UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Jakelyne Machado Lima

ALGORITMO DE ESCALONAMENTO DE GRADES COMPUTACIONAIS PARA MINIMIZAR O PROCESSAMENTO CENTRALIZADO NA MONTAGEM DE GENOMAS BACTERIANOS OBTIDOS POR SEQUENCIAMENTO DE DNA DE NOVA GERAÇÃO (NGS).

Belém
2012

Jakelyne Machado Lima

ALGORITMO DE ESCALONAMENTO DE GRADES COMPUTACIONAIS PARA MINIMIZAR O PROCESSAMENTO CENTRALIZADO NA MONTAGEM DE GENOMAS BACTERIANOS OBTIDOS POR SEQUENCIAMENTO DE DNA DE NOVA GERAÇÃO (NGS).

Dissertação de Mestrado apresentada para obtenção do grau de Mestre em Ciência da Computação pelo programa de Pós-Graduação em Ciência da Computação. Instituto de Ciências Exatas e Naturais.
Universidade Federal do Pará.
Área de concentração: Redes de Computadores.
Orientador: Prof. Dr. Antônio Jorge Gomes Abelém.

Belém
2012

ALGORITMO DE ESCALONAMENTO DE GRADES COMPUTACIONAIS PARA MINIMIZAR O PROCESSAMENTO CENTRALIZADO NA MONTAGEM DE GENOMAS BACTERIANOS OBTIDOS POR SEQUENCIAMENTO DE DNA DE NOVA GERAÇÃO (NGS).

Dissertação apresentada para obtenção do grau de Mestre em Ciência da Computação. Programa de Pós-Graduação em Ciência da Computação. Instituto de Ciências Exatas e Naturais. Universidade Federal do Pará.

Banca Examinadora

_____

Prof. Dr. Antônio Jorge Gomes Abelém
Faculdade de Computação – UFPA – Orientador

_____

Prof. Dr. Eduardo Coelho Cerqueira
Faculdade de Engenharia de Computação – UFPA - Membro

_____

Prof. Dr. Artur Luiz da Costa Silva
Instituto de Ciências Biológicas – UFPA - Membro

A Guilherme Damasceno Silva, meu porto seguro e amor da minha vida.

AGRADECIMENTOS

Á Deus sempre em primeiro lugar, por estar vencendo mais esta etapa de minha vida, apesar dos obstáculos que encontrei, graças a ele consegui superar.

Aos meus pais, e minha irmã pelo incentivo e amor incondicional.

A todos os colegas do GERCOM que me apoiaram em todos os momentos.

A todos os colegas da Bioinformática, em especial à amiga Louise Cerdeira por todo apoio e paciência.

Ao meu orientador um agradecimento especial, pela confiança e apoio nos momentos mais difíceis.

A CAPES e FAPESPA pelo apoio financeiro.

“Quanto mais informação, mais responsabilidade. Dádiva ou maldição?”

(Autor Desconhecido)

“Se as pessoas soubessem quão duramente eu trabalhei para obter o meu domínio, não pareceria tão maravilhosa depois de tudo.”

Michelangelo

RESUMO

Com o avanço das técnicas de sequenciamento do genoma, um grande volume de dados passou a ser gerado e, em decorrência desses avanços, vários desafios computacionais começaram a surgir, dentre eles a disponibilização de novos *softwares* distribuídos e adequados para a manipulação de uma grande quantidade de dados genômicos, além do armazenamento, disponibilização e gerenciamento desses dados. No contexto do mapeamento do DNA é possível identificar que a etapa de montagem, é considerada como um dos pontos que exige a utilização de um grande poder computacional, pois nesta etapa utiliza-se grande quantidade de dados e a utilização de *softwares* específicos para concluir o processo é de extrema importância. Sem a utilização de *softwares* específicos o processo, como um todo, acaba sendo afetado pela demora e pela grande centralização de dados. O presente trabalho tem como principal objetivo propor o algoritmo de escalonamento *scheduler* desenvolvido para otimizar a utilização do software de montagem *ABySS*, para uso no contexto de grades computacionais, objetivando minimizar o tempo e a centralização dos dados na etapa de reconstrução do genoma. Os testes com o software distribuído *ABySS* e com o escalonador s*heduler* foram realizados no simulador *SimGrid*, que gera simulações de aplicações distribuídas em ambientes heterogêneos para verificar a viabilidade, flexibilidade e escalabilidade para o processo de montagem com dados de *corynebacterium* de mapeamentos já realizados pela Universidade Federal do Pará.

PALAVRAS-CHAVE: Mapeamento do DNA, Reconstrução do Genoma, Grades Computacionais, *ABySS*, Algoritmo de Escalonamento, Sequenciamento, *Corynebacterium*, *SimGrid*.

ABSTRACT

With the progress of the genome sequential techniques a great volume of data became generated, and in consequence of those progresses, several computational challenges began to appear, among them the viability of new *software* distributed and adequate for the manipulation of a great amount of genomic data, besides the storage, disponibilization and administration of those data. In the context of the DNA mapping, it is possible to identify that the assembly, stage is considered as one of the points that demands the use of a great computational power, since the coming files of the sequential stage possess great amounts of data, as it is necessary to analyze countless times the same data sequence, the use of specific software to conclude the process is of extreme importance, without the use of specific software, the process as a whole ends being affected by the delay and for the great centralization of data. This work has as main objective to describe the task *scheduling* algorithm designed to optimize, the use of *software* assembly *ABySS*, the concepts of grid computing, aiming to minimize the time and the centralization of data in step reconstruction of the genome. The tests with the software distributed with the ABySS and scheduler developed SimGrid, were performed in the simulator, which generates simulations of distributed applications in heterogeneous environments to verify the feasibility, flexibility and scalability to the assembly process with corynebacterium data mappings already made by the University Federal do Pará.

KEYWORDS: DNA Mapping, Assembly of DNA, Computational Grids, ABySS, Scheduling Algorithm, Sequential, Corynebacterium, SimGrid.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ABySS	Assembly By Short Sequences
BOT	Bag of Tasks
COBOL	Common Business Oriented Language
CPU	Central Processing Unit
CT	Completion Time
DNA	Ácido Desoxirribonucleico
FPLTF	Fastest Processor to Largest Task First
LIBTS	Library Tasks Sheduling
MB	MegaByte
MSG	Meta-Simgrid
NGS	Next Generation Sequencing
TBA	Time to Become Available
UFPA	Universidade Federal do Pará
WQ	Worqueue
WQR	Worqueue with Replication

SUMÁRIO

1 **INTRODUÇÃO**

1.1 MOTIVAÇÃO

O aumento da disponibilidade de computadores com grande poder computacional interligados a redes de alta velocidade, tem possibilitado a agregação de recursos dispersos geograficamente para executar aplicações em grande escala. Essa agregação de recursos é denominada de computação em grade (Foster, 2002).

A computação em grade (*grid computing*) é um modelo que propõe a utilização de recursos computacionais de diversas máquinas situadas em localizações diversas, inclusive em continentes separados para resolver problemas que exigem grande poder computacional (Foster, 2003).

A computação em grade vem se destacando em cenários onde existe a necessidade de se utilizar aplicações com requisitos de desempenho cada vez maiores, como a mineração de dados, previsão do tempo, biologia computacional, processamento de imagens médicas, entre outras (Berman et al., 2003).

As técnicas de sequenciamento do genoma estão avançando de forma considerável e gerando grandes quantidades de dados que precisam ser tratados utilizando recursos com grande poder computacional sendo assim, os recursos de grades computacionais podem ser utilizados para solucionar essa demanda por execução de aplicações paralelas ou distribuídas com alto poder computacional (Morozova and Marra, 2008).

A etapa de montagem do genoma ainda possui algumas limitações, pois os dados gerados pelos sequenciadores de nova geração possuem características que precisam ser levadas em consideração, como o tamanho das leituras e o volume de dados, o que aumenta de forma exponencial o tempo de processamento, podendo às vezes, inviabilizar a montagem, portanto, faz-se necessário implementar técnicas objetivando minimizar essas limitações.

Como a computação em grade utiliza recursos heterogêneos, faz-se necessário a utilização do escalonamento de tarefas, objetivando determinar o tempo de início das tarefas, a localização dos recursos e outras informações importantes. A utilização de algoritmo eficiente para gerir os recursos é fator crucial para diminuir o tempo do término das tarefas (Bittencourt et al., 2006).

1.2 ORGANIZAÇÃO

Além deste capitulo introdutório, a dissertação está estruturada em sete capítulos que discutem as principais questões referentes à bioinformática, grades computacionais e como esses dois termos se relacionam, enfatizando as técnicas de sequenciamento de genoma e os algoritmos de escalonamento utilizados para agilizar o processo.

O Capítulo 2 aborda as principais questões referentes à bioinformática, incluindo um breve histórico, tratamento de dados, uma explanação sobre a etapa de reconstrução do genoma, as técnicas utilizadas e a finalização da etapa de montagem.

No Capítulo 3 é apresentada uma introdução sobre grades computacionais, um breve histórico, suas principais características, os desafios encontrados na área e sua contribuição para a bioinformática.

O Capítulo 4 apresenta os trabalhos relacionados ao escalonamento de tarefas, e a montagem de genomas distribuídos incluindo os desafios para a implementação e as principais soluções.

O Capítulo 5 apresenta o modelo de algoritmo de escalonamento proposto, sendo descrita sua implementação em conjunto com o *software* de montagem *ABySS.*

O Capítulo 6 descreve os cenários e experimentos realizados, além de apresentar os resultados e uma análise sobre estes.

Finalmente, o Capítulo 7 finaliza o trabalho com as contribuições, uma avaliação geral da proposta e sugestões de alternativas de trabalhos futuros.

1.3 OBJETIVOS

O objetivo principal desta dissertação é propor um algoritmo de escalonamento de tarefas, *Sheduler*, desenvolvido para otimizar a utilização do *software* de montagem *ABySS* para o uso no contexto de grades computacionais, objetivando minimizar o tempo e a centralização dos dados na etapa de reconstrução do genoma. O *software* utilizado para a montagem de genomas, o *ABySS*, trabalha de forma paralelizada, e a sua utilização tem como objetivo ajudar e facilitar o desempenho da etapa de reconstrução

do genoma e, por consequência, melhorar a eficiência do processo de mapeamento do DNA como um todo, utilizando em conjunto às técnicas de grades computacionais para melhorar o desempenho na execução de tarefas que requerem grande poder computacional, e aplicar essas técnicas *in loco* na Universidade Federal do Pará.

Desta forma, para alcançar o objetivo principal exposto, os seguintes objetivos específicos precisam ser superados:

- Analisar o funcionamento da etapa de montagem no mapeamento do DNA;
- Analisar o funcionamento do escalonamento de tarefas;
- Analisar o funcionamento do *software* de montagem;
- Investigar quais as propostas/ferramentas semelhantes já existentes;
- Definir métodos para implantação do escalonamento de tarefas;
- Definir métodos para a utilização dos conceitos de grades computacionais;
- Especificar e analisar os requisitos do algoritmo a ser implementado;
- Desenvolver o algoritmo de escalonamento;
- Agregar o escalonador ao *ABySS*;
- Validação com dados reais de montagens já realizadas pela UFPA, utilizando ferramentas de simulação em grades computacionais (*Simgrid*).

2. BIOINFORMÁTICA

A bioinformática desenvolve e aplica técnicas de computação, como mineração de dados e utilização de algoritmos com o objetivo de aumentar a compreensão dos processos biológicos. As principais ferramentas nesta área incluem alinhamento de sequências genômicas, procura e caracterização de genes, montagem de genoma, predição de estrutura de proteínas, predição de expressão gênica, interações proteína-proteína, entre outras. Com isso a bioinformática cobre a aquisição, processamento, armazenamento, distribuição, análise e interpretação de informações biológicas. (Gibas & Jambeck, 2001; Catanho & Miranda, 2007). Neste capitulo são tratadas as questões sobre o surgimento da bioinformática, o sequenciamento de nucleotídeos, como os dados são tratados e finalmente sobre a montagem do genoma.

2.1 O SURGIMENTO DA BIOINFORMÁTICA

Com o sequenciamento do genoma do vírus ΦX174 (Sanger et al., 1977) houve a necessidade do tratamento dos dados obtidos, não de forma manual, mas com a ajuda de computadores. Para tal, nove computadores foram conectados e a cada um foi dada uma porção única do genoma. Os pesquisadores envolvidos, movidos pela carência de automatização na montagem dos dados oriundos de tecnologias de sequenciamento, escreveram o primeiro programa na linguagem de programação COBOL que ajudou na compilação e análise dos dados das sequências de DNA (McCallum & Smith, 1977).

Contudo, o processo ainda despendia muito trabalho manual. Para auxiliar nas análises do genoma do vírus ΦX174, o cientista Roger Staden (1977) desenvolveu o primeiro pacote de programas interativos específicos para o uso genômico e focado em usuários com pouca ou nenhuma experiência com computadores. Esse programa desenvolvido (*staden package*) ainda é utilizado na atualidade (Staden et al., 2000).

Atualmente, o campo da bioinformática está em pleno desenvolvimento. A bioinformática envolve a união de diversas linhas de conhecimento: A engenharia de *softwares*, a matemática, a estatística, a ciência da computação e a biologia molecular (Lesk, 2008). Essa área é a parte central para a interpretação de dados genômicos e a geração de hipóteses a serem testadas em torno desses dados. Graças ao seu surgimento e ao progresso nesse campo, foi possível a criação de bancos de dados gerenciadores de informações genômicas, plataformas computacionais eficientes para a interpretação dos resultados obtidos e o sequenciamento dos primeiros genomas completos.

Através de dados gerados com *softwares* específicos, a bioinformática pôde ser aplicada em várias etapas do processo de mapeamento de DNA, inclusive na análise de diferentes genes e no diagnóstico sobre a constituição das proteínas.

2.2 SEQUENCIAMENTO DE NUCLEOTÍDEOS

A totalidade de DNA presente em uma célula é chamada de genoma. O genoma de uma célula pode encontrar-se dividido em moléculas fisicamente distintas. Dentro da genética moderna, o gene é uma sequência de DNA que codifica uma

proteína. O conjunto de genes comuns a todas as células da espécie é chamado de genoma central, enquanto que todo o DNA que é comum a todos os genomas da espécie é chamado de arcabouço da espécie, que não é fisicamente contínuo, mas sim, permeado de sequências variáveis (Chiapello et al., 2005).

A descoberta da estrutura tridimensional do DNA por Watson, Crick e Wilkins em 1953, apesar de ter marcado a biologia não outorgou, de imediato, a capacidade de "ler" a informação contida num organismo. A primeira técnica de sequenciamento manual foi desenvolvida apenas 20 anos mais tarde por Sanger e Coulson (Sanger & Coulson, 1975; Sanger et al., 1977).

Em 1986, o laboratório de Leroy Hood em Caltech, Califórnia, juntamente com a empresa *Applied Biosystems*, lançaram o primeiro sequenciador automático (Smith et al., 1986), que utilizava um computador para análises dos dados gerados.

Nesse período, houve um espantoso crescimento na capacidade e velocidade da geração de dados genômicos. As técnicas evoluíram significativamente desde a resolução de um pequeno genoma de um fago (5.386 pb) até o sequenciamento do genoma humano completo com aproximadamente 3 bilhões de pares de bases (Lander et al., 2001; Venter et al., 2001).

Foram desenvolvidas novas tecnologias de sequenciamento (segunda geração de sequenciadores) que tem promovido o avanço em diversas áreas da biologia moderna, dentre elas, a genômica. Além do que, a massiva quantidade de dados gerados, o aumento na acurácia dos dados, alta cobertura de genomas e rapidez na execução das técnicas vinculadas aos sequenciamento de próxima geração (sigla NGS, do inglês *Next Generation Sequencing*) permitiram a produção em larga escala de genomas completos (Ronaghi, 2001).

A redução significativa nos custos do sequenciamento foi decisiva para a implantação da tecnologia como o estado da arte em diversos centros de sequenciamento no mundo (Morozova & Marra, 2008). A comunidade científica nacional e internacional acompanha atentamente a publicação dos resultados produzidos por esses novos equipamentos, o lançamento das nanotecnologias de sequenciamento baseadas em análise *single molecule* lançadas em 2010, e a tecnologia de nano poro ainda se encontra em fase de desenvolvimento. Estas metodologias têm sido chamadas pela comunidade científica de sequenciamento de terceira geração (Imelfort et al., 2009).

2.3 TRATAMENTO DE DADOS

O pré-processamento de dados, composto por filtro de qualidade e correção de erros de sequenciamento, é essencial para aumentar a acurácia das montagens por prevenir que leituras incorretas ou com qualidade baixa façam parte do processo de montagem de genomas.

No caso de sequências obtidas a partir das plataformas de próxima geração de sequenciamento (NGS), apesar da elevada cobertura, deve-se avaliar a qualidade das bases, assim é possível trimar as leituras e aplicar filtros de qualidade. Para isso, temos scripts e ferramentas que realizam este tratamento de qualidade como, por exemplo, o Galaxy (Blankenberg et al., 2010) e ShortRead (Morgan et al., 2009).

Já os erros de sequenciamento podem causar problemas na montagem de genomas com abordagem *de novo*, pois a geração de *contigs* é altamente sensível a esses erros (Salmela, 2010; Schröder et al., 2009). Desta forma, para obter melhores resultados é preciso realizar a correção de erros antes da montagem de genomas, o que tornará os dados mais fidedignos (Schröder et al., 2009).

2.4 MONTAGEM DE GENOMAS

Uma vez geradas as leituras pelo sequenciador é necessário juntá-las de maneira lógica ou montá-las. Ao longo dos anos diversas ferramentas foram desenvolvidas para abordar esta questão, como por exemplo, os montadores Phrap (Green, 1994), Arachne (Batzoglou et al., 2002) e o Celera (Myers et al., 2000). Sem entrar nos detalhes de cada ferramenta ressalta-se que, compartilham entre si um paradigma comum, sendo referido como *overlap layout consensus* (Pevzner et al., 2001). Esta abordagem é bastante semelhante à usada para se resolver um quebra-cabeça (Figura1), como sendo descrito abaixo.

O primeiro passo consiste em alinhar duas a duas, de forma exaustiva, e estabelecer que os pares de leituras devem apresentar uma sobreposição consistente de uma leitura com a outra. Isso é análogo à procura de peças de um quebra-cabeça que se encaixam entre si e têm cores que combinam. A principal dificuldade nesta fase é distinguir as sobreposições inexatas devido a erros de sequenciamento e similaridades

dentro do genoma, tais como regiões de repetição altamente conservadas (Phillippy et al., 2008).

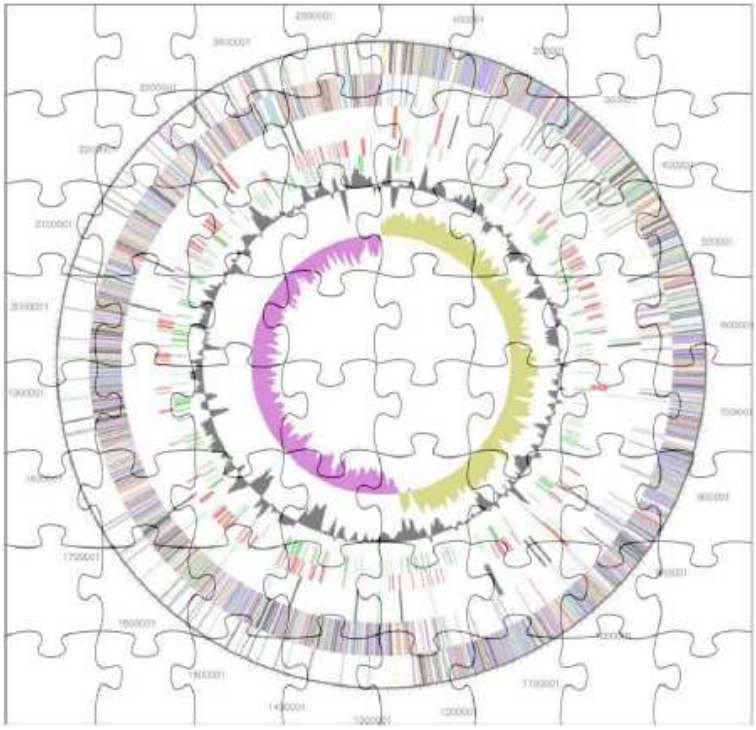

Figura 1- Analogia entre processo de montagem de genomas e de um quebra-cabeça. Fonte: Quebra – cabeça (http://contigs.files.wordpress.com/2010/04/assembly-puzzle1.jpg).

O alinhamento das sequências é uma área da bioinformática muito estudada, que consiste em fornecer o alinhamento ideal entre duas sequências em função de uma pontuação (*score*). A maioria dos métodos são baseados no algoritmo de Needleman-Wunsch (Needleman & Wunsch, 1970) que explora a dinâmica espacial dos possíveis alinhamentos entre as sequências. Muitas extensões já foram concebidas, como por exemplo, para alinhamentos múltiplos (Higgins & Sharp, 1988), alinhamentos locais (Smith & Waterman, 1981) ou pesquisas rápidas em grandes bases de dados (Altschul et al., 1990). Atualmente têm se buscado técnicas que possam ser executadas de forma muito mais rápida em processadores paralelos (Fagin et al., 1993) e em se tratando de leituras curtas geradas por plataformas de sequenciamento de segunda geração uma das soluções encontradas para a manipulação de milhares de sequências de forma simultânea é a utilização de grades computacionais (Bateman & Wood, 2009).

Os montadores detectam um agrupamento de leituras com alinhamentos consistentes entre suas extremidades (Figura 2) formando, com isso, sequências

contíguas (*contigs*). Isso seria equivalente à formação parcial de uma imagem com a união de peças em um quebra-cabeça. Em ambas as montagens, genoma e quebra-cabeça, o processo pode ser interrompido em regiões ambíguas, onde diversas continuações ou lacunas (*gaps*) são possíveis e onde nenhuma peça foi encontrada com conexão (Zerbino, 2009).

Finalmente o montador tenta ordenar e orientar os *contigs* uns com os outros de forma *de novo*, ou seja, sem o auxilio de uma sequência de referência. Retornando a metáfora do quebra-cabeça, corresponderia a identificar cantos e partes relativas de uma imagem com a outra. No final da fase de montagem, um conjunto de *contigs scaffolded* estarão disponíveis. No entanto, para fins práticos, é desejável remover a maioria das lacunas possíveis, convergindo eventualmente para um conjunto de cromossomos íntegros, ou seja, que não apresentem quebras. Esta fase, chamada de acabamento, pode ser muito cara e demorada, o que dependerá da estratégia utilizada para fechamentos das eventuais lacunas presentes no *scaffold* do genoma (Cole et al., 2008).

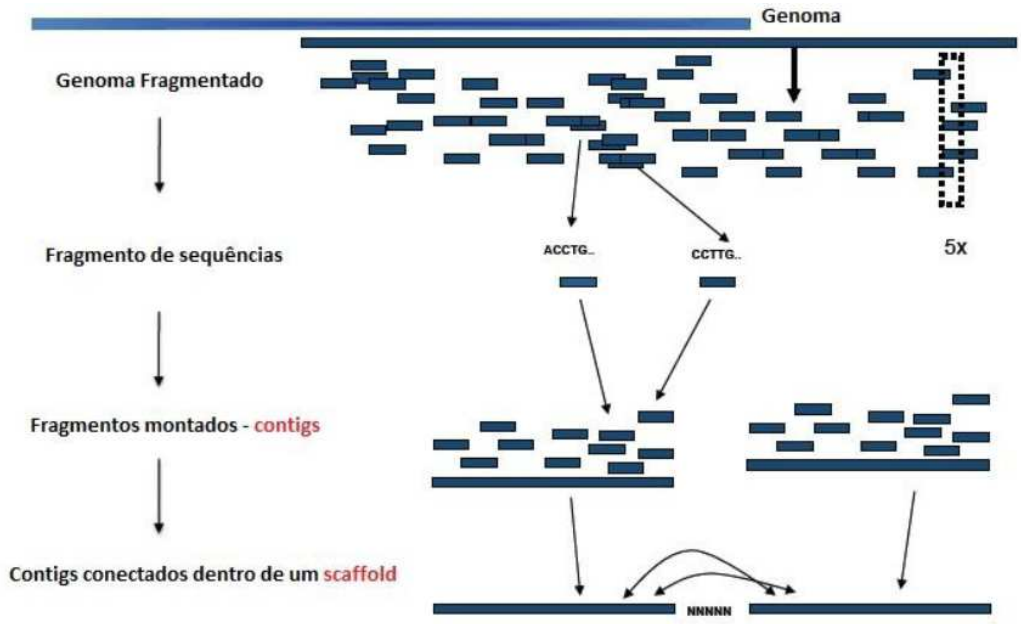

Figura 2 - Processo de montagem do genoma.

A montagem de genomas pode ser medida pelo tamanho e acurácia de seus *contigs* e *scaffolds*, além de indicadores estatísticos, com maior e menor sequência, e N50, que é o tamanho da menor sequência de um conjunto de *contigs* que contém o

menor número de *contigs* cujos tamanhos somados representam 50% da montagem (Miller et al., 2010).

2.4.1 Montagem por Referência

Basicamente, a montagem por referência consiste no mapeamento das leituras obtidas a partir do sequenciamento contra um genoma de referência (Figura 3), preferencialmente, de um organismo filogeneticamente próximo possibilitando assim o alinhamento de grande parte das leituras. Contudo, as configurações do alinhamento também influenciarão na quantidade de leituras utilizadas, desta forma, os parâmetros como profundidade de cobertura e quantidade de *mismatches* permitidos devem ser definidos com base em informações do sequenciamento: cobertura estimada e qualidade das bases (Li et al., 2010). O mapeamento utilizando a referência propicia a identificação de substituições de nucleotídeos, principalmente, com o uso de plataformas NGS em função da alta cobertura de sequenciamento (Miller et al., 2010).

Após a montagem é possível observar regiões do genoma de referência que não apresentaram cobertura, representando *gaps*, que podem ocorrer em função da presença de uma sequência de nucleotídeos na referência e que não existe no organismo sequenciado ou pelo fato da região não ter sido sequenciada.

Quanto a problemas na montagem com uso de referência pode-se citar a representação de regiões repetitivas, como por exemplo: caso o genoma de referência utilizado possua duas destas regiões e o organismo sequenciado apresente apenas uma, durante o mapeamento contra a referência, as duas regiões da referência serão cobertas, o que pode gerar erros de montagem (Figura 3).

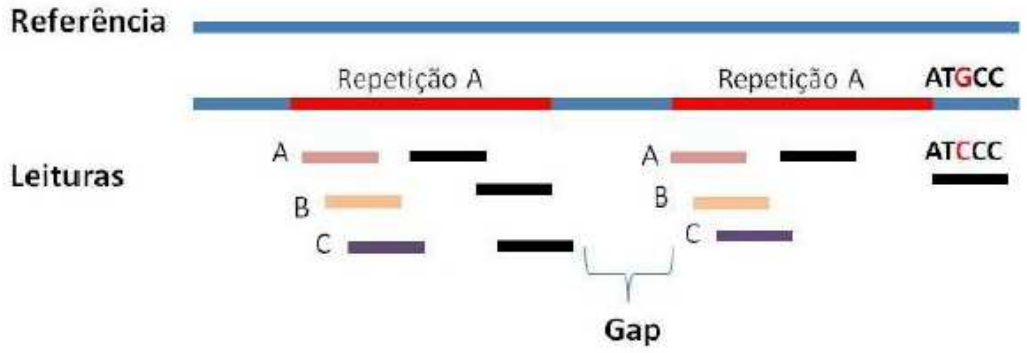

Figura 3 - Alinhamento das leituras contra um genoma de referência.

2.4.2 Montagem *de novo* ou *ab initio*

Consiste na reconstrução das sequências do genoma sem o auxílio de qualquer outra informação além das leituras geradas pelo processo de sequenciamento. Nesta estratégia pode ser realizado um alinhamento por similaridade entre as próprias leituras, ou através da sobreposição de *k-mers*. Possibilitando no final do alinhamento a formação de sequências contíguas (*contigs*). A maioria dos montadores NGS baseiam-se na Teoria dos grafos, onde seus vértices e arestas poderão representar uma sobreposição, um *k-mer* ou uma leitura variando de acordo com a estratégia utilizada. (Miller et al., 2010).

2.4.2.1 Dificuldade na montagem *de novo* de genomas

As limitações das abordagens de montagem *de novo* associam-se diretamente às limitações tecnológicas relacionadas às características dos dados gerados pelos sequenciadores de segunda geração, como o tamanho das leituras e o volume de dados gerados, que aumenta exponencialmente o tempo de processamento e, às vezes, inviabiliza a montagem. Dentro deste cenário vários problemas podem ocorrer como agrupamento de regiões repetidas, regiões onde o sequenciamento obteve baixa qualidade, compressão de base no sequenciamento, e até regiões com baixa cobertura devido ao caráter aleatório do sequenciamento (Ewing & Green, 1998).

Um dos exemplos clássicos de problemas de montagem *de novo*, é de encontrarmos um caminho em um grafo de sobreposições que visite cada vértice uma única vez (caminho Hamiltoniano) ou cada aresta uma única vez (caminho Euleriano), e que muitas vezes acarreta a perda da conectividade entre sequências muito distantes (Pevzner & Tang, 2001). Sendo estes problemas mais complexos e comuns em montagens com leituras curtas, pois a quantidade de leituras é maior, já o comprimento é bem menor, o que aumenta exponencialmente o tamanho do problema. (Pevzner et al., 2001).

2.5 FINALIZAÇÃO DA MONTAGEM DE GENOMAS

Ao final do processo de montagem um conjunto de *contigs scaffolded* estarão disponíveis. Entretanto, para fins práticos, é desejável a remoção da maior quantidade de *gaps* possíveis, eventualmente convergindo para uma montagem de cromossomos contínuos.

Esta etapa é chamada de finalização e pode ser cara e demorada, como demonstrado no desenvolvimento do Projeto do Genoma Humano. Anos depois do seu lançamento oficial (*The International Human Genome Sequencing Consortium*, 2001), o projeto ainda possui regiões repletas de *gap*s. (Cole et al., 2008).

3. GRADES COMPUTACIONAIS

As pesquisas nas mais diversas áreas têm exigido cada vez mais a necessidade de utilização de recursos de alto desempenho, principalmente, quando se trata da utilização de grandes quantidades de dados. O conceito de Grades Computacionais surgiu como uma importante área da computação distribuída pelo seu foco no compartilhamento de recursos em grande escala, aplicações inovadoras e de alto desempenho. Neste capítulo são tratadas as questões relacionadas às grades computacionais, suas origens, os desafios existentes na área e aplicações para a área de bioinformática.

3.1 HISTÓRICO

Na década de 90, pesquisadores no campo de computação paralela constataram que os computadores mais rápidos da época não estavam executando as tarefas com um bom desempenho, então surgiu à ideia de distribuir o processamento com o objetivo de aumentar a capacidade computacional.

O conceito de computação em grade (*grid computing*) foi apresentado em 1998, por Ian Foster e Carl Kasseman quando propuseram a grade computacional como uma grande infraestrutura de *software* e *hardware* que fornece um serviço confiável e

consistente, que permite acesso a grandes capacidades computacionais distribuídas geograficamente. Como mostra a Figura 4.

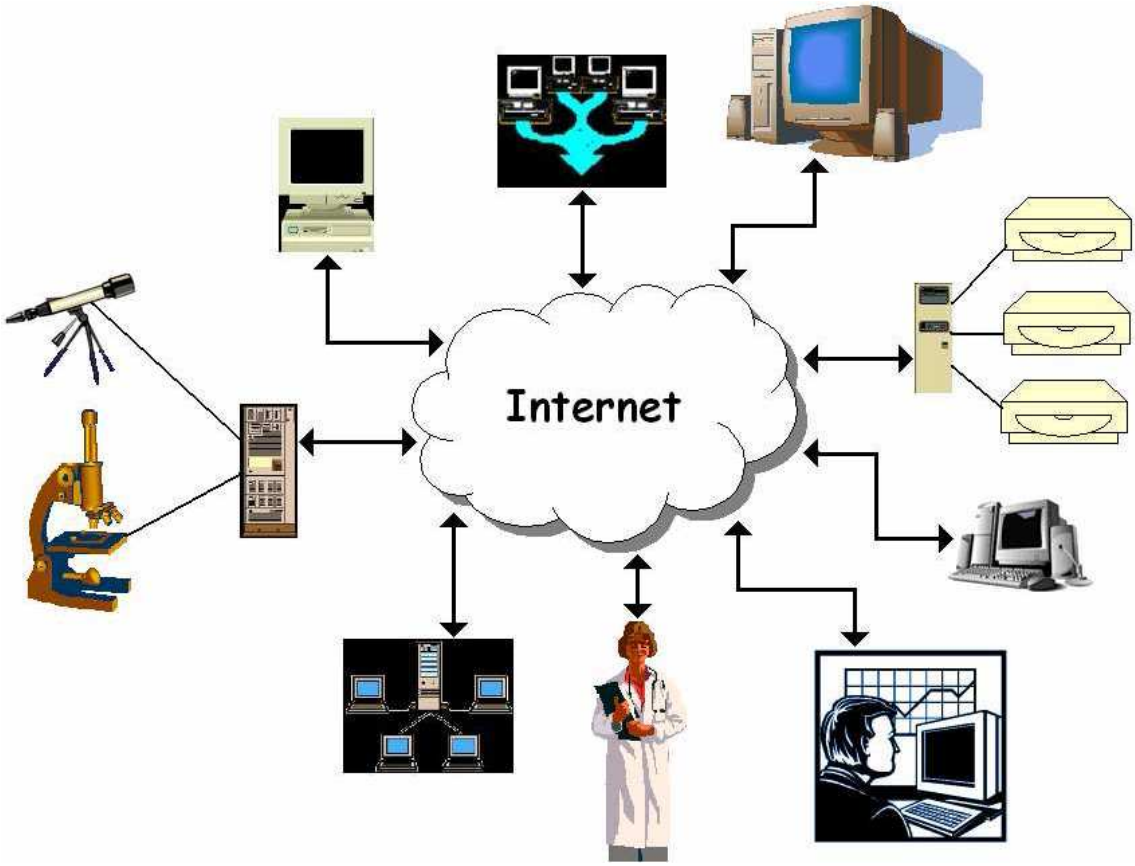

Figura 4 - Grid Computing

O nome Grid Computing, foi proposto baseado nas malhas de redes de energia elétrica (p*ower grids*), onde a utilização da energia é transparente para o usuário, ou seja, a energia é fornecida de forma eficiente adequando-se a demanda de cada usuário e sem saber de qual local ela foi gerada. (Foster, 2002).

3.2 DESAFIOS NAS GRADES COMPUTACIONAIS

A computação em grade é um modelo que propõe o uso de recursos computacionais de várias máquinas situadas em localizações diversas, inclusive em continentes separados, para resolver problemas que exigem grande poder computacional (Foster, 2002). As grades atualmente são compostas, além de computadores, por grandes repositórios de dados, equipamentos científicos controlados remotamente, dispositivos de visualização, sensores, entre outros, que oferecem possibilidades inéditas de cooperação entre os cientistas. Algumas das principais características dessa

arquitetura são: a abstração, a flexibilidade, a escalabilidade e a tolerância a falhas. Para se chegar nesse cenário de confiabilidade e eficiência, é necessário que o ambiente de grade leve em consideração as características envolvidas no uso dos recursos utilizados dinamicamente avaliando cada caso de acordo com a aplicação a ser utilizada. A Figura 6 ilustra o modelo de computação em grade.

Figura 5 - Relacionamento e Heterogeneidade dos Recursos da Grade Computacional

A natureza heterogênea dos recursos distribuídos e conectados a uma rede nem sempre segura são um dos grandes desafios encontrados na utilização das grades computacionais, além da necessidade de utilização de um escalonamento de tarefas cuidadoso que objetiva a melhoria e a eficiência da utilização dos recursos.

Esse modelo de computação distribuída em larga escala teve um crescimento significativo no momento em que a demanda por aplicações cientificas sofisticadas levou ao desenvolvimento e armazenamento dos sistemas de computação atuais. Além disso, atualmente existe a necessidade de ambientes que permitam a colaboração intensa entre diversos grupos de pesquisa espalhados geograficamente.

3.3 TIPOS DE GRIDS

Diversas nomenclaturas têm sido adotadas para classificar os tipos de *grid*s, as mais utilizadas são as definições de *grids* organizacionais e a definição de *grids* por

utilização de recursos. As principais definições e explicações para diferentes tipos de *grids* são apresentadas neste tópico. (Abbas, 2003)

***Grids* Departamentais**: São utilizados por um grupo de pessoas restrito dentro da empresa, onde os recursos não são compartilhados internamente. Possuem duas definições:

- *Cluster Grids*: um ou mais sistemas trabalhando juntos para prover um único ponto de acesso para os usuários.
- *Infra Grids*: grid que otimiza recursos dentro da empresa e não utiliza outro parceiro interno.

***Grids* Empresariais**: os recursos são disponibilizados por toda empresa e prove serviços para todos os usuários. Possuem três classificações:

- *Entreprise Grids*: é utilizado por empresas com acesso global ou que precisem acessar recursos externos a um local corporativo único.
- *Intra Gri*ds: diferentes recursos são compartilhados por diversos grupos dentro da empresa.
- *Campus Grid*s: os recursos são compartilhados de forma cooperativa, várias estações de trabalho dispersas são utilizadas localizadas em vários domínios administrativos, em departamentos ou por toda a empresa.

***Grids* Globais**: são *grids* implementados sobre a Internet pública. Podem ser estabelecidos principalmente por organizações para ajudar as negociações e compras. Essa categoria possui duas definições:

- *Global Grids*: os recursos podem ser distribuídos para usuários em qualquer lugar do mundo para computação e colaboração.
- *Inter Grids*: os recursos de computação podem ser compartilhados e armazenados pela Web pública.

***Grids* de Computação**: são desenvolvidos para prover o acesso a recursos computacionais. A classificação é feita de acordo com o *hardware* computacional implantado.

- *Desktop Grids*: compartilham recursos de computadores pessoais.
- *Server Grids*: apenas incluem os servidores

- *High-Perfomance/Cluster Grid*: *grids* constituídos por supercomputadores e *clusters*.

***Grids* de Dados**: *grids* desenvolvidos com o objetivo de requerer acesso ou processar dados.

***Grids* de Utilidades**: são *grids* de recursos computacionais comerciais que são mantidos e gerenciados por um provedor de serviço.

3.4 ESCALONAMENTO

Devido à demanda exigida pelos sistemas distribuídos, houve a necessidade de criação de novos escalonadores denominados Globais, diferentemente dos escalonadores locais utilizados pelos sistemas operacionais; os escalonadores globais precisam lidar com os diversos recursos disponíveis em um ambiente distribuído. A tarefa de um escalonador é maximizar a utilização dos recursos computacionais e minimizar os custos de comunicação em um conjunto de máquinas diferentes, objetivando cumprir metas de desempenho e performance, por isso, sua principal função é escolher quando e quais processos tem acesso a determinados recursos do sistema. (Maheswaran, 1999). As principais metas que precisam ser cumpridas são (Jain, 1991):

Aumentar o *throughtout* do sistema: nada mais é do que a vazão do sistema que é medido de acordo com o número de processos finalizados por unidade de tempo.

Diminuir o tempo de resposta: o tempo é definido pela diferença entre o término da execução da tarefa e o instante de chegada da tarefa na fila de processos.

Aumentar a utilização de recursos: o escalonador pode ser utilizado para que os recursos do sistema sejam utilizados ao máximo como: CPU, memória ou rede.

Balancear a carga do sistema: não subutilizar recursos, ou seja, distribuir os processos de acordo com a capacidade dos recursos.

Nas grades computacionais, os escalonadores possuem funções importantes, pois precisam selecionar recursos e realizar o escalonamento de processos, ao mesmo tempo em que outras requisições chegam, tendo que garantir tempo de execução e custo de recursos. A Figura 6 ilustra a relação do escalonador, usuário e recursos em um ambiente distribuído.

Figura 6 - Relação escalonador, usuário e recursos no ambiente distribuído.

No escalonamento de tarefas em grades computacionais as tarefas que compõem uma aplicação podem ter dependências entre si. Quando as dependências existem, as tarefas formam grafos direcionados para representar as dependências. Quando as dependências não existem, as tarefas formam grafos vazios, ou seja, grafos que não possuem arestas, e são mais conhecidos como *Bag-of-Tasks* (*BoT*). Na Figura 7 é ilustrado um grafo direcionado com dependência entre as tarefas, enquanto na Figura 8 é ilustrado um grafo vazio de uma aplicação *BoT*. (Batista, 2010)

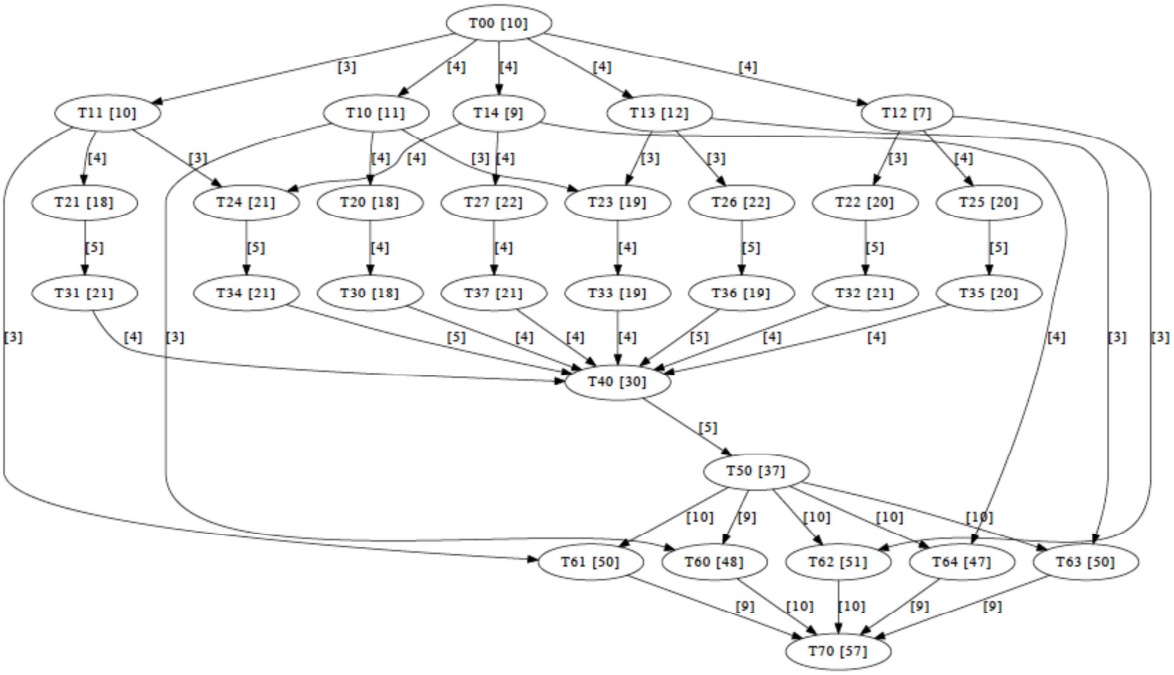

Figura 7 - Aplicação com Tarefas Dependentes

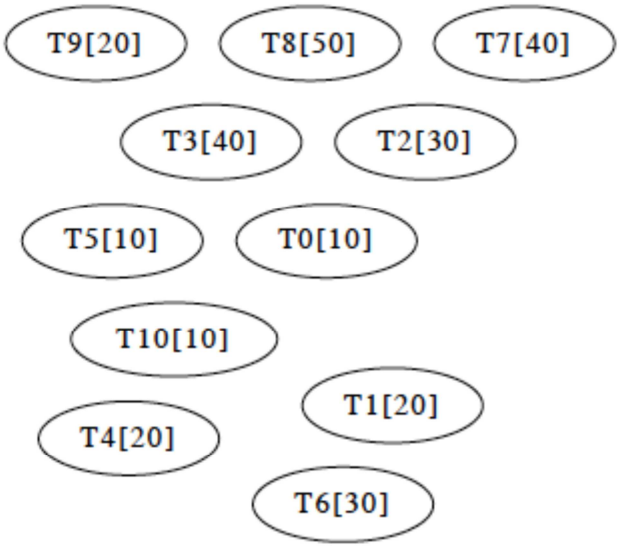

Figura 8 - Aplicação com Tarefas Independentes

3.5 GRADES COMPUTACIONAIS NA BIOINFORMÁTICA

As ciências biológicas passaram por uma revolução significativa nos últimos anos e os avanços na biotecnologia, apoiada por um salto enorme na computação, proporcionaram uma riqueza de dados em todos os níveis biológicos.

A grande demanda computacional criada por aplicações na área da bioinformática vem impulsionando cada vez mais a utilização de tecnologias que possam tratar grandes quantidades de dados. Um exemplo de utilização de solução em grade na bioinformática pode ser exemplificado pelo Programa e-Ciência do Reino Unido, no Projeto denominado Biologia Integrativa, que possuía como objetivo integrar informações biológicas em um sistema único contendo as descrições de funções biológicas de doenças cardíacas e câncer que juntos são responsáveis por mais 60% de mortes no Reino Unido (Gavaghan, 2005). A utilização das grades computacionais no campo da biologia vem facilitando a manipulação dos dados e otimizando a utilização dos recursos nessa área que vem crescendo e se destacando no cenário tecnológico.

4. TRABALHOS RELACIONADOS

Este capítulo apresenta os principais trabalhos encontrados na literatura relacionados à proposta desta dissertação. Os principais algoritmos de escalonamento de tarefas utilizados em grades computacionais são apresentados, e o *software ABySS* utilizado para montagem de genomas.

4.1 PRINCIPAIS POLÍTICAS DE ESCALONAMENTO PARA GRADES COMPUTACIONAIS

As grades computacionais no contexto atual estão sendo utilizadas como nova alternativa para se prover grande capacidade de processamento. Neste capítulo serão vistos os principais algoritmos existentes para realizar o escalonamento de tarefas em grades computacionais. O problema de escalonamento se torna mais desafiador devido a algumas características únicas pertencentes à computação em grade, tais como, a sua heterogeneidade e alta dinamicidade. (Dong, 2006).

4.1.1 WQR

O algoritmo *Workqueue with Replication* (*WQR),* foi criado para solucionar o problema de obtenção de informações sobre a aplicação e os recursos da grade em tarefas independentes. No wqr as tarefas são selecionadas de forma aleatória e enviadas ao processador quando estes estiverem disponíveis. Quando um processador fica disponível e não existem mais tarefas a serem executadas, o wqr começa sua fase de replicação para as tarefas que ainda estão executando, quando uma tarefa original finaliza sua execução primeiro que suas réplicas, estas são interrompidas. Caso contrário, quando alguma réplica finaliza primeiro, a tarefa original e as demais réplicas dela são interrompidas. O uso da replicação pode ocasionar consumo extra de CPU, porém, baixo já que ocorre nos processadores disponíveis. (Da Silva, 2003)

O ponto negativo é que, para alcançar um bom desempenho o wqr desperdiça ciclos de CPU com as réplicas de tarefas que são canceladas. Isto é, a CPU utilizada não era útil para o processamento de aplicativos. No entanto, quando a heterogeneidade de tarefas é baixa a dos processadores é elevada, o wqr desperdiça poucos ciclos. No entanto, com a heterogeneidade das tarefas alta e heterogeneidade dos processadores baixa, o consumo de recursos extra do wqr pode ser significativo.

4.1.2 *Dynamic FPLTF*

O *Dynamic Fastest Processor to Largest Task First* (*Dynamic* FPLTF) (Menascé et al.,1995), possui uma grande habilidade de se adaptar à dinamicidade e heterogeneidade de ambientes como as *Grids.* O *Dynamic* FPLTF necessita de três informações do sistema para escalonar as tarefas: *Task Size*, *Host Load* e *Host Speed*. *Host Speed* representa a velocidade relativa da máquina, o *Host Load* representa a fração de UCP da máquina que não está disponível para a aplicação, ou seja, está sendo usada por outros usuários ou aplicações. Vale ressaltar que, o valor de *Host Load* varia com o tempo, dependendo da carga que está sendo imposta à máquina por outros usuários ou aplicações. Finalmente, *Task Size* representa o tempo necessário para uma máquina com *Host Load* igual a zero, ou seja, totalmente disponível, complete a execução da tarefa.

No início do algoritmo uma variável TBA (*Time to Become Available*) é inicializada com zero. Tal variável representa o tempo necessário para que um

determinado recurso esteja disponível. As tarefas são organizadas em ordem decrescente de tamanho, desse modo. a maior tarefa é a primeira a ser alocada. Cada tarefa é alocada para o *host* que provê o menor tempo de execução CT (*Completion Time*) para ela, onde:

$$CT = TBA + Task\ Cost$$

$$Task\ Cost = (Task\ Size / Host\ Speed) / (1 - Host\ Load).$$

Quando uma tarefa é alocada para uma máquina, o valor de TBA correspondente é incrementado com *Task Cost*. As tarefas são alocadas até que todas as máquinas do *grid* estejam ocupadas. Após isso, a execução da aplicação é iniciada. Assim que uma tarefa completa sua execução, todas as tarefas que estão executando no momento são desescalonadas e escalonadas novamente, até que cada máquina do *grid* tenha pelo menos uma tarefa alocada. Isso continua até que todas as tarefas sejam completadas. Essa estratégia tenta minimizar os efeitos do dinamismo do *grid*. Uma máquina, que a princípio, é mais rápida e não está sobrecarregada recebe mais tarefas que uma máquina também rápida, porém, com bastante carga. Essa variação na carga pode comprometer a execução da aplicação já que as tarefas escalonadas para essa máquina iriam executar mais lentamente que o previsto. O processo de reescalonar tarefas tenta corrigir este problema alocando as tarefas maiores para as máquinas mais rápidas no momento do escalonamento. Esta política apresenta bom desempenho, contudo, é muito complicado de ser implementado na prática, pois necessita de muita informação do ambiente. Tais informações geralmente são difíceis de obter com precisão e frequentemente não estão disponíveis devido a restrições administrativas, tornando a solução inviável em vários casos.

4.1.3 *Sufferage*

O *Sufferage* é um algoritmo dinâmico que tem mostrado atingir bom desempenho em ambientes heterogêneos e dinâmicos quando considerada a existência de informação completa (Ahrnad,1995).

A ideia básica de escalonamento *Sufferage* (Casanova et al., 2000; Maheswaran et al.,1999), como o próprio nome reflete, é determinar quanto cada tarefa seria prejudicada (“sofreria”) se não fosse escalonada para o processador que a executaria de forma mais eficiente. O valor de *sufferage* de uma tarefa é definido pela diferença entre seu segundo melhor e seu melhor CT (*Completion Time*), este calculado

da mesma forma como mostrado na descrição da política *Dynamic FPLTF*. A *sufferage* consiste na ideia de atribuir a melhor máquina à aplicação que seria mais prejudicada se essa atribuição não fosse feita. A tendência é que cada tarefa seja atribuída à máquina que oferece o menor tempo para completá-la. No início do algoritmo, quando todas as máquinas estão livres, o valor de *sufferage* é calculado para cada tarefa da fila. A tarefa que possuir o maior valor de *sufferage* é alocada para a máquina que a executaria de forma mais eficiente, ou seja, a que proporcionou o melhor *completion time* para a tarefa. Em seguida, o procedimento é repetido várias vezes até que todas as tarefas sejam alocadas. Se uma máquina proporcionar o melhor CT para uma tarefa, mas estiver ocupada executando outra tarefa, os valores de *sufferage* de cada uma das tarefas é comparado e a que possuir o maior fica alocada na máquina, a que possuir o menor valor volta para a fila de tarefas prontas. Com o dinamismo do *grid*, como variação de carga dos processadores, os valores de *sufferage* das tarefas variam durante toda a execução da aplicação. Dessa forma, o algoritmo s*ufferage* é utilizado diversas vezes durante a execução da aplicação até que todas as tarefas tenham sido escalonadas. Toda vez que uma tarefa é completada, todas as tarefas que ainda não começaram a executar, são desescalonadas e o algoritmo é utilizado novamente para agendar as tarefas restantes, usando os valores de *sufferage* atuais, mas dessa vez, com a nova carga das máquinas. Isso se repete até que todas as tarefas tenham sido alocadas para alguma máquina. O problema deste algoritmo é o mesmo que FPLTF dinâmico, pois necessita de muita informação para calcular os tempos de conclusão das tarefas. Ele precisa saber o tamanho da tarefa, a carga dos processadores e a velocidade, além de ser utilizados diversas vezes durante todo o processo, o que demanda tempo para a conclusão das tarefas.

4.2 *ABySS* (ASSEMBLY By SHORT SEQUENCES)

A adoção de instrumentos paralelos para o sequenciamento do DNA levou ao recente desenvolvimento de algoritmos que realizam a leitura de sequências curtas (*short-reads*). Um dos desafios das ferramentas disponíveis é a incapacidade para reunir de forma eficiente grandes quantidades de dados gerados a partir de projetos de sequenciamento em larga escala. (Simpson et al., 2009)

Para resolver esta limitação foi desenvolvido por Shaun D. Jackman, o *ABySS*, um montador paralelo de sequências curtas que consegue executar a montagem de

milhões ou bilhões de sequências, através de computação paralela; foi desenvolvido em C++ e possui licença gratuita de utilização. Aceita como arquivos de entrada os formatos: *Fasta*, *Fastaq*, Qseq, *Sam* e *BM*, e como arquivos de saída: *Fasta* e *Graphvizdod*.

A principal característica do *ABySS* em relação aos montadores tradicionais é a utilização do Grafo de Debruijn para realizar a comparação e a montagem das leituras (Pevzner et al., 2001).

Em 1995, Idury e Waterman introduziram o uso de um grafo para representar uma sequência de montagem. Seu método consiste na criação de um vértice para cada palavra detectada, em seguida, conectar os vértices correspondentes à sobreposição de *k-mers*, onde k pode ser representado por uma sequência com número específico de bases, onde o vértice de origem corresponde ao prefixo k-1 da região de sobreposição correspondente (*k-mer*), e o vértice de destino ao sufixo k-1 da mesma com isso, a reconstrução da sequência consiste no caminho que percorre todas as arestas. Pevzner et al., (2001), propuseram uma formalização ligeiramente diferente do grafo de sequência, o chamado grafo de Debruijn, segundo o qual, o *k-mers* são representados como arcos ou arestas e as sobreposição dos *k-mers* aderem às suas pontas.

O método clássico de montagem de fragmentos baseia-se na noção de um grafo de sobreposições. A sequência de DNA da Figura 9*a* consiste de quatro segmentos únicos A, B, C e D e uma repetição R. Cada leitura corresponde a um vértice do grafo de sobreposições e dois vértices são conectados por um arco, se as leituras correspondentes se sobrepuserem como observamos na Figura 9b (Lemos et al., 2003). É possível visualizar a construção deste grafo representando uma sequência de DNA como uma ― "linha" com regiões repetidas cobertas por uma "cola" que liga tais regiões (Figura 9c). O grafo de Debruijn neste caso (Figura 9d) consiste em 5 arcos, onde cada repetição corresponde a um arco ao invés de uma coleção de vértices no grafo de sobreposições (Figura 9b). É possível notar que o grafo de Debruijn (Figura 9d) representa as repetições de uma forma muito mais simples que o grafo de sobreposições (Figura 9b). É importante ressaltar que um grafo de Debruijn é centrado no *k-mer*, o que significa que sua topologia não é afetada pela fragmentação das leituras (Zerbino, 2009).

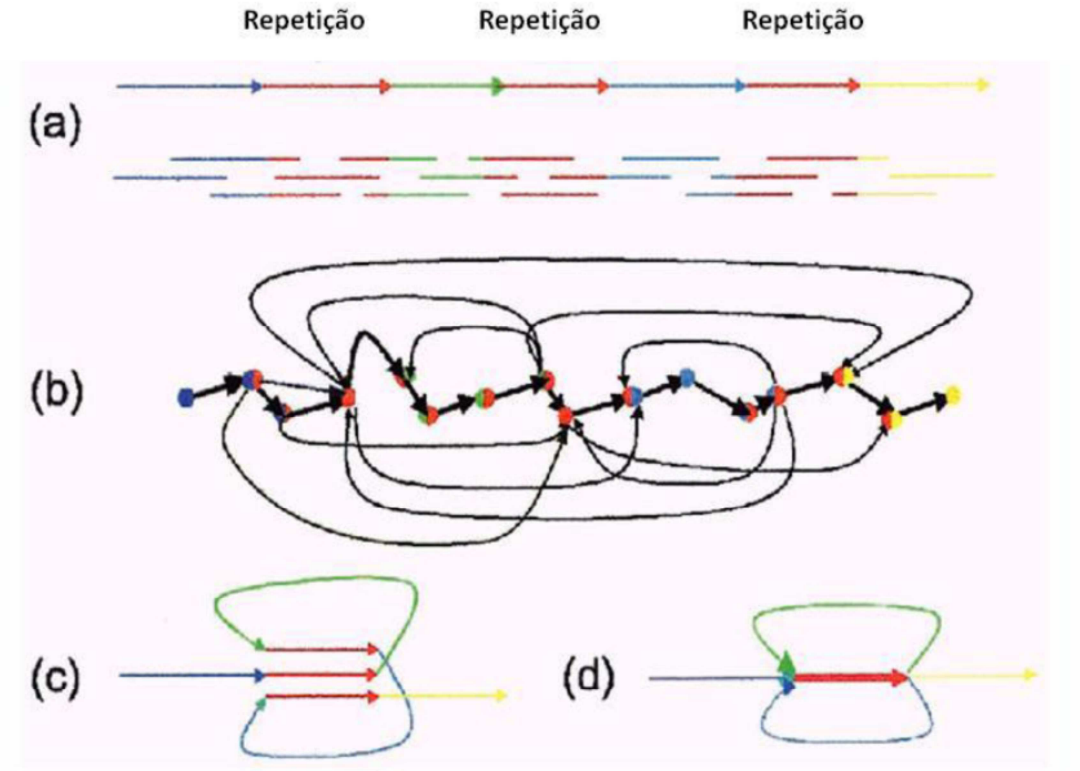

Figura 9 - (a) Sequência de DNA com três repetições R; (b) o grafo do layout; (c) construção do grafo de Debruijn colando as repetições; (d) o grafo de Debruijn.

5. O ALGORITMO DE ESCALONAMENTO *"SCHEDULER"*

O aumento na disponibilidade de computadores mais poderosos e redes de alta velocidade estimulou a utilização de técnicas eficientes para alocação de tarefas em aplicações paralelas, conhecida como escalonamento de tarefas. O desenvolvimento de algoritmos de escalonamento de tarefas tem como objetivo alocar tarefas de uma aplicação em recursos computacionais para atingir um bom desempenho, ou seja, minimizar o tempo do termino das aplicações e assim otimizar o processamento de uma forma geral.Neste capitulo será descrito o processo o algoritmo desenvolvido denominado *sheduler*, seu funcionamento, sua relação com o *software* de montagem *ABySS* e suas aplicações

O escalonador de tarefas foi desenvolvido com o objetivo de otimizar o tempo no processo de montagem do mapeamento do DNA, utilizando em conjunto às técnicas de grades computacionais para alcançar os resultados desejados.

O escalonador foi implementado em C e utiliza uma política de escalonamento que não necessita de informações sobre o ambiente ou sobre a aplicação para processar as tarefas. Assim que os recursos (processadores) estão disponíveis, tarefas são atribuídas a eles. A distribuição das tarefas é realizada de forma aleatória, bastando apenas que o recurso esteja disponível e que a tarefa exista para ser escalonada; ao final da execução da tarefa, o processador responsável envia o resultado do processamento ao escalonador e este, por sua vez lhe atribui uma nova tarefa, fazendo com que todas as tarefas sejam processadas, como mostra a Figura 10.

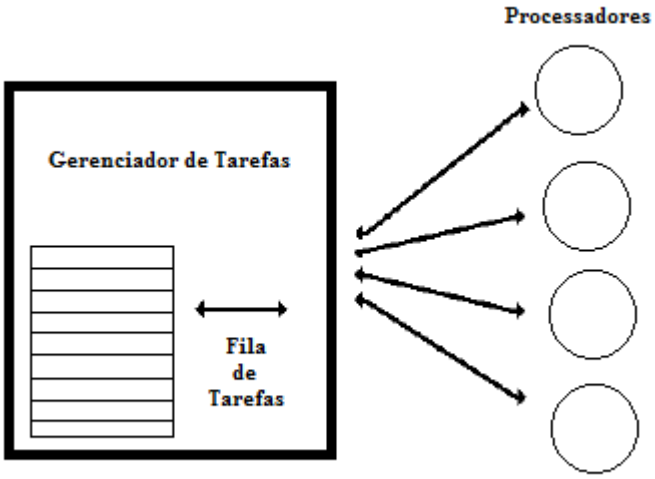

Figura 10 - Gerenciador de Tarefas

Os processadores são dedicados, ou seja, foram utilizados somente para o processamento das tarefas do mapeamento do DNA, desta forma, os processadores mais velozes receberão mais tarefas, já que ficaram disponíveis mais rapidamente; existe uma possibilidade remota de um processador ficar sobrecarregado com uma tarefa que não pode processar, já que os processadores possuem configurações bem próximas, caso isso ocorra o processamento como um todo não será afetado, já que os outros processadores estarão disponíveis, tal fato apenas pode afetar o tempo final de processamento de todas as tarefas.

O algoritmo faz o agendamento de tarefas usando como entrada as seguintes variáveis: identificação da tarefa, duração da tarefa, tempo de execução da tarefa, tempo de espera da tarefa.

```
#include <stdio.h>
#include <stdlib.h>

struct processos {
    int id;                 /* Id do processo*/
    int tdp;                /* Tempo de duração do processo*/
    int exec;               /* Tempo de execução do processo*/
    int tep;                /* Tempo de espera do processo*/
    struct processos *prox;
};
```

Figura 11 - Variáveis de entrada do *Sheduler*.

O *sheduler* começa a sua execução verificando a lista de processos que serão executados pelos processadores disponíveis, cada tarefa possui seu *Id* de identificação e o tempo mínimo para que esse processo seja executado. A lista de processos contém o conjunto de dados genômicos provenientes do sequenciamento do DNA, previamente já realizado, como mostra a Figura 11.

```
/* Inicia a lista de processos*/
struct processos *init_processos (int id, int tdp) {
    struct processos *proc;
proc = (struct processos*)malloc(sizeof(struct processos));
  if (proc == NULL) {
        printf("Erro Fatal: Falha Alocacao de memoria.\nFinalizar.\n");
        exit(1);
    };
    proc->id = id;
    proc->tdp = tdp;
    proc->exec = 0;
    proc->tep = 0;
    proc->prox = NULL;
    return(proc);
};
```

Figura 12 - Lista dos processos para execução

Após a verificação de processos disponíveis, o *sheduler* inicia o escalonamento das tarefas, esse escalonamento é realizado verificando a disponibilidade de processadores para iniciar o escalonamento; no inicio do escalonamento os processadores recebem tarefas aleatórias para serem processadas, quando um processador finaliza a execução, um aviso de disponível é enviado para o *sheduler* junto

com o tempo de processamento desta tarefa e o tempo de espera até que a tarefa tenha sido escalonada, e assim o processador se torna disponível para que outras tarefas possam ser atribuídas a ele, todo esse processo ocorre na função “sh” do algoritmo, que só para de executar quando não existem mais tarefas disponíveis. A Figura 13 mostra o código da função “sh” e os parâmetros utilizados.

```
void sh (struct processos *proc) {
    int tempo = 0, inicio, fim;
  struct processos *tmp = proc;
  printf("\tSheduler\n");
    printf("\n");
  while (tmp != NULL) {
    inicio = tempo;
    tempo += tmp->tdp;
    fim = tempo;
    printf("Processo: %d\tTempo de Duracao: %d\tTempo de Espera: %d\tRetorno: %d
\n", tmp->id, tempo, inicio, fim);
    tmp = tmp->prox;
  };
    printf("\n\n");
};
```

Figura 13 - Função sh

Quando todos os processos são devidamente escalonados, uma nova saída com todas as informações sobre os processos é gerada e a partir dessa saída, podemos verificar o tempo de espera que cada tarefa levou para começar a ser escalonada, o tempo de execução que cada processador levou para concluir a tarefa, o número de tarefas que cada processador conseguiu executar e o seu *id* de identificação. Como mostra a Figura 14.

	Processador 1	Processador 2	Processador 3	Processador 4	Processador 5	Processador 6
Id	1	2	3	4	5	6
Tempo de espera	90	70	80	30	35	20
Tempo de execução	100	90	100	80	93	92
nº de tarefas	1000	800	900	700	850	840

Figura 14 - Saída de Informações dos Processos

5.1 *SHEDULER* E O *ABySS*

O *ABySS* como já mencionado no capítulo anterior é um montador de sequências curtas de DNA, a versão utilizada para os testes com o *sheduler* foi a versão

paralelizada 1.3.4. Esse montador foi escolhido, pois o seu código fonte é aberto para alterações e por possuir uma versão paralela, diferente, por exemplo, do montador Velvet que somente publicou uma versão paralela recentemente.

Para um melhor aproveitamento dos dados genômicos, o *sheduler* desenvolvido foi adaptado ao *software* de montagem de sequências curtas *ABySS*, esse *software* faz toda a parte biológica, ou seja, os dados provenientes do sequenciamento são verificados quanto a repetições, *gaps* e em seguida fracionados em partes aleatórias para o processamento, e somente a partir deste momento o escalonador *sheduler* é executado para gerenciar o processamento e a alocação de tarefas; após o processamento, os dados são enviados novamente para os montador *ABySS*, que através das tabelas geradas pelo escalonador com o *id* dos processos consegue montar novamente o genoma e gerar um arquivo para posterior anotação, como mostra a Figura 15.

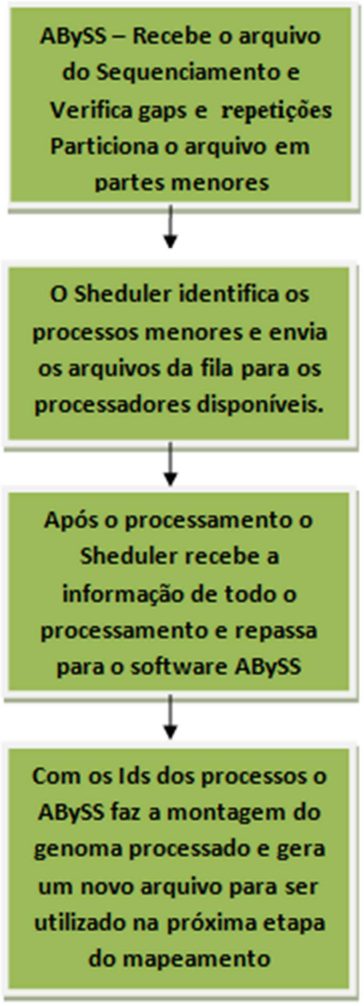

Figura 15 - Diagrama de funcionamento do *software ABySS* com o *Sheduler*

5.2 APLICAÇÃO DO *SHEDULER*

O uso do *sheduler* visa melhorar o processamento e minimizar o tempo de espera em ambientes distribuídos. A utilização do escalonador no mapeamento do DNA agregado ao *software* de montagem *ABySS*, se mostrou bastante eficiente e significativo quando comparado com os *software* já desenvolvidos na literatura, pois apesar de objetivarem melhorar o processamento em ambientes distribuídos, não utilizam um montador paralelo de sequências curtas agregado, o que afeta de forma significativa o processamento por conta do tempo gasto para realizar a comunicação entre o escalonador e o *software* de montagem, já o *sheduler*, de acordo com os testes realizados, melhora de maneira eficiente o tempo e o processamento das tarefas, pois trabalham de forma conjunta e eficiente.

O s*heduler* trabalha com partes menores e distribui o processamento entre os processadores disponíveis, se por alguma falha ou problema no arquivo, uma das partes não for processada de forma correta, os outros processos não são interrompidos, e o arquivo volta para fila de espera para ser processado novamente, sem que tudo é controlado pelo Id dos processos.

Utilizando o *sheduler* em conjunto com o *software ABySS*, toda a parte biológica continua a ser executada com a agregação do escalonamento, diminuindo o gargalo que ocorre na etapa de reconstrução do genoma, pois existe uma grande quantidade de dados concentrados e que exigem um alto poder de processamento.

6. AMBIENTE DE SIMULAÇÃO

Este capítulo tem o objetivo de avaliar o funcionamento do escalonador desenvolvido. Para a realização dos testes com o algoritmo de escalonamento desenvolvido foi definido como melhor procedimento a simulação, pois as variáveis, a diferença entre as cargas de processamento e o volume dos dados poderiam ser manipulados com mais controle. O simulador escolhido foi o *Simgrid* (Casanova et al.2001), pela fácil adaptação a diferentes condições e testes. Para comparar o desempenho do escalonador, foram considerados apenas os algoritmos dinâmicos

devido à sua melhor adequação em grades. Os algoritmos são: *Dynamic FPLTF*, *Sufferage* e *Workqueue*. Eles foram escolhidos por serem bem conhecidos e estudados. O escalonador também foi comparado com a utilização do *software ABySS* na sua versão original para verificarmos o comportamento do escalonador utilizado no *software* de montagem.

6.1 O SIMULADOR

O *Simgrid* foi desenvolvido por Henri Casanova em um projeto de doutorado em 1999, a maior motivação para a construção da ferramenta foi à necessidade de se utilizar a simulação, ao invés de experimentos reais, no estudo prático de algoritmos de escalonamento para aplicações computacionais distribuídas e heterogêneas.

O *Simgrid* é uma ferramenta que fornece um conjunto de funcionalidades que podem ser utilizadas no estudo prático de algoritmos de escalonamento, ou seja, em ambientes heterogêneos distribuídos, e provê ainda ambientes de programação construídos a partir de um único núcleo de simulação. Apesar disso, não disponibilizam políticas internas de escalonamento de tarefas, a implementação dos algoritmos deve ser feita pelo próprio usuário. (Legrand, A., et al. 2003, *Simgrid* Project Web Page 2012).

No processo de simulação, a modelagem dos recursos é um componente importante, a um cluster ou uma máquina, por exemplo, podendo ser atribuídas variáveis do tipo latência, taxa de serviço e poder computacional. O poder computacional é definido no simulador como sendo o número de unidades de trabalho por unidades de tempo.

6.1.1 Biblioteca *Libts* (*library tasks scheduling*)

A *LIBTS* foi desenvolvida para disponibilizar a implementação dos algoritmos de escalonamento de tarefas, no *SimGrid*. O *SimGrid* é uma ferramenta de simulação de aplicações em ambientes distribuídos heterogêneos que não disponibiliza políticas internas de escalonamento de tarefas, além disso, a implementação dos algoritmos deve ser feita pelo próprio usuário. A biblioteca *LIBTS* utiliza as funções do módulo MSG disponibilizadas pelo *SimGrid* e executa com aplicações do tipo *Master-Slave*. Em uma

aplicação *Master-Slave* um computador é definido como *Master* (mestre) e os outros como *Slaves* (escravos). A função do *Master* é controlar o envio das tarefas para os *Slaves.* O escalonamento de tarefas tem sido uma questão importante para melhorar a execução paralela em sistemas distribuídos. O modelo *Master-Slave* tem sido aplicado com sucesso em muitas aplicações paralelizadas em diferentes domínios de aplicações. Na Figura 16 é apresentada uma visão geral do *SimGrid* e como a LIBTS foi adicionada para interagir com os módulos do *SimGrid*. A biblioteca é composta pelo módulo principal "Escalonamento", que contém os algoritmos: *WQ, WQR, Sufferage, Dynamic FPLTF*. O usuário que desejar desenvolver sua própria política de escalonamento pode utilizar a *LIBTS*, desde que crie a aplicação seguindo os padrões dos componentes do módulo *MSG*. (Franco et al., 2011).

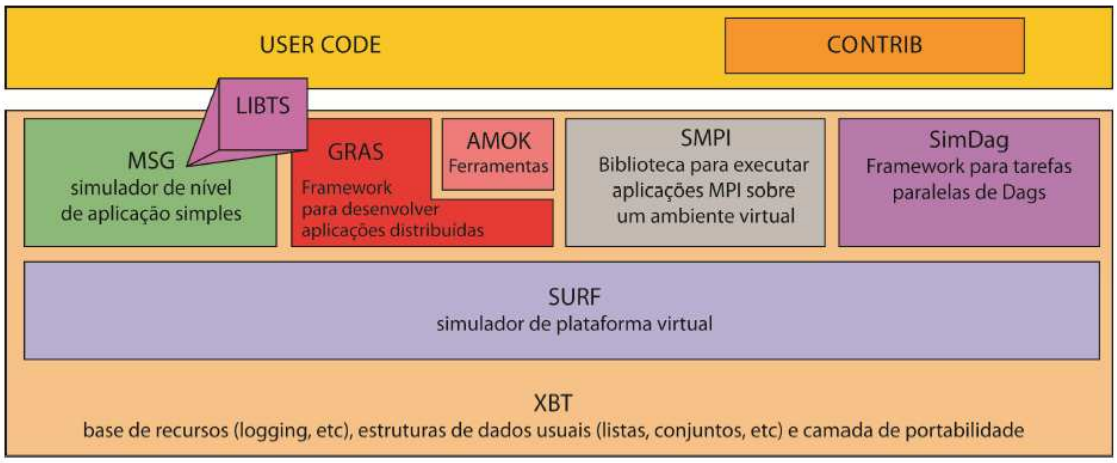

Figura 16 - *Simgrid* com a Biblioteca LIBTS

6.2 AMBIENTES PARA AS SIMULAÇÕES

O principal objetivo de nossos experimentos é avaliar o desempenho do algoritmo de escalonamento desenvolvido no que diz respeito principalmente, ao tempo de realização de cada tarefa na montagem do genoma. As experiências ajudam a avaliar a influência da heterogeneidade das redes (diferentes velocidades), heterogeneidade das tarefas (variação de tamanho) e a granularidade das tarefas (número de tarefas por máquina). Foram realizadas 10 simulações e a carga nas máquinas são simuladas a partir de dados genômicos disponibilizados pelo Instituto de Ciencias Biológicas da

Universidade Federal do Pará, de um genoma procarioto de sequências curtas de *Corynebacterium pseudtuberculosis* I19.

O cenário de grid que foi construído no simulador é composto de seis unidades de processamento, onde o poder de processamento varia entre 800 *Mflops* e 3500 *Mflops*, todas as máquinas estando interconectadas. Para a realização das tarefas existe uma máquina chamada "*scheduler*" que recebe essas tarefas, realiza o escalonamento e submete ao nó escolhido para a execução. A máquina *"scheduler"* está conectada ao *front-ends* e seu poder computacional é irrelevante, pois não executa nenhuma tarefa de processamento, já que a velocidade do escalonamento é determinada pela máquina onde a simulação é executada, após o término do processamento os resultados obtidos são enviados para a máquina novamente *sheduler*.

Figura 17 - Cenário de Grid

Os experimentos são realizados com processadores de capacidades variáveis e as tarefas são heterogêneas. Na simulação pressupõe-se que as máquinas estão 100% disponíveis para concluir todas as tarefas de forma simultânea; a latência e a largura de banda não são levadas em consideração na simulação, já que podem causar um impacto significativo na performance da aplicação.

O genoma utilizado para os testes de simulação possui de 40 a 50 Mb, e através do *software ABySS*, o genoma é fragmentado em pequenas partes de tamanhos variados e são distribuídos pelo escalonador para as máquinas disponíveis na grade computacional, os resultados obtidos são enviados para a máquina com o *software ABySS* que coleta os resultados, agrupa e a saída é o resultado esperado. A variação das máquinas e tarefas tem por objetivo avaliar o impacto no desempenho do escalonador, pois podem ocorrer momentos em que há mais tarefas a serem executadas do que máquinas disponíveis e o oposto também é considerado verdadeiro.

6.3 TESTES E RESULTADOS

Nesta seção, são apresentados os resultados obtidos da avaliação de desempenho do escalonador desenvolvido. As avaliações foram feitas através de simulações em cenários distintos.

A Figura 18 mostra o tempo de execução médio do algoritmo de escalonamento. Cada ponto de dados resume seis níveis de máquinas e heterogeneidade de tarefas

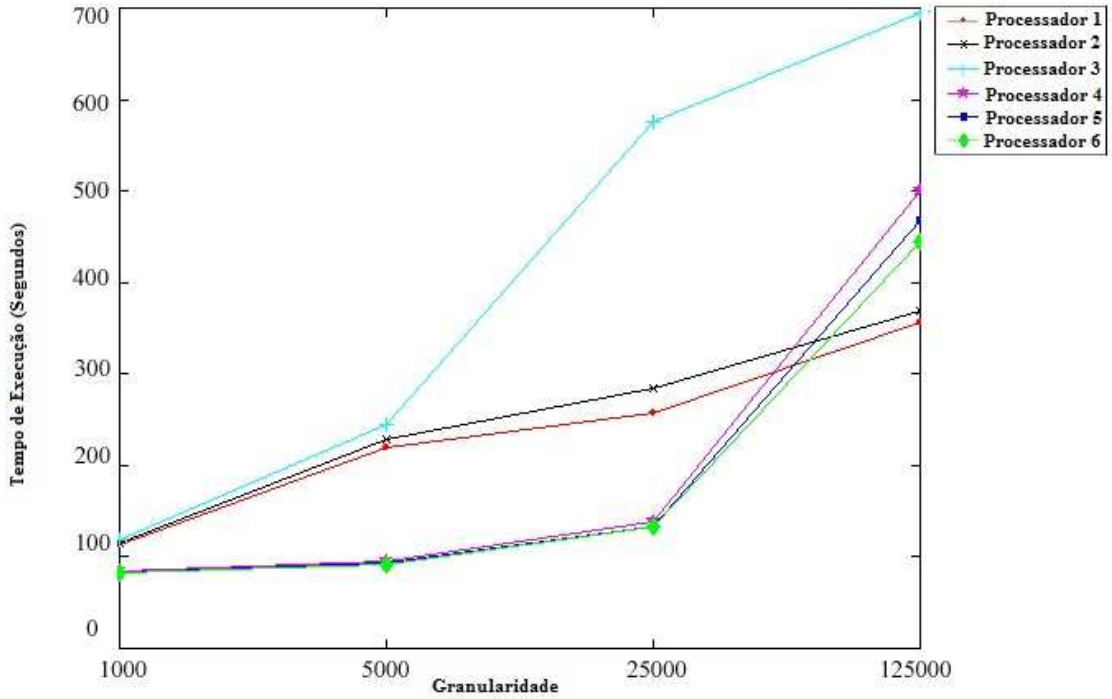

Figura 18 - Desempenho do Algoritmo quanto à granularidade.

A tendência que pode ser observada é que em situações com maior número de tarefas por máquina e menos granularidade a performance melhora, quando há muitas tarefas o escalonador de forma dinâmica consegue manter todos os processadores ocupados na maior parte do tempo, este gráfico apresenta este comportamento porque os processadores são heterogêneos e as tarefas podem ser alocados de forma aleatória sem impacto para o processamento como um todo, como mostra a Figura 18, somente

no fim da execução da aplicação a situação muda devido à carga de desequilíbrio e prejudica o desempenho. Outro ponto que deve ser considerado é o tempo de execução das tarefas que é relativamente alto, pois as tarefas são heterogêneas e trata-se de dados genômicos.

A Figura 19 apresenta o impacto da heterogeneidade de máquinas na grade, cada ponto mostra os níveis de heterogeneidade das tarefas; de acordo com a análise, podemos concluir que o escalonador responde de forma satisfatória ao processamento de acordo com o tempo, pois conseguiu alocar todas as tarefas e estas foram processadas no tempo médio sem causar impacto no processo final, mesmo quando as máquinas são bastante heterogêneas. Como mostra a Figura 19, podemos observar que o processador 3 possui um desempenho em relação ao tempo diferente dos outros processadores, pois quanto maior a heterogeneidade das máquinas existe a possibilidade de uma máquina com capacidade inferior de processamento executar uma tarefa "grande", causando um impacto apesar de pequeno sob o tempo de execução das tarefas, porém, os outros processadores se mantiveram com um desempenho satisfatório.

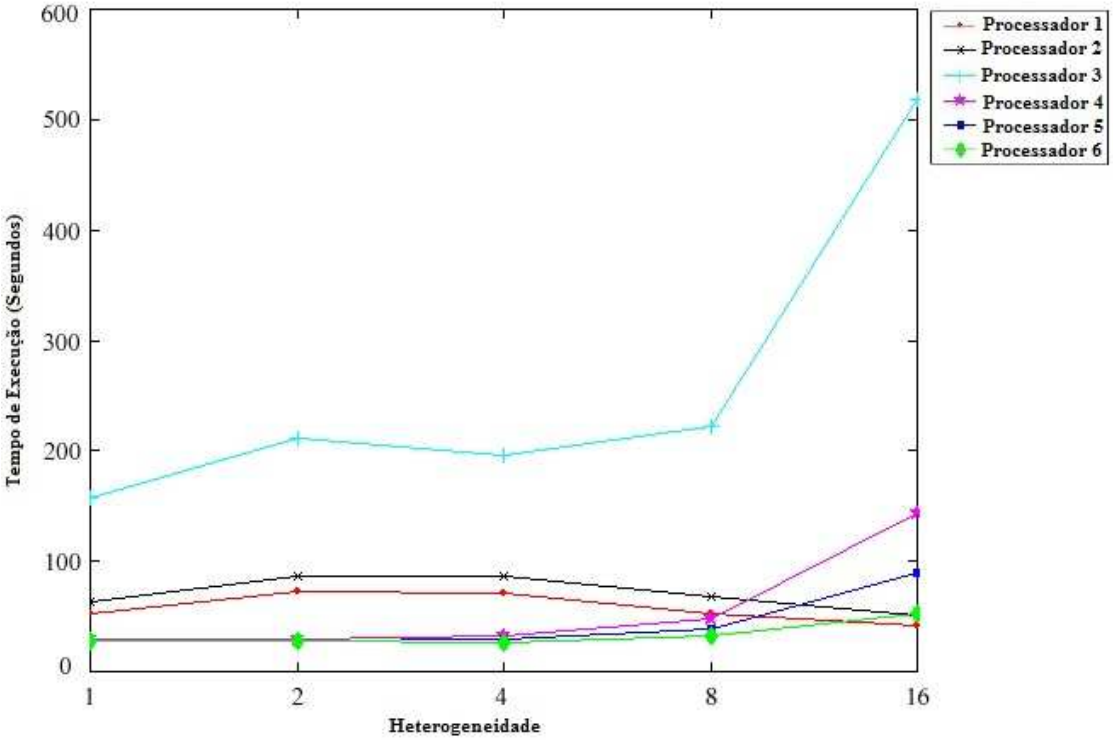

Figura 19 - Heterogeneidade dos Processadores

A Figura 20 mostra o impacto da heterogeneidade de tarefas sobre o desempenho do algoritmo de escalonamento, cada ponto mostra as tarefas que foram

executadas; de acordo com a análise, podemos concluir que a heterogeneidade de tarefas não causou um impacto significativo no desempenho do escalonador proposto, mostrando sua eficiência em tarefas heterogêneas.

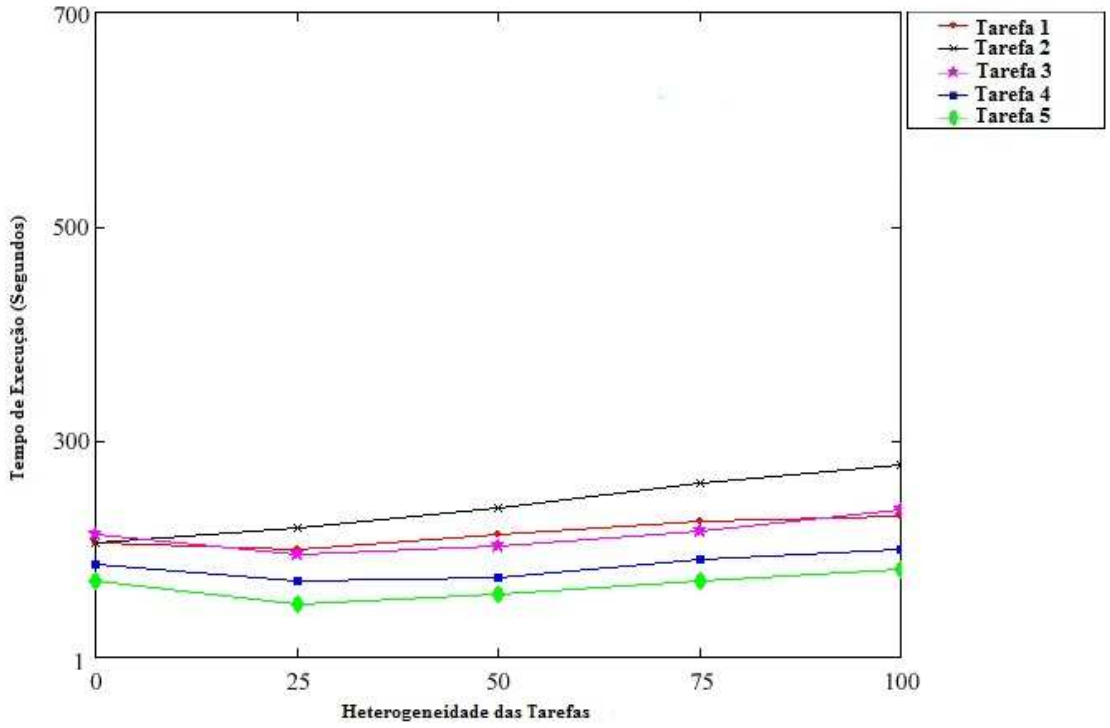

Figura 20 - Heterogeneidade das Tarefas

Na Figura 21 é apresentado o resultado da simulação com as mesmas tarefas, porém utilizando o *software ABySS* para distribuir as tarefas entre os processadores. Os testes mostram que o tempo de processamento das tarefas executadas pelos processadores com o *ABySS* foi superior, pois quando comparado com o tempo que o escalonador proposto *sheduler* conseguiu processar a mesma quantidade de tarefas se mostrou inferior, isso ocorre porque o software *ABySS* não possui um escalonador eficiente para alocar as tarefas para os processadores disponíveis. Isso implica dizer que o escalonador desenvolvido pode ser considerado mais eficiente que o escalonador utilizado pelo *ABySS* em relação ao tempo de execução das tarefas heterogêneas.

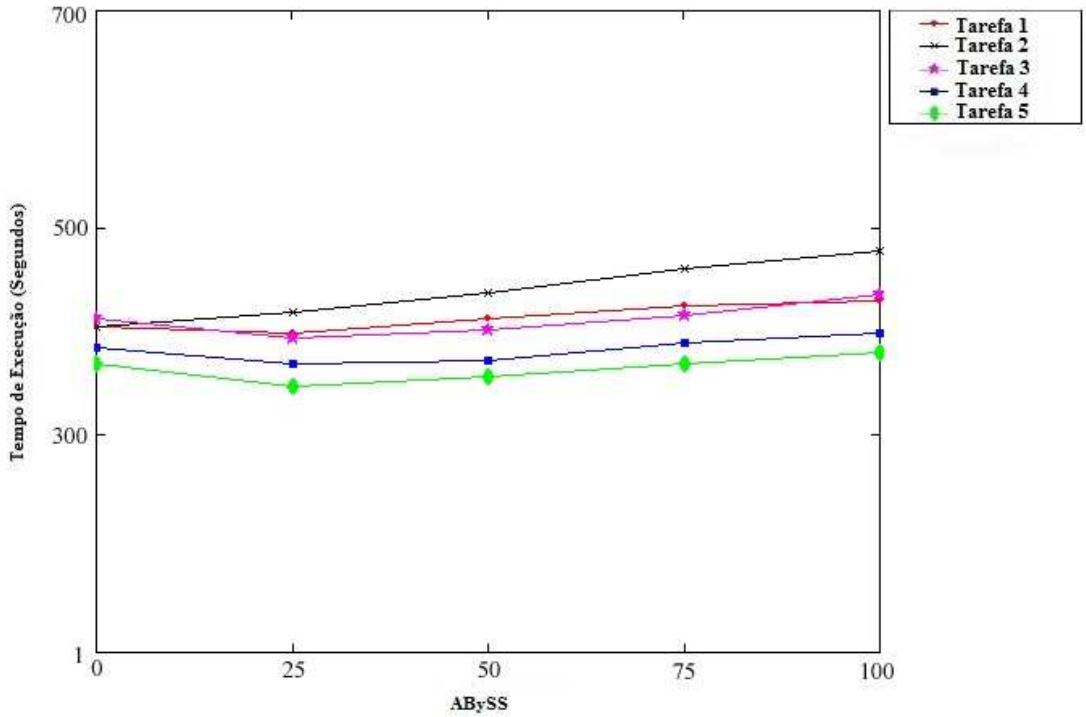

Figura 21 Processamento utilizando somente o *ABySS*

Na Figura 22, o escalonador *sheduler* foi comparado às políticas de escalonamento mais utilizadas em grades computacionais a *WQR*, a *suferage* e a *dynamic FPLTF*, descritas nos capítulos anteriores; de acordo com os testes realizados o algoritmo desenvolvido se mostrou um pouco mais lento que o *suferage* no início, mas na média das simulações acabou sendo um pouco mais rápido que os outros algoritmos, isso ocorreu principalmente, no caso do *suferage*, pela grande quantidade de acessos ao escalonador durante o processamento, causando um impacto no tempo total de execução das tarefas; em relação ao *WQR*, o *sheduler* se mostrou mais eficiente principalmente por conta da heterogeneidade das tarefas utilizadas, já que, o *WQR* na sua fase de replicação utiliza bastante tempo quando se trata de tarefas heterogêneas, e em relação ao *Dynamic FPLTF*, o *sheduler* se mostrou mais eficiente por conta da grande quantidade de dados sobre o ambiente exigido pelo algoritmo, como os testes foram realizados através de simulação, as informações que foram colocadas de forma aleatórias o que prejudicou a eficiência do processo de escalonamento como um todo.

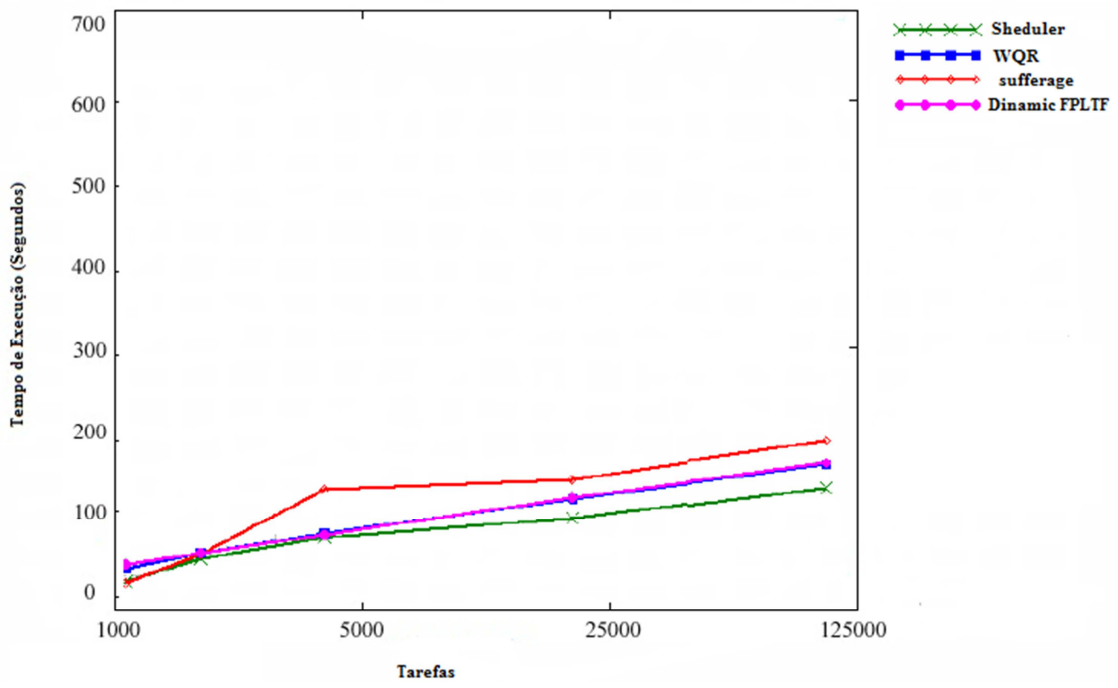

Figura 22 - *Sheduler* comparado com outras políticas de escalonamento.

7. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este capítulo apresenta as conclusões gerais do trabalho e apresenta algumas possibilidades de trabalhos futuros. A Seção 7.1 apresenta as conclusões gerais do trabalho, a Seção 7.2 apresenta os trabalhos futuros.

7.1 CONCLUSÕES GERAIS

A principal contribuição desta dissertação foi propor o s*heduler*, um algoritmo de escalonamento de tarefas para grades computacionais usado na reconstrução do genoma com o *software ABySS* para ajudar e facilitar o processo de mapeamento do DNA, mais especificamente na etapa de montagem do genoma, já que esta etapa é considerada o grande "gargalo" de todo o processo de mapeamento, pois lida com grande volume de dados e necessita de um grande poder computacional. Usando as técnicas de grades computacionais aliadas ao escalonador desenvolvido, os testes com as simulações se mostraram satisfatórias, pois atingiram o objetivo principal do escalonador de descentralizar o processamento dos dados e minimizar o tempo da montagem como um todo.

O algoritmo mostrou ser eficaz em simulações com tarefas heterogêneas já que os dados foram processados de forma correta e, não ocorreu nenhum impacto sobre a performance dos processadores. No entanto, quando a granularidade estava alta identificou-se um pequeno desequilíbrio, causando uma leve queda no desempenho do processador. Os testes foram realizados com um grupo heterogêneo de máquinas e o escalonador respondeu eficazmente ao desafio de processamento, mesmo quando havia baixa no desempenho devido a uma máquina lenta ter recebido uma tarefa grande, pois os outros processadores acabavam compensando a ligeira perda de tempo na tarefa como um todo no final da montagem.

Ainda como contribuição desta dissertação foi publicado um artigo sobre o escalonador, na revista *Frontiers in Genetics* 2012, apresentando os resultados sobre como a utilização de escalonadores em grades computacionais pode melhorar os estudos na área da bioinformática, e principalmente quando grande quantidade de dados precisam ser processados.

7.2 TRABALHOS FUTUROS

Como trabalho futuro, pretende-se implementar um módulo de prioridade no *sheduler*, com o intuito de priorizar as tarefas mais pesadas para os processadores com maior poder computacional, evitando assim, a queda de performance dos processadores.

Pretende-se fazer a implantação e teste do *sheduler* desenvolvido na nova versão distribuída do *software* de montagem Velvet, para verificar seu comportamento em outros *softwares* de montagem de genomas.

Pretende-se adaptar o *sheduler* para realizar testes reais in loco na Universidade Federal do Pará, utilizando o *cluster Penguin*, implantado recentemente no Instituto de Ciências Biológicas, com o objetivo de verificar o desempenho do *sheduler* em ambientes com situações reais de mapeamento do DNA.

Após o término dos testes e implementação de novos módulos, o *sheduler* será disponibilizado para que possa ser utilizado pela comunidade acadêmica em geral.

REFERÊNCIAS

Abbas, A. (2003). *Grid Computing: a Practical Guide to Technology and Applications*. Charles River Media, Inc.

Ahrnad, I. (1995). Resource Management in Parallel and Distributed Systems with Dynamic Scheduling. Concurrency: Practice and Experience, v. 7, p. 587–590.

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

Bateman A, Wood M. (2009). Cloud computing. Bioinformatics. 25(12):1475.

Batista, Daniel Macêdo. Escalonamento de Tarefas Dependentes para Grades Robustos às Incertezas das Informações de Entrada. PhD thesis, Unicamp - Universidade Estadual de Campinas. Instituto de Computação, Campinas, São Paulo, Brasil, Fevereiro 2010.

Batzoglou, S., Jaffe, D.B., Stanley, K., Butler, J., Gnerre, S., Mauceli, E., Berger, B., Mesirov, J.P., Lander, E.S. (2002). ARACHNE: a whole-genome shotgun assembler, Genome Res. 12:177–189.

Berman, F., Fox, G., and Hey, A. J. G. (2003). Grid Computing: Making the Global Infrastructure a Reality.1st edition. New York: John Wiley & Sons.

Bittencourt, L. F., and Madeira, E. R. M. (2006). A dynamic approach for scheduling dependent tasks on the Xavantes grid middleware. In Proceedings of the 4th international workshop on Middleware for grid computing (MCG 2006). ACM Press.

Blankenberg, D, Gordon, A, Von Kuster, G, Coraor, N, Taylor, J, Nekrutenko, A. (2010). Manipulation of FASTQ data with Galaxy. Bioinformatics. 26:1783-5.

Casanova, H. (2001). *Simgrid*: a toolkit for the simulation of application scheduling. Proceedings First IEEEACM International Symposium on Cluster Computing and the Grid: 430-437.

Catanho, M., Miranda, AB. (2007). Comparando genomas: bancos de dados e ferramentas computacionais para a análise comparativa de genomas procarióticos. *RECIIS.* Rio de Janeiro, v.1, n.2, Sup.1, p.Sup335-Sup358.

Chiapello, H., Bourgait, I., Sourivong, F., Heuclin, G., Gendrault-Jacquemard, A., Petit, MA., El., Karoui, M. (2005). Systematic determination of the mosaic structure of bacterial genomes: species backbone versus strain-specific loops. BMC Bioinformatics, 6 : 171.

Cole, C. G., McCann, O. T., Oliver, J. E. C. K., Willey, D., Gribble, S. M., Yang, F., McLaren, K., Rogers, J., Ning, Z., Beare, D. M., et al.. (2008). Finishing the finished human chromosome 22 sequence. Genome Biology. 9:R78.

Da Silva, D. P., Cirne, W., and Brasileiro, F. V. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. In Euro-Par, pages 169-180, 2003.

Dong, Fangpeng and Selim G. Akl. Scheduling algorithms for grid computing: State of art and open problems. Technical report, School of Computing, Queen's University, Kingston, Ontario, 2006.

Ewing B, Green P. (1998). Base-calling of automated sequencer traces using phred. II. Error probabilities. Genome Research. 8(3):186-194.

Fagin, B., Watt, J. G., and Gross, R. (1992). A special-purpose processor for gene sequence analysis. Computer applications and biosciences. 9(2):221–226.

Foster, I. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Ed. Fran Berman, Geoffrey C Fox, and Anthony J G Hey. International Journal of High Performance Computing Applications. 15(3): 200-222.

Foster, I., Kesselman. C., Nick, J. M., and Tuecke, S.. (2002). Grid services for distributed system integration. Computer. 35(6): 37-46.

Foster, I., and Kesselman, C. (2004). Grid 2: Blueprint for a New Computing Infrastructure. 2nd edition. San Francisco: Morgan Kaufmann.

Foster, I., Zhao, Y., Raicu, I., and Lu, S.. (2008). Cloud Computing and Grid Computing 360-Degree Compared. 2008 Grid Computing Environments Workshop. 5: 1-10.

Franco, B. P., Spolon, R., Cavenaghi, A. M., Lobato, S. R. Biblioteca de Escalonamento de Tarefas em Grid Computacional – LIBTS, 2011.

Gavaghan, D.J, A.C. Simpson, S. Lloyd, D.F. Mac Randal, and D.R.S. Boyd. Towards a grid infrastructure to support integrative approaches to biological research. Phil. Trans.R. Soc. A, 363(1833):1829 – 1841, August 2005.

Gibas, C., Jambeck, P. Developing Bioinformatics Computer Skill. 1ª edição. O'Reilly & Associates, Inc. 2001.

Green, P. (1994). Documentation for Phrap.

Higgins, D. and Sharp, P. (1988). Clustal: a package for performing multiple sequence alignment on a microcomputer. Gene. 73:237–244.

Imelfort, M., Batley, J., Grimmond, S., Edwards, D. (2009). Genome sequencing approaches and successes. *Methods Mol. Biol.* 513: 345-358.

Jain, H. (1991). The Art of Computer Systems Performance Analysis: Tecniques for Experimental Design Measurement, Simulation, and Modeling. Wiley. Ahrnad, I. (1995). Resource Management in Parallel and Distributed Systems with Dynamic Scheduling. Concurrency: Practice and Experience, v. 7, p. 587–590.

Lander ES, Linton LM, Birren B, Nusbaum C, Zody MC, Baldwin J, Devon K, Dewar K, Doyle M, FitzHugh W, Funke R, Gage D, Harris K, Heaford A, Howland J, Kann L, Lehoczky J, LeVine R, McEwan P, McKernan K, Meldrim J, Mesirov JP, Miranda C, Morris W, Naylor J, Raymond C, Rosetti M, Santos R, Sheridan A, Sougnez C, Stange-Thomann N, Stojanovic N, Subramanian A, Wyman D, Rogers J, Sulston J, Ainscough R, Beck S, Bentley D, Burton J, Clee C, Carter N, Coulson A, Deadman R, Deloukas P, Dunham A, Dunham I, Durbin R, French L, Grafham D, Gregory S, Hubbard T, Humphray S, Hunt A, Jones M, Lloyd C, McMurray A, Matthews L, Mercer S, Milne S, Mullikin JC, Mungall A, Plumb R, Ross M, Shownkeen R, Sims S, Waterston RH, Wilson RK, Hillier LW, McPherson JD, Marra MA, Mardis ER, Fulton LA, Chinwalla AT, Pepin KH, Gish WR, Chissoe SL, Wendl MC, Delehaunty KD, Miner TL, Delehaunty A, Kramer JB, Cook LL, Fulton RS, Johnson DL, Minx PJ, Clifton SW, Hawkins T, Branscomb E, Predki P, Richardson P, Wenning S, Slezak T, Doggett N, Cheng JF, Olsen A, Lucas S, Elkin C, Uberbacher E, Frazier M, Gibbs RA, Muzny DM, Scherer SE, Bouck JB, Sodergren EJ, Worley KC, Rives CM, Gorrell JH, Metzker ML, Naylor SL, Kucherlapati RS, Nelson DL, Weinstock GM, Sakaki Y, Fujiyama A, Hattori M, Yada T, Toyoda A, Itoh T, Kawagoe C, Watanabe H, Totoki Y, Taylor T, Weissenbach J, Heilig R, Saurin W, Artiguenave F, Brottier P, Bruls T, Pelletier E, Robert C, Wincker P, Smith DR, Doucette-Stamm L, Rubenfield M, Weinstock K, Lee HM, Dubois J, Rosenthal A, Platzer M, Nyakatura G, Taudien S, Rump A, Yang H, Yu J, Wang J, Huang G, Gu J, Hood L, Rowen L, Madan A, Qin S, Davis RW, Federspiel NA, Abola AP, Proctor MJ, Myers RM, Schmutz J, Dickson M, Grimwood J, Cox DR, Olson MV, Kaul R, Raymond C, Shimizu N, Kawasaki K, Minoshima S, Evans GA, Athanasiou M, Schultz R, Roe BA, Chen F, Pan H, Ramser J, Lehrach H, Reinhardt R, McCombie WR, de la Bastide M, Dedhia N, Blöcker H, Hornischer K, Nordsiek G, Agarwala R, Aravind L, Bailey JA, Bateman A, Batzoglou S, Birney E, Bork P, Brown DG, Burge CB, Cerutti L, Chen HC, Church D, Clamp M, Copley RR, Doerks T, Eddy SR, Eichler EE, Furey TS, Galagan J, Gilbert JG, Harmon C, Hayashizaki Y, Haussler D, Hermjakob H, Hokamp K, Jang W, Johnson LS, Jones TA, Kasif S, Kaspryzk A, Kennedy S, Kent WJ, Kitts P, Koonin EV, Korf I, Kulp D, Lancet D, Lowe TM, McLysaght A, Mikkelsen T, Moran JV, Mulder N, Pollara VJ, Ponting CP, Schuler G, Schultz J, Slater G, Smit AF, Stupka E, Szustakowski J, Thierry-Mieg D, Thierry-Mieg J, Wagner L, Wallis J, Wheeler R, Williams A, Wolf YI, Wolfe KH, Yang SP, Yeh RF, Collins F, Guyer MS, Peterson J, Felsenfeld A, Wetterstrand KA, Patrinos A, Morgan MJ, de Jong P, Catanese JJ, Osoegawa K, Shizuya H, Choi S, Chen YJ. (2001). Initial sequencing and analysis of the human genome. Nature. 409(6822):860-921.

Legrand, A., Marchal, L., and Casanova, H.. (2003). Scheduling distributed applications: the *SimGrid* simulation framework. Third IEEEACM International Symposium on Cluster Computing and the Grid (CCGrid 2003). Proceedings: 138-145.

Lemos, M., Basílio, A., Casanova, A. (2003). Um Estudo dos Algoritmos de Montagem de Fragmentos de DNA. PUC Rio.

Lesk, AM. Introdução à Bioinformática. (Ed.). Artmed. 2ª edição. 2008.

Li, H., Homer, N. (2010). A survey of sequence alignment algorithms for next-generation sequencing. Briefings Bioinformatics. 11:181–197.

Maheswaran, M. S. Ali and H. Siegel et al. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. Heterogeneous Computing Workshop, 1999.

McCallum D & Smith M. (1977). Computer processing of dna sequence data. J. Mol. Biol. 116:29-30.

Menascé, D. Saha D. and S. Porto et al. *Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architectures*. Journal of Parallel and Distributed Computing, pp. 1-18. 1995.

Miller, R.J., Koren, S., Sutton, G. (2010). Assembly algorithms for next-generation sequencing data. Genomics.

Myers, E.W., Sutton, G.G., Delcher, A.L., Dew, I.M., Fasulo, D.P., Flanigan, M.J., Kravitz, S.A., Mobarry, C.M., Reinert, K.H., Remington, K.A., Anson, E.L., Bolanos, R.A., Chou, H.H., Jordan, C.M., Halpern, A.L., Lonardi, S., Beasley, E.M., Brandon, R.C., Chen, L.,. Dunn, P.J, Lai, Z., Liang, Y., Nusskern, D.R., Zhan, M., Zhang, Q., Zheng, X., Rubin, G.M., Adams, M.D., Venter, J.C. (2000). A whole-genome assembly of Drosophila, Science. 287: 2196–2204.

Morgan, M. et al. (2009). ShortRead: a bioconductor package for input, quality assessment and exploration of high-throughput sequence data. Bioinformatics. 25:2607–2608.

Morozova, O., and Marra, M. A. (2008). Applications of next-generation sequencing technologies in functional genomics. Genomics. 92(5):255-264.

Needleman, S. and Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology. 48:443–453.

Pevzner, P.A., Tang, H., Waterman, M.S. (2001). An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences. 98(17):9748-9753.

Pevzner, P. A. and Tang, H. (2001). Fragment assembly with double barreled data. Bioinformatics. 17:S225–S233.

Phillippy, A. M., Schatz, M. C., and Pop, M. (2008). Genome assembly forensics: finding the elusive mis-assembly. Genome Biology. 9:R55.

Ronaghi M. 2001. Pyrosequencing Sheds Light on DNA Sequencing. Genome Research. 11(1): 3-11.

Salmela, L. (2010).Correction of sequencing errors in a mixed set of reads. Bioinformatics. 26:1284-1290.
Sanger F, Air GM, Barrell BG, Brown NL, Coulson AR, Fiddes CA, Hutchison CA, Slocombe PM & Smith M. (1977). Nucleotide sequence of bacteriophage phi x174 dna. Nature. 265:687-695.

Sanger F & Coulson AR. (1975). A rapid method for determining sequences in dna by primed synthesis with dna polymerase. *J. Mol. Biol.* 94:441-448.

Schröder, J., Schröder, H., Puglisi, SJ., Sinha, R., Schmidt, B. (2009). SHREC: a short-read error correction method. Bioinformatics. 25: 2157-63.

Silva A. et al. (2011). Complete Genome Sequence of Corynebacterium pseudotuberculosis I19, a Strain Isolated from a Cow in Israel with Bovine Mastitis. Journal of Bacteriology. 193(1):323-324.

Simpson, J.T., Wong, K., Jackman, S.D., Schein, J.E., Jones, S.J., and Birol, I. (2009). *ABySS*: a parallel assembler for short read sequence data. Genome research. 19(6):1117-23.

SimGrid. "*SimGrid* Project. Toolkit for simulation of distributed applications in heterogeneous distributed environments". (2012). URL: http://*simgrid*.gforge.inria.fr/

Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. Journal of Molecular Biology. 147:195–197.

Staden R, Beal KF & Bonfield JK. The staden package, 1998. (2000). *Methods Mol. Biol.* 132:115-130.

The International Human Genome Sequencing Consortium (2001). Initial sequencing and analysis of the human genome. Nature. 409:860–921.

Venter JC, Adams MD, Myers EW, Li PW, Mural RJ, Sutton GG, Smith HO, Yandell M, Evans CA, Holt RA, Gocayne JD, Amanatides P, Ballew RM, Huson DH, Wortman JR, Zhang Q, Kodira CD, Zheng XH, Chen L, Skupski M, Subramanian G, Thomas PD, Zhang J, Gabor Miklos GL, Nelson C, Broder S, Clark AG, Nadeau J, McKusick VA, Zinder N, Levine AJ, Roberts RJ, Simon M, Slayman C, Hunkapiller M, Bolanos R, Delcher A, Dew I, Fasulo D, Flanigan M, Florea L, Halpern A, Hannenhalli S, Kravitz S, Levy S, Mobarry C, Reinert K, Remington K, Abu-Threideh J, Beasley E, Biddick K, Bonazzi V, Brandon R, Cargill M, Chandramouliswaran I, Charlab R, Chaturvedi K, Deng Z, Di Francesco V, Dunn P, Eilbeck K, Evangelista C, Gabrielian AE, Gan W, Ge W, Gong F, Gu Z, Guan P, Heiman TJ, Higgins ME, Ji RR, Ke Z, Ketchum KA, Lai Z, Lei Y, Li Z, Li J, Liang Y, Lin X, Lu F, Merkulov GV, Milshina N, Moore HM, Naik AK, Narayan VA, Neelam B, Nusskern D, Rusch DB, Salzberg S, Shao W, Shue B, Sun J, Wang Z, Wang A, Wang X, Wang J, Wei M, Wides R, Xiao C, Yan C, Yao A, Ye J, Zhan M, Zhang W, Zhang H, Zhao Q, Zheng L, Zhong F, Zhong W, Zhu S, Zhao S, Gilbert D, Baumhueter S, Spier G, Carter C, Cravchik A, Woodage T, Ali F, An H, Awe A, Baldwin D, Baden H, Barnstead M, Barrow I, Beeson K, Busam D, Carver A, Center A, Cheng ML, Curry L, Danaher S, Davenport L, Desilets R, Dietz S, Dodson K, Doup L, Ferriera S, Garg N, Gluecksmann A, Hart B, Haynes J, Haynes C, Heiner C,

Hladun S, Hostin D, Houck J, Howland T, Ibegwam C, Johnson J, Kalush F, Kline L, Koduru S, Love A, Mann F, May D, McCawley S, McIntosh T, McMullen I, Moy M, Moy L, Murphy B, Nelson K, Pfannkoch C, Pratts E, Puri V, Qureshi H, Reardon M, Rodriguez R, Rogers YH, Romblad D, Ruhfel B, Scott R, Sitter C, Smallwood M, Stewart E, Strong R, Suh E, Thomas R, Tint NN, Tse S, Vech C, Wang G, Wetter J, Williams S, Williams M, Windsor S, Winn-Deen E, Wolfe K, Zaveri J, Zaveri K, Abril JF, Guigó R, Campbell MJ, Sjolander KV, Karlak B, Kejariwal A, Mi H, Lazareva B, Hatton T, Narechania A, Diemer K, Muruganujan A, Guo N, Sato S, Bafna V, Istrail S, Lippert R, Schwartz R, Walenz B, Yooseph S, Allen D, Basu A, Baxendale J, Blick L, Caminha M, Carnes-Stine J, Caulk P, Chiang YH, Coyne M, Dahlke C, Mays A, Dombroski M, Donnelly M, Ely D, Esparham S, Fosler C, Gire H, Glanowski S, Glasser K, Glodek A, Gorokhov M, Graham K, Gropman B, Harris M, Heil J, Henderson S, Hoover J, Jennings D, Jordan C, Jordan J, Kasha J, Kagan L, Kraft C, Levitsky A, Lewis M, Liu X, Lopez J, Ma D, Majoros W, McDaniel J, Murphy S, Newman M, Nguyen T, Nguyen N, Nodell M, Pan S, Peck J, Peterson M, Rowe W, Sanders R, Scott J, Simpson M, Smith T, Sprague A, Stockwell T, Turner R, Venter E, Wang M, Wen M, Wu D, Wu M, Xia A, Zandieh A, Zhu X. (2001). The sequence of the human genome. Science. 291:1304-1351.

Zerbino, D. (2009). Genome assembly and comparison using de Bruijn graphs. Ph.D. thesis, University of Cambridge.