exemplo, os parâmetros $n/2/F/F_{max}$ indicam, respectivamente, um número arbitrário de trabalhos (n) em um *flow shop*[6] (F) de duas máquinas (2) com o intuito de minimizar o *flow-time* máximo (tempo total que um recurso gasta para executar um determinado trabalho) (F_{max}).

Além das informações contantes na notação, Conway et al também descreveram outros parâmetros utilizados em um problema de escalonamento. Em seguida, são citados os principais parâmetros identificados por eles:

- *ready-time*, *release-time* ou *arrival-time*: tempo relacionado à liberação do trabalho por um recurso externo para execução no recurso de execução;
- *due-date*: tempo para a última operação do trabalho ser completada;
- *waiting time* ($W_{i,j}$): o tempo precedente à j-ésima operação do trabalho i, ou seja, o tempo que o trabalho deve esperar após o término da execução da [j - 1]-ésima operação antes de começar a [j]-ésima operação;

[6]Todos os trabalhos são atribuídos às máquinas na mesma ordem.

- *completion time* (C_i): tempo em que a última operação de um trabalho i foi completada;
- *flow time* (F_i): tempo total que o recurso utiliza para realizar um determinado trabalho J_i;
- *lateness* (L_i): diferença algébrica entre o tempo em que a última operação do trabalho foi completada e o tempo em que a última operação deveria estar completa (*due-date*). Assim como as variáveis *tardiness* e *earliness*, esta variável é um modo diferente de comparar o tempo de fato no qual o trabalho foi finalizado com o tempo desejado (*due-date*);
- *tardiness* (T_i): max(0, L_i): refere-se ao atraso na execução do trabalho, considerando somente os valores maiores do que zero;
- *earliness* (E_i): min(0, $-L_i$): refere-se ao término do trabalho em um tempo menor do que o previsto (*due date*);

Os parâmetros *lateness*, *tardiness* e *earliness* estão relacionados a funções que atribuem penalidades à execução do trabalho quando não é terminado no tempo devido (*due date*).

Os algoritmos de escalonamento são caracterizados por duas propriedades: a preempção e a granularidade. A preempção refere-se à interrupção de um trabalho para a execução de outro trabalho e à atribuição de 'fatias' de tempo para a execução desses trabalhos. Quanto à granularidade, temos uma definição em três níveis: granularidade fina, granularidade média e granularidade grossa, que se referem aos trabalhos (programas) com apenas algumas linhas de intruções até trabalhos com milhares de instruções e complexos segmentos de código.

Para Casavant e Kuhl [22] a definição de uma instância de um problema de escalonamento é composta por três componentes principais: o(s) consumidor(es), o(s) recurso(s) e a política de escalonamento. Assim, para entender o problema de escalonamento e os seus efeitos no ambiente é necessário avaliar a política de escalonamento e o seu relacionamento com os consumidores e os recursos. O relacionamento entre esses componentes é mostrado na Figura 4.1.

Brucker [17] apresenta um problema de escalonamento como a atividade de encontrar uma solução que minimize uma função de custo total, que pode ser de dois tipos: funções objetivo de gargalo e funções objetivo de somatório. Essas funções são consideradas critérios de otimização classificando-se em duas famílias: *minimax* e *minisum* [105]. A primeira refere-se à minimização de um valor máximo

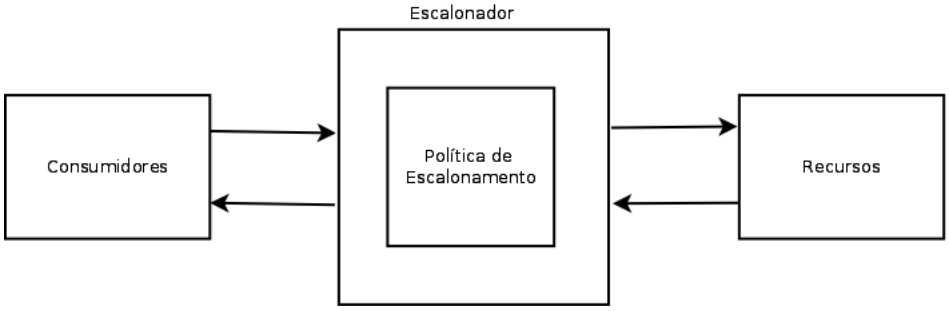

Figura 4.1: Relacionamento entre os componentes

C de um conjunto de funções e o segundo à minimização da soma das funções ($\sum f_i(C)$). Mostramos essas funções nas equações 4.1 e 4.2, respectivamente.

$$f_{max}(C) := max\{f_i(C_i)|i = 1, ..., n\} \tag{4.1}$$

$$\sum f_i(C) := \sum_{i=1}^{n} f_i(C_i) \tag{4.2}$$

Máquinas Paralelas

No mesmo trabalho, Conway et al estudam a distribuição de trabalhos em máquinas paralelas. Eles consideram trabalhos independentes que chegam simultaneamente ao conjunto de máquinas que executarão o processamento. A capacidade das m máquinas é descrita por uma matriz n x m, onde n é o número de trabalhos, m é o número de máquinas e cada célula p_{ij} da matriz representa o tempo de processamento de uma operação (tarefa) do trabalho i na máquina j. O caso mais simples ocorre quando todas as máquinas têm características idênticas.

No caso de máquinas diferentes, os valores das células de uma linha da matriz são diferentes e, desse modo, admite-se que uma máquina pode ser incapaz de realizar um trabalho em razão do tempo proibitivo de processamento. O principal problema do escalonamento é: os trabalhos individuais podem ou não ser atribuídos a mais de uma máquina? Se não é possível, o problema é dividido em subconjuntos para a determinação de uma sequência de execução dos trabalhos. Se é permitida a divisão de um trabalho em tarefas, então a obtenção de melhores escalonamentos é favorável.

Considerando que um trabalho é constituído de diversas operações (tarefas) idênticas, o mesmo tempo de processamento é disponibilizado para cada uma delas. Para mostrar um ponto positivo da divisão de tarefas entre máquinas, visualiza-se o exemplo da Figura 4.2(a), onde temos uma matriz que representa um problema

de escalonamento de dois trabalhos em duas máquinas (2x2). Na matriz, cada célula representa o tempo de processamento do trabalho na respectiva máquina. Neste caso para o trabalho 1 temos um tempo de processamento de 2 unidades nas máquinas 1 e 2. O mesmo tempo é observado para o trabalho 2.

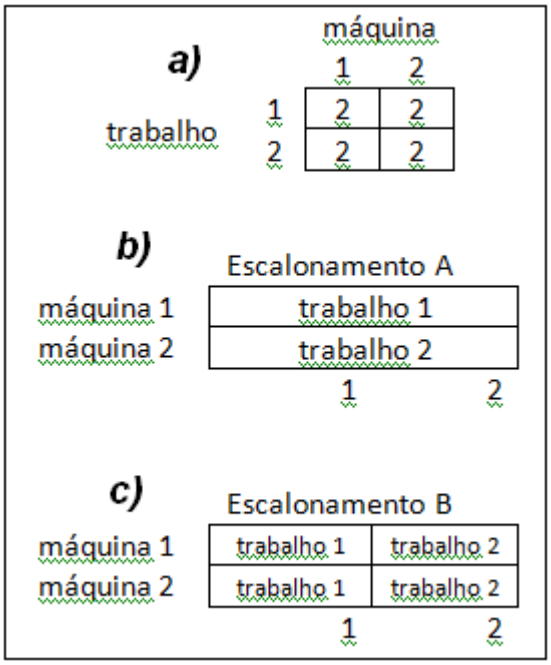

Figura 4.2: Escalonamento de um Problema 2/2 em Duas Máquinas

Conforme a Figura 4.2(b), se um trabalho é atribuído a cada máquina, um escalonamento A tem *mean flow time*($\overline{F}$) igual a 2,0. Se as duas máquinas são utilizadas para processar cada um dos trabalhos conforme escalonamento B (Figura 4.2(c)), o $\overline{F}$ reduz-se para 1,5.

Generalizando: se existirem m trabalhos para processar em m máquinas com p de tempo de processamento para cada um, temos um total de m trabalhos a serem realizados e terminados simultaneamente, independente da forma como os trabalhos são atribuídos individualmente às máquinas. Assim, se cada trabalho é atribuído a cada uma das m máquinas, então todos os trabalhos terminam simultaneamente no tempo p e $\overline{F} = p$. De outra forma, se cada trabalho é dividido entre todas as m máquinas, o primeiro trabalho terminará no tempo p/m (tempo de processamento de um trabalho em uma máquina dividido pelo número de máquinas). O segundo será encerrado no tempo $2p/m$ e o terceiro em $3p/m$, até o último que será terminado no tempo p. Utilizando essa estratégia consegue-se um tempo de finalização do trabalho menor que o resultado de um trabalho atribuído

a cada máquina, *excetuando-se o último*. O resultado dessa atribuição é traduzido na Equação 4.3:

$$\overline{F} = \frac{p}{m}(\frac{1}{m} + \frac{2}{m} + \frac{3}{m} + ... + \frac{m}{m}) = \frac{m+1}{2m}p \tag{4.3}$$

Assim, qualquer escalonamento pode ser melhorado utilizando a atribuição de tarefas para máquinas idênticas.

4.2 Classificação de um Problema de Escalonamento

Nesta seção detalharemos como classificar problemas de escalonamento.

Dados do Trabalho

Graham et al [58] classificam um problema de escalonamento por uma notação de três campos, $\alpha|\beta|\gamma$. Para um determinado trabalho J_i, podemos identificar:

- número de operações m_i;
- um ou mais tempos de execução das operações, com p_{ij} indicando o tempo de execução de uma operação do trabalho *i* na máquina *j*, omitindo-se o segundo índice quando todas as máquinas são idênticas (ex: p_i);
- uma data de início r_i, na qual o trabalho torna-se disponível para processamento;
- uma data de término quando o trabalho J_i é completado;
- um peso w_i atribuindo uma importância relativa ao trabalho;
- uma função real não decrescente chamada de *função de custo (f_i)*, que determina o custo de completar o trabalho J_i no tempo *t*.

Ambiente de Processamento (α)

O ambiente de processamento é caracterizado por uma cadeia de caracteres formada por dois parâmetros $\alpha = \alpha_1\alpha_2$, com o valor $\circ$ representando o símbolo de vazio. Se $\alpha_1 \in \{\circ, P, Q, R\}$, cada trabalho J_i é constituído de uma operação simples que pode ser processada em qualquer máquina M_i e o tempo de processamento de J_i na máquina M_i é igual a p_{ij}. Os quatro valores do conjunto são definidos da seguinte forma:

- se $\alpha_1 = \circ$, temos uma única máquina e o tempo de processamento $p_{1j} = p_j$;

- se $\alpha_1 = P$, o ambiente é caracterizado por máquinas paralelas idênticas e o tempo de processamento é representado $p_{ij} = p_i$ para $i = 1, ..., m$;
- se $\alpha_1 = Q$, o ambiente é caracterizado por máquinas paralelas uniformes, com tempo de processamento $p_{ij} = q_i p_i$, sendo *q* um fator de velocidade da máquina M_i, para $i = 1, ..., m$;
- se $\alpha_1 = R$, as máquinas do ambiente de processamento são paralelas e diferentes.

O segundo parâmetro, α_2, representa o número de máquinas do ambiente de processamento. Assim, se tivermos α_2 igual a um número inteiro e positivo, o número *m* de máquinas será constante. Se $\alpha_2 = \circ$, então o número de máquinas será variável. Obviamente, a condição $\alpha_1 = \circ$ (vazio) será somente verificada se tivermos uma única máquina no ambiente ($\alpha_2 = 1$).

Características do Trabalho (β)

As características de um trabalho pertencente a um problema de escalonamento são determinadas, no máximo, por um conjunto β de seis elementos - β_1, β_2, β_3, β_4, β_5, β_6 - e corresponde ao segundo campo (β) da notação de Graham et al.

O primeiro elemento, β_1 refere-se à preempção, se é permitida ou não. Em um algoritmo não preemptivo, a execução do trabalho, uma vez iniciada, é realizada até o término. Um algoritmo preemptivo pode suspender a execução do trabalho, armazenando o seu estado e alterando sua prioridade ou transferindo o trabalho para outro recurso [29]. A relação $\beta_1 \in \{pmtn, \circ\}$ respresenta o conjunto de símbolos para o elemento. Assim, *pmtn* indica que a preempção é permitida no problema de escalonamento e o símbolo de vazio ($\circ$) indica que a preempção não é permitida.

A utilização de *s* recursos para a execução de um trabalho é definida pelo elemento β_2, com $\beta_2 \in \{res, res1, \circ\}$. Para o valor *res* temos a restrição de utilização de recursos limitada pelo conjunto R_h (*h = 1,...,s*). Assim, cada trabalho J_i requer o uso de r_{hj} unidades de R_h para a sua execução. Se temos o valor *res1*, a presença de um único recurso é assumida. O simbolo de vazio indica que não existem restrições de recursos para a execução do trabalho.

As relações de precedência são indicadas pelo terceiro elemento do conjunto β, ($\beta_3 \in \{prec, tree, \circ\}$). As restrições impostas a β_3 podem ser sucintamente estipuladas por um *DAG (Directed Acyclic Graph)* cuja representação é *G=(V,A)*, onde *V =1,2,...,n* é o conjunto dos trabalhos com $(i, k) \in A$ se e somente se J_i deve ser completado antes de J_k começar, tendo como representação a expressão $J_i \rightarrow J_k$. Em um problema de escalonamento, o segundo elemento é representado por $\beta_3 = prec$ para

um DAG. Para relações de precedência resultantes de outras estruturas (ávores, por exemplo), podemos representar o segundo elemento informando explicitamente a estrutura. Assim, para uma árvore temos $\beta_3 =$ **tree**. Caso não haja precedência, o elemento β_3 não aparece no conjunto β ou é representado por $\circ$.

O próximo elemento, β_4, refere-se ao tempo de início do trabalho (r_i), ou seja, o tempo no qual o trabalho inicia efetivamente a sua execução. Se temos $\beta_4 = r_i$, então os trabalhos terão tempos de início de sua execução explicitamente determinados, podendo haver diferença nessa determinação.

Um limite superior para o número de máquinas presentes no ambiente de processamento é atribuído ao elemento β_5. Portanto, para $\beta_5 \in \{m_j \leq \overline{m}, \circ\}$, a constante $\overline{m}$ determina o número máximo de máquinas para o processamento do trabalho. Limites desse tipo são instituídos para trabalhos nos quais $\alpha_1 = J$, onde, neste caso, ***J*** representa um *job shop*[7].

Em β_6 é definido se cada operação do trabalho J tem uma unidade de tempo para processamento. Desse modo, para $\beta_6 \in \{p_{ij} = 1, \underline{p} \leq p_{ij} \leq \overline{p}, \circ\}$. Se $p_{ij} = 1$, cada operação tem uma unidade de tempo de processamento para a sua execução. Para $\underline{p} \leq p_{ij} \leq \overline{p}$ limites inferior e superior com relação a p_{ij} são especificados para o processamento da operação. Para $\beta_6 = \circ$ nenhum limite é especificado, com a operação utilizando o tempo de processamento necessário para a sua execução completa.

Critérios de Otimização (γ)

O terceiro campo (γ) da notação refere-se ao critério de otimização do trabalho. Dado um problema de escalonamento, podemos computar para cada trabalho J_i o valor do tempo C_i (*completion time*) para completar o trabalho; o tempo de atraso L_i ($C_i - d_i$) em relação ao tempo definido de término do trabalho, onde d_i é o prazo determinado para finalização do trabalho, admitindo valores negativos, caso o trabalho termine antes do prazo estimado, ou o atraso máximo T_i de finalização do trabalho (considerando somente valores positivos). Tal critério de otimização geralmente envolve a minimização de uma função de maximização ($f_{max} \in \{C_{max}, L_{max}\}$, onde C_{max} refere-se ao tempo máximo de finalização do trabalho J_i (finalização da última operação O_i em execução) e L_{max} é o atraso máximo do trabalho em relação ao prazo previsto de finalização (atraso da última operação O_i em execução).

Abaixo listamos exemplos de utilização da notação de Graham et al:

[7] trabalho i composto de uma sequência n_i de operações O_{ij} (O_{i1}, O_{i2},..., O_{in_i}) que são processadas nesta ordem, ou seja, com restrições de precedência da forma $O_{ij} \rightarrow O_{i,j+1}$ ($j = 1, ..., n_i - 1$)

1. Minimizar o tempo máximo de atraso referente à execução de um trabalho J_i em uma máquina, sendo este trabalho sujeito a regras de precedência: $1|prec|L_{max}$;
2. Minimizar o tempo total de finalização de um trabalho J_i, com um número variável de máquinas paralelas e heterogêneas, permitindo a preempção: $R|pmtn|\sum C_i$;
3. Minimizar o tempo máximo de finalização de um trabalho em um ambiente composto por três máquinas, caracterizado como *job shop*, com unidades de tempo de processamento: $J3|p_{ij}|C_{max}$.

4.2.1 Tipos de Escalonamento

Segundo Brucker [17], existem dois tipos de problemas de escalonamento ligados à natureza da chegada dos trabalhos aos recursos: *estáticos e dinâmicos*.

- *Estáticos*: chegada simultânea dos trabalhos aos recursos que estão prontos para executar esses trabalhos;
- *Dinâmicos*: os recursos executam os trabalhos continuamente, sendo intermitente a solicitação para a execução de trabalhos.

Num contexto mais amplo, Casavant [22] elabora uma taxonomia de escalonamento baseada em níveis hierárquicos, conforme mostrado na Figura 4.3. Na sequência, efetuamos uma breve descrição para cada nível dessa taxonomia.

Níveis da Taxonomia de Casavant

Global x Local: um escalonamento global refere-se ao problema de decidir em quais máquinas de um sistema distribuído executaremos um determinado trabalho, enquanto o escalonamento local é pertinente às políticas de escalonamento de sistema operacional de uma máquina.

Estático x Dinâmico: considerando um escalonamento global, um escalonamento estático atribui um conjunto de tarefas a uma configuração particular de *hardware*. Não são utilizadas informações sobre as alterações nessa configuração do *hardware* e as características que descrevem o conjunto de tarefas permanecem fixas. No escalonamento dinâmico, ao contrário, não se tem conhecimento prévio das características dos trabalhos e do ambiente que será utilizado para processamento das operações desses trabalhos. As decisões de alocação e ordenação dos trabalhos são tomadas, portanto, em tempo de execução.

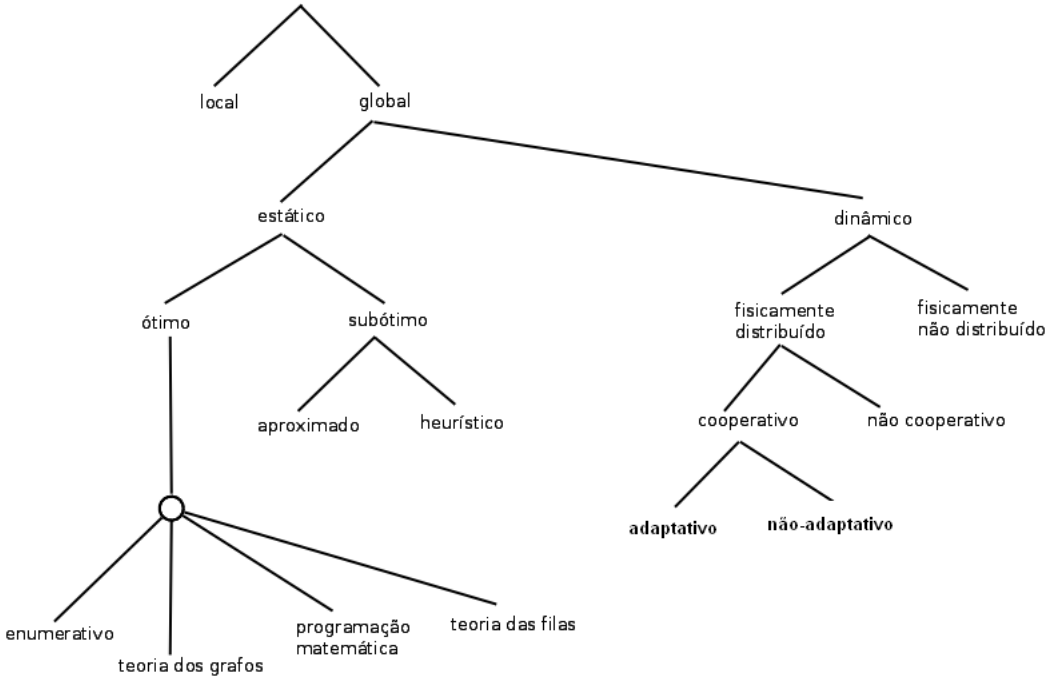

Figura 4.3: Classificação Hierárquica do Escalonamento (adaptado de Casavant [22])

Escalonamento Estático

Como mencionado, no escalonamento estático os critérios e os parâmetros para a alocação das tarefas não se alteram e são conhecidos antecipadamente. Nessa categoria, os próximos níveis são:

- *Ótimo x Subótimo*: se o estado do sistema e todos os recursos necessários para a execução das tarefas são conhecidos, uma atribuição ótima pode ser conseguida com a utilização de uma função critério para o escalonamento (por exemplo: minimizar o *completion time (C)*, maximizar a utilização dos recursos ou maximizar o *throughput*[8]). Algumas técnicas de solução ótima são:

  - enumeração do espaço de soluções e pesquisa;
  - aplicação da Teoria dos Grafos;
  - programação matemática, e;
  - teoria das filas

  Como o problema do escalonamento de tarefas é NP-completo [55], na prática, usam-se duas formas genéricas de soluções para problemas subótimos: aproximada e heurística.

[8] quantidade de trabalhos que um sistema pode executar em um determinado período de tempo.

- *Aproximada x Heurística*:

  A primeira estratégia procura por uma solução ”boa” no espaço de soluções e é determinada pelos seguintes fatores:

  1. disponibilidade de uma função para avaliar a solução;
  2. tempo requerido para avaliar a solução;
  3. a habilidade para avaliar o valor de uma solução ótima, baseada em alguma métrica;
  4. disponibilidade de um mecanismo para limitar o espaço de soluções.

  Para as soluções que utilizam *heurísticas* assumem-se características mais realísticas a respeito do sistema. As heuríticas requerem maior quantidade de tempo e de outros recursos do sistema para executar as suas funções. Parâmetros referentes ao desempenho do sistema, por exemplo, são monitorados indiretamente a fim de prover um método mais simples de interação e de cálculo dos escalonamentos possíveis.

Escalonamento Dinâmico

Nesse tipo de escalonamento são usadas informações mais reais sobre o estado do sistema. A decisão sobre a alocação das tarefas é feita no momento em que a tarefa começa seu ciclo de execução no sistema. A seguir, são destacados os principais níveis desse tipo de escalonamento:

- *Distribuído x Não-distribuído*: a decisão sobre a distribuição das tarefas é feita em um único processador ou está distribuída entre os processadores.
- *Cooperativo x Não-cooperativo*: para escalonadores distribuídos, pode haver a cooperação entre os processadores que adotam determinado comportamento consultando os pares envolvidos na execução das tarefas ou se cada processador possui uma política independente de tomada de decisões.
- *Adaptativo x Não-adaptativo*: se os parâmetros e os algoritmos mudam dinamicamente de acordo com o comportamento anterior e atual do sistema, podemos classificar essa política como um escalonamento adaptativo. Por outro lado, podemos considerar sempre os mesmos parâmetros como entrada para um determinado algoritmo de escalonamento, sem modificá-los. Neste caso, temos uma política não-adaptativa.

4.3 Complexidade de Problemas de Escalonamento

Para um problema de escalonamento representado pela notação $P|prec|C_{max}$[9], temos os seguintes parâmetros:

- a existência de uma ordem de precedência entre os trabalhos (*prec*);
- os trabalhos que podem ser executados em máquinas paralelas (*P*); e
- uma função objetiva relacionada à minimização do tempo de término desses trabalhos (C_{max}).

A solução do problema está relacionada com a busca de um valor C_{max} que seja menor ou igual a *k*, com k inteiro e positivo. Assumindo, portanto, que os valores de C_i (tempos de finalização dos trabalhos *i*) e de m_i (máquina onde o trabalho *i* é processado) são representados por código binário, constituindo dois vetores de entrada com tamanho limitado para a entrada do problema, podemos verificar em tempo polinomial se os C_i e u_i determinam um escalonamento possível para $C_{max} \leq k$.

Garey e Jonhson [55] enumeram vários problemas NP-completos de escalonamento, entre eles o sequenciamento de tarefas em um único processador e o processamento de tarefas em vários processadores. Dizer que um problema de escalonamento é NP-Completo significa afirmar que obter um algoritmo de complexidade polinomial para sua resolução é tão difícil quanto encontrar um algoritmo polinomial para resolução do problema do caixeiro viajante. A seguir, selecionamos dois problemas NP-completos descritos pelos autores. O primeiro problema refere-se ao sequenciamento de tarefas em um processador, com datas de início e fim determinadas. O segundo refere-se à alocação de tarefas em máquinas heterogêneas não relacionadas, tendo como função minimizar o *makespan*[10].

Problema 1: Sequenciamento de Tarefas

Para esse problema, é definido um conjunto de tarefas *T*, com cada $t \in T$, um inteiro positivo *l(t)*, referente ao tamanho da tarefa, um inteiro positivo, *r(t)*, referente ao tempo de início da tarefa e um inteiro positivo, *d(t)*, referente ao término da tarefa. Para essas definições, tem-se as seguintes afirmações:

- para quaisquer tarefas distintas t_1 e t_2, se temos uma função ou critério de otimização σ tal que $\sigma(t_1) > \sigma(t_2)$, então $\sigma(t_1) \geq \sigma(t_2) + l(t_2)$;

[9] Notação do problema conforme Graham et al, vide Seção 4.2.
[10] Tempo entre a submissão da tarefa e a sua finalização.

- para todas as tarefas $t \in T$, $\sigma(t) \geq r(t)$;
- para todas as tarefas $t \in T$, $\sigma(t) + l(t) \leq d(t)$.

Os requisitos do escalonamento S, representados pela função σ, podem ser descritos da seguinte maneira:

1. atribuição de um tempo de início único para cada tarefa t do conjunto, decorrente da relação um-por-um da função σ;
2. duas tarefas não podem executar simultaneamente, pois se a tarefa t_1 é escalonada para executar após a tarefa t_2 ($\sigma(t_1) > \sigma(t_2)$), então o tempo mais cedo no qual a tarefa t_1 pode ser escalonada, pertence a qualquer momento após o término da tarefa t_2, ou seja, $\sigma(t_1) + l(t_1)$;
3. nenhuma tarefa pode iniciar antes do seu tempo de início; e
4. nenhuma tarefa pode terminar depois do seu tempo de término estabelecido.

Este tipo de problema é característico de um escalonamento local, no qual o fator determinante para a sua complexidade são os requisitos impostos pelos tempos de início e término das tarefas, concomitantemente à concorrência por um único processador. O tratamento desse problema está definido geralmente na implantação de algoritmos nos escalonadores de um computador.

Problema 2: Escalonamento em Máquinas Heterogêneas

Para o escalonamento distribuído, o problema de minimizar o *makespan*, $R||C_{max}$ da máquina mais carregada é NP-Difícil [55]. Na sequência, descreve-se a formalização referente à alocação de *n* tarefas independentes a *m* máquinas paralelas não-relacionadas, com um tempo de processamento diferente de cada tarefa em cada máquina.

Assim, um conjunto $J = j_1, j_2, ..., j_n$ de *n* tarefas alocadas a um conjunto $I = i_1, i_2, ..., i_m$ de *m* máquinas e uma matriz m por n, com p_{ij} tempos de processamento, onde p_{ij} é o tempo que a i-ésima máquina consome para executar a *j*-ésima tarefa. Considerando, também, $X = x_{ij}$ representando um conjunto de variáveis de decisão e atribuindo $x_{ij} = 1$, caso a *j*-ésima tarefa seja alocada para a máquina *i*, e *0*, caso contrário, tem-se a seguinte representação matemática, definida pelas Equações 4.4, 4.5, 4.6 e 4.7.

$$min(C_{max}) \tag{4.4}$$

considerando

$$\sum_{j=1}^{n}(p_{ij}x_{ij}) \leq C_{max}, i = 1, 2, ..., m \tag{4.5}$$

$$\sum_{i=1}^{m} x_{ij} = 1, j = 1, 2, ..., n \tag{4.6}$$

$$x_{ij} = 0 \text{ ou } 1, i = 1, 2, ..., m; j = 1, 2, ..., n \tag{4.7}$$

Com a apresentação desses dois problemas, enfatizamos a dificuldade e a importância que o escalonamento de tarefas exerce no contexto de ambientes computacionais paralelos e/ou distribuídos. Várias abordagens e heurísticas colaboraram para o desenvolvimento e a implementação de algoritmos que resultaram em um aumento do *throughput* e na melhora de desempenho dos sistemas.

4.4 Levantamento Bibliográfico

Nesta seção, apresentamos e discutimos a bibliografia estudada neste trabalho, que aborda diversas soluções relacionadas ao problema do escalonamento de tarefas em sistemas distribuídos. Para cada trabalho, descrevemos seus objetivos, a metodologia empregada e os resultados, além de discutir as ideias de cada texto que são mais ligadas ao trabalho desenvolvido nesta dissertação.

Como critério de seleção dos textos, escolhemos trabalhos relacionados ao escalonamento de tarefas independentes em ambientes heterogêneos, incluindo, também, textos com métodos passíveis de utilização no p2pBIOFOCO.

"A Taxonomy of Scheduling in General-Purposed Distributed Computing Systems"
Autores: T. L. Casavant and J. G. Kuhl [22]
Ano: 1988

- *Objetivo*: propor uma taxonomia híbrida (horizontal e hierárquica) para o problema do gerenciamento de recursos, com o objetivo de padronizar uma terminologia e estabelecer mecanismos de classificação do problema de escalonamento.
- *Metodologia*: avaliação de sistemas de classificação e taxonomias já propostas.
- *Resultados*: compilação das tendências e das ideias sobre o gerenciamento de recursos apresentadas até a data de publicação do artigo (1988), para então

propor uma taxonomia plausível e útil que oriente o estudo do gerenciamento de recursos em sistemas computacionais. A taxonomia tem como foco a classificação dos métodos de escalonamento em sistemas mono e multiprocessados, estendendo-se aos sistemas distribuídos, conforme comentário na conclusão do trabalho.

- *Discussão*: na análise das vantagens e das desvantagens de inclusão de determinada categoria de escalonamento em nosso trabalho, a utilização da hierarquia de Casavante e Kuhl foi útil para identificarmos, a partir das características do sistema p2pBIOFOCO (independência das tarefas, heterogeneidade dos recursos), os elementos positivos e negativos de cada tipo de escalonador a ser utilizado. Assim, por exemplo, sabemos que um escalonador estático tem a vantagem de executar o algoritmo de alocação dos processos aos recursos em tempo de compilação, aumentando a eficiência do ambiente de execução. No entanto, uma das principais desvantagens é a falta de métodos para eficientemente estimar os tempos de execução das tarefas e os atrasos de comunicação presentes no sistema.

"Task Scheduling in Networks"
Autores: C. Phillips, C. Stein and J. Wein [85]
Ano: 1996

- *Objetivo*: propor algoritmos aproximados para o cálculo do *makespan* e do *tempo médio* de finalização das tarefas em sistemas distribuídos, considerando o tempo do tráfego das informações na rede.
- *Metodologia*: definição de teoremas e de lemas, com a utilização de indução matemática, da teoria dos grafos e da análise de algoritmos.
- *Resultados*: algoritmos aproximados para o escalonamento distribuído, considerando o tempo de comunicação entre as máquinas no cálculo do tempo total de execução das tarefas do sistema.
- *Discussão*: consideramos que a inclusão de algoritmos para cálculo do tráfego de informações no p2pBIOFOCO adicionaria uma complexidade desnecessária para o atual estágio do projeto. Assim, abdicamos de aplicarmos, no momento, as ideias dos autores no sistema. No entanto, com o aumento do número de nós e da complexidade da rede, esse trabalho pode ser analisado novamente para cálculo do impacto do tráfego das informações na rede.

"On the Robustness of the Soft State for Task Scheduling in Large-scale Distributed Computing Environment"

Autores: H. Tada, M. Imase e M. Murata [103]
Ano: 2008

- *Objetivo*: propor dois métodos para o controle da utilização de CPU e do tempo de execução das tarefas na execução do algoritmo WQR (*Workqueue with Replication* [99]) em um sistema distribuído.
- *Metodologia*: aplicação de duas abordagens: a primeira baseada no uso de *timeout* para a execução de uma tarefa e a última no emprego de *soft state method* [87] (protocolo para troca de mensagens com o objetivo de monitorar o estado dos *hosts*) para solução de problemas originados quando há cancelamento de tarefas ou falhas na execução da tarefa.
- *Resultados*:
  - o método *soft state* é mais robusto do que o método *task timeout*, considerando as falhas de execução das tarefas;
  - o desempenho do método *task timeout* depende do valor do *timeout*, que é difícil de calcular;
  - o método *soft state* gasta menos ciclos de CPU quando a taxa de falhas na execução das tarefas é baixa;
  - o método *soft state* tem baixo desempenho quando a perda de mensagens ultrapassa determinado limite.
- *Discussão*: o trabalho cita dois problemas conhecidos sobre o controle de tráfego de mensagens em sistemas distribuídos: o tempo de expiração da mensagem (*timeout*) e a inclusão de regras de segurança com a utilização de *firewalls*. Em nosso sistema, as ideias de Tada e et al são úteis em razão da utilização do algoritmo *WorkQueue with Replication* (WQR) e das características da rede (existência dos *firewalls* entre as instituições envolvidas - UnB e EMBRAPA) na qual executamos a submissão dos trabalhos.

"A Decentralized Scheduling and Load Balacing Algorithm for Heterogeneous Grid Environments"
Autores: M. Arora, S. K. Das e R. Biswas [6]
Ano: 2002

- *Objetivo*: desenvolver um algoritmo para escalonamento de tarefas com balanceamento de carga entre os recursos de um ambiente distribuído, utilizando vários parâmetros relacionados aos recursos dos nós e à criação de métricas de desempenho baseadas em funções de maximização e minimização.

- *Metodologia*: desenvolvimento do algoritmo, com a utilização de MPI (*Message Passing Interface*), e execução de simulações utilizando parâmetros definidos para os recursos de cada nó, além de estipulação dos parâmetros de desempenho. O conjunto de simulações foi executado em três grupos:
  - com heterogeneidade de recursos e com latência da comunicação constante;
  - com capacidade dos recursos constante e com variação da latência da comunicação entre os nós;
  - com heterogeneidade de recursos e com variação da latência.

  Na execução do algoritmo, são consideradas duas filas: uma fila local contendo as tarefas que serão executadas no nó e uma fila externa, referente às tarefas que serão migradas para outros nós com base na carga do nó atual. Neste algoritmo, os autores utilizam a comparação entre a *latência de comunicação entre os nós* e o *tempo considerado para esvaziar a fila local de tarefas*. Se o dobro do tempo de comunicação for menor do que o tempo de esvaziamento da fila local, uma nova tarefa é requisitada de um nó vizinho. Assim, essa heurística determina o seguinte: quanto maior a carga no nó, menor a disponibilidade para aceitar novas tarefas. No algoritmo de Arora et al (transcrito em seguida), podemos visualizar a rotina principal de execução do algoritmo.

algoritmo 1 Arora et al

```
1: loop
2:   if (α ← nova tarefa) then
3:     tempo ← tempo atual do sistema
4:     executaRemoto ← avaliaCarga(α,tempo) {Verifica carga local no nó}
5:     if (excecutaRemoto) then
6:       tempo_FL ← tempo para esvaziar a fila local
7:       ∀j ∈ Vizinho(P_i)
8:       if (2 * Comm_i^j ≤ tempo_FL) then
9:         RequererEnvio(j, Comm_i^j, tempo_FL) {Utiliza MPI para envio de solicitação de um tarefa ao nó vizinho}
10:      end if
11:      Receber(tempo_FL) {Aguarda resposta do envio}
12:      Balancear(S, R) {
13:    end if}
14:  end if
15: end loop
```

- *Resultados*:

  - mantendo-se a latência de comunicação entre os nós e as características das tarefas constantes, somente com a variação da capacidade dos recursos, os autores observaram o aumento do desempenho, quando os recursos são incrementados (como seria esperado);
  - com a variação na latência e mantendo-se os recursos constantes, o aumento nos custos da comunicação degradaram o parâmetro de desempenho;
  - houve degradação do tempo de execução da tarefa com a maior especificidade de requisitos para a execução da tarefas.

- *Discussão*: o foco na descentralização do processo de escalonamento com critérios de balanceamento de carga, concomitante à criação de uma política de localização de nós iniciada na máquina que recebe os trabalhos do usuário, são características que também fazem parte da arquitetura do p2pBIOFOCO. O algoritmo desenvolvido, implementado com a utilização de MPI, tem algumas funções e estruturas de dados (filas), com a possibilidade de aproveitamento na construção de um escalonador. Como observação ao trabalho dos autores, o mecanismo de localização dos nós - baseado na execução de *pings* e execução de procedimentos de roteamento com restrições ao tempo de latência na comunicação entre os nós - poderia ser substituído pela implementação de uma DHT, considerado um método de busca mais independente, mais eficiente e não acoplado à solução.

"Trading Cycles for Information: Using Replication to Schedule bag of tasks Applications on Computacional Grids"
Autores: D. P. da Silva, W. Cirne e F. V. Brasileiro [99]
Ano: 2003

- *Objetivo*: desenvolver e implementar o algoritmo *Workqueue with Replication* (WQR), utilizando como base o algoritmo clássico *Workqueue*, com o propósito de minimizar o problema do escalonamento dinâmico de tarefas independentes (neste caso, caracterizadas como *Bag of Tasks*-BoT) em sistemas heterogêneos, dispensando a dependência de qualquer tipo de informação sobre as tarefas ou sobre os recursos do sistema.

- *Metodologia*: análise do desempenho do algoritmo, efetuando comparações com os algoritmos que utilizam as informações sobre os recursos do ambiente. Os autores utilizaram um ambiente distribuído virtual, criado pela ferramenta Simgrid [20], e realizaram os experimentos aplicando os algoritmos

Dynamic Fastest Processor to Largest Task First (*Dynamic FPLTF*), *Sufferage*, *Workqueue* básico, WQR2x, WQR3x e WQR4x. O primeiro algoritmo foi alterado pelos autores para tornar-se uma versão dinâmica do conhecido algoritmo FPLTF [79]. Assim como o algoritmo Dynamic FPLTF, o algoritmo *Sufferage* [21] necessita de informações do sistema, como o tamanho da tarefa, a carga do nó e a velocidade do processador, para a atribuição das tarefas aos *hosts*. Seu método utiliza o segundo melhor *completion time* resultante da execução de uma tarefa em um nó. O próximo algoritmo, Workqueue, não precisa de qualquer tipo de informação para o escalonamento, atribuindo as tarefas arbitrariamente aos nós. Com relação aos algoritmos WQRNx (onde N é o *threshold* de replicação das tarefas), as simulações foram realizados com os fatores de replicação 2, 3 e 4.

- *Resultados*: a execução dos escalonadores com os mesmos parâmetros de heterogeneidade de máquinas e de tarefas, alterando-se somente a granularidade das aplicações, demonstra o seguinte:

  1. Na primeira simulação, na qual as tarefas tem tamanho médio de 5000 segundos, o tempo médio de execução dos algoritmos se mantém próximo. A partir desse ponto, há uma degradação acentuada do algoritmo Worqueue. Até 25000 segundos, o algoritmo WQR tem um melhor desempenho. No entanto, ao aumentar a granularidade da aplicação (tarefas até 125000 segundos), os algoritmos dinâmicos melhoram o desempenho. Em contrapartida, o algoritmo WQR aumenta o tempo médio de execução da aplicação em decorrência da atribuição de tarefas maiores a máquinas mais lentas.
  2. No segundo cenário, com o incremento do índice de heterogeneidade das máquinas, o desempenho dos algoritmos baseados em informações dos recursos melhora em relação ao algoritmo WQR. Para minimizar a discrepância, o número de cópias das tarefas é aumentado no WQR.
  3. Na terceira simulação, com variação no valor do índice de heterogeneidade do tamanho das tarefas, o algoritmo WRQ apresenta melhor desempenho que os outros algoritmos, quando aumenta-se o número de cópias da tarefas para 3 e 4.
  4. Por fim, o último resultado do trabalho apresenta uma relação entre o consumo de CPU do algoritmo WQR quando incrementamos o número de cópias. Verifica-se o aumento acentuado desse consumo em relação ao crescimento da granularidade da aplicação.

- *Discussão*: a simplicidade de sua implementação, as características das tarefas (*BoT*) e a independência de informações sobre os recursos do sistema determinaram a escolha deste algoritmo para o p2pBIOFOCO. Dessa forma, verificaremos se os tempos observados na aplicação do algoritmo apresentam resultados proporcionais aos experimentos realizados na ferramenta de simulação (Simgrid). No nosso trabalho, utilizamos a aplicação de Bioinformática BLAST, definindo como tamanho das tarefas o tamanho de arquivos fragmentados obtidos a partir de um arquivo FASTA. Mais detalhes sobre a execução do algoritmo e sobre a sua implementação no p2pBIOFOCO serão discutidas na Seção 6.2.2.

"Low Complexity Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Environments"
Autores: E. Ilavarasan e P. Thambidurai [63]
Ano: 2007

- *Objetivo*: desenvolver um algoritmo dinâmico baseado em escalonamento em lista [11], com baixa complexidade e bom desempenho para o escalonamento de tarefas em sistemas computacionais heterogêneos. O algoritmo PETS (*Performance Effective Task Scheduling*) apresenta complexidade de tempo igual a $O(e)(p+\log v)$ para aplicações modeladas como DAGs (*Directed Acyclic Graph*), em que v é o número de tarefas, e é o número de arestas (representando a dependência entre as tarefas) e p representa o número de processadores.
- *Metodologia*: desenvolvimento de uma ferramenta para geração de DAGs (*Directed Acyclic Graphs*) e aplicação de problemas do mundo real (Decomposição LU, Transformações de Fourier e dinâmica molecular), comparando o PETS a três algoritmos que aplicam o mesmo método de escalonamento em lista, *LMT-Levelized Min Time*, *CPOP-Critical Path On a Processor* e *HEFT-Heterogeneous Earlist Finish Time*. As métricas de comparação foram as seguintes:
  - *Schedule Length Ratio (SLR)*: relação ente o *makespan* e a soma dos custos para computar as tarefas no caminho crítico;
  - aumento da rapidez de processamento (*speedup*): relação existente entre o tempo de execução sequencial e o tempo de execução em paralelo das tarefas;

[11] alocação de tarefas utilizando uma lista ordenada de processos baseada em prioridades, executada em dois passos: 1^0 - seleção do processo com prioridade mais alta; 2^0 - atribuição do melhor recurso ao processo selecionado

  - eficiência: relação entre o valor do *speedup* e o número de processadores usados no escalonamento;
  - número de ocorrências dos escalonamentos de melhor qualidade: número de vezes que cada algoritmo produz um escalonamento de qualidade melhor em relação aos outros algoritmos, e;
  - tempo de execução dos algoritmos: tempo para obtenção de um resultado para um determinado grafo de tarefas.

- *Resultados*: de acordo com os autores, o tempo de execução do algoritmo, após a realização dos experimentos foi de $O(e)(p + \log v)$, produzindo um tempo de execução melhor comparando-o aos tempos $O(p^2.v^2)$, $O(p^2.v)$ e $O(p^2.v)$ dos algoritmos LMT, HEFT e CPOP, respectivamente.

- *Discussão*: o estudo desse algoritmo teve como objetivo oferecer um contraponto à maioria dos algoritmos avaliados, que possuem como foco central o tratamento de tarefas independentes. Para o caso de precedência entre tarefas, um algoritmo com baixa complexidade e com boa aproximação à solução ótima é um grande desafio, em decorrência do tempo de execução não polinomial para esse tipo de problema.

"A Peer-to-Peer Meta-Scheduler for Service-Oriented Grid Environments"
Autores: K. Dörnemann, J. Prenzer e B. Freisleben [38]
Ano: 2007

- *Objetivo*: implementar um metaescalonador distribuído para sistemas baseados em computação em grade [51] [54], que seja tolerante a falhas, escalável, com balanceamento automático de carga e de fácil gerenciamento.

- *Metodologia*: combinação de tecnologias utilizadas em computação em grade orientada a serviços com técnicas empregadas em ambientes *peer-to-peer*. Utilização do algoritmo ACO (Ant Colony Optimization), para resolver problemas de otimização, baseado no comportamento natural das formigas e adaptado para redes P2P.

- *Resultados*: os autores utilizaram como métricas o desvio padrão observado na distribuição dos trabalhos entre os nós e o tempo requerido para a distribuição dos trabalhos entre os nós. Concluíram que um desvio menor do que 5 trabalhos representou uma boa qualidade no balanceamento dos trabalhos entre os nós do sistema. O tempo de 50 segundos para o escalonamento de 2000 trabalhos para 11 nós foi considerado rápido o bastante para a utilização

em aplicações do mundo real. A implementação do algoritmo de roteamento também foi avaliada e considerada eficiente em decorrência dos poucos *hops* verificados no processo de escalonamento do trabalho.

- *Discussão*: o enfoque dado pelos autores ao escalonador foi a descentralização, resultado da sua distribuição entre todos os nós do sistema. Em nossa proposta, também sugerimos a distribuição do escalonador entre todos os nós do sistema, mas com a possibilidade de alternar o algoritmo de escalonamento, conforme o comportamento do sistema. No texto, observamos que não há referência ao contexto das aplicações, sendo consideradas, provavelmente, aplicações que são próprias para execução em plataformas distribuídas.

"Scheduling Independent Tasks on Metacomputing Systems"
Autores: H. A. James, K. A. Hawick e P. D. Coddington [68]
Ano: 1999

- *Objetivo*: desenvolver um modelo de escalonamento hierárquico para o gerenciamento em larga escala de trabalhos em um sistema metacomputacional[12].
- *Metodologia*: estudo de políticas de escalonamento para obtenção de um desempenho próximo ao desempenho ótimo de execução de tarefas independentes, com a análise da execução de cinco algoritmos de escalonamento. A utilização de cinco tipos de distribuição das tarefas com variação de seus tempos no intervalo entre 0,1 e 1,0 segundo. Algoritmos utilizados: *Round-Robin*, *Clustered Round-Robin*, *Minimal Adaptive* (obtém um conjunto de informações mínimas sobre o desempenho dos nós para a atribuição dos trabalhos remanescentes), *Continual Adaptive* (análise de desempenho dos nós antes de enviar os "jobs") e FCFS (*first-come, fist-serve*). Utilização de distribuições dos tempos de execução das tarefas, a saber: Uniforme, Normal, Poisson, Bimodal e Randômica.
- *Resultados*: se a distribuição dos tempos de execução é conhecida, como no caso de distribuições homogêneas, o algoritmo *clustered round-robin* é o mais indicado (menor tempo de latência e de *overhead* de predição). Para aquelas distribuições onde existe simetria ou alto grau de clusterização (ex.: Poisson e normal), o algoritmo com melhor desempenho foi o *continual adaptive*. Para distribuições que possuem aleatoriedade em sua configuração (aleatória e bimodal), a escolha do algoritmo FCFS, conforme os resultados, é a melhor.

[12]Um sistema metacomputacional é um sistema executado em um metacomputador. Conforme Foster et al [52], um metacomputador é um computador virtual criado pela interligação de vários recursos distribuídos geograficamente, através de redes de alta velocidade.

- *Discussão*: os aspectos do trabalho que se destacaram foram a caracterização das tarefas como independentes, as distribuições dos tempos de execução e a comparação efetuada entre os algoritmos:
  - a independência das tarefas faz parte do escopo do nosso trabalho;
  - as distribuições sugeridas para a chegada de tarefas ao sistema, considerando os intervalos dos tempos de execução, tentam simular a execução em um ambiente real. Dessa forma, tentamos estabelecer semelhanças entre as distribuições observadas (aleatórias, normais e de Poisson) e o padrão de submissão de tarefas ao p2pBIOFOCO;
  - a comparação entre os 5 algoritmos estudados produz várias informações que são úteis para o nosso trabalho, especialmente, com relação às distribuições dos tempos de execução, à utilização de algoritmos com informações sobre o tempo de execução das tarefas e as vantagens e as desvantagens identificadas em cada abordagem.

"Load Sharing in Distributed Systems"
Autores: Y.-T. Wang e R. J. T. Morris [108]
Ano: 1985

- *Objetivo*: propor uma taxonomia de algoritmos de balanceamento de carga em sistemas distribuídos que represente uma dicotomia básica entre as abordagens do balanceamento iniciado pelo recurso e do balanceamento iniciado pelo servidor.
- *Metodologia*: aplicação de um parâmetro de desempenho criado pelos autores e denominado *Q-factor* (*quality of load sharing*) para avaliar a eficiência de um algoritmo no balanceamento de carga. O parâmetro tenta demonstrar o comportamento do balanceamento de carga no sistema, destacando deficiências referentes à existência de nós ociosos enquanto existem trabalhos esperando para serem executados. Seu cálculo tem como parâmetros principais o tempo médio de resposta de todos trabalhos executados quando utiliza-se o algoritmo *first-come first-served* (FCFS) e o tempo médio máximo de resposta do recurso i quando aplicado determinado algoritmo A.
- *Resultados*:
  - a escolha do algoritmo de balanceamento de carga é uma decisão crítica no projeto do sistema distribuído;

  - para o mesmo nível de troca de informações, os algoritmos de escalonamento executados no servidor têm mais chance de serem melhores em desempenho do que os algoritmos executados nos recursos;
  - alguns dos algoritmos estudados, como *multiserver cyclic server* [13], subestimados em estudos anteriores, produziram soluções efetivas de acordo com os resultados apresentados pelos autores.

- *Discussão*: os autores criaram um parâmetro (*Q-factor*) para uma distribuição mais justa dos trabalhos entre os nós de uma rede, propiciando um balanceamento de carga. Esse fator, com a apropriada adaptação para sistemas heterogêneos, poderia ser aplicado ao nosso sistema para a obtenção da qualidade do balanceamento na implementação de determinado algoritmo. No estudo, ressaltamos que o desempenho dos algoritmos não está atrelado somente ao desempenho intrínseco de sua execução, mas também aos vários parâmetros determinados pelos autores em suas simulações.

"'Scheduling in a dynamic heterogeneous distributed system using estimation error"
Autores: A. J. Page, thomas M. K. and T. J. Naughton [84]
Ano: 2008

- *Objetivo*: apresentar um algoritmo de escalonamento que trata características restritivas do escalonamento de tarefas (heterogeneidade das tarefas, heterogeneidade de recursos, DAG's com restrições de comunicação e tarefas, custos de comunicação, etc) através da técnica de estimativa de erro, obtida dinamicamente. O algoritmo é definido como um par de processos, com um processo executando o gerenciamento das tarefas (centralizado em um servidor) e o processo de escalonamento, propriamento dito, distribuído para cada nó.

- *Metodologia*: geração de uma matriz de tempo estimado de computação (ETC) de cada tarefa em cada processador, utilizando duas técnicas para geração da matiz: *k-nearest neighbours* (k-NN) e *smoothed average* com *benchmark analítico*[14].

- *Resultados*: um escalonador simples que inicia o processo de escalonamento sem o conhecimento prévio dos recursos do sistema ou da complexidade dos

[13] Algoritmo no qual cada servidor visita os recursos de maneira cíclica, exaustivamente ou até determinado limite, removendo e disponibilizando determinado número de trabalhos

[14] Teste que avalia o desempenho das máquinas disponíveis com a execução de determinado tipo de código [71].

trabalhos a serem submetidos. Os testes (efetuados com até 90 processadores utilizando problemas das áreas de Ciência da Computação, Bioinformática e Engenharia Biomédica) demonstraram que o algoritmo é eficiente para obtenção de um baixo *makespan*, executando tão bem quanto um algoritmo complexo estabelecido sobre uma heurística evolucionária.

- *Discussão*: apesar da centralização do gerenciador de tarefas, o método de estimativa de erro parece-nos apropriado para utilização também em escalonadores descentralizados, com a execução de uma instância do escalonador em cada nó e a adaptação do módulo de gerenciamento para a sua execução em todos os nós. Notamos, também, que a flexibilidade do gerenciador de tarefas para tratamento de tarefas independentes e com diversos graus de dependência é um conceito útil para futuras versões do p2pBIOFOCO.

"Scheduling Independent Tasks Sharing Large Data Distributed with BitTorrent"
Autores: B. Wei, G. Fedak e F. Capello [110]
Ano: 2005

- *Objetivo*: propor um modelo que selecione o melhor protocolo entre o BitTorrent e o FTP (File Transfer Protocol) [60], de acordo com o tamanho do arquivo a ser distribuído e com o número de nós do sistema. Propor uma evolução no protocolo BitTorrent que permita a identificação de mais padrões de comunicação.

- *Metodologia*: a simulação da distribuição dos arquivos foi registrada em *traces* obtidos da rede e da CPU. A simulação da rede utilizou os tempos de transferência dos arquivos registrados em cada nó. O desempenho de cada CPU foi modelado usando ferramenta apropriada para a apuração da disponibilidade da CPU durante um período de 51 dias. As tarefas foram definidas pelo número de operações que executam. No escalonamento das tarefas foram utilizadas duas heurísticas: uma básica (*Round Robin*) e outra baseada em informações dos nós (Min-Min, Max-Min e *Sufferage*).

- *Resultados*: a implementação do modelo que seleciona o melhor protocolo de acordo com o tamanho do arquivo a ser distribuído e com o número de nós foi feita com sucesso. As heurísticas baseadas em algum tipo de informação obtida do sistema (carga da CPU, taxa de transferência de dados, etc) conseguiram um incremento de até três vezes na rapidez de execução dos trabalhos, quando comparado o protocolo BitTorrent adaptado para MinMin, MaxMin e

Sufferage com o protocolo FTP-*Round Robin*, além de aumento de 1,5 quando comparado com o protocolo *BitTorrent-Round Robin*.

- *Discussão*: a principal ideia do texto, referente à distribuição dos dados utilizando *BitTorrent* ou FTP, é uma das principais alterações que propusemos para o p2pBIOFOCO. Tendo em vista a submissão dos dados referentes aos arquivos FASTA, a atualização das bases biológicas de dados, geralmente na faixa de *gigabytes*, e a disponibilização dos resultados das tarefas aos nós, a alternativa parece-nos viável para aumentar o desempenho do sistema com relação à transferência dos dados.

"Algorithms for Scheduling Independent Tasks"
Autores: S. K. Sahni [95]
Ano: 1976

- *Objetivo*: estudar quatro problemas de escalonamento de tarefas:

  1. sequenciamento de trabalhos em um único processador com *deadlines*;
  2. sequenciamento de trabalhos em *m* processadores idênticos para minimizar o tempo de término;
  3. minimização do *minimum flow time* com a utilização de escalonamento SPT (*shortest process time first*);
  4. sequenciamento de trabalhos em dois processadores idênticos para minimizar *weigthed mean flow time*. Utilizar algoritmos de programação dinâmica para obtenção de soluções ótimas e apresentação de três técnicas genéricas para obter soluções aproximadas.

- *Metodologia*: descrição dos problemas de escalonamento utilizados no estudo; descrição e análise dos algoritmos ótimos existentes para solução desses problemas; utilização das técnicas *digit truncation*[15], *dynamic interval partitioning*[16] e *static interval partitioning* para elaboração de algoritmos aproximados que resolvam os problemas apresentados obtendo uma boa solução.

- *Resultados*: para os problemas (1) *Job Sequencing with Deadlines* (JSD) e (3) Weighted Mean Flow Time (WMFT) os autores produziram algoritmos de

[15]Método utilizado para a redução de alguns ou de todos os números de um problema, para que possa ser executado em tempo polinomial sobre a instância reduzida [94].

[16]No método *interval partition*, o espaço de soluções é dividido em intervalos e, para cada intervalo, consideramos somente uma das soluções possíveis contidas no intervalo.

complexidade O(n^2/ϵ), em que ϵ é igual ao fator de aproximação, que garantiram uma solução próxima da solução ótima. Quando generalizaram os métodos aplicados para uma quantidade de processadores maior que 2 ($m > 2$), os algoritmos não foram eficientes;

- *Discussão*: para o p2pBIOFOCO, a contribuição deste trabalho resume-se ao escalonamento local das tarefas em máquinas da rede *peer-to-peer* ou, no máximo, restrita a pares de máquinas com os mesmos recursos, pois os algoritmos estudados consideram somente um número de processadores $m \leq 2$. Assim, a eficiência provada para o algoritmo aproximado que trata o problema JSD (*Job Sequencing with Deadlines*), por exemplo, poderia ser aplicada aos nós individualmente, constituindo um procedimento de alocação global de tarefas (executado num primeiro passo) somado às propostas de Sahni para um ou dois nós do sistema.

"The Performance of Bags-of-Tasks in Large-Scale Distributed Systems"
Autores: A. Iosup, O. Sonmez, S. Anoep e D. Epema [64]
Ano: 2008

- *Objetivos*:

  1. propor uma abordagem sistemática para avaliar o escalonamento de BoTs em grandes sistemas de computação distribuída, com a identificação de três classes de políticas de escalonamento;
  2. propor um modelo de trabalho para soluções baseadas em BoTs, validando o modelo através de informações colhidas de sistemas de computação distribuída;
  3. realizar uma investigação realística e compreensiva do desempenho de BoTs em sistemas distribuídos.

- *Metodologia*: descrição dos modelos de sistemas *clusters* e dos trabalhos submetidos ao sistema; apresentação de arquiteturas de gerenciamento de recursos, com inclusão dessas arquiteturas no conjunto de *clusters*; definição de políticas de seleção de tarefas; definição de políticas de escalonamento de tarefas; definição do modelo para BoTs utilizando quatro aspectos: submissão do usuário, padrões de chegada do BoT, tamanho do BoT e características das tarefas que compõem o BoT; simulação da execução dos BoT em sistema distribuído usando o simulador DGSim; avaliação das execuções utilizando as métricas *makespan* do BoT e NSL (Normalized Sheduled Length).

- *Resultados*: os resultados foram organizados de acordo com os impactos do modelo de carga do trabalho adotado e das políticas de escalonamento e seleção de tarefas no desempenho de aplicações BoT em sistemas distribuídos. Desse modo, enumera-se esses impactos da seguinte maneira:

  - impacto da política de escalonamento: políticas do tipo (U,*), ou seja, que não utilizam informações sobre os recursos - *Unkonowledgement* - e são independentes em relação a informações sobre as tarefas (U,*) são as mais adaptadas às aplicações BoTs;
  - impacto das características da carga de trabalho (*workload*): considerando os três padrões de chegada de trabalhos ao sistema: realístico, com intervalos variáveis entre as chegadas das tarefas; cíclico controlado, com a chegadas de todas as tarefas em intervalos de tempo iguais; e, *all at T-0*, com a chegada de todas as tarefas ao mesmo tempo. Verifica-se no último caso um aumento substancial do *makespan*. A chegada das tarefas em intervalos regulares mostrou-se mais favorável em cinco tipos de escalonamento em relação ao modelo realístico.
  - impacto de informações dinâmicas do sistema: políticas baseadas em predição têm um impacto maior no desempenho que outras políticas de escalonamento;
  - impacto da política de seleção de tarefas: é importante para os sistemas com carga acima de 50 por cento. A seleção de tarefas por ordem de chegada tem melhor desempenho em várias configurações de carga;
  - impacto da arquitetura de gerenciamento de recursos: escalonador centralizado com gerenciamento local de processos.

- *Discussão*: o trabalho de Iosup et al é centrado na criação de um modelo para aplicações BoTs em sistemas de computação distribuída. Características como o padrão de chegada das tarefas ao sistema, tipos de aplicação usuária e as propriedades das tarefas são analisadas com o objetivo de encontrar uma estatística real de distribuição das tarefas. No texto são discutidas, também, questões importantes relativas ao desempenho das BoTs e ao impacto das políticas de escalonamento. Neste ponto, enquadramos o nosso projeto, em razão de suas similaridades com a pesquisa desenvolvida. Nosso sistema é utilizado por aplicações BoTs e está sujeito à inclusão e ao estudo de políticas de seleção de tarefas e de escalonamento, como mencionado no texto dos autores.

"Scheduling Data-Intensive Bags of Tasks in P2P Grids with BitTorrent-enabled Data Distribution"
Autores: C. Briquet, X. Dalem, S. Jodogne e P.-A. Marneffe [15]
Ano: 2007

- *Objetivo*: propor a combinação de várias tecnologias e padrões existentes para a execução de BoTs em sistemas *peer-to-peer*.
- *Metodologia*: utilização do protocolo de compartilhamento de dados BitTorrent (BT) para a transferência de dados; *caching* de dados nos recursos computacionais; utilização do método *Storage Affinity*; elaboração de um novo algoritmo de escalonamento de tarefas *Temporal Tasks Grouping*, baseado no compartilhamento de dados observado nas tarefas; e replicação de dados.
- *Resultados*: o uso do protocolo BitTorrent para transferir grande quantidade de dados compartilhados, diminui ou evita a formação de gargalos no sistema; o suporte de *caching* evita a transferência de dados recentemente usados pelo recurso; um algoritmo de escalonamento e seleção de recursos minimiza a quantidade de dados de entrada para uma dada tarefa; a combinação de todas as técnicas apresentadas mais a utilização do algoritmo proposto, *Temporal Taks Grouping* (TTG), é eficiente para diversas configurações do sistema.
- *Discussão*: o trabalho tem pontos em comum com o sistema p2pBIOFOCO. A transferência de arquivos e a utilização do algoritmo proposto no módulo de escalonamento agregam ganho de desempenho na execução das tarefas submetidas ao sistema. A utilização da técnica de *caching* pode ser implementada para o problema de submissão de tarefas referentes às sequências de pesquisa duplicadas para os nós que já executaram ou estão executando uma tarefa que tem essas sequências como entradas.

"Improving Peer-to-Peer Performance through Server-Side Scheduling"
Autores: Y. Qiao, F. E. Bustamante, P. A. Dinda, S. Birrer e D. Lu [86]
Ano:2008

- *Objetivo*: mostrar como aplicar a heurística SRPT (*Shortest Remaining Processing Time*) no escalonamento de trabalhos em sistemas *peer-to-peer* de compartilhamento de arquivos. Verificar o incremento apresentado no desempenho em relação aos algoritmos FCFS (*First Come Fist Served*) e PS (*Processor Sharing*) e como a aplicação dessa heurística reduz significativamente o tempo médio de resposta. Apresentar um algoritmo adaptado a partir do SRPT.

- *Metodologia*: análise detalhada para predição do tempo dos serviços de cada requisição e introdução de um novo módulo de predição nas políticas de escalonamento do SRPT. A efetividade das políticas de predição do algoritmo adaptado é avaliada com a utilização do método *trace-driven simulations*. Experiência real do método de predição proposto através de sua implementação em uma rede *eDonkey*.
- *Resultados*: sob várias cargas de trabalho, algoritmos baseados em SRPT produzem tempos de resposta mais rápidos para os usuários, sem sacrificar os critérios de eqüidade (*fairness*).
- *Discussão*: possível uso do algoritmo de escalonamento no p2pBIOFOCO, com a finalidade de diminuição do tempo de resposta dos nós, com o seguintes objetivos:
  - diminuir o impacto de sobrecargas para o algoritmo quando grande número de requisições é feito no *peer* servidor;
  - analisar os nós sob a perspectiva de servidores, estudando a carga de trabalho quando submetidos a uma grande quantidade de requisições;
  - analisar o padrão de distribuição de chegada dos trabalhos, tamanhos dos trabalhos e tempo para execução dos serviços;
  - entender a utilização dos nós como servidores dentro da estrutura das redes *peer-to-peer*;
  - estudar a integração da política local do escalonador com a política definida para os serviços do sistema;
  - verificar o módulo de predição no contexto do *peer-to-peer*.

"Comparison of Heuristics for Scheduling Independent Tasks on Heterogeneous Distributed Environments"
Autores: H. Izakian e A. Abraham e V. Snasel [66]
Ano:2009

- *Objetivo*: o principal objetivo dos autores é comparar algumas heurísticas já conhecidas e propor uma nova heurística (min-max) que apresente resultados melhores no escalonamento das tarefas, minimizando o *makespan* e o *flowtime*;
- *Metodologia*: comparação de desempenho das heurísticas utilizando o *benchmark* proposto por Braun et al [13], com a utilização, também, de um modelo

de simulação com um tempo estimado para computar uma matriz de 512 trabalhos em 16 máquinas. As instâncias do *benchmark* foram classificadas em 12 tipos diferentes de matrizes, de acordo com as três seguintes métricas: heterogeneidade de tarefas, heterogeneidade de máquinas e consistência;

- *Discussão*: neste trabalho destacamos os resultados da implementação do algoritmo min-max, expressando um desempenho maior, se comparado ao algoritmo workqueue (WQR).

Resumimos na tabela 4.1 os principais pontos observados nos trabalhos mencionados, destacando as características que foram incorporadas ao p2pBIOFOCO.

Os trabalhos incluídos na tabela 4.2 referem-se aos trabalhos pertinentes à classificação dos escalonadores e ao estudo de algoritmos para escalonamento não distribuído.

Nas tabelas 4.3, 4.4, 4.5 e 4.6 agrupamos os escalonamento conforme a arquitetura do sistema distribuído.

4.5 Escalonamento com Heurística Min-Max

Para o entendimento dessa heurística, descrevemos nesta seção algumas informações sobre o conceito de mapeamento de tarefas com a apresentação de algumas definições. Logo após, discutimos dois algoritmos básicos que utilizam o tempo de execução e o tempo de processamento como métricas e critérios para o escalonamento das tarefas.

Em geral, em um sistema computacional heterogêneo são necessários esquemas para atribuir tarefas às máquinas (*matching)* e para computar a ordem de execução dessa tarefas (*scheduling)*. O processo de encontrar e escalonar as tarefas é conhecido como mapeamento (*mapping*). Um mapeamento dinâmico é executado assim que as tarefas chegam para o processo de atribuição e escalonamento, enquanto no mapeamento *estático* conhecemos antecipamente o conjunto de tarefas e as suas características [77].

O *tempo esperado de execução* e_{ij} da tarefa t_i na máquina m_j é definido como o tempo que a máquina m_j levou para executar a tarefa t_i considerado que a máquina m_j não tinha nenhuma carga na atribuição da tarefa. Quanto ao *tempo esperado* c_{ij} para completar a tarefa t_i na máquina m_j é definido como o tempo de processamento (*wall-clock*) no qual a máquina m_j completa a tarefa t_i. Em seguida, são explicadas as duas heurísticas utilizadas pela heurística Min-Max.

Tabela 4.1: Algoritmos de Escalonamento - Resumo em Ordem Cronológica

Autores	Arquiteturas			Escalonadores		
	Arquitetura	Sistema	Tarefas	Algoritmos	Objetivo	Comentários para o p2pBIOFOCO
James (1999) [68]	cluster	Distributed Java Platform [70]	independentes	*Round Robin, Clustered RR, Minimal Adaptive, Continual Adaptive e FCFS*	escalonamento hierárquico para gerenciamento em larga escala.	método para estimativas e obtenção dos tempos de execução das tarefas; estudo dos padrões da chegada das tarefas ao sistema.
Arora (2002) [6]	grid	Implementado com MPI	independentes	algoritmo de Arora et al	escalonamento de tarefas baseado em funções de maximização e minimização, com balanceamento de carga.	funções e estruturas de dados (filas) para a construção do escalonador; estudo sobre o impacto da latência de comunicação entre os nós.
Cirne (2003) [99]	grid	Ourgrid	*bag of tasks*	*WQR - Workqueue with Replication*	minimizar o problema do escalonamento dinâmico de tarefas independentes.	independência de informações sobre o sistema; *bag of tasks*; simplicidade de implementação.
Wei (2005) [110]	cluster	*LRI Simulation Cluster*	parameter sweep	DP-RR *Dual Protocol with Round Robin*	com base na política de escalonamento, seleção do melhor protocolo, FTP ou BitTorrent, para a difusão dos dados.	distribuição dos arquivos com sequências biológicas utilizando um dos protocolos mencionados.
Ilavarasan (2007) [63]	Distribuído	heterogêneo	com precedência	PETS - *Performance Effective Task Scheduling*	desenvolvimento de um algoritmo de escalonamento dinâmico, baseado em lista, com baixa complexidade e bom desempenho.	implementação de algoritmo para tratamento de dependência de tarefas; contraponto para os algoritmos estudados, mostrando um modelo para tarefas não independentes.
Dornemann (2007) [38]	grid/P2P	*P2P Meta-Scheduler* baseado no FreePastry P2P [92]	independentes	ACO - *Ant Colony Optimization*	implementação de um metaescalonador, com balanceamento automático, tolerante a falhas, escalável e de fácil gerenciamento	distribuição da política de escalonamento entre os nós do sistema; implementação de um metaescalonador com integração dos escalonadores propostos.
Briquet (2007) [15]	grid/P2P	LBG - Lightweight Bartering Grid [16]	*bag of tasks*	TTG - *Temporal Tasks Grouping*	proposta de combinação de várias técnicas para a execução de *bag of tasks* em sistemas *peer-to-peer* e implementação de algoritmo de difusão de dados baseado nos protocolos FTP e BitTorrent	utilização no p2pBIOFOCO do método de difusão dos dados, implementado no algoritmo proposto; possibilidade de execução de caching, para as sequências mais utilizadas na execução da ferramenta de bioinformática.
Tada (2008) [103]	grid	Heterogêneo	*bag of tasks*	WQR - *Workqueue With Replication*	proposta de método para controle da utilização da CPU, concomitante ao tempo de execução das tarefas quando aplicado o algoritmo WQR	métodos propostos: *soft state (SS)* e *task timeout* (TT) são duas propostas que podem ser aplicadas ao algoritmo WQR implementado no p2pBIOFOCO, para avaliação do ganho de desempenho e de tolerância a falhas.
Qiao (2008) [86]	P2P	Gnutella [90]	independentes	SRPT - *Shortest Remaining Processing Time*	estudo da aplicação da heurística em sistema *peer-to-peer*; comparação com algoritmos FCFS (*Fist Come Fist Served*) e PS (*Processing Sharing*	trabalho utilizado como referência para entender a carga submetida aos nós de um sistema *peer-to-peer* quando os nós atuam como servidores.
Iosup (2008) [64]	cluster/grid	DGSim [65]	*bag of tasks*	ECT, FPLT, RR, WQR, DFPLT, ECT-P, STFR e FPF	avaliação do escalonamento de *bag of tasks* em sistemas distribuídos	centralizado na análise do padrão de chegada das tarefas ao sistema, suas características e a submissão dessas tarefas pelo sistema; aplicação dos conceitos quando o sistema p2pBIOFOCO está sob utilização plena, com submissão de grande quantidade de requisições.
Page (2008) [84]	distribuída	Heterogêneo	independentes	Utiliza funções de custo FA, FE e FZ	implementação de um algoritmo de escalonamento com políticas de tratamento para restrições impostas na alocação das tarefas	potencial aproveitamento do sistema gerenciador de recursos (RMS - *Resource Management System*) proposto no texto para sistemas *peer-to-peer* com algumas adaptações para seu uso descentralizado
Izakian (2009) [66]	distribuída	Heterogêneo	independentes	Min-Max	comparação de várias heurísticas conhecidas e proposta de uma nova heurística para minimização do *makespan* e do *flowtime*	implementação da heurística Min-Max comparando-a ao algoritmo WQR, com a avaliação do desempenho dos dois algoritmos.

- *Minimum Execution Time (MET)*: também conhecida como LBA (*Limited Best Assignment*), essa heurística atribui cada tarefa para a máquina de melhor expectativa de tempo de execução daquela tarefa, desconsiderando a disponibilidade da máquina [5]. No entanto, esse método pode causar o desbalanceamento de carga, produzindo um desempenho ruim.
- *Minimum Completion Time (MCT)*: atribui cada tarefa para a máquina que possui o *minimum completion time* daquela tarefa [5]. Desse modo, algumas tarefas podem ser atribuídas para máquinas que não possuem o menor tempo

Tabela 4.2: Escalonamento não distribuído e Classificação

Autores	**Arquiteturas**			**Escalonadores**		
	Arquitetura	Sistema	Tarefas	Algoritmos	Objetivo	Comentários para o p2pBIOFOCO
Sahni(1976) [95]	não distribuída	diversos	independentes	diversos	revisar e estudar quatro problemas de escalonamento: sequenciamento de trabalhos com prazos finais, sequenciamento de trabalhos para $m \geq 2$ processadores, minimizar *mean flow time* do escalonamento com menor tempo de finalização e minimizar *mean flow time* ponderado para $m \geq 2$.	escalonamento local das tarefas em máquinas da rede *peer-to-peer*.
Casavant e Kuhl(1988) [22]	diversas	diversos	diversas	diversos	propor uma taxonomia híbrida (horizontal e hieráquica) para o problema do gerenciamento de recursos.	classificação dos algoritmos utilizados no p2pBIOFOCO.

Tabela 4.3: Escalonamento para Sistemas Grid

Escalonador	Sistema	Objetivo	Autores
Algoritmo de Arora et al	Distribuído (*n nodes*)	melhorar escalonamento considerando *overhead* da coordenação entre nós	Arora et al
WQR - *Worqueue with Replication*	Ourgrid	oferecer alternativa simples ao escalonamento dinâmico de tarefas	Cirne et al
WQR - modificado	Ourgrid	minimizar problemas de detecção de falhas e de cancelamento das tarefas	Tada et al

Tabela 4.4: Escalonamento para Sistemas P2P

Escalonador	Sistema	Objetivo	Autores
SRPT - *Shortest Remaining Processing Time*	Gnutella	analisar desempenho dos nós de sistemas P2P, quando atuam como "servidores"	Qiao et al
Worqueue (WQ)	p2pBIOFOCO	fragmentar arquivos de Bioinformática e alocar os fragmentos parar recursos disponíveis no sistema	Ribeiro

Tabela 4.5: Escalonamento para Sistemas Grid/P2P

Escalonador	Sistema	Objetivo	Autores
ACO - Ant Colony Optmization	Metascheduler (base FreePastry)	implementar escalonador tolerante a falhas, escalável e de fácil gerenciamento	Dornemann et al
TTG - Temporal Tasks Grouping	LBG - Lightweight Bartering Grid	implementar método de difusão de dados e combinar várias técnicas para executar *BoT*	Briquet et al

de execução, em decorrência do alto tempo em que a tarefa fica no estado de pronta (*ready*), para a sua execução. A intuição do procedimento é evitar o desempenho pobre da estratégia de atribuição das tarefas às próximas máquinas disponíveis (não considerando o menor tempo tempo de execução) e do procedimento MET, que não considera o tempo de prontidão da máquina

Tabela 4.6: Escalonamento para Sistemas Distribuídos em Geral

Escalonador	Sistema	Objetivo	Autores
Min-Max	Distribuído	minimizar *makespan* e *flow-time* comparando com outras heurísticas	Izakian et al
DP-RR (Dual Protocol - Round Robin)	Cluster	implementar método eficiente para difusão de dados	Wei et al

(*ready time*) [13].

Neste ponto, podemos mencionar o algoritmo *switching algorithm* (*SA*) que utiliza as duas heurísticas em sua execução. A heurística MET pode potencialmente criar um desbalanceamento de carga entre as máquinas por atribuir mais tarefas para algumas máquinas do que para outras, enquanto a heurística MCT tenta balancear a carga atribuindo tarefas para a máquina com menor tempo para completar a tarefa [56]. Se as tarefas não possuem um padrão de chegada ao escalonador, é possível a utilização da heurística MET, com o custo do balanceamento até um determinado limite. Se esse custo for ultrapassado, troca-se o método de escalonamento para o MCT, amenizando, portanto, o impacto do desbalanceamento do algoritmo aplicado anteriormente. Desse modo, o algoritmo *SA* tenta prover ao escalonador as propriedades desejáveis do MET e do MCT.

Diante do exposto, a heurística Min-Max [66], assim como o algoritmo SA, utiliza, também, as duas heurísticas mencionadas (MCT e MET) em sua implementação. Executada em dois passos, a heurística minimiza primeiro um critério de avaliação para, em seguida, maximizar um segundo critério utilizando-os como métricas do escalonamento. No algoritmo 2 vemos a implementação da heurística.

O primeiro passo refere-se à utilização da métrica concernente ao tempo mínimo de processamento da tarefa (linhas 1 a 6). Então, conforme visto na definição do algoritmo MCT, temos a definição de um conjunto U de tarefas com a obtenção de um conjunto de tempos mínimos caracterizados por:

$$M = min(tempo\_minimo(T_i, M_i)|(1 \leq i \leq n, 1 \leq j \leq m)$$

Em seguida, executa-se o segundo passo da heurística (linhas 8 a 18): calcula-se o menor tempo para completar a tarefa em cada máquina e dividi-se o menor tempo de execução (tempo esperado para a execução da tarefa na máquina mais rápida) pelo tempo para completar a tarefa da máquina selecionada, sendo o maior valor incluído no mapeamento entre tarefas e máquinas.

O resultado da execução é um par máquina-tarefa (T_i, M_j) no qual a máquina selecionada pode executar a sua tarefa correspondente, com um tempo de execução mais baixo se comparado com as outras máquinas. Assumindo que $C_{i,j}(i \in$

Algorithm 2 Implementação da Heurística Min-Max (Adpatado de Maheswaran [77])

1: **for all** tarefas t_i no trabalho J_v (em ordem arbitrária) **do**
2: **for all** máquinas m_j (em uma ordem arbitrária fixada) **do**
3: e_{ij} = *tempo de execução da tarefa* t_i *na máquina* m_j
4: **end for**
5: **end for**
6: para cada tarefa em J_v encontre o menor tempo de execução da tarefa e a máquina responsável por esse tempo (máquina m_l)
7: **repeat**
8: **for all** tarefas t_i no trabalho J_v (em ordem arbitrária) **do**
9: **for all** máquinas m_j (em uma ordem arbitrária fixada) **do**
10: $c_{ij} = e_{ij} + r_{ij}$
11: **end for**
12: **end for**
13: para cada tarefa em J_v encontre o menor tempo para completar a tarefa e a máquina responsável por esse tempo (máquina m_l)
14: encontre a tarefa que produz o maior valor (Max) como resultado da divisão entre o seu tempo mínimo de execução e o tempo para completar a tarefa na máquina m_l
15: executar a atribuição da tarefa à máquina m_l
16: excluir a tarefa do trabalho J_v
17: atualizar *ready time* das tarefas restantes
18: atualizar *completion time* para as tarefas restantes
19: **until** até todas as tarefas em J_v estarem mapeadas

$1, 2, ..., m; j \in 1, 2, ..., n)$ é o tempo de execução da *j*-ésima tarefa na *i*-ésima máquina e $W_i (i \in 1, 2, ..., m)$ é a carga anterior da máquina M_i então, na equação 4.8, temos o tempo requerido para M_i completar as tarefas atribuídas a ela. Desse modo, o *makespan* e o *flowtime* podem ser estimados conforme as equações 4.9 e 4.10.

$$\sum C_i + W_i \tag{4.8}$$

$$makespan = max \sum C_i + W_i, i \in 1, 2, ..., m \tag{4.9}$$

$$flowtime = \sum_{i=1}^{m} C_i \tag{4.10}$$

Na Figura 4.4, vemos uma atribuição de tarefas efetuada no passo 1 da heurística Min-Max.

	machines		
assignment	m_1 CT	m_2 CT	m_3 CT
$t_1 \rightarrow m_2$	3	<u>2</u>	<u>2</u>
$t_2 \rightarrow m_2$	4	<u>3</u>	4
$t_3 \rightarrow m_3$	5	7	<u>3</u>
$t_4 \rightarrow m_1$	<u>4</u>	8	7

Figura 4.4: Atribuição das tarefas às máquinas do sistema considerando MCT (*minimum completion time*) [14].

4.6 Escalonamento com Agrupamento de Tarefas

O último método detalhado neste trabalho foi proposto por Briquet et al [15] com a intenção de combinar as políticas de escalonamento e de transferência de dados em uma arquitetura P2P não estruturada. Os autores propõem uma combinação de padrões de escalonamento de tarefas, técnicas de *caching*, técnicas de replicação e a utilização do protocolo BitTorrent para a transferência dos dados (ver Seção 6.2). Neste método é utilizado um modelo de grade (*grid*) para o envio e o recebimento dos tempos de execução das tarefas, com o emprego de um método chamado de *escambo* ou *permuta*.

No contexto de sistemas P2P, os recursos de um nó são utilizados por outros nós, criando um sistema de trocas de recursos entre os nós. Essa troca de recursos é um modo de potencializar o tempo de execução dos trabalhos, com a utilização do consumo sincronizado de múltiplos recursos de diversos nós. A arquitetura LBG

(Lightweight Bartering Grid Architecture), resultado do trabalho de Briquet et al, é um exemplo de utilização desse modelo de trocas para a construção de um sistema P2P, no qual são permutadas mensagens com informações sobre o tempo de execução das tarefas nos nós.

As aplicações submetidas ao LBG P2P são classificadas como *bag of tasks* (BoT) com a utilização intesiva de dados, caracterizando-as como tarefas PHD (*processors of huge data*0 [96]. Essas aplicações estão relacionadas a diversas áreas que utilizam grandes quantidades de dados no processamento, e mais especificamente, no contexto deste trabalho, à área da Bioinformática.

Em sistemas P2P, os tempos de execução e a transferência de dados podem ser variáveis e os conflitos no uso dos recursos podem ocorrer mais frequentemente do que o desejado. Briquet et al utilizaram técnicas de *caching* de dados, de replicação de dados e de afinidade de localização para a distribuição das tarefas no sistema, com a finalidade de amenizar os problemas oriundos dos conflitos mencionados.

A *replicação* dos dados é uma técnica que utiliza a execução de réplicas das tarefas em nós, assim que o escalonador tenha ciência deles, sem utilizar qualquer métrica para cálculo do custo da realização da tarefa em determinado nó. Os algoritmos de escalonamento que usam esse conceito são mais simples, executados assincronamente e conseguem um bom desempenho. Como exemplo dessa técnica temos o algoritmo proposto em Cirne et al [99], discutido na Seção 6.2.

Outra técnica importante para o escalonamento de tarefas e transferência de dados é a *afinidade de localização*, responsável pelo escalonamento de tarefas para os recursos computacionais onde a maioria dos dados de entrada requeridos já estão à disposição da aplicação. Um dos algoritmos mais conhecidos que implementa esta técnica é o *Storage Affinity* [96] presente no sistema *OurGrid* [4].

Com a implantação das técnicas mencionadas, os autores propõem a implementação do *TTG (Temporal Tasks Grouping)*. Trata-se de um método de escalonamento baseado no agrupamento de tarefas que utilizam temporariamente o mesmo conjunto de dados. Junto com as estratégias apontadas anteriormente, provê uma solução para execução de tarefas que têm uma grande quantidade de dados de entrada. Para a compreensão deste método, seus autores descrevem dois procedimentos principais para a execução do escalonamento: a *seleção de tarefas* e a *seleção de recursos*.

4.6.1 Seleção de Tarefas

Na seleção de tarefas é efetuada a ordenação das tarefas. Com isso, os autores começam definindo uma aplicação *bag of tasks* como o conjunto de tarefas $\theta =$

$\theta_0, ..., \theta_{n-1}$ e o conjunto dos arquivos de entrada da tarefa θ_i como Δ_i , considerando Δ_i^j seu j^{th} arquivo de entrada e $|\Delta_i^j|$ o tamanho em bytes dessa entrada.

Duas tarefas são *relacionadas* quanto têm ao menos uma entrada em comum. Assim, existe uma tarefa θ_i e uma tarefa θ_k nas quais encontramos, ao menos, uma entrada de dados em comum ($\exists j, l / \Delta_i^j = \Delta_k^l$). Um conjunto de tarefas θ está no estado *conectado* se toda tarefa θ_i do conjunto está relacionada ao menos com outra tarefa θ_k, ou seja, $\forall i \exists k$, com θ_i e θ_k relacionadas. De acordo com essas definições, qualquer *bag of tasks* pode ser dividida em conjuntos de tarefas conectadas pela aplicação do algoritmo de fechamento transitivo (*transitive closure algorithm*).

Uma sequência $\sigma(\theta)$ de um conjunto de tarefas θ é uma ordenação de θ. Uma subsequência $\overline{\sigma}_s(\theta)$ é uma seção desta ordenação com tamanho definido por $|\overline{\sigma}_s(\theta)|$. A distância entre dois conjuntos de entrada de dados das tarefas, θ_i e θ_k é igual a soma dos dados de entrada da tarefa θ_k que não são compartilhados com a tarefa θ_i: $d(\Delta_i, \Delta_k = \sum_l |\Delta_k^l, \forall l / \Delta_k^l \notin \Delta_i|)$.

Com essa definições, Briquet et al propõem o algoritmo *Temporal Tasks Grouping* para o escalonamento das tarefas. A algoritmo trabalha com a sequência de tarefas definida anteriormente ($\sigma(\theta)$), sendo aplicado para todas as sequências de uma *bag of tasks*, que são, em seguida, ordenadas.

Portanto, duas tarefas são ditas *iguais* quando elas têm todos os seus dados em comum, ou seja, $d(\Delta_i, \Delta_k = 0)$ (dois arquivos FASTA com as mesmas sequências). Uma igualdade de subsequência de um conjunto de tarefas θ é uma extensão da igualdade de subsequência de tarefas, significando que as tarefas imediatamente antes e depois de uma subsequência podem não compartilhar essa igualdade. Na proposta do TTG, a tarefa básica é a ordenação dessas subsequências de uma sequência em ordem não crescente de tamanho da subsequencia $|\overline{\sigma}_s(\theta)|$, conforme a Figura 4.5.

A consequência desse agrupamento é a maximização das transferências executadas pelo protocolo BitTorrent. Ele também assegura que os maiores grupos de tarefas com igualdade de dados sejam escalonados primeiro, assim que vários nós estejam disponíveis. Então, várias subsequências de tamanho similar podem ser ordenadas usando um algoritmo de vizinhança mais próxima[17], usando como métrica a distância entre as tarefas ($d = (\Delta_i, \Delta_k)$). Após a seleção de tarefas , o agrupamento espacial das tarefas é considerado para a seleção dos recursos que executarão as tarefas, completando, assim, o procedimento de escalonamento do TTG. A principal rotina de execução da heurística é mostrada no algoritmo 3.

[17]Algoritmo simples que utiliza um procedimento guloso para resolver o problema do caixeiro viajante, encontrando uma solução subótima.

Dados de entrada (arquivos) de uma BoT

Δ_0	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	Δ_7
$\{g\}$	$\{i\}$	$\{g\}$	$\{r\}$	$\{g\}$	$\{g\}$	$\{d\}$	$\{r\}$

Dados das tarefas ordenados

Δ_0	Δ_2	Δ_4	Δ_5	Δ_3	Δ_7	Δ_1	Δ_6
$\{g\}$	$\{g\}$	$\{g\}$	$\{g\}$	$\{r\}$	$\{r\}$	$\{i\}$	$\{d\}$

Figura 4.5: O conjunto θ de tarefas de uma aplicação *bag of tasks* (considerando 1 arquivo por tarefa) são agrupados por igualdade de subsequências e ordenados em ordem não crescente de tamanho dessas subsequências ($|\overline{\sigma}_s(\theta)|$).

algoritmo 3 Agrupamento

1: {Passo 1: seleciona as sequencias de dados iguais que pertencem a tarefas diferentes e associa essas sequencias a um mesmo metadado(tarefas relacionadas)}

2: $array\_tarefas \leftarrow selecionatarefas$
3: **for** $i = 0$ to $i < tamanho\_array\_tarefas$ **do**
4: $conjunto\_de\_metadados\_das\_tarefas[i] \leftarrow obtem\_metadados\_da\_tarefa\_(i)$ {o conjunto de metadados refere-se as descrições de cada arquivo de entrada utilizado na execução da tarefa}
5: **conjunto de sequencias com tarefas relacionadas** $\leftarrow$ atualiza tarefas relacionadas(conjunto de metadados da_tarefa)
6: **end for**
7: ordena o conjunto de sequencias, em ordem não crescente, pelo tamanho de sequencias
8: em cadas item do conjunto de sequencias, ordena as tarefas em ordem não crescente de tamanho das entradas

4.6.2 Seleção de Recursos

O *agrupamento espacial* de tarefas está relacionado à atribuição explícita de tarefas realizada localmente e à atribuição implícita quando a alocação é efetuada em nós remotos do sistema nos quais temos o dado armazenado em *cache* de algum recurso do *host*.

Se no escalonamento, temos que Δ_{R_x} é o conteúdo de um dado em *cache* no recurso R_x (dado acumulado pela execução de tarefas anteriores), um recurso (R_x) é alocado em decorrência da minimização da distância, $d(\Delta_i, Delta_{R_x})$, entre o dado em $cache$ de cada recurso e o conjunto de dados de entrada Δ_i da tarefa a ser escalonada. Esta distância corresponde ao custo de transferência de escalonar a tarefa θ_i no recurso R_x. A distância mínima computada será pequena como resultado de

várias execuções das tarefas no recurso, propiciando assim o armazenamento da maioria dos dados de Δ_i utilizados pela tarefa. Essa minimização é equivalente à maximização do *storage affinity*.

4.7 Resumo do Capítulo

Apresentamos neste capítulo os principais conceitos de escalonamento de tarefas e realizamos um levantamento bibliográfico do estado da arte, no qual identificamos métodos, heurísticas e algoritmos que são utilizados em escalonamento de tarefas independentes em sistemas distribuídos. Dois desses métodos foram selecionados para o p2pBIOFOCO: replicação de tarefas (Cirne et al) e transferência eficiente de dados (Baohua et al). Por fim, discutimos dois outros métodos que também podem ser utilizados no p2pBIOFOCO: heurística Min-Max (Izakian et al) e agrupamento de tarefas (Briquet et al). No capítulo 6 detalharemos os métodos de replicação de tarefas e de transferência eficiente de dados que foram implementados neste trabalho.

Capítulo 5

p2pBIOFOCO

Neste capítulo apresentamos as principais características do p2pBIOFOCO. Na Seção 5.1, apresentamos a arquitetura projetada inicialmente para esse sistema. Em seguida, na Seção 5.2 descrevemos as modificações implementadas no sistema.

5.1 Arquitetura Original do p2pBIOFOCO

O sistema p2pBIOFOCO [88] foi desenvolvido utilizando técnicas baseadas em P2P para a execução distribuída de aplicações de Bioinformática. Tendo em vista que tais aplicações demandam, geralmente, uma grande quantidade de processamento e geram uma enorme quantidade de dados. O sistema provê a execução paralela das tarefas, com a finalidade de reduzir substancialmente o tempo de obtenção dos resultados. Na primeira versão, o sistema utilizou a plataforma JXTA para a formação de uma rede que integrava as instituições, UnB, UCB e Embrapa Recursos Genéticos e Biotecnologia, que utilizam os serviços de Bioinformática.

De forma geral, o funcionamento do sistema abrangia os seguintes passos:

1. leitura de um arquivo no formato FASTA contendo as sequências biológicas;
2. segmentação desse arquivo FASTA em arquivos menores, sendo os tamanhos dos arquivos dependentes do número de sequências armazenadas no arquivo;
3. distribuição dos arquivos às máquinas disponíveis no p2pBIOFOCO usando um algoritmo básico denominado Workqueue (WQ);
4. execução das aplicações de Bioinformática como serviços nas máquinas de destino; e
5. disponibilização dos resultados em um repositório, para consulta dos usuários do sistema.

Na Figura 5.1 observamos um exemplo de submissão das sequências do arquivo FASTA ao sistema p2pBIOFOCO.

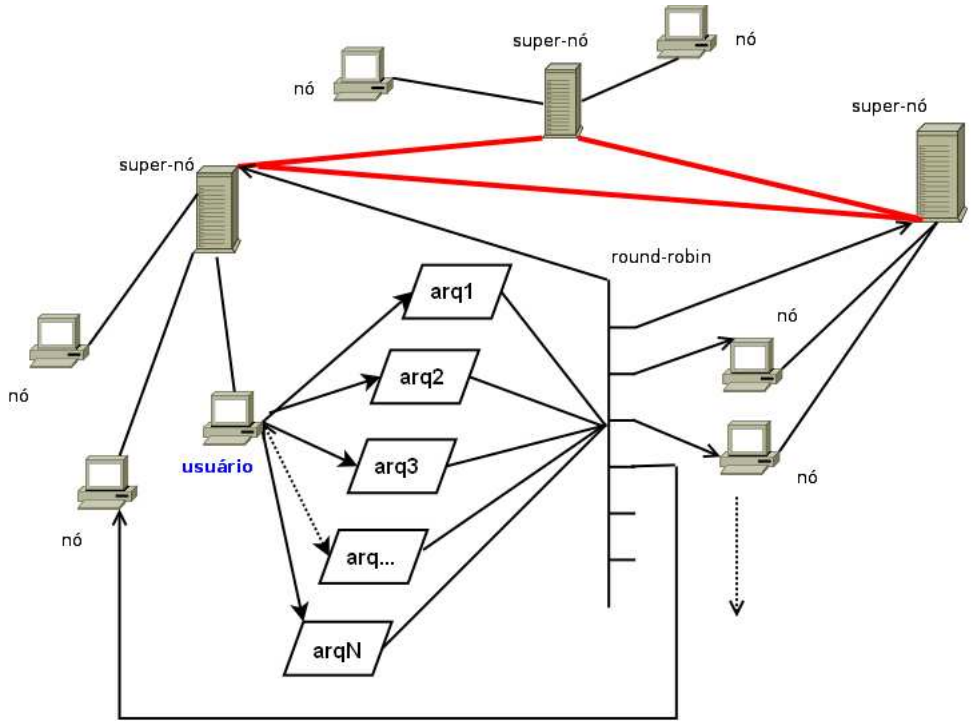

Figura 5.1: Submissão do Arquivo FASTA, segmentado em *n* arquivos, ao p2pBIOFOCO.

A primeira arquitetura do sistema p2pBIOFOCO utilizou a principal característica de modelos P2P que é caracterizar cada máquina pertencente à rede como um (*peer*) *cliente-servidor*. Isso significa que cada *peer* da rede oferece os mesmos serviços presentes nos outros *peers*. Essa abordagem de sistema distribuído resolve em grande parte duas situações clássicas dos sistemas centralizados, dependentes do paradigma cliente-servidor: escalabilidade e tolerância a falhas. No entanto, tais sistemas apresentam deficiências nos aspectos de segurança e de confiabilidade.

Nessa arquitetura do p2pBIOFOCO foi adotada uma topologia de sistemas P2P baseda em super-nós. Nesse esquema, alguns dos *peers* da rede (super-nós) são selecionados em decorrência de apresentarem maiores recursos (CPU, memória, armazenamento e largura de banda). Caso essses super-nós não estejam ativos, é possível, também, a atribuição do *status* de super-nó a qualquer *peer* do sistema. Vemos na Figura 5.2 uma representação desse modelo.

As principais funções dos super-nós são a criação de uma estrutura de rede virtualmente sobreposta à estrutura da rede física e a construção de um mecanismo de roteamento de mensagens com uma estrutura de dados básica denominada *tabela hash distribuída* (DHT). Essa estrutura oferece uma interface distribuída para o armazenamento e a recuperação dos dados e recursos pertencentes à rede P2P. Ela

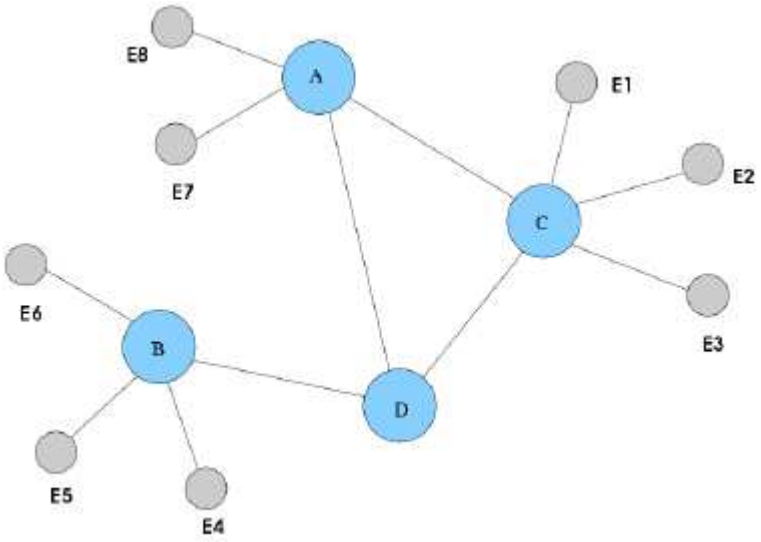

Figura 5.2: Topologia de Sistemas P2P com Super-Nós, sendo A, B, C e D os super-nós.

é utilizada como um substrato para a construção de sistemas P2P, controlando os processos de pesquisa e localização, aumentando a escalabilidade da rede. Assim, as mensagens podem ser roteadas entre os nós de uma rede P2P com um número mínimo de pulos (*hops*) entre os *peers*, de acordo com a implementação da DHT.

Para compor a rede sobreposta[18] do sistem p2pBIOFOCO foi utilizado o JXTA, um projeto da Sun Microsystems Inc. [80] cujo principal objetivo é a criação de um padrão que permita o desenvolvimento de sistemas distribuídos baseados na tecnologia P2P, possibilitando a integração de diferentes dispositivos eletrônicos (*desktops*, *mainframes*, celulares, etc.) com o uso de protocolos independentes de linguagem e de implementação, para a execução em diversos ambientes.

Com a evolução do projeto JXTA, a versão 2.0 propunha uma abordagem híbrida que combinava o uso de uma DHT fracamente consistente com um andarilho *rendezvous* de extensão limitada (*limited range rendezvous walker*). Cada *rendezvous* do sistema possui uma lista, ordenada por ID, de *rendezvous* conhecidos, chamada *rendezvous peer view* (RPV). Os nós *rendezvous* não são obrigados a manter um índice consistente entre todos os nós, originando o termo fracamente consistente. Se a entrada ou saída dos nós é baixa, então o RPV mantém-se estável e a DHT tende a atingir a consistência entre todos os nós, alcançando, consequentemente, um bom desempenho. Na Figura 5.3 temos a RPV representada como uma tabela e como um círculo DHT.

[18]Rede sobreposta é uma rede virtual criada sobre uma rede física, que possui as seguintes propriedades: garantia de recuperação de dados; horizonte de pesquisa representado tipicamente por $O(logN)$, onde N é o número de nós da rede; balanceamento automático; e, auto-organização [39].

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

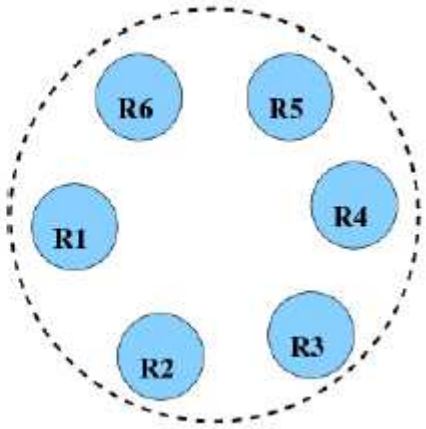

Figura 5.3: RPV (*Rendezvous Peer View*) como uma tabela e um círculo DHT.

O sistema p2pBIOFOCO foi projetado com a capacidade de definir, carregar, exportar e executar serviços usando a infraestrutura mencionada, executando aplicações de Bioinformática remotamente. No seu desenvolvimento, foi dada prioridade à elaboração de módulos fracamente acoplados, com o objetivo de facilitar a integração ou a remoção de novos serviços e de novos componentes. Na Figura 5.4 exibimos uma estrutura modular que representa a arquitetura original do p2pBIOFOCO.

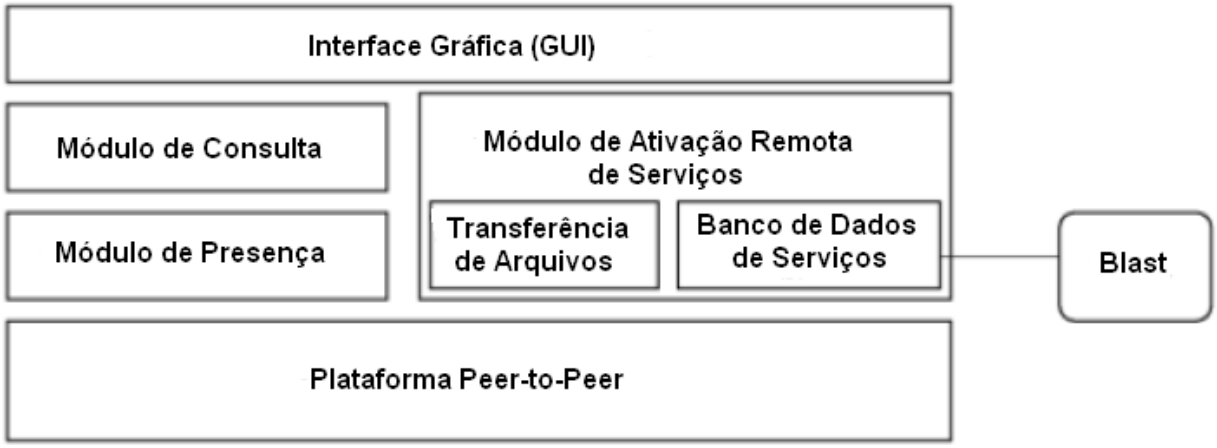

Figura 5.4: Estrutura Modular do Sistema p2pBIOFOCO

Os módulos do sistema possuem as seguintes funções:

- Plataforma *Peer-to-Peer*: implementa a interface entre a aplicação e a plataforma JXTA. É o principal módulo do sistema, pois permite a integração entre a plataforma e os módulos do sistema. Qualquer módulo desenvolvido para o sistema deve utilizar as chamadas às funções oferecidas pela plataforma. Dessa maneira, pode-se efetuar a execução dos serviços e das tarefas comuns aos sistemas P2P, tais como a publicação e a localização de recursos, a lo-

calização dos nós, o envio de mensagens e o compartilhamento de arquivos. O módulo ainda permite a abstração da camada de rede pública através da carga do arquivo de configuração, no qual estão definidas as propriedades de identificação dos nós para a plataforma.

- *Módulo de Consulta*: cria, envia, recebe e processa consultas aos serviços de Bioinformática na rede P2P. Cada serviço oferecido pelo sistema é exportado para a plataforma JXTA mediante a publicação de um *anúncio* conhecido como *BioService*, que deverá manter uma classe associada e integrada ao pacote do sistema. Particularmente, com relação às aplicações de Bioinformática, serão executadas chamadas ao sistema operacional do ambiente para execução do programa. Após a definição dos programas que executam os serviços, as informações básicas sobre os serviços são persistidas em um arquivo no formato XML [48]. Na Figura 5.5 temos um exemplo dos dados contidos no arquivo, que, no caso, referem-se à execução de dois serviços, BLAST e HelloWorld[19].

```
<?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 5.5: Informações locais presentes no arquivo (serão exportadas para a rede P2P).

Quando o p2pBIOFOCO é iniciado, o arquivo é lido, os serviços são carregados em memória e armazenados em um banco de dados local. O módulo de ativação remoto então envia um *anúncio* para a rede P2P informando aos outros nós sobre a disponibilidade do serviço. Na Figura 5.6, observamos um diagrama simplificado desse processo.

- *Módulo de Presença*: obtém informações sobre o estado dos nós na rede (ativos e inativos). É permitida no sistema apenas a apresentação do nó para a rede informando seu estado de atividade. A divulgação de presença faz uso de dois *pipes* (*propagate pipes*) e centraliza-se basicamente no papel do *rendezvous* para a divulgação dos anúncios de presença. Os anúncios de presença são enviados periodicamente a todos os nós conectados ao sistema. Desse modo, são

[19]Informa que o p2pBIOFOCO está em execução

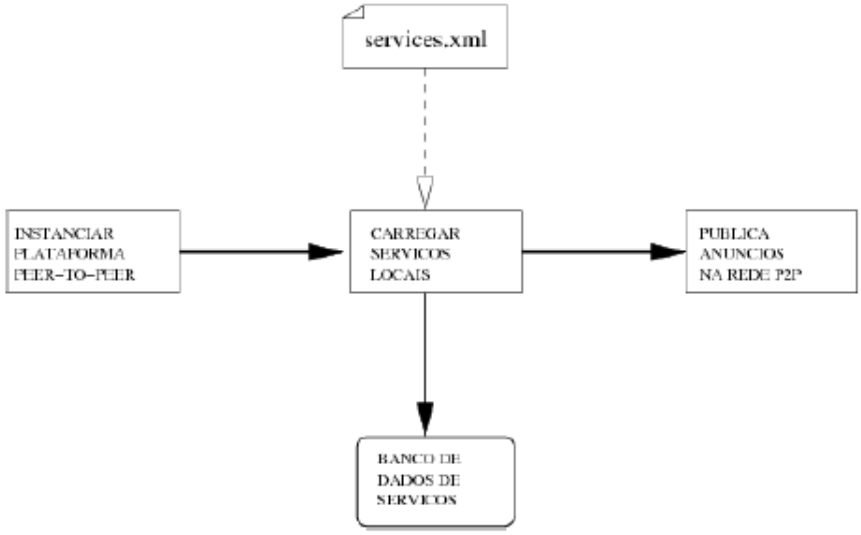

Figura 5.6: Processo de anúncio de um serviço na rede P2P utilizando a plataforma JXTA.

mantidas listas de presença em cada nó, atualizadas constantemente entre os super-nós com o objetivo de manter a consistência das listas em todo o sistema.

- *Módulo de Ativação Remota de Serviços*: é o módulo mais crítico do sistema p2pBIOFOCO, pois executa remotamente os serviços submetidos à rede P2P. Assim, aplicações de Bioinformática são executadas de forma distribuída. Na sua implementação, o módulo foi dividido em dois outros módulos.

  No primeiro módulo, *Banco de Dados de Serviços*, foram criados bancos de dados com as informações sobre os serviços locais, carregados no início da aplicação, e com informações recebidas pela rede acerca dos serviços remotos executados nos outros nós.

  O sistema executa o programa BLAST com a submissão de arquivos pela rede, executada pelo outro módulo, chamado *Transferência de Arquivos*. Esses arquivos contêm as sequências biológicas (consultas) que serão comparadas com os bancos de dados biológicos presentes em cada nó ativo do sistema.

  A informação enviada pelo módulo tem os seguintes argumentos:

  - o comando a ser executado;
  - os parâmetros de execução da aplicação de Bioinformática;
  - o endereço do repositório FTP (para o qual serão enviados os resultados);
  - conta de acesso do repositório FTP;
  - o nome da sequência para processamento.

Na Figura 5.7, visualizamos um desenho básico do processo.

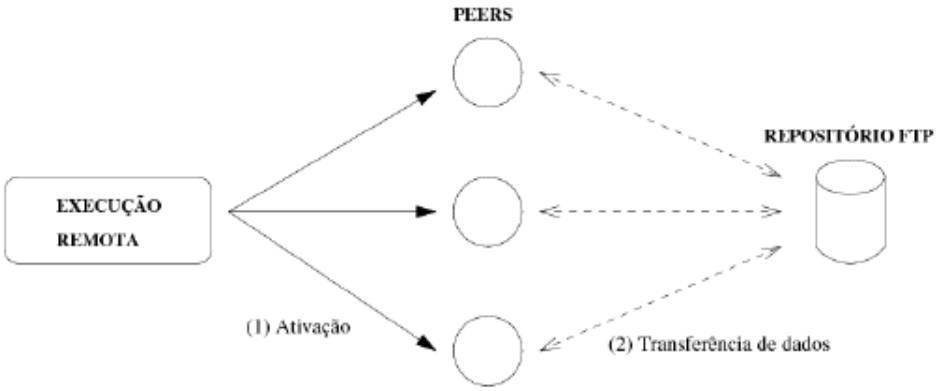

Figura 5.7: Ativação do Módulo de Execução Remota do Sistema p2pBIOFOCO.

- *Interface Gráfica*: fornece uma interface interativa para a pesquisa e execução de serviços remotos, concomitante à exibição dos nós ativos do sistema.

Limitações do p2pBIOFOCO

Após concluída a primeira versão do sistema p2pBIOFOCO, alguns problemas foram encontrados:

- *XML*: o uso de XML como base para os protocolos P2P está entre as desvantagens e as limitações da rede JXTA. Apesar do formato XML dar grande flexibilidade e portabilidade ao JXTA, o uso desse formato de dados cria uma sobrecarga de processamento, pois o desempenho da plataforma JXTA está estreitamente vinculado ao desempenho do interpretador XML utilizado. Além disso, existe uma sobrecarga na rede, pois uma mensagem transmitida entre dois nós geralmente é encapsulada em várias outras camadas de documentos XML. Isto faz com que as mensagens transmitidas na rede JXTA sejam grandes, ocasionando um grande tráfego de dados na rede de comunicação. A transmissão de uma série de mensagens pequenas pode inundar a rede, consumindo a largura de banda e os recursos do *rendezvous*. Isto dificulta a inclusão de novos colaboradores do projeto ou mesmo a utilização da plataforma por programadores não habituados ao paradigma P2P;
- *Descentralização*: apesar da implementação do sistema estar baseada na tecnologia P2P, a concentração de alguns serviços críticos do p2pBIOFOCO em alguns nós do sistema, os super-nós, não atende ao grau de tolerância a falhas provido pela tecnologia. No sistema atual, um certo grau de tolerância a fa-

lhas é mantido em decorrência da descentralização dos nós vinculados a cada super-nó. No entanto, existe a possibilidade de implantação de um arquitetura mais descentralizada com a implantação de uma DHT e a atribuição das mesmas responsabilidades (execução dos serviços, escalonamento, etc) para cada nó do sistema. Essa descentralização diminui os pontos de falhas do sistema e aumenta a resistência a ataques maliciosos do tipo DDoS (*Distributed Denial of Service*);

- *Escalonamento*: o sistema p2pBIOFOCO não possui algoritmos de escalonamento mais elaborados, dependendo de uma formulação básica de atribuição de tarefas usando o método Workqueue (WQ). Como mencionado em Ribeiro e co-atuores [88], a inclusão de algoritmos mais sofisticados pode melhorar o desempenho do sistema, considerando o tempo total de execução das aplicações de Bioinformática, sendo esta a justificativa básica da presente dissertação.

5.2 Arquitetura Modificada do p2pBIOFOCO

Ribeiro [89] propôs e implementou modificações para o p2pBIOFOCO, visando aprimorar o seu desempenho. Esta dissertação tem como foco o aumento do desempenho do sistema com a inclusão de novos métodos de escalonamento e de transferência de dados. Desse modo, dá continuidade e complementa alguns dos aspectos explorados na primeira arquitetura do p2pBIOFOCO.

5.2.1 Substituição do JXTA

Apesar de utilizado em vários projetos, a plataforma JXTA apresenta algumas restrições que a inviabilizam como uma plataforma confiável para a implantação e utilização em uma rede P2P com requisitos de desempenho, segurança e confiabilidade. Dentre as limitações da plataforma, destacam-se a sobrecarga de comunicação imposta pelo tráfego de mensagens XML, a complexidade das seis camadas de protocolos e a falta de confiabilidade na entrega das mensagens na rede. Por esta razão, decidimos implantar uma nova plataforma P2P baseada na DHT Kademlia [78]. Esta escolha foi em decorrência do Kamdemlia ser simples, com tempo $O(\lg n)$ para a pesquisa de todo o espaço da rede (conjunto de *n peers*) e da maior descentralização proporcionada ao sistema.

No Kademlia, os recursos são representados por um conjunto de chave/valor, seguindo a estrutura de uma DHT. Como exemplo, um arquivo pode ser classificado como um valor associado a uma chave *hash* de 160 bits. Para armazenamento

desse arquivo, o Kademlia utiliza a operação XOR para localização do nó no qual o arquivo será armazenado. A utilização da métrica XOR está baseada na execução dessa operação e no retorno de um número inteiro, sendo utilizado o de menor valor para atribuição do recurso ao nó.

O método é utilizado para resolver o problema de armazenamento de determinado recurso que não possui um ID igual ao do nó de destino ou possui somente uma parte do ID de 160 bits. Além disso, toda mensagem transmitida através do protocolo do Kademlia possui o ID do nó emissor, permitindo ao nó receptor a gravação ou atualização da existência do nó emissor em sua tabela de roteamento. Com esse método, diminui-se o tráfego de mensagens na rede, poupando recursos de rede. No caso do p2pBIOFOCO, o Kademlia foi usado principalmente para a localização dos nós, aproveitando a mesma operação de XOR que é empregada na localização dos recursos.

A Figura 5.8 mostra uma árvore binária representando os nós presentes no espaço de 160 bits, como tratado pelo algoritmo do Kademlia. O círculo negro indica a localização do nó 0011. As linhas ovais demarcam os espaços nos quais o nó 0011 deve ter um contato.

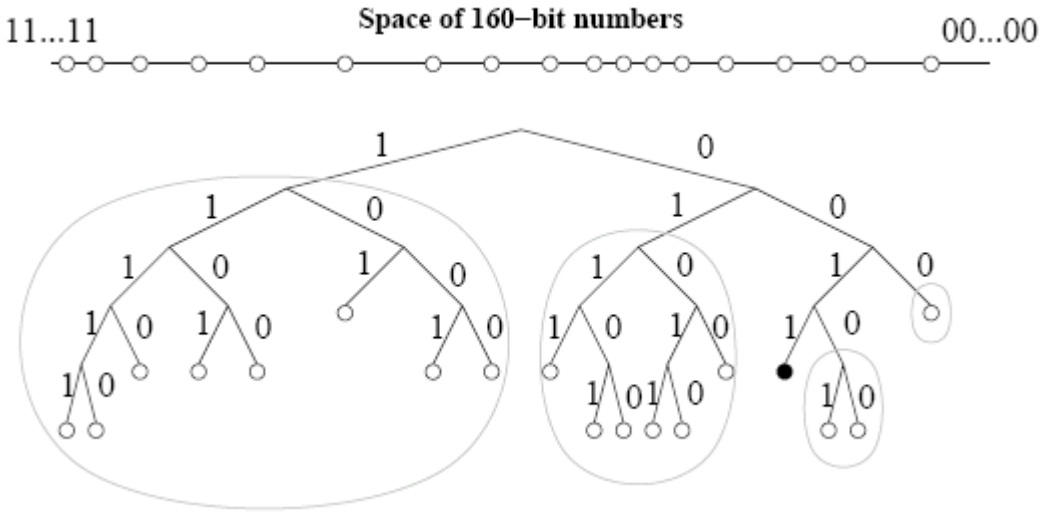

Figura 5.8: Árvore Binária Gerada pela DHT Kademlia.

O protocolo garante que cada nó do sistema conheça no mínimo um nó em cada uma das subárvores da DHT Kademlia, se essa subárvore contém pelo menos um nó. Com essa garantia, qualquer nó pode localizar qualquer outro nó pelo seu ID. Para localizar um nó por seu ID, podemos usar, por exemplo, o nó com prefixo *0011*, que localiza o nó com prefixo *1110* através de consultas interativas, ao melhor nó

que ele conhece, encontrando nós conhecidos em subárvores cada vez menores, e, assim, alcançando o nó alvo da pesquisa em menor tempo.

Na Figura 5.9 o segmento de linha acima da árvore representa o espaço de IDs e mostra como as pesquisas convergem para o nó alvo da pesquisa. Abaixo do segmento estão representadas mensagens RPC (*remote procedure call*) feitas pelo nó 0011. A primeira mensagem é enviada para o nó 101, conhecido pelo nó 1110. As mensagens subsequentes são para os nós retornados pelas mensagens executadas anteriormente. Esse comportamento garante que os nós conheçam os vários nós em sua vizinhança, e conheçam poucos nós à medida que a distância entre os IDs aumenta, conferindo a propriedade de limitação da quantidade de informação do roteamento em cada nó.

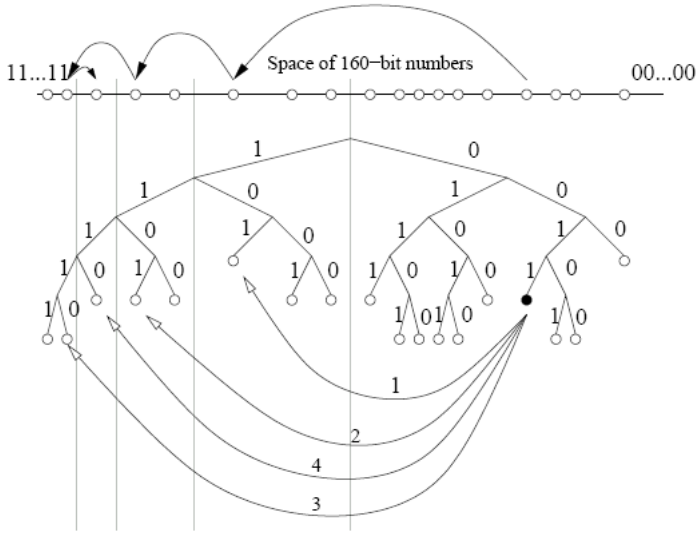

Figura 5.9: Pesquisa na Árvore Binária do Kademlia.

O protocolo Kademlia oferece diversas propriedades que estão presentes em outras DHTs. A diferença está no fato de encontrarmos a maioria dessas propriedades na implementação do Kademlia [9]. Entre essas propriedades destacam-se:

- *Simetria*: decorrente da simetria oferecida pela métrica XOR. A distância entre o *nó x* e o *nó y* é mesma nos dois sentidos, ou seja, $d(x, y) = d(y, x)$;
- *Descentralização*: é inteiramente distribuído, não atribuindo grau de maior importância para qualquer um dos nós;
- *Unidirecionalidade*: Para qualquer ponto x e uma distância arbitrária $s > 0$, existe exatamente um ponto y tal que $d(x, y) = s$. Portanto, independentemente do nó de origem, as pesquisas pelo mesmo nó convergirão para o mesmo caminho;

- *Pesquisas paralelas*: uma das mais vantajosas propriedades é a possibilidade de enviar a pesquisa da mesma chave para diversos nós. Desse modo, a ocorrência de *timeouts* em um caminho não atrasará necessariamente o processo de pesquisa, garantindo maior desempenho e confiabilidade em sistemas com alta incidência de *churn rates*[20];
- *Entradas dos buckets*[21]: flexibilidade dada ao usuário para escolher um nó arbitrário que está no *bucket*. Pode-se, também, criar qualquer critério para escolha dos nós, baseado, por exemplo, em confiabilidade e baixa latência;
- *Baixo tráfego de estabilização*: Ao contrário de outras DTHs, Kademlia praticamente não causa *overhead* na rede para atualização das conexões. A atualização está implícita nas pesquisas realizadas pelos nós;
- *Segurança*: devido à sua natureza descentralizada, DHT's baseadas no Kademlia são resistentes a ataques de serviço.

Novos Protocolos

Com a mudança do JXTA, adotamos os protocolos UDP, UDP_MCAST, TCP, o HTTP Puro e o HTTP/REST (Representational State Transfer) [46] para a troca de informações entre os nós. Com essa medida, procuramos construir um mecanismo de troca de informações mais flexível para o sistema p2pBIOFOCO, possibilitando a análise de qual protocolo é o mais interessante para determinada configuração do sistema. Assim, para requisitos definidos através de interfaces com a camada de comunicação, utilizamos o protocolo HTTP, enquanto que a utilização dos protocolos TCP e UDP nos permite um tratamento de mais baixo nível orientado a conexões ou não [31].

Na Figura 5.10 temos um diagrama UML no qual destacam-se os protocolos usados no p2pBIOFOCO, com a possibilidade, ainda, de utilização do protocolo UDP. Os nós podem oferecer acesso aos três tipos de comunicação, situação representada pela classe *EndPoint* e suas herdeiras *TcpEndPoint*, *HttPEndPoint* e *RestEndPoint* presentes no diagrama. A definição é efetuada em um arquivo de configuração presente em cada nó, sendo atribuída ao nó quando iniciamos o processo de execução e de anúncio do nó na rede *peer-to-peer*.

O diagrama da Figura 5.11, mostra a interação entre os módulos de comunicação do p2pBIOFOCO, evidenciando a sequência de envio e de recebimento de

[20]Taxa de saída de nós de um sistema em um determinado período de tempo

[21]*k-bucket* é uma lista com os k endereços (contatos) dos possíveis nós vizinhos, gravada em cada nó do sistema.

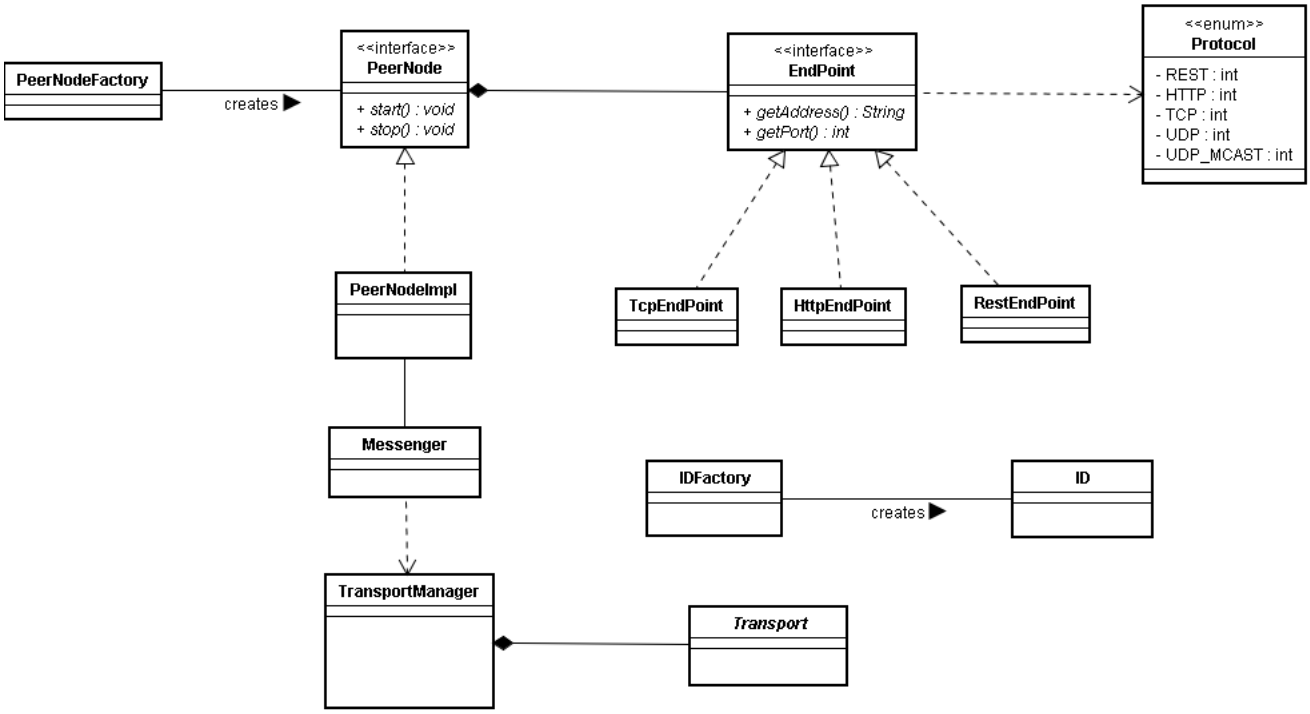

Figura 5.10: Diagrama UML de Mensagens e Protocolos da nova arquitetura do p2pBIOFOCO

mensagens através desses módulos. O principal módulo do diagrama é a classe *Messenger*, responsável pela obtenção do tipo de protocolo que será usado na transferência dos dados (TCP, HTTP puro ou HTTP rest) e responsável pelo controle do tráfego das mensagens.

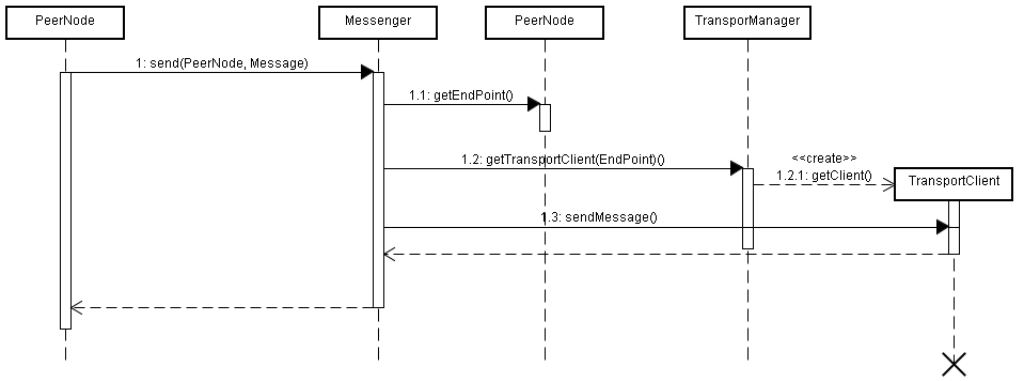

Figura 5.11: Diagrama de criação dos pontos de conexão da rede (*endpoints*).

5.2.2 Descentralização

Como mencionamos no capítulo anterior, uma plataforma mais aberta, tolerante a falhas e escalável foi implementada na arquitetura nova do p2pBIOFOCO. Na Figura 5.12, demonstramos como ficou a configuração dos nós na nova configuração.

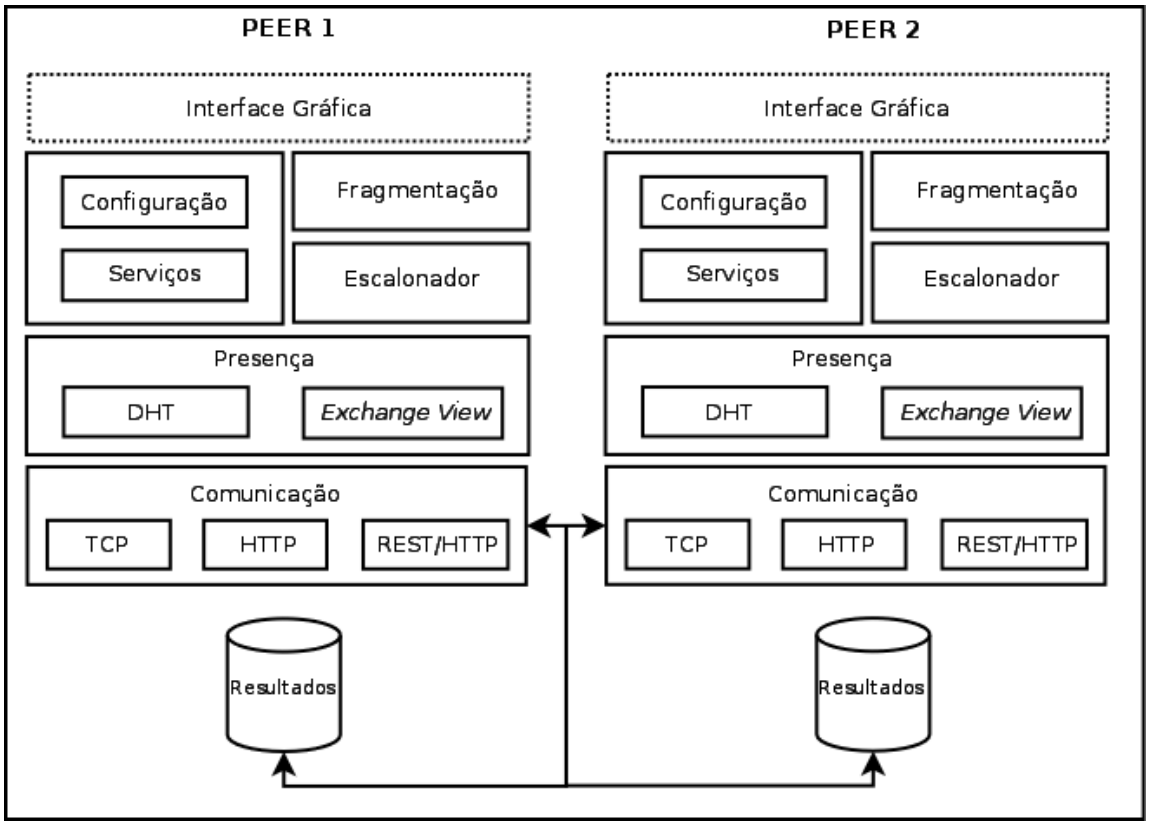

Figura 5.12: Arquitetura Modificada do p2pBIOFOCO

Podemos observar, então, que o papel dos super-nós, presente na arquitetura anterior, foi excluído. Cada nó participante do sistema tem, portanto, as atribuições de cliente-servidor. Dessa maneira, cumprem-se dois dos principais requisitos dos sitemas P2P: a tolerância a falhas e a escalabilidade. No sistema anterior, apesar de caracterizado como um sistema P2P, ainda tínhamos um certo grau de pontos de falha com a implementação dos super-nós. Independentemente de atribuirmos o papel de super-nó a qualquer nó do sistema, quando houver a saída de um deles, a forma atual, com maior grau de descentralização, evita a execução de algoritmos para a seleção de um novo super-nó. O único custo para o sistema é a necessidade de atualizações de suas tabelas de roteamento, que também eram efetuadas no modelo anterior.

5.2.3 Novos Módulos

- *Interface Gráfica*: responsável por permitir ao usuário a submissão dos arquivos FASTA ao p2pBIOFOCO. Pode ser acessada via WEB informando no *browser* o endereço de qualquer nó do sistema ou invocada utilizando uma interface escrita em Swing Java, habilitada através de um parâmetro na execução do processo de inicialização do nó;
- *Configuração*: é composto de atributos de configuração do nó e de arquivos no formato JSON (*JavaScript Object Notation*) [48] ou XML, armazenados no sistema de arquivos da máquina. Contém valores constantes sobre alguns atributos dos nós (identificador e porta, por exemplo). Valores que sofreram alterações durante a execução do nó, são persistidos na parada normal da aplicação;
- *Fragmentação*: conjunto de classes que manipulam os arquivos FASTA para segmentação e formatação das sequências, preparando-as para envio e execução nos nós de destino;
- *Comunicação*: possui uma suíte de protolocos - TCP, UDP, HTTP e BitTorrent - e um conjunto de serviços de comunicação para permitir o envio e o recebimento das mensagens no p2pBIOFOCO. Implementa, também, serviços de gerenciamento e controle dessas mensagens;
- *Serviços*: suas classes carregam os serviços solicitados nas tarefas submetidas ao nosso P2P. Com a utilização das funções presentes neste componente, serviços são criados, excluídos e alterados. O *status* de cada tarefa pertencente a determinado serviço, é monitorado para informação ao sistema de sua conclusão normal ou não;
- *DHT*: a estrutura de tabela *hash* distribuída no nosso sistema é empregada para a localização das máquinas participantes do P2P. Neste componente, com o uso da tabela, instituímos o serviço de localização dos nós;
- *Presença*: o processo de identificação dos pontos de comunicação do sistema (*endpoints*), ao qual chamamos de troca de visões (*exchange view*);
- *Escalonador*: ordenação dos trabalhos, com definição de suas prioridades, alocação das tarefas às máquinas baseado ou não em informações oriundas dos nós do sistema, conforme a heurística ou algoritmo implementado.

Para o módulo de presença, temos os seguintes passos, detalhados no diagrama de sequência Figura 5.13:

1. o nó chama o processo de identificação dos nós na rede (armazenados na tabela de roteamento);
2. após a identificação, são criados os observadores (*listeners*) para os pontos ativos da rede;
3. o módulo de transporte, responsável pelo envio e recebimento de mensagens, inicia o processo de observação do estado de cada nó ativo da tabela de roteamento, através da troca de mensagens;

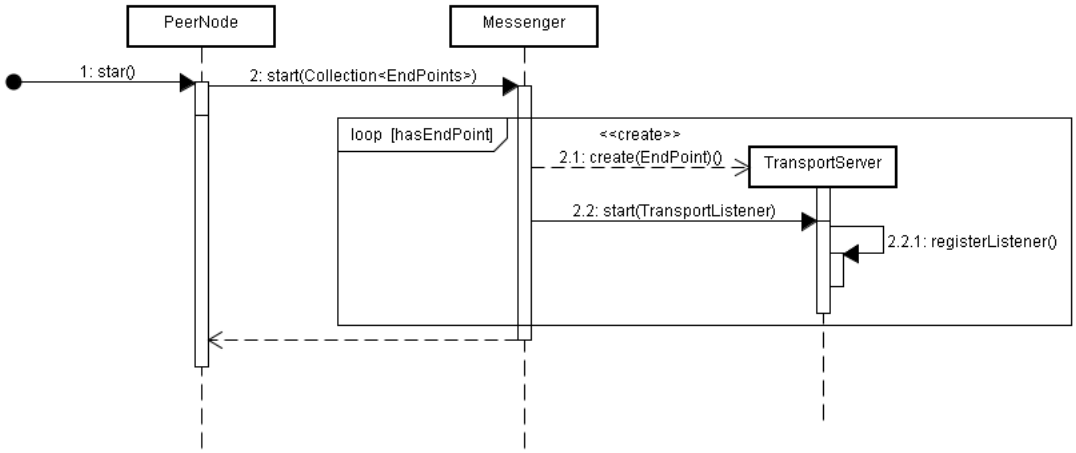

Figura 5.13: Diagrama: Criação dos Pontos de Comunicação no p2pBIOFOCO.

Cada nó, após a sua instanciação na rede, tenta atualizar a sua tabela de roteamento emitindo um comando *ping* para os nós cadastrados na sua tabela. Após esse primeiro contato, a tabela é atualizada com os nós ativos, expondo para o nó que iniciou a pesquisa os pontos de acesso na rede (IP's e portas). O objetivo desse método - a atualização de todas as tabelas dos nós participantes do sistema - é alcançado com a complementação da troca das visões de todos os nós. Cada nó envia para os nós conhecidos a sua tabela de nós ativos. Dessa forma, conseguimos uma estabilização do sistema com cada nó conhecendo todos os nós participantes do sistema. Na Figura 5.14, temos uma visualização desse processo de troca de informações entre os nós.

Na Figura 5.15, vemos a integração entre os módulos descritos.

5.2.4 Detalhes de Implementação

Nesta seção vamos detalhar algumas funções implementadas no p2pBIOFOCO, seguindo a ordem dos módulos apresentados na Seção 5.2.3. Assim, começamos

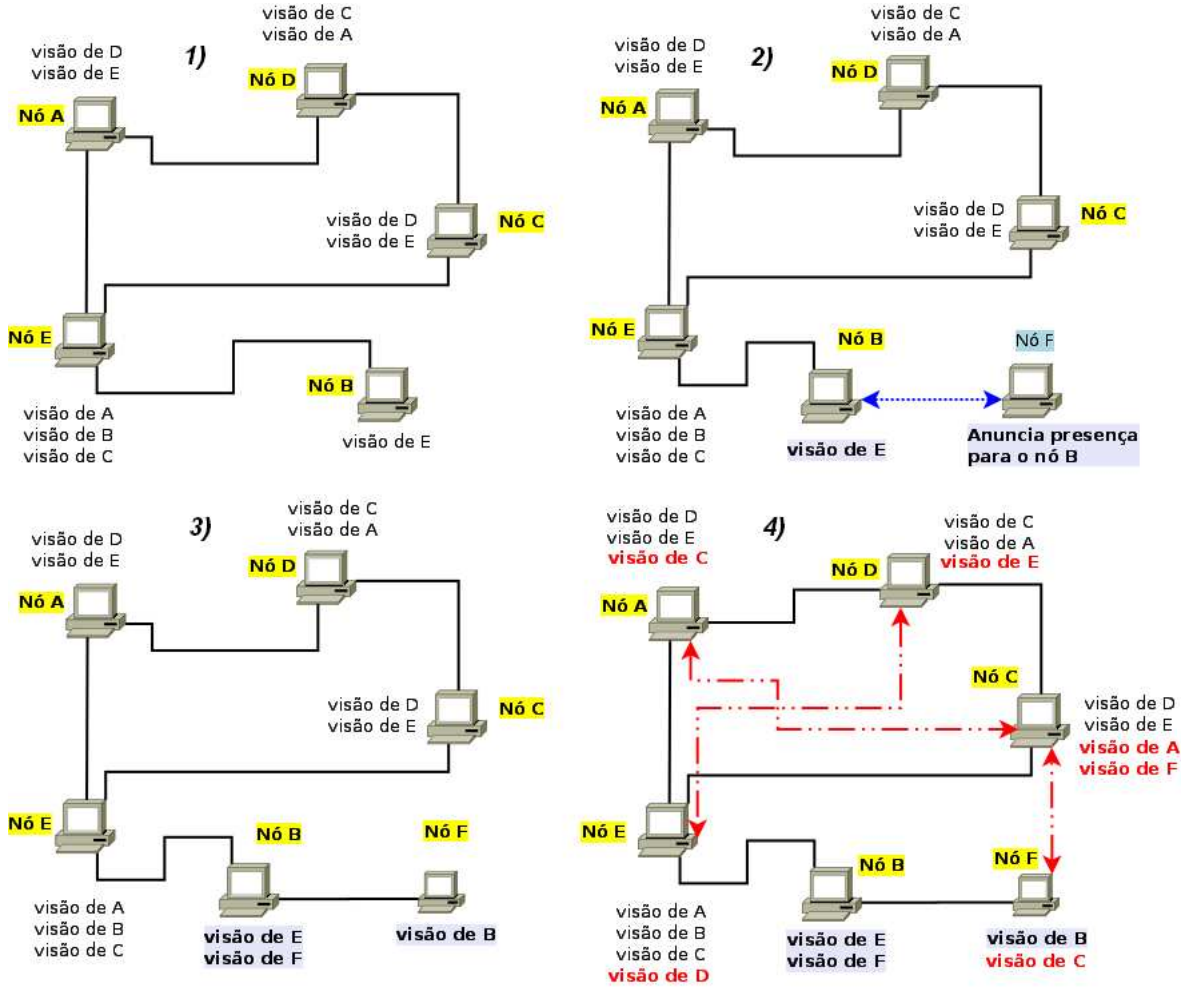

Figura 5.14: Troca da Tabela de Roteamento entre os Nós: em 5.14-1 é exibido o estado no qual os nós tem as visões de seus nós vizinhos; em 5.14-2 o nó F anuncia sua presença para o nó B; o nó F obtém a visão do nó B (5.14-3); os nós atualizam suas visões efetuando a troca de informações conforme 5.14-4

o detalhamento a partir da interface gráfica até chegarmos à tabela distribuída utilizada para a localização dos nós. As particularidades do escalonador serão discutidas no capítulo 6.

Configuração

As duas principais classes Java para a configuração do p2pBIOFOCO são: AppConfig e FSXmlPeerConfigurator. Nestas classes, são carregados os principais atributos para a execução do p2pBIOFOCO. Elas utilizam quatro arquivos XML (*app.xml*, *seeds.xml*, *peer.xml* e *services.xml*), com dados sobre a aplicação, sobre os nós participantes do P2P, sobre o nó e sobre os serviços executados em cada nó.

Na Figura 5.16, vemos os dados do arquivo *app.xml*, no qual temos o comando a ser executado e o caminho relativo no sistema de arquivos onde estão armazenados

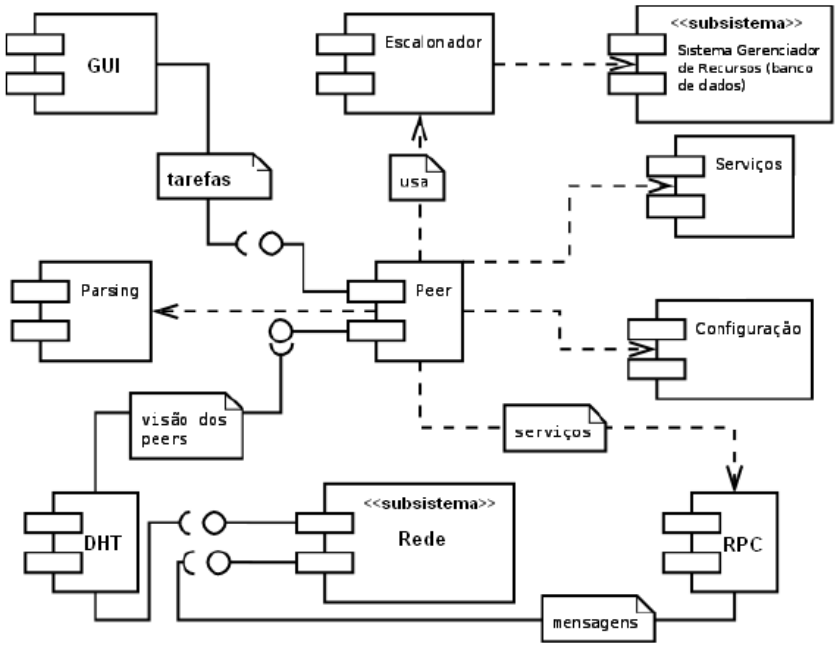

Figura 5.15: Módulos do Sistema p2pBIOFOCO

os dados de entrada e saída das tarefas. Neste caso, temos o parâmetro *spool-directory*, que se refere ao local onde estão localizados os dados de entrada (arquivos com as sequências) e o parâmetro *output-directory* que se refere ao local onde os dados de saída da execução do comando. O comando de execução da tarefa está representado pelo parâmetro *path*, que representa um arquivo *shell script* (.sh), no qual temos a chamada para o serviço (como, por exemplo, a execução do comando *blastall*). O arquivo é executado por meio da chamada do método *exec()* da classe *java.lang.Runtime*, pertencente à API oficial da plataforma Java. Como exemplo dessa chamada, temos a seguinte linha de código:
Process p = Runtime.getRuntime.exec("<comando>")

```
<app>
    <spool-directory>/tmp</spool-directory>
    <output-directory>output</output-directory>
    <path>/home/biofoco/blast.sh</path>
</app>
```

Figura 5.16: Exemplo de arquivo *app.xml* com alguns dados da aplicação que instancia o nó.

O arquivo *seeds.xml* contém a *visão* que cada nó do p2pBIOFOCO tem de seus pares. Para a carga inicial do sistema, os arquivos já estão com alguns dos nós do sistema informados. Com a descoberta dos nós, através da utilização do algoritmo

do Kademlia, o arquivo é atualizado. Na Figura 5.17, temos um exemplo dos dados contidos no arquivo.

```
<seeds>
    <seed>
        <endpoints>
            <endpoint>tcp://164.41.17.165:7171</endpoint>
            <endpoint>tcp://164.41.17.165:7272</endpoint>
            <endpoint>tcp://164.41.17.169:7171</endpoint>
            <endpoint>tcp://164.41.17.191:7272</endpoint>
        </endpoints>
    </seed>
</seeds>
```

Figura 5.17: Exemplo de arquivo *seeds.xml* com os nós cadastrados.

Os nós do sistema têm seus dados configurados no arquivo *peer.xml*. Ao instanciar um nó, o identificador (ID) e o nome do nó são carregados e constituem uma identidade única do nó para o p2pBIOFOCO. Caso o sistema não consiga obter este atributo, é gerado um identificador aleatório de mesmo tamanho e formato para o nó, sendo este considerado o seu novo ID. Um dado importante acrescentado a este arquivo é o atributo *protocolo*, pelo qual informamos o tipo de protocolo que o p2pBIOFOCO utilizará para o envio e recebimento de mensagens. Com essa mudança esperamos tornar o p2pBIOFOCO mais flexível para o tratamento de mensagens que trafegam pela rede, além de possibilitar o uso de tecnologias e serviços WEB (procotolos REST e HTTP). Com o protocolo UDP_MCAST (*UDP com multicast*) pretendemos agilizar o envio de mensagens que se referem a estrutura do p2pBIOFOCO e à sua manutenção como uma rede P2P. Como exemplo, poderíamos citar o envio de várias mensagens para a verificação da existência dos nós constantes do arquivo *seeds.xml*. A Figura 5.18 mostra um exemplo do arquivo.

Por fim, citamos o arquivo *service.xml*, onde estão cadastrados os serviços que o nó pode executar. Ao receber uma mensagem remota de solicitação de execução do serviço, o nó carrega uma classe Java referente ao serviço que está sendo solicitado. Assim, para a execução de serviços no p2pBIOFOCO, temos duas opções: execução de comandos referentes às aplicações instaladas no computador ou a execução de aplicações implementadas na linguagem Java. Para as aplicações em Java, dentre os parâmetros podemos destacar o valor constante referente ao comando Java e os parâmetros de configuração da máquina virtual Java e de execução do programa. Exemplo: ”java -Dconfig.dir=conf BlastService -gui”, que significa a chamada do serviço do Blast (*BlastService*), utilizando uma interface gráfica (*gui*) e tendo o caminho do arquivo de configuração igual a *conf*. Cada serviço tem o seu código de identificação para o p2pBIOFOCO. Assim, na Figura 5.19, mostramos que o serviço

```
<peerConfig>
    <peerID>3f744c7b725feb5aa12517bee270442f35734140</peerID>
    <peerName>peer1</peerName>
    <endpoints>

        <endpoint>
            <protocol>REST</protocol>
            <port>7171</port>
        </endpoint> -->

        <endpoint>
            <protocol>HTTP</protocol>
            <port>7272</port>
        </endpoint>

        <endpoint>
            <address>172.18.17.165</address>
            <protocol>TCP</protocol>
            <port>7171</port>
        </endpoint>

        <endpoint>
            <protocol>UDP_MCAST</protocol>
            <address>235.255.0.1</address>
            <port>5000</port>
        </endpoint>

    </endpoints>

</peerConfig>
```

Figura 5.18: Exemplo de arquivo *peer.xml* com a configuração de um nó.

Blast é identificado pelo código ”192”, enquanto o serviço Fatorial é identificado pelo código ”193”.

```
    <services>
        <service id="192" name="BLAST">
            br.biofoco.p2p.services.BlastService
        </service>
        <service id="193" name="FATORIAL">
            br.biofoco.p2p.services.FatorialService
        </service>
    </services>
```

Figura 5.19: Exemplo de arquivo *services.xml* com dois serviços cadastrados.

Fragmentação dos Arquivos

O nó que submeterá os arquivos para processamento remoto executa o serviço de segmentação do arquivo FASTA de acordo com a quantidade de sequências do arquivo e envia esses arquivos para os nós remotos de acordo com o algoritmo de escalonamento ativado (Workqueue ou WQR, predefinido nos parâmetros do sistema antes de iniciá-lo). Portanto, a unidade básica da fragmentação do arquivo é o *contig*. O tamanho e a granularidade dos arquivos que serão submetidos remotamente ao p2pBIOFOCO são determinados pelo tamanho da sequência e pela quantidade de sequências existentes no arquivo FASTA.

A fragmentação do FASTA é realizada pela classe *FastaParser*, que tem como atributo principal a lista de arquivos que será produzida no processo de segmentação. Seu método mais importante é o método *parse*, no qual o caracter *">"* é utilizado como base para o particionamento do arquivo e criação dos arquivos que representam as tarefas que serão executadas remotamente. Os parâmetros de entrada do método são o arquivo FASTA que será submetido à fragmentação e o diretório no qual serão gravados os arquivos com as sequências. Na Figura 5.20, mostramos um diagrama simples deste processo. Cada sequência é representada como um arquivo na figura. No entanto, podemos parametrizar a aplicação para o fragmentação do arquivo FASTA em arquivos que possuam mais de uma sequência. Esse procedimento dependente do algoritmo de escalonamento que será utilizado.

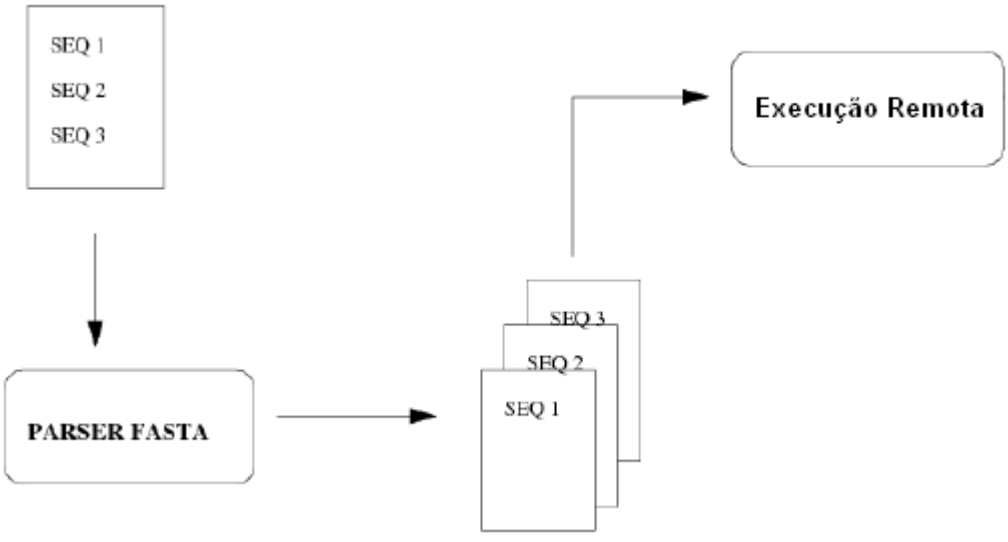

Figura 5.20: Particionamento do Arquivo FASTA para Envio à Execução Remota.

Comunicação - Protocolo JSON

JSON (Java Serialization Object Notation) [33] é um formato de arquivo utilizado inicialmente para persistir objetos JavaScript em sistemas de arquivos, bancos de dados, ou para transmissão via rede.

Entretanto, devido à sua expressividade, legibilidade, independência de linguagem de programação, e tamanho relativamente reduzido, JSON tem sido cada vez mais utilizado como uma alternativa ao XML para a transferência e armazenamento de dados estruturados e auto-descritivos em diferentes plataformas e linguagens de programação. Sítios comerciais como Google, Yahoo! e Amazon, dentre muitos outros, utilizam JSON em seus protocolos de Serviços Web devido à sua facilidade de implantação e adoção. O formato JSON encontra-se descrito na RFC 4627 [61]. Na Figura 5.21, temos uma representação dos dados em JSON.

```
{
        "nome": "Anna",
        "sobrenome": "Carolina",
        "idade": 25,
        "endereço": {
                "cidade": Brasília,
                "uf": "DF"
        }
}
```

Figura 5.21: Exemplo do Formato de Dados em JSON.

Na Seção 5.2.1, discutimos a utilização dos protocolos TCP, UDP, HTTP, REST e UDP_MCAST. A implementação destes protocolos, como foi dito, permite ao p2pBIOFOCO flexibilização na comunicação entre os nós, característica que não estava presente na primeira versão. Nesta subseção apresentamos detalhes sobre o protocolo utilizado na construção das mensagens do p2pBIOFOCO.

Na versão anterior, devido à utilização do JXTA, as mensagens tinham como base o padrão XML. Fonseca e Simões [48] relacionam algumas desvantagens do XML para a representação e a transferência de dados:

- a sintaxe é redundante e possui grande quantidade de informações com impacto à leitura humana, à eficiência das aplicações e aos custos de armazenamento, que tornam-se mais elevados;
- a construção de *parsers* não é trivial, exigindo que bons *parsers* processem dados aninhados arbitrariamente, com testes adicionais para detecção de dados

mal formatados ou erros de sintaxe, com carga adiconal significativa para os usos mais básicos do XML;

- os requisitos básicos de processamento não admitem um grande conjunto de tipos de dados. Por exemplo, não existe mecanismo em XML para que o número 3,14159 seja entendimento como um número de ponto flutuante, exigindo para isso linguagens adicionais (ex: XML *Schema* [107]);

Para o transporte de dados no p2pBIOFOCO, adotamos o JSON como uma alternativa ao XML. Simões e Fonseca [48] demonstraram que, para mensagens de até um megabyte, o desempenho do JSON é melhor em relação ao XML devido à baixa complexidade para a especificação dos dados, possibilitando um processamento mais rápido nos *parsers*. Com mensagens maiores, no entanto, devido à perda de escalabilidade do JSON (menos flexibilidade para descrição dos dados), o XML tornar-se a opção mais indicada para o transporte de dados. A maioria das mensagens trocadas entre os nós do p2pBIOFOCO tem tamanho pequeno, sendo a transferência dos dados dos arquivos executada via FTP ou Torrent. Assim, o padrão adotado foi o JSON, com possibilidade de utilização do XML, caso haja necessidade de transferência de mensagens maiores e mais complexas. Na Figura 5.22, mostramos um exemplo simples de troca de mensagens utilizando JSON.

```
--> {“comando”: “eco”, “params”: [“JSON-RPC”], “identificação”: 1}
<-- {“resultado”: “JSON-RPC”, “erro”: zero, “identificação”: 1}
```

Figura 5.22: Exemplo de Troca de Mensagens Utilizando o Padrão JSON.

Interface Gráfica

A interface gráfica, escrita em Java, ou disponibilizada como uma página WEB, permite ao usuário a submissão de suas pesquisas. Ao término da execução, um relatório com as informações é gerado e apresentado ao usuário. As funcionalidade básicas em relação à primeira versão do p2pBIOFOCO foram mantidas, pois as maiores alterações deste trabalho estão centralizadas em funções que são transparentes ao usuário. Na Figura 5.23 vemos a interface utilizada pelo usuário para submissão do arquivo FASTA ao p2pBIOFOCO.

Serviços

O principal objeto do componente Serviços é a interface *Service*. Nela estão definidas as principais funções pertinentes à manutenção dos serviços no p2pBIOFOCO.

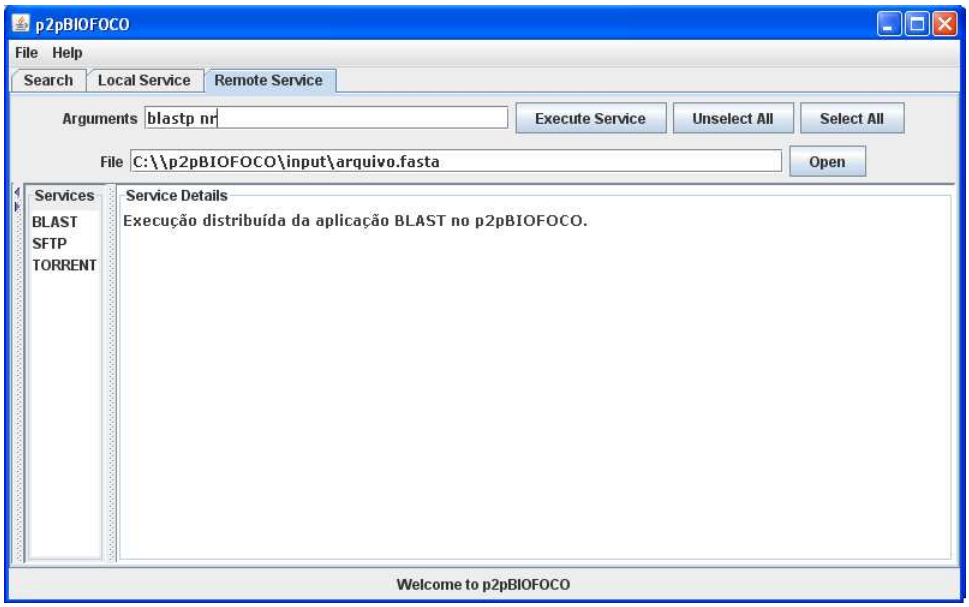

Figura 5.23: Interface Gráfica com o serviço BLAST para execução no p2pBIFOCO.

Seu principal método, *execute*, refere-se à execução dos serviços solicitados remotamente por outros nós. Uma classe abstrata, *AbstractService*, implementa a interface mencionada, tornando obrigatória a extensão desta classe para qualquer novo serviço incluído no p2pBIOFOCO. Este método é pertinente aos casos em que os serviços são novas classes Java pertencentes ao pacote de serviços do p2pBIOFOCO.

Por exemplo, se quisermos implementar um serviço que execute uma outra ferramenta de Bioinformática para comparação de sequências, criamos a classe *ServiceOutra* extendendo a classe *AbstractService* e implementamos no método *execute* o código de execução do serviço. Em seguida, cadastramos o serviço no arquivo *services.xml*. Após a atualização do código do p2pBIOFOCO com o novo serviço e sua distribuição para as máquinas, a próxima execução do sistema contemplará a nova ferramenta como uma opção para o usuário do P2P. Um objeto importante para este componente é a classe *ServiceResult*. Nesta classe, estão encapsulados o identificador da tarefa (ID) e o estado de execução dessa tarefa.

DHT (*Distributed Hash Table*)

As principais classes para a implementação do método de pesquisa dos módulos estão no pacote *dht* do p2pBIOFOCO. Na nossa implementação, temos as seguintes classes: *AbstractDHT*, *MessageBuilder*, *RandomWalkDHT* e *SeedProvider*.

Na classe *AbstractDHT*, temos a definição dos principais comandos utilizados no protocolo do Kademlia (PING, STORE, FIND_NODE e FIND_VALUE). Nesta

versão do p2pBIOFOCO, habilitamos a execução do comando PING e implementamos o comando EXCHANGE_VIEW para a troca de visões dos nós. Com a execução do comando PING, descobrimos os nós que estão ativos no P2P e armazenamos seus identificadores e valores em uma tabela. Periodicamente, o comando EXCHANGE_VIEW é enviado pelos nós para recebimento da tabela de cada nó com a informação dos nós conhecidos por ele.

5.3 Resumo do Capítulo

Neste capítulo apresentamos a nova arquitetura do p2pBIOFOCO e detalhamos seus módulos principais. Com a implementação de novos protocolos de comunicação, o sistema tornou-se mais flexível, possibilitando a utilização de outras plataformas (WEB, REST). Com o JSON, as mensagens que trafegam pelo sistema são menores, com ganhos no desempenho e menor utilização da rede. A implementação da DHT Kamdemlia tornou o mecanismo de localização dos nós mais eficiente e mais rápido. No próximo capítulo, destacaremos as duas principais alterações incluídas na nova arquitetura, com a inclusão de um novo método de escalonamento e de um método para a transferência eficiente de dados.

Capítulo 6

Métodos de Escalonamento e de Transferência de Dados

Como já mencionado, uma das maiores necessidades do sistema atual é a implementação de algoritmos de escalonamento distribuído que permitam a alocação de tarefas para os nós localizados e com seus identificadores armazenados na DHT. Na Seção 6.1, mostramos como o módulo de escalonamento foi incluído no p2pBIOFOCO. Na Seção 6.2, descrevemos detalhadamente o método de escalonamento WQR, mencionado na Seção 4.4. Em seguida, detalhamos as características mais importantes de sua implementação. Da mesma forma, na Seção 6.3, descrevemos o método DP-RR e os seus detalhes de implementação no p2pBIOFOCO.

6.1 Incluindo Escalonamento no p2pBIOFOCO

No Capítulo 4, vimos que o escalonamento refere-se ao processo de alocação de recursos por um determinado período de tempo. No nosso caso, as máquinas integrantes do sistema p2pBIOFOCO são os recursos, enquanto as requisições para esses recursos são as sequências biológicas (dados de entrada das tarefas) submetidas para cada nó do sistema. Um trabalho J_k refere-se a um arquivo FASTA com determinado número de sequências s_i pertencentes ao conjunto S de sequências ($S = s_1, s_2, ..., s_i$). O conjunto de máquinas $M = m_1, m_2, ..., m_j$ refere-se às máquinas ativas que compõem o p2pBIOFOCO em determinado momento. O objetivo da implementação do escalonador selecionado é a minimização do *makespan* das tarefas, obtendo como resultado final a redução do tempo de execução de um trabalho no p2pBIOFOCO.

No p2pBIOFOCO, o módulo de escalonamento recebe uma lista de nós e atribui uma tarefa, obtida de uma FIFO, a cada um deles. Após a execução do algoritmo de

escalonamento, cada nó tem uma tarefa selecionada de acordo com o critério de otimização do algoritmo. Depois da alocação das tarefas, a lista de nós é enviada para o módulo de comunicação. Este módulo tem como função a construção das mensagens, com seus respectivos parâmetros, para envio aos nós remotos. As tarefas ficam pendentes na FIFO, aguardando a disponibilidade de um nó para executá-la. Na Figura 6.1, temos uma representação geral do processo de escalonamento efetuado no sistema p2pBIOFOCO.

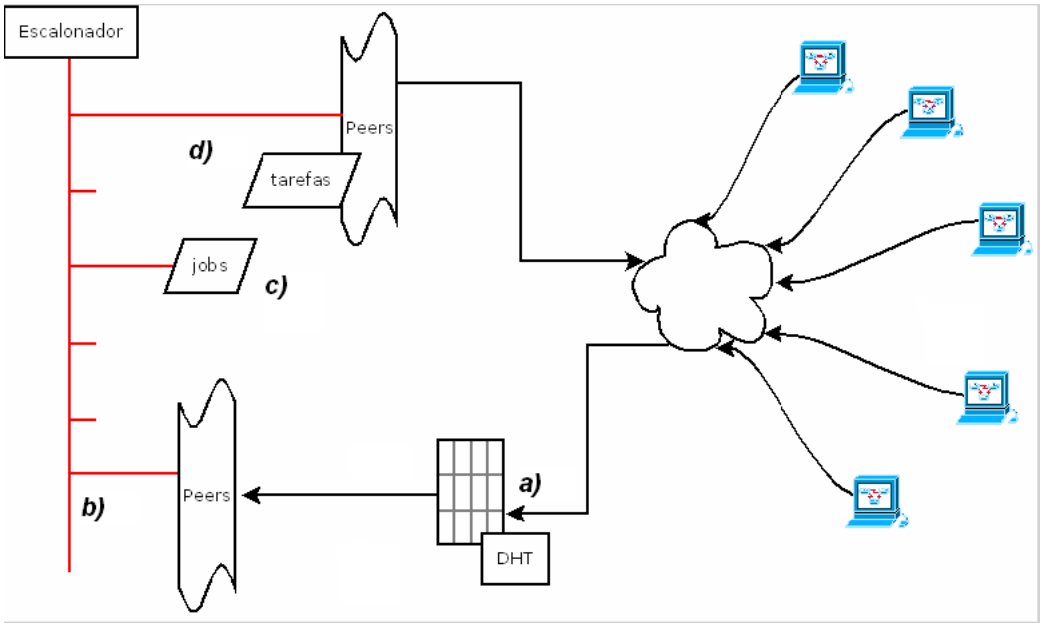

Figura 6.1: a) os identificadores dos nós são armazenados na DHT; b) os identificadores são repassados ao módulo de escalonamento; c) os trabalhos são submetidos ao módulo de fragmentação do escalonador; d) o escalonador executa a alocação de tarefas aos nós.

O algoritmo implementado neste trabalho é característico de aplicações que possuem tarefas independentes, ou seja, de tarefas que não dependem de intercomunicação entre as mesmas. Assim, as pesquisas das sequências realizadas pelas ferramentas de Bioinformática serão enviadas aos nós da rede e executadas, sem a necessidade de uma tarefa aguardar o resultado de outra.

Todas as tarefas devem terminar o seu trabalho para a produção de um resultado válido, ou seja, todas as execuções paralelas devem terminar normalmente para a consolidação de seus resultados. Se, por algum motivo, alguma tarefa pertencente a um trabalho (grupo de arquivos particionados do arquivo FASTA) não terminar normalmente a sua execução, a tarefa será enviada novamente para execução em um nó diferente daquele em que estava executando.

Para o escalonamento, consideramos a execução dos escalonadores nos *peers* que submeterão as tarefas, com a flexibilidade de escolha do escalonador. Na Fi-

gura 6.2 vemos dois escalonadores implementados no p2pBIOFOCO - Workqueue e WQR - com a possibilidade de inclusão de outros, neste caso, representados pelos escalonadores Min-Max e TTG.

Figura 6.2: Serviço de Escalonamento

6.2 Escalonamento de Tarefas usando WQR

Nesta seção, inicialmente descrevemos com detalhes o método WQR e, em seguida, os pontos principais da implementação.

6.2.1 Descrição Detalhada do Método

Como os outros métodos de escalonamento estudados neste trabalho, o WQR recebe uma lista de nós ativos localizados pela DHT e atribui a cada nó uma tarefa do trabalho. Após a execução de $n - m$ tarefas, onde n é o número de tarefas e m o número de máquinas, com $n > m$, o método WQR efetua a duplicação de uma tarefa em execução e envia essa cópia para outro nó disponível do sistema.

A quantidade de cópias é limitada, possuindo um valor limite de replicação definido de acordo com o tamanho do sistema e de acordo com a complexidade e o número das tarefas. A principal característica desse escalonamento é a troca do *overload* de mensagens que deixam de transitar pela rede (para atualização de informações utilizadas em escalonadores mais dinâmicos) pelo consumo de CPU [99]. Quando uma das cópias encerra normalmente a execução de sua tarefa, a execução das outras cópias é cancelada, concluindo a tarefa com êxito. A execução do método é descrita em seguida.

Método WQR

1: seleciona o trabalho
2: obtém a lista de tarefas do trabalho
3: executa algoritmo WQ
4: **for** $i = 0$ to $i <$ tamanho da lista com tarefas originais e replicadas **do**
5: verifica se a tarefa pode ser replicada {a inclusão da réplica depende da quantidade de tarefas que estão sendo executadas e da quantidade de falhas na tentativa de executá-las}
6: se afirmativo, replica a tarefa {a quantidade de tarefas em execução mais a tarefa replicada não pode ultrapassar o limite de réplicas}
7: aloca a tarefa para o nó disponível
8: **end for**

Um dos principais argumentos defendido pelos autores do algoritmo é o seu bom desempenho com relação aos algoritmos que utilizam informações sobre o sistema. Apesar do gasto extra de ciclos de CPU, em decorrência da duplicação das tarefas, a compensação quando ao custo dispendido com a troca de informações entre os nós, ainda tem suas vantagens. O custo de troca de mensagens envolve várias interações entre os nós, mais o custo para a manutenção de um sistema gerenciador dos recursos. Outro problema, indicado pelos autores, refere-se ao monitoramento do sistema distribuído em decorrência de restrições de segurança implantadas pelos administradores para acesso aos recursos, como, por exemplo, a existência de *firewalls*.

Com a utilização da replicação, espera-se que as tarefas que estão atrasando a execução do trabalho em consequência de sua atribuição a um nó sobrecarregado ou com recursos limitados, sejam copiadas e atribuídas aos nós com maior probabilidade de encerrá-las antes do fim da execução da tarefa original. Nesse esquema, a alta heterogeneidade dos recursos e a baixa heterogeneidade das tarefas são importantes para o custo adicional de ciclos de CPU. O custo, dessa forma, não é muito relevante, pois a probabilidade de uma cópia da tarefa ser atribuída a um nó com baixa utilização de seus recursos, é maior, possibilitanto a execução mais rápida da tarefa.

No p2pBIOFOCO identificamos que as tarefas são muito semelhantes, pois referem-se à execução de uma ferramenta de Bioinformática que tem como entrada sequências biológicas de tamanhos entre 1K e 2K, com pouca variação, portanto, no tamanho. Quanto à complexidade da tarefa, cada sequência é comparada com todo o banco de dados biológicos, mantendo-se o tempo de execução das sequências dentro de um intervalo proporcional ao tamanho das sequências, que não se diferenciam muito em seu tamanho. Caracterizadas as tarefas dessa maneira, se

houver baixa heterogeneidade dos recursos (máquinas de mesma configuração) e cargas semelhantes (decorrente da baixa heterogeneidade das tarefas), o consumo de CPU será alto. Como resultado, haverá pouco impacto no desempenho em decorrência da baixa probabilidade de atribuição de uma cópia de uma tarefa a um nó não sobrecarregado e com grande capacidade de recursos.

Devemos observar a questão referente à tolerância a falhas. Importante para um sistema distribuído consistente, ela não foi incluída no método WQR. No entanto, encontramos em [103] a proposta de dois métodos - *soft state* e *timeout* - para tratamento do problema. Em decorrência da importância do assunto, detalhamos a execução dos dois métodos com o objetivo de implementá-los futuramente no método WQR.

Na Seção 4.4, Tada et al [103] observaram que a implementação do método WQR apresentava uma deficiência com relação ao tratamento de falhas na execução das tarefas e ao cancelamento explícito de tarefas em decorrência da existência de barreiras nas redes (*NATs* ou *firewalls*). Essa complicação deve-se ao fato da comunicação iniciar-se no nó que solicitou a execução da tarefa, com o envio da solicitação de cancelamento da tarefa. No entanto, em algumas situaçoes, não é permitido ao nó que executa o escalonador o envio da mensagem de cancelamento para os outros nós, submetidos a restrições de comunicação. Desse modo, decidiram incluir dois métodos para preenchimento dessa lacuna na definição do método WQR. O primeiro método é chamado de *task timeout*, enquanto o seguinte refere-se ao método *soft state*.

Com a instituição do *task timeout*, assume-se um valor de *timeout* no escalonador para execução da tarefa que ele submeteu. Se não houver resposta até o término do tempo determinado, a tarefa é copiada e escalonada para outro nó, garantindo assim a execução de todas as tarefas. O problema surge quando se estabelece qual será o valor do *timeout*. Um valor subestimado resulta em replicação de tarefas que estão quase terminando sua execução, aumentando os custos de ciclos de CPU e de comunicação. Se o valor for superestimado, haverá problemas com a identificação de tarefas que falharam durante a sua execução. O cancelamento explícito da tarefa e de suas cópias é iniciado pelo nó do escalonador.

No segundo método, *soft state*, trabalha-se com a atualização das informações sobre a execução das tarefas e o envio de informações periódicas é iniciado pelo nó remoto, contornando o problema da reconfiguração da rede (inclusão de regras de *firewall*, NATs). Ao submeter a tarefa, o escalonador inicia a contagem do *timeout*, aguardando mensagens de atualização da execução da tarefa. Essas mensagens são enviadas pelo nó remoto ao escalonador para evitar que as reconfigurações da

rede atrasem o escalonador no envio de uma mensagem de cancelamento explícito da tarefa para outros nós, quando a tarefa é encerrada.

Se o escalonador não receber uma mensagem do nó para o qual foi submetida a tarefa até terminar o *timeout* de excução da tarefa, a tarefa é copiada e enviada para outro nó. Do mesmo modo, se o nó que recebeu a tarefa para execução não recebe uma mensagem do escalonador (a tarefa foi submetida a um nó que, durante o tempo de sua execução, foi submetido a restrições de acesso), informando que ainda está esperando pela execução da tarefa, a execução da tarefa é cancelada pelo nó remoto e uma mensagem de requisição de uma nova tarefa é enviada para o escalonador.

6.2.2 Detalhes da Implementação

Para a implementação e a execução de qualquer algoritmo de escalonamento no p2pBIOFOCO, duas estruturas essenciais são necessárias: a lista de máquinas ativas e a fila de tarefas. A lista de máquinas disponíveis para determinado nó é obtida pelo Kademlia e a fila de tarefas é preenchida com todas a tarefas prontas para execução.

Para a execução remota dessas tarefas, enviamos os parâmetros de execução das tarefas e os dados utilizados em suas execuções (nome do programa chamado, parâmetros para execução do programa, as sequências do arquivo FASTA para execução da pesquisa, banco *nr*). Caso a base de dados *nr* não esteja presente no nó, utilizaremos o método *Dual Protocol - Round Robin* (DP-RR), descrito na Seção 6.3, para o envio do *nr* aos nós de execução do p2pBIOFOCO.

O tamanho e a granularidade das tarefas são importantes para o tipo de escalonador que será usado no p2pBIOFOCO. Se na primeira versão do sistema, a prioridade era o processamento paralelo da aplicação BLAST, obtendo bons resultados com a redução substancial do tempo de execução da pesquisa genômica na base de dados *nr*, neste trabalho, a racionalização na utilização dos recursos e o aumento do desempenho são os próximos níveis para melhorar a eficiência do p2pBIOFOCO no processamento distribuído de tarefas.

Assim, se assumirmos que a granularidade das tarefas seja de uma tarefa por máquina, com tamanhos de tarefa muito próximos (tarefas por exemplo de 10MB), o resultado do trabalho estará limitado ao tempo de execução da máquina mais lenta (considerado que todas as tarefas são enviadas aos nós ao mesmo tempo).

A fragmentação do arquivo FASTA é executada pela classe *FastaParser*, produzindo arquivos com tamanhos entre 400 *bytes* e 4K *bytes*. Esse fracionamento determina um alto grau de homogeneidade das tarefas, em razão da pequena va-

riação no tamanho dos arquivos. Na realização dos experimentos, Capítulo 7, descreveremos o impacto dessa propriedade na execução dos algoritmos WQ e WQR.

Os principais atributos de uma tarefa no p2pBIOFOCO são o estado, o identificador, o atributo *tool* e o arquivo que a tarefa representa para o p2pBIOFOCO. Para o caso específico referente à execução do algoritmo WQR, incluímos mais um atributo que se refere à quantidade de cópias que o algoritmo pode criar. O atributo *tool* refere-se à aplicação que será executada pelo nó remoto. A execução do BLAST por exemplo atribui o valor "BLAST" ao atributo *tool*. O identificador da tarefa é único para cada tarefa. Através do identificador, podemos obter o estado atual da tarefa na máquina em que ela está sendo executada. A tarefa apresenta seis estados em seu ciclo de existência no p2pBIOFOCO:

- *READY*: a tarefa foi alocada pelo escalonador, mas está pendente de execução, aguardando a liberação dos recursos do nó para a sua execução;
- *RUNNING*: a tarefa foi alocada e enviada pelo escalonador para algum nó e está em execução;
- *CANCELLED*: a tarefa teve sua execução cancelada de forma controlada, ou para liberação de recursos ou, como no caso do escalonador com replicação de tarefas porque uma de suas cópias já havia terminado a tarefa;
- *ABORTED*: a tarefa terminou de forma inesperada;
- *FAILED*: a tarefa obteve um erro como resultado de sua execução;
- *FINISHED*: a tarefa terminou normalmente

O primeiro algoritmo de escalonamento do p2pBIOFOCO foi construído como uma fila de tarefas que são atribuídas aleatoriamente aos nós disponíveis. Assim, para submetermos um arquivo FASTA a partir de qualquer nó do sistema, o arquivo era dividido em arquivos menores, conforme processo de fragmentação mencionado no Capítulo 5. Se existissem mais tarefas do que máquinas, as tarefas permaneciam na fila à espera do término de alguma das tarefas em execução. Se houvesse mais máquinas do que tarefas, todas as tarefas eram atribuídas às máquinas, independente da situação da máquina, se carregada ou não. Se alguma máquina se tornasse indisponível durante o processo de alocação, a próxima máquina receberia a tarefa. A vantagem desse método era a sua simplicidade, ao custo de um desempenho ruim em decorrência da atribuição de tarefas grandes para máquinas com recursos limitados e de tarefas simples para máquinas com grande poder de processamento, deixando-as ociosas.

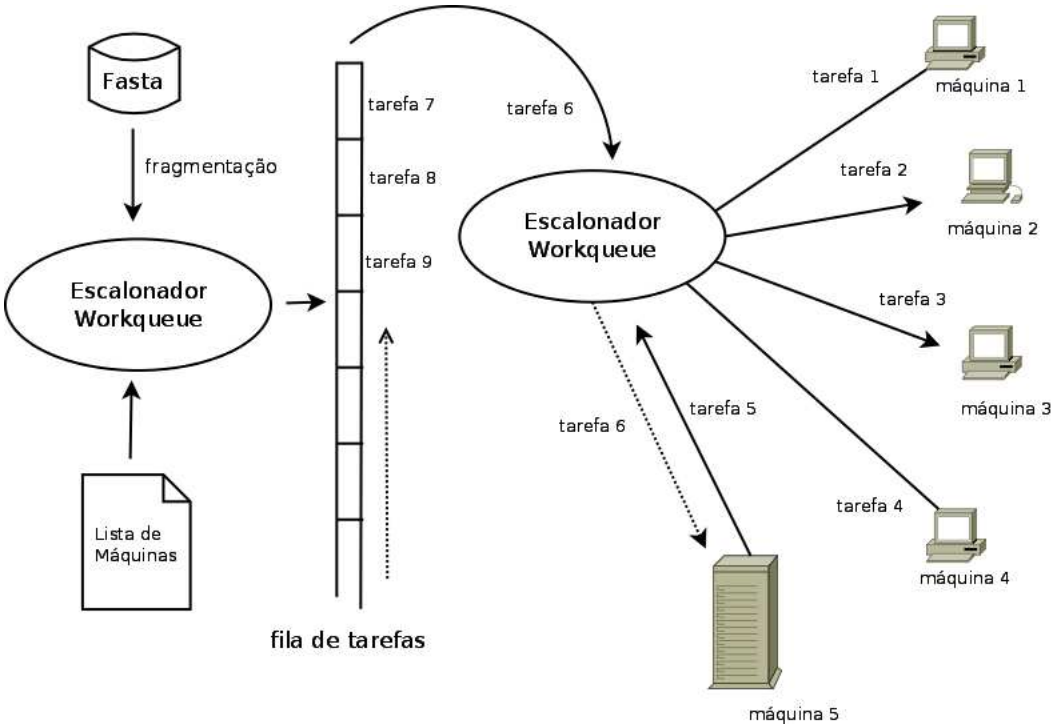

Figura 6.3: Atribuição de tarefas utilizando o escalonador *Workqueue* sem replicação.

Na Figura 6.3, vemos a execução do algoritmo WQ no ambiente do p2pBIOFOCO. As linhas contínuas a partir do escalonador até as máquinas indicam que a tarefa está em execução e que nenhuma outra tarefa é atribuída a mesma máquina (tarefas 1 e 4). As linhas que se iniciam no escalonador em direção às máquinas indicam a alocação da tarefas (tarefas 2 e 3). Ao terminar a tarefa 5, a máquina envia ao escalonador uma mensagem de término da tarefa. O escalonador, então, seleciona a próxima tarefa da fila de tarefas e a envia para a máquina 5. Na figura, a linha pontilhada representa o envio da tarefa 6, que foi retirada da fila e está sendo alocada pelo escalonador.

Em nossa implementação do WQR, enviamos todas as tarefas para uma fila FIFO (*first in first out*). Após a inclusão de todas as tarefas na fila, o módulo de escalonamento começa o seu trabalho alocando uma tarefa para cada nó ativo do sistema, utilizando a lógica do algoritmo WQ.

Esse procedimento é repetido até a fila de tarefas pendentes reproduzir a situação em que temos uma última tarefa em execução em cada nó do p2pBIOFOCO. Nesse estágio, após o término de uma tarefa, o nó que terminou a tarefa torna-se ocioso e inicia-se a replicação das tarefas, que ainda estão em execução em outros nós. A tarefa replicada, então, é enviada ao nó ocioso para execução na tentativa de executá-la mais rapidamente.

Quando a tarefa é terminada, o nó que finalizou a tarefa envia uma mensagem

ao nó cliente informando o estado *FINISHED* para a tarefa. Na Figura 6.4, vemos o algoritmo em execução. As máquinas ligadas ao algoritmo estão enviando tarefas e recebendo informações sobre as execuções dessas tarefas. Finalizada uma tarefa, o algoritmo atribui uma nova tarefa ao nó ocioso, retirando uma tarefa da fila de tarefas daquele nó.

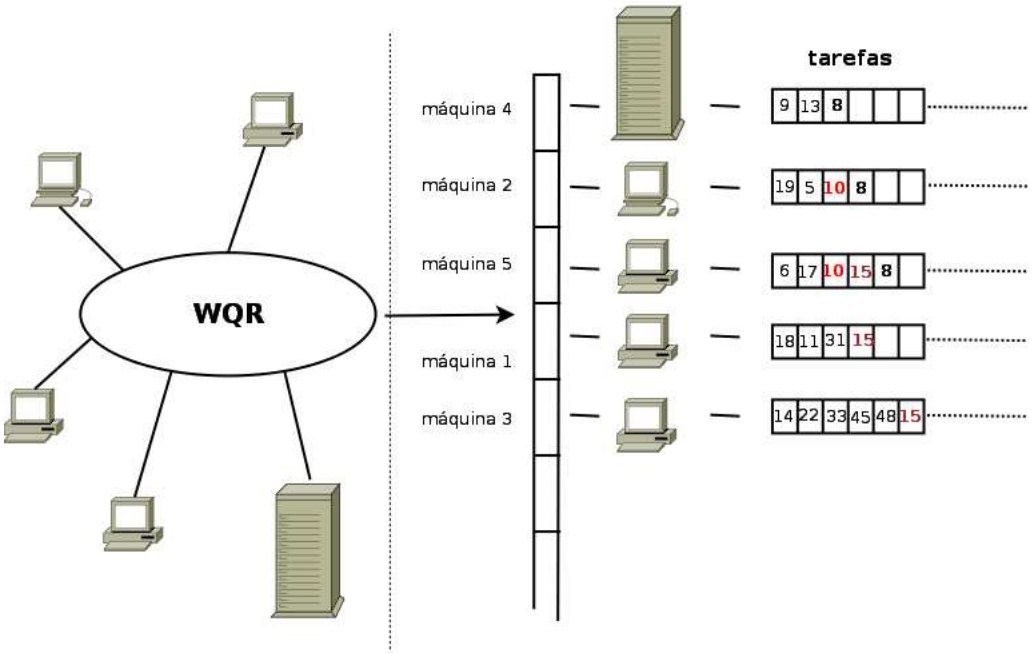

Figura 6.4: WQR em Execução.

Para evitar a replicação ilimitada de tarefas, implementamos um fator de replicação de no máximo três tarefas replicadas no P2P. De acordo com o trabalho de Cirne et al [99], acima desse limite, a replicação, que ocasiona a troca de ciclos de CPU pelo intercâmbio de informações, depedendo do cenário, não produz bons resultados, em decorrência do gasto de CPU e do aumento do tráfego de tarefas pela rede.

A implementação do método WQR no p2pBIOFOCO determina um ganho de desempenho com relação ao método básico WQ. Por outro lado, a replicação dos dados pode aumentar bastante os custos de execução das tarefas, considerando os tempos de transferência dos dados e a sobrecarga na infraestrutura, resultando em gargalos e degradação do desempenho do sistema. Esse problema não foi discutido no trabalho de Cirne et al [99], sendo importante na análise do processo de replicação de tarefas. Se constatado que a utilização do método, devido ao crescimento do tamanho dos dados, impacta negativativamente o desempenho de um sistema P2P, com é o caso do p2pBIOFOCO, outro algoritmo deve ser escolhido para o escalonamento das tarefas.

6.3 Transferência de Dados usando DP-RR

Nesta seção, inicialmente descrevemos, com detalhes, o método DP-RR e, em seguida, detalhamos a importância de sua implementação no p2pBIOFOCO.

6.3.1 Descrição Detalhada do Método

Para a transferência dos arquivos que serão transmitidos pelo sistema, a solução de Wei et al [110] tem um bom desempenho, conforme estudo dos autores. Sua implantação no sistema atende aos serviços de transferência de arquivos entre os nós. Em algumas situações, esses arquivos apresentam tamanhos consideráveis e em outras não, caracterizando uma situação na qual podemos utilizar o algoritmo mencionado. Dividimos, então, as funções em duas tarefas:

- o escalonamento de tarefas com a transmissão de sequências grandes ou pequenas; e,
- o suporte à manutenção, atualização e consolidação das bases de dados presentes no sistema.

Na Figura 6.5 vemos um esquema que demonstra as duas abordagens referentes ao escalonamento de tarefas e dados.

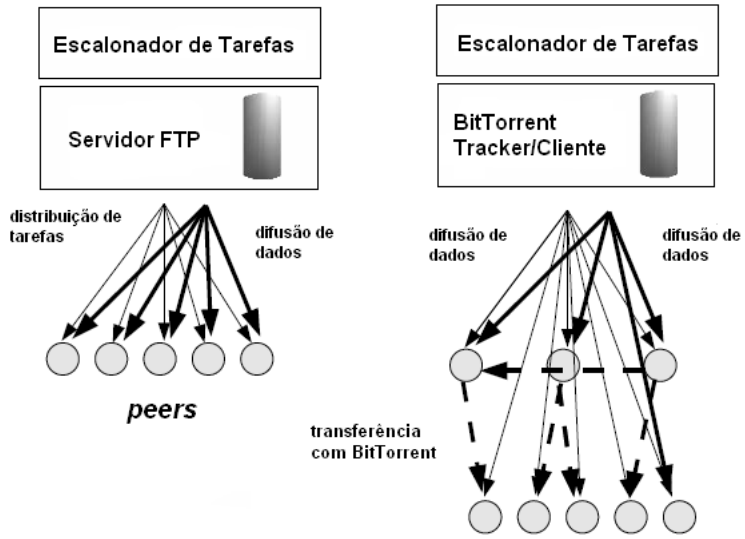

Figura 6.5: Comparação entre as duas arquiteturas de escalonamento de tarefas e dados: escalonamento centralizado de dados e tarefas e escalonamento centralizado de tarefas e difusão colaborativa de dados [110].

O algoritmo citado é apropriado para aplicações chamadas *parameter sweep*. Tais aplicações surgiram de um modelo proposto por Abaramson et al [1] para composição de aplicações com alto *throughput* em *grids* globais [19]. É uma combinação de modelos paralelos de tarefas e dados, com aplicações formuladas para conter um grande número de tarefas independentes, trabalhando em conjuntos diferentes de dados. A execução do modelo geralmente envolve o processamento de n tarefas independentes em m computadores onde n é muito maior que m, situação que pode ocorrer no processamento das tarefas do p2pBIOFOCO.

O BitTorrent [26], usado na solução proposta por Wei et al, é um protocolo popular para a distribuição de arquivos que evita os gargalos geralmente encontrados em soluções que utilizam os servidores FTP quando existe a transferência de arquivos muito grandes e uma grande demanda por arquivos. A ideia por trás do BitTorrent é a cooperação entre os requisitantes do mesmo arquivo, que enviam 'pedaços' (*chunks*) do arquivo entre eles. O primeiro nó que obter toda a informação completa sobre o arquivo cria um arquivo especial rotulado de *.torrent*. Esse arquivo possui uma lista de todas as assinaturas dos pedaços do arquivo e a localização de um coordenador central, o *tracker*, que auxilia os nós a se conectarem entre si. Os *trackers* são responsáveis pela conservação de uma lista com informações sobre os nós: endereço IP, porta, arquivos recebidos, etc. Quando um nó solicita um arquivo, a primeira requisição é enviada ao *tracker* pedindo a lista dos nós que estão recebendo o arquivo. Assim, todas as decisões sobre a recepção do arquivo são efetuadas localmente com base em informações coletadas dos nós vizinhos.

O escalonador DP-RR é definido da seguinte maneira:

- a aplicação $A = T_1, ..., T_n | n > 0$, com cada tarefa sendo composta de um número de operações Ω_i;
- cada tarefa T_i é associada a um conjunto de entrada de dados $\theta(T_i) = I_1^i, I_2^i, ..., I_l^i$ onde cada I_j^i é um dado de entrada;
- conjunto global de entrada de dados da aplicação, representado por $\theta(A) = \underbrace{\bigcup}_{1<i<|T|} \theta(T_i)$;

Observando as definições do escalonador, podemos identificar pontos relacionados ao p2pBIOFOCO, para aplicação neste trabalho. O conjunto de tarefas definido no primeiro item acima pode ser considerado como o conjunto de arquivos FASTA e as bases de dados submetidas pelos usuários do sistema. As operações Ω_i referem-se a cada sequência do arquivo. Aos dados de entrada relacionamos as sequências

de cada arquivo FASTA, constituindo, portanto, o conjunto de entrada de dados. Finalmente, o conjunto de todas as sequências e de bases de dados compõe o conjunto global de dados alvo do escalonamento.

Para a caracterização da aplicação, os autores do algoritmo consideram quatro parâmetros:

1. *Granularidade da Aplicação*: é especificada pelo par (tamanho dos dados de entrada; número de nós ativos do sistema). No contexto do p2pBIOFOCO, esses parâmetros referem-se ao tamanho de cada sequência do arquivo FASTA e à quantidade de nós obtidos pela DHT;

2. *Tamanho da Aplicação*: refere-se à quantidade de tarefas da aplicação com relação ao número de processadores disponíveis. Para o p2pBIOFOCO, são a quantidade de sequências obtidas da segmentação do arquivo FASTA e a quantidade de nós disponbilizados pela DHT;

3. *Taxa de Computação (CIOR-Computation/file IO Ratio)*: Baohua et al [110] adotaram uma abordagem flexível com relação aos protocolos e determinaram essa razão como o *tamanho dos dados de entrada/soma do número de operações*. Essas informações correspondem ao tamanho do arquivo FASTA (soma de todas as sequências do arquivo) dividido pela soma do número de tarefas produzidas pela segmentação do arquivo, conforme a expressão da equação 6.1. Por exemplo, se temos um arquivo com 200 sequências, de tamanho 20MB, o parâmetro é calculado como 20/200, tendo como resultado 0,10 MB por sequência, ou seja, 0,10 MB por tarefa.

$$CIOR = \frac{tamanho(\theta(A))}{\sum_{1<i<|T|} \Omega_i} \tag{6.1}$$

4. *Taxa de Compartilhamento dos Dados (SDR-Shared Data Ratio)*: equivale à quantidade de dados que são compartilhados entre as tarefas. O parâmetro é computado como o volume de dados da aplicação sobre o volume de dados de entrada distribuído por cada tarefa (equação 6.2).

$$SDR = \frac{tamanho(\theta(A))}{\sum_{\substack{1<i<|T| \\ 1<k<|I(T_i)|}} tamanho(I_i^K)} \tag{6.2}$$

No escalonamento da aplicação $\boldsymbol{A}$ com a utilização dos protocolos BitTorrent e FTP, alguns parâmetros foram utilizados para avaliação e análise do desempenho.

O *tempo estimado de execução* (ETC) de uma tarefa em um nó é definido como o tempo que o processador P_j utiliza para executar a tarefa T_i, dado o número de operações e o poder de processamento de P_j. O tempo esperado de transferência do conjunto de entradas da tarefa T_i é o tempo gasto para transferir esse conjunto de dados até o nó P_j antes da tarefa poder iniciar a sua execução. Por fim, o tempo esperado de fim da tarefa (*completion time*) c_1 para o nó P_j é definido como o tempo de *clock* do processador no qual a tarefa é encerrada, dado que todos os dados estão presentes e todas as tarefas previamente atribuídas já terminaram. Além disso, foi considerado o *makespan* como métrica de desempenho do escalonador, com o objetivo de minimizá-lo.

Wei et al fizeram várias análises experimentais de transferências de arquivos e concluíram que a utilização de somente um deles traz benefícios, considerando o tamanho dos dados. Para a escolha de um dos protocolos na execução da aplicação, a fixação criteriosa de um determinado ponto é importante para a troca de um protocolo por outro. Assim, decidiram expressar esse parâmetro com base no número de nós e no tamanho dos arquivos. Para a utilização do FTP, propuseram o cálculo mostrado na equação 6.3, assumindo que a banda de rede é dividida igualmente entre os requisitantes dos arquivos.

$$t(N, S) = \alpha_{ftp} + \frac{N.S}{\beta_{emissor}} \tag{6.3}$$

Na equação 6.3, α_{ftp} e $\beta_{emissor}$ são duas constantes referentes à latência do protocolo e à capacidade de banda de rede do nó que envia os dados, respectivamente. S refere-se ao tamahno dos dados e N ao número de nós. O modelo do padrão de comunicação e transferência do BitTorrent é representado na Equação 6.4, que descreve um modelo em árvore para o padrão de distribuição.

$$t(N, S) = (\beta_{bittorrent} + \frac{\alpha.S}{\beta_{emissor}}).log_{\alpha}(N) \tag{6.4}$$

Assim, $\beta_{bittorrent}$, $\beta_{emissor}$ e α são constantes que representam respectivamente a latência do protocolo, a disponibilidade banda de rede no emissor e o número de folhas da árvore, sendo S o tamanho dos dados e N o número de nós receptores.

Na execução do escalonamento pelo método DP-RR, sugerimos um gerenciamento de dados semelhante ao apresentado na arquitetura LBG (*Lightweight Bartering Grid*) [16] para a mudança de protocolo entre o BitTorrent e o FTP. Essa troca seria gerenciada por componentes chamados *data managers* e cada nó do sistema possuíria no mínimo desses componentes, que executariam as seguintes funções:

- armazenamento dos dados no nó local ou remoto;

- compartilhamento dos arquivos com utilização do BitTorrent (*tracking* e *seeding* nos consumidores e *seeding* somente nos recursos) e compartilhamento FTP (nós consumidores somente);
- *download* BitTorrent e FTP (atribuição dos nós quando atuam como recursos).

6.3.2 Detalhes da Implementação

Para a implementação do DP-RR utilizamos a biblioteca Java BitTorrent (jBittorrent) [40]. A API foi escrita em Java e apresenta as implementações básicas do procotolo *BitTorrent*. Sua integração ao p2pBIOFOCO proporcionou a distribuição dos arquivos entre os nós, servindo de alternativa à transferência sftp.

O principal motivo para a implementação do protocolo foi o ganho de desempenho para a transferência de arquivos grandes que são entradas para a comparação das sequências informadas pelo usuário. A transferência ftp ou sftp constitui um gargalo para o *download* dos arquivos que são utilizados nas máquinas do P2P. Arquivos grandes que são necessários à execução das comparações são, dessa forma, transferidos de forma eficiente com a utilização do protocolo *BitTorrent*.

Na nossa implementação, todo nó do sistema é considerado um *tracker*. Além disso, são considerados também *seeders* pois, após a baixa do arquivo, anunciam-se no *tracker* para fazerem parte do grupo de nós que compartilham o arquivo. No p2pBIOFOCO, os arquivos resultantes da fragmentação do FASTA são pequenos e não são considerados na transferência executada com o *BitTorrent*. Nosso interesse está na transferência de arquivos grandes que são necessários à execução das comparações que; no caso específico do p2pBIOFOCO, é o caso do arquivo *nr*. Tendo em vista o tamanho grande do arquivo *nr*, consideramos a sua compactação dentro do processo de transferência do arquivo entre os nós. Na Figura 6.6, mostramos a sequência de envio dos dados utilizando o protocolo *BitTorrent*.

Então, temos os seguintes passos, numerados na Figura 6.6:

1. o nó interessado em transferir os dados para execução remota das tarefas cria o arquivo *torrent*, por exemplo, da base de dados *nr* compactada;
2. em seguida anuncia-se ao *tracker* informando que possui uma cópia completa do arquivo *nr* (compactado);
3. o arquivo *torrent* é enviado para todos os nós ativos do sistema;
4. ao receberem o arquivo *torrent*, cada nó processa o arquivo *torrent* e comunica-se com o *tracker*, formando o grupo de nós que compartilharão o arquivo *nr* no processo de *download*;

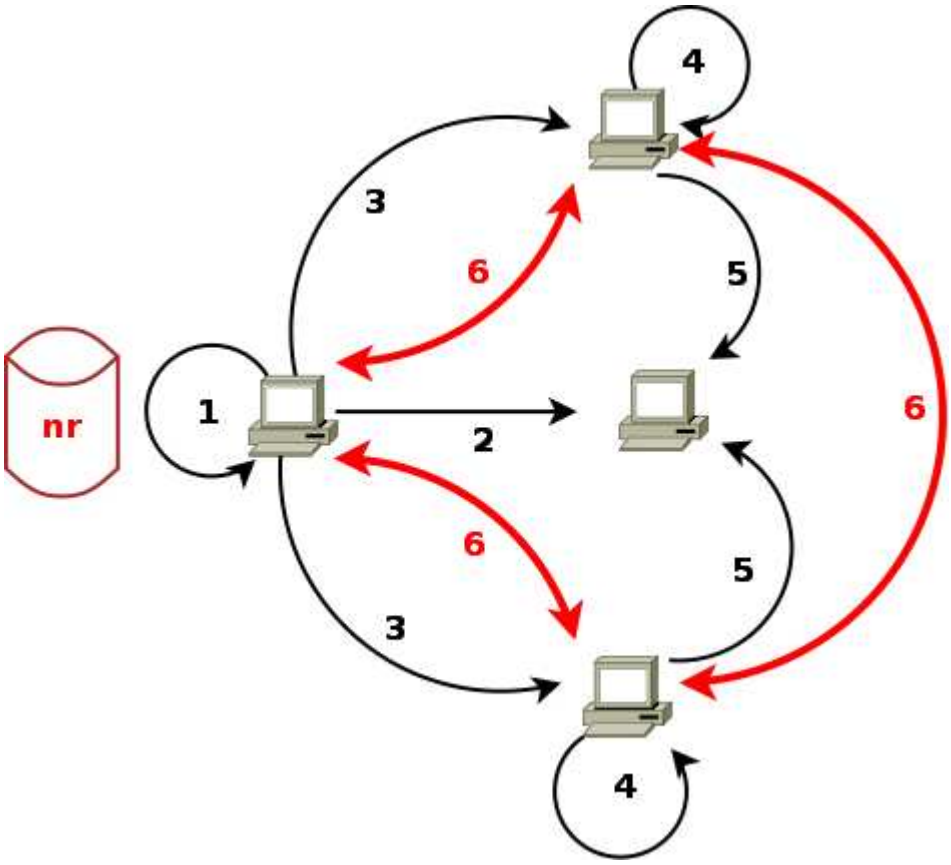

Figura 6.6: *DP-RR*: Transferência dos Dados utizando *BitTorrent*.

5. logo após a inclusão do nó interessado no *tracker*, os nós comunicam-se entre si;

6. o processo de *download* do arquivo *nr* compactado é iniciado entre os nós.

Na implementação da API Java BitTorrent no p2pBIOFOCO, definimos uma lista de *trackers* para evitar a existência de um único *tracker*. Isto evita que a falha em um nó se torne um problema para o *download* dos dados. A opção para utilização do protocolo *BitTorrent*, a princípio, está disponível na interface gráfica acessada pelo usuário. Ao escolhê-la, o usuário tem que estar ciente do tamanho do arquivo que está transferindo para não obter um desempenho baixo com a transferência de arquivos pequenos com a utilização do Bittorrent. Estamos implementando a troca automática entre sftp e BitTorrent na transferência dos arquivos com base na identificação do tamanho dos arquivos.

Portanto, para o p2pBIOFOCO, temos uma opção de escalonamento baseada em um protocolo duplo de transferência de dados (*Dual Protocol*) com uma política *round robin* de atribuição das tarefas aos nós. O algoritmo flexibiliza as vantagens de utilização de um protocolo ou de outro e compõe, em conjunto com o gerencia-

mento de dados, uma opção de distribuição dos arquivos de sequências que serão enviados aos nós executores das pesquisas (*queries*).

6.4 Resumo do Capítulo

Neste capítulo, discutimos as duas principais contribuições desta dissertação para o p2pBIOFOCO: a inclusão de um método de escalonamento (WQR) e a implementação de um método eficiente para a transferência de dados. A escolha flexível dos escalonadores depende da implementação de um sistema gerenciador de recursos. Assim, neste trabalho, implementamos somente o método WQR para avaliarmos o ganho de desempenho na execução distribuída das tarefas. Com a implementação do BitTorrent, disponibilizamos ao p2pBIOFOCO uma solução eficiente para a transferência de grandes bases de dados, com a opção de uso dos protocolos ftp/sftp ou BitTorrent conforme o tamanho dos arquivos.

Capítulo 7

Experimentos e Discussão

Neste capítulo, discutimos os resultados obtidos dos experimentos realizados nesta dissertação. Na Seção 7.1, descrevemos o ambiente de testes. Nas seções 7.2, 7.3 e 7.4 discutimos e comparamos os resultados obtidos dos experimentos.

7.1 Ambiente de Testes

O ambiente dos nossos experimentos está representado na Tabela 7.1, na qual temos 10 máquinas conectadas por uma rede local: três máquinas Intel com a mesma configuração (laico1, 2 e 3); duas máquinas AMD (laico7 e 9), de configurações idênticas, e cinco máquinas com recursos diferentes (laico4, 5, 6, 8 e lab-pos12).

Tabela 7.1: Configuração das máquinas utilizadas nos experimentos de escalonamento

Máquina	**Recursos**									
	CPU						**Discos**		**Memória**	
	Fabr	Mod	Veloc	Cache	Bits	Qtde	Tam	Tipo	Tam	Tipo
laico1	Intel	Pentium 4	1,7 GHz	256KB	64	1	40GB	IDE	256MB	SDRAM
laico2	Intel	Pentium 4	1,7 GHz	256KB	64	1	40GB	IDE	256MB	SDRAM
laico3	Intel	Pentium 4	1,7 GHz	256KB	64	1	40GB	IDE	256MB	SDRAM
laico4	Intel	Core2 Duo	1,8 GHz	2MB	64	1	150GB	SCSI	1GB	SDRAM
laico5	Intel	Pentium 4	3,0 GHz	2MB	64	1	80GB	IDE	1GB	SDRAM
laico6	Intel	Core2 Duo	2,5 GHz	3MB	64	1	160GB	SCSI	2GB	SDRAM
laico7	AMD	Sempron(tm) 2400+	1,6 GHz	256KB	64	1	36GB	IDE	2GB	DDR2
laico8	AMD	Sempron(tm) 3000+	1,8 GHz	128KB	64	1	36GB	IDE	2GB	DDR2
laico9	AMD	Sempron(tm) 2400+	1,6 GHz	256KB	64	1	36GB	IDE	2GB	DDR2
lab-pos12	AMD	Sempron(tm) 2400+	1660 MHz	256KB	64	1	36GB	IDE	2GB	DDR2

Os experimentos foram feitos com BLAST, de modo que as tarefas referem-se à execução remota do BLAST em cada máquina para cada arquivo de entrada (sequência), após a alocação executada pelo algoritmo de escalonamento. Foi utilizada a versão 2.14 do BLAST, obtida do sítio do NCBI. O banco de dados *nr*, com o qual

foram executadas as comparações, tem um tamanho de 6GB. A versão da máquina virtual Java utilizada foi a *JDK 6 Update 20*, obtida da Oracle.

A coleta de dados foi realizada com o registro das execuções em arquivos de *log*, contendo informações sobre o comportamento do algoritmo de escalonamento em resposta à execução das tarefas no p2pBIOFOCO. O tempo de execução dos algoritmos foi registrado na máquina do usuário que iniciou as tarefas, disparando um *timer* no momento de submissão das tarefas aos nós remotos. As tarefas foram submetidas sempre a partir da mesma máquina para evitar a influência de aspectos relacionados ao *hardware* na execução da submissão e processamento das tarefas.

A realização do teste em uma única máquina foi utilizada para a comparação com os testes distribuídos realizados no p2pBIOFOCO, com o banco de dados *nr* e as sequências de consulta (Tabela 7.2). Como no trabalho de Ribeiro [88], verificamos o custo de execução do BLAST para um único arquivo FASTA, com todas as sequências sendo pesquisadas no banco de dados *nr* da máquina, utilizando somente uma máquina específica do ambiente de testes para o trabalho.

Tabela 7.2: Execução não distribuída BLAST - Máquina laico4

Tarefas	**Tempo de Execução (em horas)**
50 sequências	3:02
100 sequências	5:45
200 sequências	11:29
400 sequências	22:10
800 sequências	45:47

Observamos que ao dobrar o número de sequências, dobramos também o tempo de execução. Além disso, já é longo o tempo para a execução de uma quantidade pequena de sequências, tornando impraticável o processamento para entradas com milhares de sequências.

Fizemos uma análise do tempo de execução das tarefas nas máquinas calculando a diferença entre a data de gravação do arquivo fragmentado na máquina remota (denominado ContigXXXX, onde XXXX é o número do *contig*), recebido via sftp, e a data de gravação do arquivo com o resultado da execução da pesquisa das sequências no *nr* (que indica o fim da execução do BLAST).

O tamanho de cada tarefa executada no p2pBIOFOCO refere-se ao tamanho do arquivo de entrada para a pesquisa no banco de dados *nr*. O tamanho médio dos arquivos submetidos a cada máquina está contido no intervalo de 400 bytes *bytes* a 3K *bytes*.

Na tabela 7.3, temos o tempo médio de execução das tarefas por máquina. Os valores são resultados da média aritmética de 50 tarefas selecionadas do grupo de 800 sequências, descartando a sequência mais longa.

Tabela 7.3: Tempo Médio de Execução das Tarefas (em minutos)

Total de Tarefas	**Máquinas**									
	laico1	laico2	laico3	laico4	laico5	laico6	laico7	laico8	laico9	lab-pos12
tempo	18,0	15,6	14,4	6,2	6,8	4,3	9,9	7,3	8,2	9,35

O tempo médio calculado mostra o impacto da execução das tarefas em máquinas mais lentas em relação às máquinas mais rápidas. No ambiente dos experimentos, a máquina mais lenta executa as tarefas em um tempo até 4 vezes maior, na média, que a máquina mais rápida (máquina laico1 em relação à máquina laico6). Portanto, na análise das execuções das sequências, as máquinas laico1 e laico6 foram as mais importantes na avaliação do tempo de execução resultante da aplicação dos algoritmos WQ e WQR.

Após o levantamento do tempo médio das tarefas nas máquinas, observamos que a atribuição de uma tarefa ao nó mais lento demorou 27 minutos para terminar a pesquisa no *nr*. Na tabela 7.4, informamos os tempos de execução mais longos das tarefas nas máquinas do p2pBIOFOCO.

Tabela 7.4: Tempo da Tarefa Mais Longa (em minutos)

	Máquinas									
	laico1	laico2	laico3	laico4	laico5	laico6	laico7	laico8	laico9	lab-pos12
tempo	27,0	22,0	21,0	8,0	12,0	5,0	15,0	10,0	17,02	22,0

7.2 Análise do Escalonador WQ

A execução do algoritmo *workqueue* mostrou-se, então, ineficiente para a execução das últimas tarefas presentes nas máquinas do P2P. Enquanto alguns nós ficam ociosos ao terminarem suas tarefas, o nó que possui esta tarefa atrasa o término do trabalho. No pior dos cenários, temos os nós mais rápidos finalizando suas tarefas e uma tarefa iniciando a sua execução no nó mais lento. Na Figura 7.1, mostramos a execução das tarefas no escalonador WQ quando todas as tarefas já foram alocadas.

Na Figura 7.1, temos uma linha vertical na qual estão indicadas as máquinas. A linhas horizontais indicam os tamanhos das tarefas, sendo que as linhas de mesmo tamanho indicam tarefas iguais. Os pontos no final de cada linha horizontal representam o término da tarefa e as linhas pontilhadas a quantidade de processamento

ainda a ser gasto na execução da tarefa. Para simplificarmos a análise, supomos que as tarefas foram alocadas no mesmo tempo $t_i = 0$. Após algum tempo, temos a situação representada na Figura 7.1.

Nesta situação, observamos que o último grupo de tarefas executando no sistema refletiu o problema de execução das últimas tarefas em nós mais lentos. A máquina 3, pela figura, é a máquina mais lenta ou a mais carregada do sistema. Nela temos somente um pequena parte de sua tarefa cumprida. Por outro lado, as máquinas 1, 5, 7 e 10 encerraram suas tarefa, tornando-se ociosas até a finalização da tarefa da máquina 3.

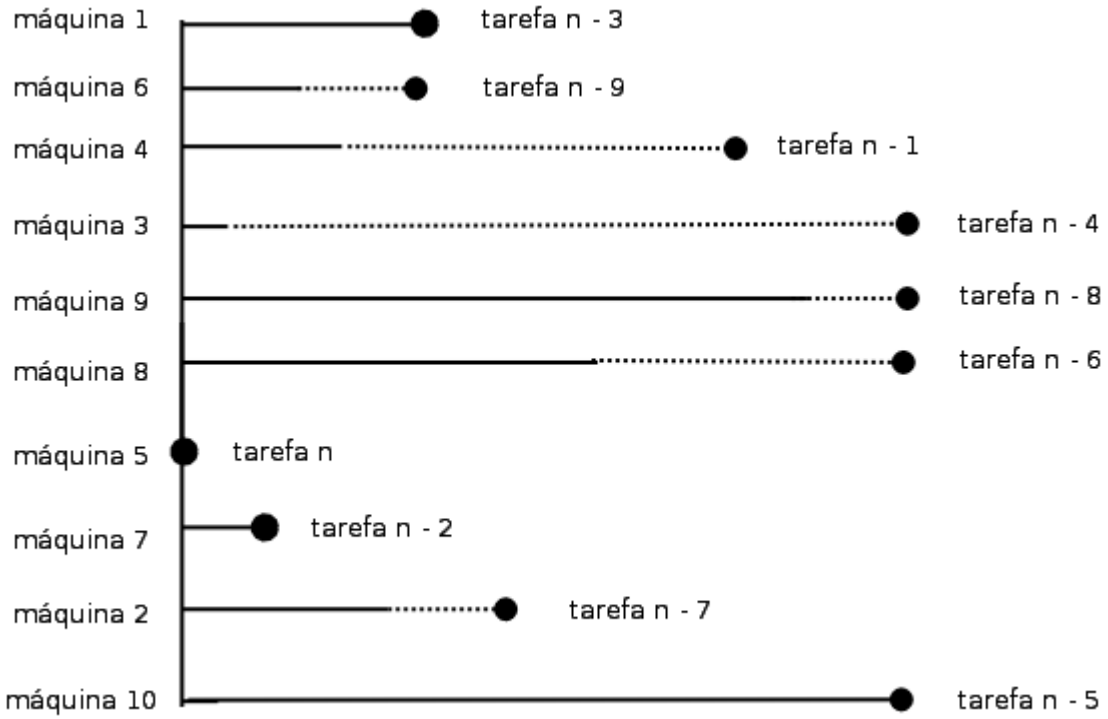

Figura 7.1: Execução das tarefas nas máquinas do P2P.

Fizemos, também, um avaliação da quantidade de tarefas processadas em cada máquina na execução do algoritmo WQ. Este número foi obtido verificando-se o número médio de arquivos gravados no diretório de resultados de cada máquina, após a execução de todas as sequências. Na tabela 7.5, vemos que a máquina laico6 foi a máquina com maior número de tarefas executadas, enquanto a máquina laico1, mostrou-se a máquina mais lenta. Construímos a tabela com o objetivo de compararmos com a Tabela 7.6, que mostra os efeitos da replicação no número de tarefas executadas por máquina com a execução do algoritmo WQ com replicação. Consideramos somente as tarefas completadas, excluindo as tarefas canceladas pelo algoritmo de replicação ou canceladas por erro ou indisponbilidade do nó.

Tabela 7.5: Tarefas por Máquina com o Algoritmo Original de Escalonamento WQ do p2pBIOFOCO.

Total de Tarefas	Máquinas									
	laico1	laico2	laico3	laico4	laico5	laico6	laico7	laico8	laico9	lab-pos12
50	3	3	4	6	7	9	4	5	5	4
100	5	6	6	13	13	18	10	11	9	9
200	10	11	12	29	25	39	18	21	16	19
400	19	22	23	55	50	77	35	45	37	37
800	38	45	47	110	96	155	73	88	74	74

7.3 Análise do Escalonador WQR

Após várias execuções do algoritmo, podemos ver na Tabela 7.6 que o número médio de tarefas executadas por máquina aumentou em algumas máquinas e diminuiu em outras. Este comportamento descreve a função principal do algoritmo WQR: a criação de cópias de tarefas em execução, até um determinado limite e alocação em outras máquinas na tentativa de obter uma máquina mais rápida para execução das tarefas.

Tabela 7.6: Tarefas por Máquina, com o WQR.

Total de Tarefas	Máquinas									
	laico1	laico2	laico3	laico4	laico5	laico6	laico7	laico8	laico9	lab-pos12
50	2	3	4	6	6	11	2	6	6	4
100	4	3	5	16	11	22	11	11	9	8
200	7	9	11	30	26	42	19	21	15	20
400	10	16	20	58	57	84	34	44	38	39
800	32	39	46	112	100	163	72	86	78	72

Como resultado da submissão das sequências (50, 100, 200, 400 e 800) aos nós do p2pBIOFOCO, com a utilização dos algoritmos WQ e WQR, observamos nas tabelas 7.7 e 7.8 os tempos médios de execução de todas as sequências.

tarefas	**(h)**
50	0:39
100	1:19
200	2:47
400	5:38
800	11:19

Tabela 7.7: Tempo - WQ

tarefas	**(h)**
50	0:28
100	1:14
200	2:37
400	5:25
800	11:04

Tabela 7.8: Tempo - WQR

A Figura 7.2, apresenta uma comparação dos tempos de execução das tarefas no p2pBIOFOCO utilizando os algoritmos WQ e WQR. O WQR tem um limite máximo de três cópias de uma tarefa, quando efetuada a rotina de replicação.

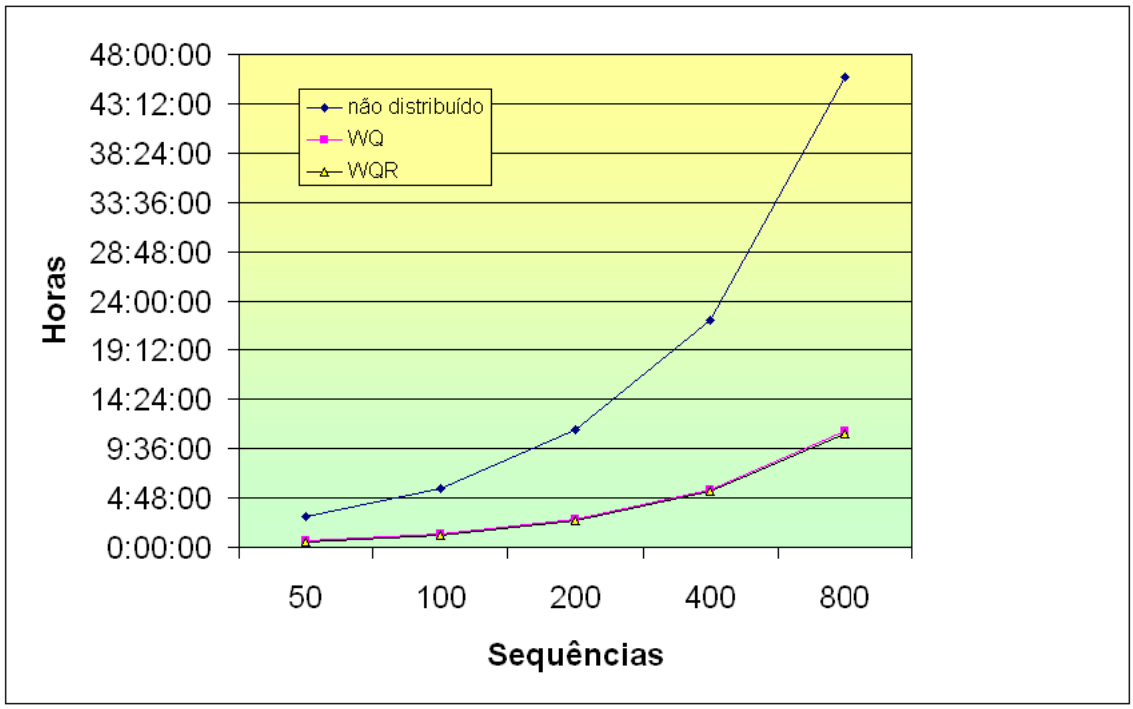

Figura 7.2: Execução não distribuída e dos escalonadores WQ e WQR no p2pBIOFOCO.

Para uma comparação mais detalhada, no gráfico Figura 7.3, vemos o tempo resultante da execução das tarefas somente para os escalonadores WQ e WQR.

7.4 Análise do Mecanismo de Transferência de Dados

O principal objetivo dos nossos testes foi a medição do tempo de transferência dos arquivos para as máquinas do p2pBIFOCO utilizando os protocolos sftp e BitTorrent e obtermos o protocolo mais apropriado para a distribuição dos arquivos ao sistema. Para isso, no ambiente de testes utilizado (Tabela 7.9), executamos a distribuição de arquivos com tamanhos de 10 MB, 50 MB, 200 MB, 700 MB e 1,5 GB, referentes a bancos de dados genômicos e aplicações de Bioinformática. Criamos, também, um serviço de transferência de arquivos no p2pBIOFOCO que envia, a partir de um nó com o arquivo o completo, o pedido de *download* desse arquivo para os outros nós do sistema (no caso do protocolo BitTorrent, pedido de *download* do arquivo *torrent*).

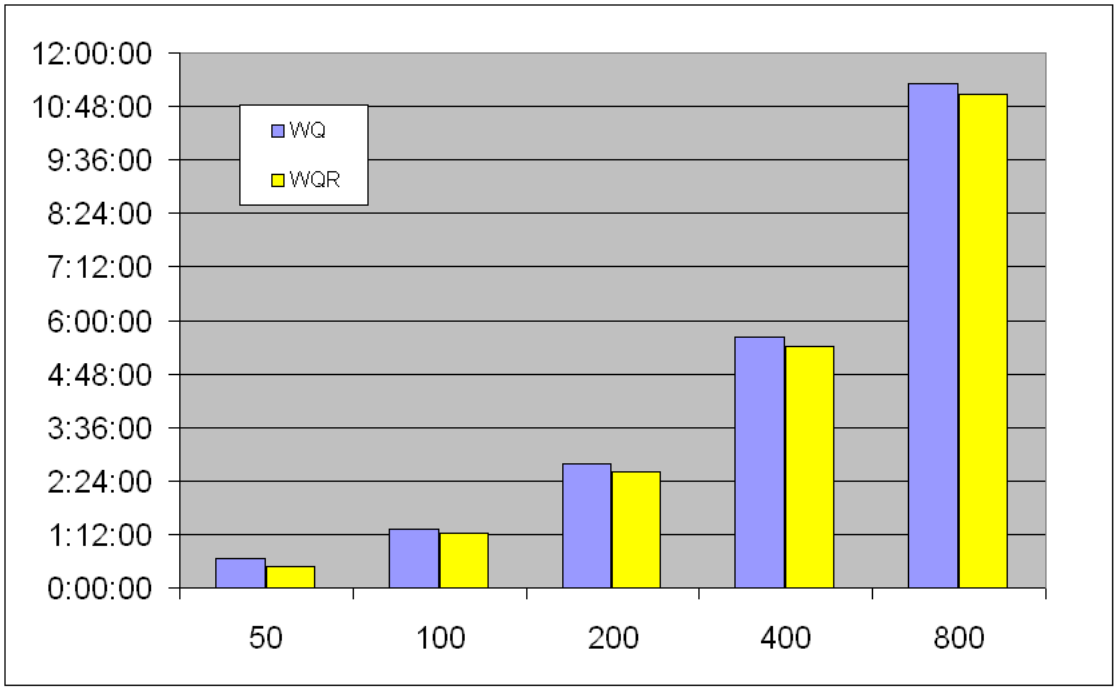

Figura 7.3: Gráfico com a execução dos escalonadores WQ e WQR no p2pBIOFOCO.

Para medirmos os tempos de todos os *downloads* efetuados na execução dos protocolos sftp e BitTorrent, disparamos um temporizador na máquina que enviou a solicitação a todos os nós, terminando sua execução quando a última máquina em processo de *download* retorna uma mensagem sobre o final do recebimento do arquivo.

Tabela 7.9: Configuração das máquinas utilizadas para transferência do arquivos

Máquina	**Recursos**									
	CPU						**Discos**		**Memória**	
	Fabr	Mod	Veloc	Cache	Bits	Qtde	Tam	Tipo	Tam	Tipo
laico1	Intel	Pentium 4	1,7 GHz	256KB	64	1	40GB	IDE	256MB	SDRAM
laico3	Intel	Pentium 4	1,7 GHz	256KB	64	1	40GB	IDE	256MB	SDRAM
laico4	Intel	Core2 Duo	1,8 GHz	2MB	64	1	150GB	SCSI	1GB	SDRAM
laico5	Intel	Pentium 4	3,0 GHz	2MB	64	1	80GB	IDE	1GB	SDRAM
laico6	Intel	Core2 Duo	2,5 GHz	3MB	64	1	160GB	SCSI	2GB	SDRAM
laico8	AMD	Sempron(tm) 3000+	1,8 GHz	128KB	64	1	36GB	IDE	2GB	DDR2
laico9	AMD	Sempron(tm) 2400+	1,6 GHz	256KB	64	1	36GB	IDE	2GB	DDR2

No método de distribuição de arquivos com o BitTorrent, um dos pontos críticos foi a confiabilidade do *tracker*. Desse modo, em nossa implementação, todos os nós do ambiente possuíam uma instância do *tracker* em execução. Além disso, utilizamos a mesma lista produzida pela DHT para constituir a lista dos *trackers*, sendo os três primeiros nós *trackers* do p2pBIOFOCO. Em nossas simulações, ao provocarmos a queda do *tracker* que estava servindo aos nós, verificamos que os nós foram reconectados ao próximo da lista.

Na tranferência por sftp, caracterizada pela transferência direta de dados, procuramos indentificar a formação de gargalos na transferência de arquivos muito grandes ou arquivos muito populares com grande quantidade de acessos ao servidor. No nosso ambiente de testes, no entanto, devido ao pequeno e constante número de nós, direcionamos nossa análise para o tamanho dos arquivos transferidos, excluindo-se a influência do número de nós.

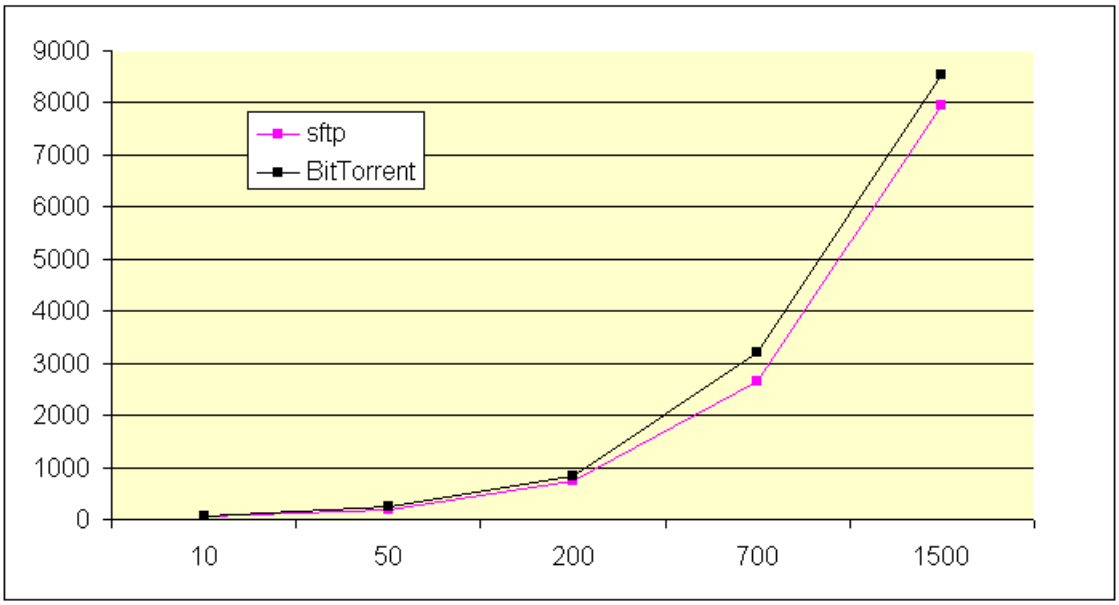

Figura 7.4: Transferência de arquivos utilizando os protocolos sftp e BitTorrent. No eixo vertical, representamos o tempo em segundos, enquanto no eixo horizontal, representamos o tamanho dos arquivos em MB.

Para a transferência com o BitTorrent, observamos que as principais funções do protocolo foram implementadas na API Java, com destaque para a seleção dos melhores *downloaders* e *uploaders*, concomitantemente à utilização das operações de *choking* e *unchoking*. No entanto, a análise dos quatro melhores *downloaders* e *uploaders* ficou limitada ao pequeno número de nós disponíveis. Este fato é verificado durante o processo de *download* dos arquivos, no qual temos praticamente os mesmos nós envolvidos no processo de recebimento dos arquivos. Em uma escala maior, teríamos uma seleção de 50 nós com a manutenção da conexão com 20 a 40 nós [67].

Não observamos, como esperávamos, um ponto de intersecção das duas curvas. O tempo de distribuição dos arquivos com o protocolo sftp foi melhor do que o tempo do protocolo BitTorrent em todas as transferências. Esse comportamento deve-se ao fato do pequeno número de nós envolvidos no experimento, que impacta sobremaneira o aspecto colaborativo do protocolo BitTorrent. Além disso, a latência do BitTorrent é maior do que a latência do protocolo sftp, sendo esse fator compen-

sado pelo incremento de nós no sistema. Funcionalmente, as transferências dos arquivos foram realizadas integralmente, com todos os nós recebendo os arquivos.

Capítulo 8

Conclusões e Trabalhos Futuros

Neste trabalho, inicialmente realizamos um levantamento de algoritmos de escalonamento que poderiam ser usados no p2pBIOFOCO. Dessa forma, implementamos um destes algoritmos, o WQR, com o objetivo de analisar os benefícios oriundos da inclusão dos algoritmos de escalonamento no ambiente do p2pBIOFOCO. A implementação do WQR foi feita de forma que novos algoritmos de escalonamento possam ser facilmente incorporados ao sistema. Além disso, implementamos o algoritmo DP-RR para transferência eficiente de dados, substituindo o sftp anterior. Os resultados dos experimentos usando o WQR mostram a melhora do desempenho do p2pBIOFOCO.

Observamos, também, que além do tamanho e da quantidade de máquinas e tarefas, um fator importante para o bom desempenho desses algoritmos é a heterogeneidade do ambiente. Na execução dos experimentos com o algoritmo WQR, o ambiente é composto por um número limitado de recursos (10 máquinas), com recursos heterogêneos em parte do ambiente (50% das máquinas) e tarefas praticamente homogêneas em tamanho, 400 *bytes* a 4K *bytes*. Essas características reduzem a perspectiva de um bom desempenho para o processo de escalonamento, pois uma das maiores vantagens desses escalonadores é a atribuição de tarefas de alto custo computacional para recursos com grande capacidade de processamento, sem penalizar os clientes que possuem tarefas menores. No entanto, mesmo com essas limitações, pudemos concluir que a utilização do algoritmo melhorou o desempenho do p2pBIOFOCO, reduzindo o tempo de execução das tarefas conforme análises realizadas.

Na execução do algoritmo WQR, a redução do tempo total de execução das tarefas não foi expressiva com relação ao algoritmo original WQ. No entanto, a pequena diferença apresentada acentuar-se-á se replicarmos as tarefas para máquinas mais rápidas em ambientes mais heterogêneos e com maior número de máquinas, que é

o caso de aplicações de bioinformática em geral. Os algoritmos de escalonamento selecionados para o p2pBIOFOCO têm sua eficiência maximizada quando temos uma heterogeneidade de tarefas e de máquinas. Desse modo, o *parser* existente no p2pBIOFOCO deve ser modificado para a divisão do arquivo FASTA em arquivos de entrada de diversos tamanhos, produzindo tarefas heterogêneas resultantes de uma fragmentação mais adequada aos requisitos dos escalonadores.

Nos experimentos com os protocolos BitTorrent e sftp para a transferência de dados, os resultados indicam que: (i) para ambientes com número pequeno de nós, como o usado nos experimentos deste trabalho, a utilização do sftp é suficiente para a transferência dos arquivos; (ii) à medida em que o sistema tem o número de nós incrementado, a utilização do protocolo BitTorrent torna-se a escolha mais apropriada, favorecendo o uso do método DP-RR.

8.1 Contribuições

- Incluir escalonamento de tarefas no p2pBIOFOCO de forma flexível de modo que novos métodos de escalonamento sejam facilmente incorporados. Particularmente, o método WQR foi implementado nesta dissertação.
- Incluir mecanismo de transferência eficiente de dados no p2pBIOFOCO, de modo que grandes volumes de informação, típicas de aplicações de Bioinformática, não degradem o desempenho do sistema. Neste trabalho, implementamos o método DP-RR, que inclui os protocolos sftp e BitTorrent.

8.2 Trabalhos Futuros

Para novas perspectivas dentro do p2pBIOFOCO, vislumbramos a integração e a utilização desse sistema como um serviço em uma arquitetura em nuvem (*cloud computing*), paralela à implementação dos outros algoritmos discutidos neste trabalho. Os serviços de Bioinformática seriam submetidos à nuvem, que resolveria a execução do serviço solicitado, neste caso, em nós do p2pBIOFOCO. Neste caso, o sistema *peer-to-peer* poderia ser enquadrado como um SaaS (*Service as a Service*), comsumindo as sequências nos nós integrantes da infraestrutura da nuvem. Com a implementação dos algoritmos, poderíamos resolver dinamicamente qual deles seria o mais adaptado a determinado tipo de serviço submetido ao p2pBIOFOCO, com base em banco de dados estatístico gravado em sua base.

Outra possibilidade a ser explorada seria a implementação de aplicações baseadas em *stream* com a partição do banco de dados na memória das máquinas

integrantes do p2pBIOFOCO e o envio das sequências ao sistema, fluindo entre as máquinas e comparando-as às sequências em memória. Em Cherniack et al [25], temos a descrição de uma arquitetura que discute esse tipo de abordagem para processamento de dados como alternativa aos tradicionais DBMSs (Database Management Systems). Para os idealizadores desse *framework*, o armazenamento de dados e o processamento devem estar no mesmo sistema. Essa perspectiva incrementaria bastante execução de aplicações como o BLAST, consumidoras intensivas de CPU (*cpu bound*), com o processamento de suas consultas em memória, incluindo o p2pBIOFOCO como uma boa opção de sistema distribuído para a execução de aplicações de Bioinformática.

Referências

[1] D. Abramson, J. Giddy, and L. Kotler. High performance parametric modeling with nimrod/g: Killer application for the global grid? In *IPDPS '00: Proceedings of the 14th International Symposium on Parallel and Distributed Processing*, page 520, Washington, DC, USA, 2000. IEEE Computer Society. 108

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990. 3

[3] P. A. Alvarez. Pipeline para tanscritomas obtidos por sequenciadores de alto desempenho, 2009. Monografia apresentada como requisito parcial para a conclusão do Curso de Bacharelado em Ciência da Computação. 20, 32

[4] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg. Ourgrid: An approach to easily assemble grids with equitable resource sharing. In *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, Seattle, WA, USA, June 2003. 70

[5] R. Armstrong, D. Hensgen, and T. Kidd. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *In 7th IEEE Heterogeneous Computing Workshop HCW 98*, pages 79–87, 1998. 65

[6] M. Arora, S. K. Das, and R. Biswas. A de-centralized scheduling and load balancing algorithm for heterogeneous grid environments. *Parallel Processing Workshops, International Conference on*, 0:499, 2002. 49, 65

[7] D. A. Bader, Y. Li, T. Li, and V. Sachdeva. Bioperf: A benchmark suite to evaluate high-performance computer architecture on bioinformatics applications. In *Proc. IEEE Int'l Symp. Workload Characterization (IISWC 05)*, pages 163–173, 2005. 24

[8] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and E. W. Sayers. Genbank. *Nucleic acids research*, 37(Database issue), January 2009. 19, 23

[9] A. Binzenhöfer, K. Tutschku, B. Graben, M. Fiedler, and P. Arlos. A p2p-based framework for distributed network management. In *EuroNGI Workshop*, pages 198–210, 2005. 83

[10] BIOFOCO. Projeto BIOFOCO. Web Site, 2009. http://www.biofoco.org/. 3

[11] BIOFOCO. Projeto BIOFOCOIII. Web Site, 2009. http://www.biofoco.org/biofoco3. 3

[12] A. Boukerche and Alba C. M. A. Melo. Computational molecular biology. In *Parallel Computing for Bioinformatics and Computational Biology*, pages 149–166. Wiley Interscience - John Wiley and Sons, 2006. 24, 28

[13] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. Parallel Distrib. Comput.*, 61(6):810–837, 2001. 63, 67

[14] L. D. Briceno, M. Oltikar, H. J. Siegel, and A. A. Maciejewski. Study of an iterative technique to minimize completion times of non-makespan machines. In *IPDPS*, pages 1–14, 2007. 69

[15] C. Briquet, X. Dalem, S. Jodogne, and P.-A. Marneffe. Scheduling data-intensive bags of tasks in p2p grids with bittorrent-enabled data distribution. In *UPGRADE '07: Proceedings of the second workshop on Use of P2P, GRID and agents for the development of content networks*, pages 39–48, New York, NY, USA, 2007. ACM. 62, 65, 69

[16] C. Briquet and P.-A. Manefe. Description of a Lightweight Bartering Grid Architecture. In *Cracow Grid Workshop*, 2006. 65, 110

[17] P. Brucker. *Scheduling Algorithms*. Springer-Verlag Heidelberg New York, fifth edition, march 2007. 36, 42

[18] Science Buddies. Web Site. http://www.sciencebuddies.org/science-fair-projects. 22

[19] R. Buyya, M. Murshed, D. Abramson, and S. Venugopal. Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost-time optimization algorithm. *Software-Practice and Experience*, 35(5):491–512, 2005. 108

[20] H. Casanova, A. Legrand, and M. Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *UKSIM '08: Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, pages 126–131, Washington, DC, USA, 2008. IEEE Computer Society. 51

[21] H. Casanova, D. Zagorodnov, F. Berman, and A. Legrand. Heuristics for scheduling parameter sweep applications in grid environments. In *HCW '00: Proceedings of the 9th Heterogeneous Computing Workshop*, page 349, Washington, DC, USA, 2000. IEEE Computer Society. 52

[22] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, 1988. 36, 42, 43, 47, 66

[23] Dolan DNA Learning Center. Biology animation library - cycle sequencing. Web Site. http://www.dnalc.org/resources/animations/cycseq.html. 19

[24] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, Tushar Ch, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *In Proceedings of the 7th Conference on USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, pages 205–218, 2006. 1

[25] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. In *CIDR 2003 - First Biennial Conference on Innovative Data Systems Research*, California, January 2003. 125

[26] B. Cohen. Incentives Build Robustness in BitTorrent. In *Proc. of the Workshop on Economics of Peer-to-Peer Systems*, may 2003. 108

[27] J. Cohen. Bioinformatics - an introduction for computer scientists. *ACM Comput. Surv.*, 36(2):122–158, 2004. 2, 5

[28] Brazilian National Genome Project Consortium, A. T. R. Vasconcelos, and co-authors. The complete genome sequence of *Chromobacterium violaceum* reveals remarkable and exploitable bacterial adaptability. *Proc. Nat. Acad. Sci*, 100(20):11660–11665, September 2003. 23

[29] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling*. Addison-Wesley Publishing Co., 1967. 34, 35, 40

[30] L. F. Costa. Bioinformatics: perspectives for the future. *GMR - Genetics and Molecular Research*, 4(3):9–19, 2004. 23

[31] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems - Concepts and Design*, volume 1. Addison Wesley, 2005. 1, 84

[32] F. Crick. On protein synthesis. *The Symposia of the Society for Experimental Biology*, 36(12):138–163, 1958. 14

[33] D. Crockford. Json: The fat-free alternative to xml. Web Site, 2006. http://www.json.org/fatfree.html. 94

[34] K. Czajkowski, I. Foster, and C. Kesselman. Resource co-allocation in computational grids. In *In Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8*, pages 219–228, 1999. 1

[35] Universidade de Brasília, Universidade Federal de Goiás, and Universiade Federal de Mato Grosso do Sul e Universidade Federal de Mato Grosso. Projeto Pb. Web Site, 2010. https://helix.biomol.unb.br/Pb/. 2

[36] Embrapa Gado de Corte, UFMS, UFMT, UEMS, UCDB, UNIDERP, UnB, UFG, and IBMP. Projeto genoma anaplasma. Web Site, 2010. https://helix.biomol.unb.br/anaplasma/servlet/IndexServlet?body=bodyHtml. 2

[37] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *COMMUNICATIONS OF THE ACM*, 51(1):107–113, 2008. 1

[38] K. Dörnemann, J. Prenzer, and B. Freisleben. A peer-to-peer meta-scheduler for service-oriented grid environments. In *GridNets '07: Proceedings of the first international conference on Networks for grid applications*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 54, 65

[39] D. Doval and D. O'Mahony. Overlay networks: A scalable alternative for p2p. In *IEEE Internet Computing*, volume 7, pages 79–82, 2003. 76

[40] B. Dubuis. Java bittorrent api. Web Site, 2005. http://sourceforge.net/projects/bitext/. 111

[41] Embrapa/CPAA, INPA, UFAM, UFPA, CEPEM(RO), UFMA, UFRR, UFTO, UFAC, and UNIFAP. Projeto genoma funcional e genética genômica de paullinia cupana (guaranazeiro). Web Site, 2010. https://helix.biomol.unb.br/GR/. 2

[42] B. Ewing, L. Hillier, M. C. Wendl, and P. Green. Base-calling of automated sequencer traces using phred. *Genome Research*, 8:175–185, 1998. 21

[43] J. A. Falkner, J. A. Hill, and P. C. Andrews. Proteomics fasta archive and reference resource. *Proteomics*, 8(9):1756–1757, May 2008. 22

[44] M. S. S. Felipe and Pb Genome Network. Transcriptome characterization of the dimorphic and pathogenic fungus *Paracoccidioides brasiliensis* by EST analysis. *Yeast*, 20(3):263–271, February 2003. 23

[45] D. F. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evl.*, 25(4):351–360, 1987. 28

[46] R. Fielding. Architectural styles and the design of network-based software architectures. Phd thesis, University of California, Irvine, California, 2000. 84

[47] Folding. Folding@home distributed computing. Web Site, 2009. http://folding.stanford.edu/. 1

[48] R. Fonseca and A. Simões. Alternativas ao XML: YAML e JSON. In José Carlos Ramalho, João Correia Lopes, and Luís Carríço, editors, *XATA 2007 — 5ª Conferência Nacional em XML, Aplicações e Tecnologias Aplicadas*, pages 33–46, February 2007. 78, 87, 94, 95

[49] NCBI National Center for Biotechnology Information. http://www.ncbi.nlm.nih.gov/genbank/genbankstats.html. 19

[50] NCBI National Center for Biotechnology Information http://www.ncbi.nlm.nih.gov. NCBI - National Center for Biotechnology Information - http://www.ncbi.nlm.nih.gov. 18

[51] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *In 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, pages 118–128, 2003. 54

[52] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997. 55

[53] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation, 1999. 1

[54] I. Foster, C. Kesselmann, and S. Tuecke. The anatomy of the grid: Enable scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3):200–222, Fall 2001. 1, 54

[55] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. 43, 45, 46

[56] S. Ghanbari and M. R. Meybodi. On-line mapping algorithms in highly heterogeneous computational grids: A learning automata approach. In *International Conference on Information and Knowledge Technology (IKT '05)*, May 2005. 67

[57] C. Gondro and B. P. Kinghorn. A simple genetic algorithm for multiple sequence alignment. *Genetics and Molecular Research*, 6(4):964–982, 2007. 28

[58] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Ronnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications*, pages 287–326, 1979. 39

[59] P. Green and B. Ewing. User's guide. Web Site. http://bldg6.arsusda.gov/mtucker/Public/Consed/phd2fasta.html. 21

[60] Network Working Group. File transfer protocol (ftp). Web Site, 1994. http://www.w3.org/Protocols/rfc959/. 58

[61] Network Working Group. The application/json media type for javascript object notation (json). Web Site, 2006. http://tools.ietf.org/html/rfc4627. 94

[62] A. Iamnitchi and I. Foster. On fully decentralized resource discovery in grid environments. In *In International Workshop on Grid Computing*, pages 51–62, 2001. 1

[63] E. Ilavarasan and P. Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer Science*, 3(2):94–103, 2007. 53, 65

[64] A. Iosup, O. Sonmez, S. Anoep, and D. Epema. The performance of bags-of-tasks in large-scale distributed systems. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 97–108, New York, NY, USA, 2008. ACM. 60, 65

[65] A. Iosup, O. Sonmez, and D. Epema. Dgsim: Comparing grid resource management architectures through trace-based simulation. In *Euro-Par '08: Proceedings of the 14th international Euro-Par conference on Parallel Processing*, pages 13–25, Berlin, Heidelberg, 2008. Springer-Verlag. 65

[66] H. Izakian, A. Abraham, and V. Snasel. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. In *CSO '09: Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization*, pages 8–12, Washington, DC, USA, 2009. IEEE Computer Society. 63, 65, 67

[67] M. Izal, G. Uroy-Keller, E. W. Biersack, P. A. Felber, A. Al Hamra, and L. Garces-Erice. Dissecting bittorrent: Five months in torrent's lifetime. In *Passive And Active Network Measurement: 5th International Workshop, PAM 2004*, pages 1–11, 2004. 121

[68] H. A. James, K. A. Hawick, and P. D. Coddington. Scheduling independent tasks on metacomputing systems. In *Parallel and Distributed Computing Systems*, 1999. 55, 65

[69] K. Kato. Impact of the next generation dna sequencers. *International journal of clinical and experimental medicine*, 2(2):193–202, 2009. 30, 31, 32

[70] T. Keane, R. Allen, T. J. Naughton, J. Mclnerney, and J. Waldron. Distributed java platform with programmable mimd capabilities. In *Scientific Engineering for Distributed Java Applications*, pages 122–131. Springer, 2003. 65

[71] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C.-L. Wang. Heterogeneous computing: Challenges and opportunities. *IEEE Computer Magazine*, 26(6):18–27, 1993. 57

[72] E. S. Lander and co-authors. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, February 2001. 23

[73] A. M. Lesk. *Introduction to Bioinformatics*. Oxford University Press, 2002. 23

[74] D. J. Lipman, S. F. Altschul, and J. D. Kececioglu. A tool for multiple sequence alignment. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 86, pages 4412–4415, 1989. 28

[75] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and evaluation of a resource selection framework for grid applications. In *Design and Evaluation of a Resource Selection Framework for Grid Applications*, pages 63–72, 2002. 1

[76] N. M. Luscombe, D. GreenBaum, and M. Gerstein. What is bioinformatics? an introduction and overview. In *Methods of Information in Medicine 40*, pages 346–358, 2001. review. 23

[77] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F.Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *J. Parallel Distrib. Comput.*, 59(2):107–131, 1999. 64, 68

[78] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. *Peer-To-Peer Systems: First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002*, 2002. 81

[79] D. A. Menascé, D. Saha, S. C. Silva Porto, V. A. F. Almeida, and S. K. Tripathi. Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. *J. Parallel Distrib. Comput.*, 28(1):1–18, 1995. 52

[80] Inc. Sun Microsystems. Jxta java standard edition v2.5: Programmer's guide. white paper, September 2007. 76

[81] D. S. Milojicic and et al. Peer-to-Peer Computing. Technical report, HP Laboratories, Palo Alto, CA, US, March 2002. 1

[82] D. W. Mount. *Sequence and Genome Analysis*. Cold Spring Harbor, 2005. 19

[83] C. Notredame. Recent evolutions of multiple sequence alignment algorithms. *PLoS Comput Biol*, 3(8):e123, 08 2007. 27

[84] A. J. Page, T. M. Keane, and T. J. Naughton. Scheduling in a dynamic heterogeneous distributed system using estimation error. *J. Parallel Distrib. Comput.*, 68(11):1452–1462, 2008. 57, 65

[85] C. Phillips, C. Stein, and J. Wein. Task scheduling in networks. *SIAM J. Discret. Math.*, 10(4):573–598, 1997. 48

[86] Y. Qiao, F. E. Bustamante, P. A. Dinda, S. Birrer, and D. Lu. Improving peer-to-peer performance through server-side scheduling. *ACM Trans. Comput. Syst.*, 26(4):1–30, 2008. 62, 65

[87] S. Raman and S. McCanne. A model, analysis, and protocol framework for soft state-based communication. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 15–25, New York, NY, USA, 1999. ACM. 49

[88] E. O. Ribeiro. p2pbiofoco: Um framework peer-to-peer para processamento distribuído do blast. Master degree thesis, Universidade de Brasília, Brasilia, Distrito Federal, Brasil, 2006. 74, 81, 115

[89] E. O. Ribeiro, M. E. M. T. Walter, M. M. Costa, R. Togawa, and G. Pappas. p2pbiofoco: Proposing a peer-to-peer system for distributed blast execution. In *HPCC '08: Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, pages 594–601, Washington, DC, USA, 2008. IEEE Computer Society. 81

[90] M. Ripeanu, A. Iamnitchi, and I. Foster. Mapping the gnutella network. *IEEE Internet Computing*, 6(1):50–57, 2002. 65

[91] M. Ronaghi, M. Uhln, and P. Nyrn. Dna sequencing: A sequencing method based on real-time pyrophosphate. *Science*, 281(5375):363–365, July 1998. 31

[92] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag. 65

[93] C. Saccone and G. Pesole. *Handbook of Comparative Genomics*. John Wiley and Sons, 2003. 18

[94] S. Sahni. *Rounding, Interval Partitioning and Separation*, chapter 10, page 1. Chapman and Hall - Taylor and Francis Group (CRC Press), 2007. 59

[95] S. K. Sahni. Algorithms for scheduling independent tasks. *J. ACM*, 23(1):116–127, 1976. 59, 66

[96] E. Santos-Neto, W. Cirne, F. Brasileiro, A. Lima, and R. Lima. Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 210–232, 2004. 70

[97] J. C. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing Company, Boston, 1997. 5, 7, 10, 21, 27

[98] M. Shirts and V. Pande. Screensavers of the world, unite! *Science*, 290:1903–1904, 2000. 1

[99] D. P. Silva, W. Cirne, and F. V. Brasileiro. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In *Euro-Par*, pages 169–180, 2003. 49, 51, 65, 70, 100, 106

[100] A. J. G. Simpson and co-authors. The genome sequence of the plant pathogen *Xylella fastidiosa*. *Nature*, 406(6792):151–157, 2000. 23

[101] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981. 26

[102] I. Stoica and et al. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM. v, vi, 1

[103] H. Tada, M. Imase, and M. Murata. On the robustness of the soft state for task scheduling in large-scale distributed computing environment. In *International Multiconference on Computer Science and Information Technology*, pages 475–480, Wisla, Poland, october 2008. 49, 65, 102

[104] J. R. ten Bosch and W. W. Grody. Keeping up with the next generation: Massively parallel sequencing in clinical diagnostics. *The Journal of Molecular Diagnostics*, 10(6):484–492, November 2008. 32

[105] V. T'kindt and J.-C. Billaut. *Multicriteria Scheduling - Theory, Models and Algorithms*. Springer, Berlin, second edition, 2006. 36

[106] J. C. Venter and co-authors. The sequence of the human genome. *Science*, 291:1304–1351, February 2001. 23

[107] W3C. Xml schema. Web Site, 2010. http://www.w3.org/XML/Schema.html. 95

[108] Y.-T. Wang and R. J. T. Morris. Load sharing in distributed systems. *IEEE Trans. Comput.*, 34(3):204–217, 1985. 56

[109] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171:737–738, 1953. 10

[110] B. Wei, G. Fedak, and F. Cappello. Scheduling independent tasks sharing large data distributed with bittorrent. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 219–226, Washington, DC, USA, 2005. IEEE Computer Society. 58, 65, 107, 109