UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

BRUNO RECKZIEGEL FILHO

Aplicação do MapReduce na Análise de Mutações Gênicas de Pacientes

Trabalho de Conclusão.

Prof. Dr. Cláudio Fernando Resin Geyer
Orientador

Porto Alegre, julho de 2013.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

DNA	Deoxyribonucleic Acid
RNA	Ribonucleic Acid
NGS	Next Generation Sequence
BDs	Bancos de Dados, Bases de Dados
OMIM	Online Mendelian Inheritance in Man
CCDS	Consensus CDS project
MR	MapReduce
MB	Megabytes
GB	Gigabytes

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O avanço obtido com o desenvolvimento de técnicas rápidas para o sequenciamento de DNA e a comercialização de máquinas sequenciadoras, permitiram vários progressos na área da genética médica. Porém, devido à grande quantia de dados produzidos por tais máquinas, métodos e programas que façam a análise de sequenciamento eficientemente e em um curto espaço de tempo são indispensáveis. Além disso, aplicações que façam o diagnóstico clínico de pacientes são vistas com extremo interese por parte de pesquisadores e médicos.

O MapReduce é um modelo de computação intensiva em dados que possibilita o tratamento de dados intensivos em um sistema de arquivos distribuído, além de abstrair o paralelismo de tarefas, através do uso de duas funções básicas (Map e Reduce), e permitir o controle de falhas. Considerando a inexistência de dependência entre tais dados, arquivos longos de todos tipos são bem aceitos para serem analisados neste contexto, sendo desmembrados em tamanhos menores e manipulados por diversas máquinas. Portanto, o uso desse modelo acaba se tornando uma possível solução viável para o propósito de análise dos dados produzidos por sequenciadores.

Considerando tais fatos, este trabalho de conclusão de graduação objetivou o desenvolvimento de um aplicativo MR , em conjunto com pesquisadores do Grupo de Processamento Paralelo e Distribuído (GPPD) da Universidade Federal do Rio Grande do Sul e pesquisadores do Hospital de Clínica de Porto Alegre (HCPA), que auxilie no diagnóstico clínico de pacientes através da automatização da análise das sequências genéticas desses pacientes (providas por máquinas sequenciadoras) e que vise a criação de uma solução escalável, considerando esse grande volume de dados a ser analisado.

Palavras-Chave: MapReduce, Computação Intensiva em Dados, Bioinformática, Análise em Genética Médica.

Application of MapReduce in the Analysis of Genetic Mutations in Patients

ABSTRACT

The advance obtained with the development of fast DNA sequencing techniques and the commercialization of sequencing machines allowed the progress of many researches in the Medical Genetics area. However, due to the big quantity of data produced by these machines, the development of methods and programs that can analyse these data efficiently and rapidly is required. Besides, diagnosis applications are viewed with extreme interest by doctors and researchers.

MapReduce is a data-intensive computing model that handles big volume of data in a distributed file system, abstracting the parallelism of tasks over these data using two basic functions (Map and Reduce) and creating a fault-tolerant system. It provides support for Big files from all types of formats, dividing these files in small pieces and distributing them to the machines being used by the architecture. Therefore, this computing model can offer a good solution to the analysis of the data volume generated by sequencing machines.

Considering these facts, the objective in this bachelor work is to develop an MR application(supported by GPPD and HCPA researchers) to assist in the clinic diagnosis of patients automatizing the analysis of the genetic sequences from these patients (provided by sequencing machines) and trying to create a scalable solution considering the great amount of data to be analyzed.

Keywords: MapReduce, Data-Intensive Computing, Bioinformatics, Medical Genetics Analysis.

1 INTRODUÇÃO

Este capítulo apresenta uma introdução ao assunto abordado neste trabalho, que é o desenvolvimento de um aplicativo que solucione um problema de análise na genética médica, de forma automatizada e escalável (considerando grandes volumes de dados de entrada). Também são discutidas as motivações e os objetivos que serão considerados.

1.1 Motivação

Como definido em (WATSON, 2007), o *DNA* (do inglês *Deoxyribonucleic Acid*) é uma molécula que codifica instruções genéticas que são utilizadas no desenvolvimento e funcionamento de todos organismos vivos conhecidos (além de muitos vírus). O processo utilizado para se obter tais dados (chamado sequenciamento de DNA) progrediu muito desde o surgimento dos primeiros métodos propostos para tal (SANGER, 1977).

A informação provida por sequências de DNA tem se tornado impresindível para o sucesso de pesquisas biológicas. Seja com o desenvolvimento de novos fármacos, produtos alimentícios, produtos agrícolas ou testes de diagnóstico clínico, a área da Biotecnologia é uma das que mais utiliza tal informação como base para seu progresso (THIEMAN et al., 2008).

Atualmente, existem diversas técnicas e máquinas que propõe soluções para o sequenciamento de DNA e nota-se que o custo delas vem decaindo consideravelmente. Segundo pesquisas feitas pelo NHGRI (National Human Genome Research Institute), a queda desses custos aconteceu, entre 2007 e 2008, devido à transição da metodologia de sequenciamento de Sanger para as de próxima geração (NGS). Na Figura 1.1 mostra-se um gráfico que evidencia a decadência do preço gasto no processo de sequenciamento de DNA, além de uma alusão à tendência de melhorias na indústria de *hardware*, definida pela Lei de Moore.

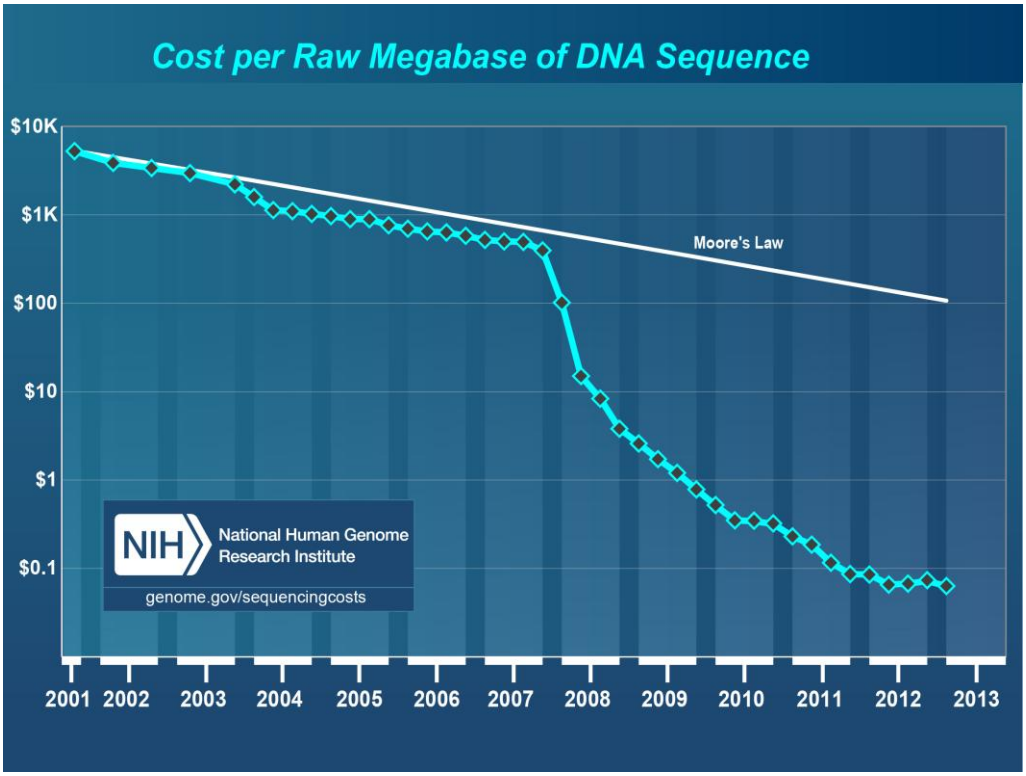

Figura 1.1: Reduções de custo no sequenciamento (NHGRI, 2013).

Além desse fator, o tempo gasto para se fazer o sequenciamento também está decaindo. Como exposto na Tabela 1.1, algumas máquinas com tecnologia de próxima geração (especificamente as que utilizam semicondutores Ion), como por exemplo o sequenciador Ion Proton (IONTORRENT, 2013b), são capazes de sequenciar milhões de pares de base em questão de horas gerando, assim, grande quantia de dados como saída. Contudo, existem poucas ferramentas que propõe uma análise desses dados, de forma automatizada e em um período curto de tempo. Como será visto a seguir, o trabalho aqui proposto foi desenvolvido com esse âmbito.

Tabela 1.1: Especificações dos sequenciadores IonTorrent.

Modelo do Sequenciador	*PGM 318*	*PI*	*PII*	*PIII*
Nº de sensores	11 M	165 M	660 M	1.2 B
Tamanho da Saída	~2 GB	~10 GB	~32 GB	~64 GB
Tempo de Execução	4~7 horas	2~4 horas	2~4 horas	2~4 horas
Tamanho médio de leitura	400 pb	200 pb	100 pb	100 pb
Nº de leituras	~5.5 M	~82 M	~330 M	~660 M

Fonte: Adaptado de (IONTORRENT, 2013a).

Com o surgimento de BDs contendo informações de teor clínico sobre mutações genéticas específicas, o desenvolvimento de sistemas computacionais que suportem médicos no diagnóstico dessas doenças em seus pacientes, através da análise de suas sequências genéticas, tem se tornado uma possível realidade.

Considerando esse contexto, o modelo de computação proposto pelo MapReduce acaba se tornando uma opção já que ele oferece, implicitamente, uma solução paralela e distribuída para o processamento de grande volume de dados, como é o caso destes sequenciadores. Além disso, as implementações existentes do modelo MR lidam automaticamente com questões como tolerância a falhas e balanceamento de carga. Desse modo, o desenvolvimento de software para tais ambientes se torna menos complexo pois os programadores não precisam considerar esses problemas em suas aplicações MR.

1.2 Objetivo

O objetivo proposto neste trabalho é o desenvolvimento de um aplicativo que auxilie médicos com a automatização do diagnóstico clínico de pacientes, através da análise de suas mutações genéticas. Além disso, seu tempo de execução deve manter-se escalável em relação à quantia de dados a ser analisada, quantia essa gerada por máquinas sequenciadoras que suportem tecnologia de semicondutores Ion. Para tanto, foi utilizada uma implementação de código aberto do modelo de computação MapReduce, chamada Hadoop (HADOOP, 2013a) (desenvolvido pela Apache Software Fundation), e algumas BDs contendo informações relevantes ao problema em questão.

1.3 Organização do Texto

O conteúdo restante deste trabalho é organizado da seguinte maneira: No capítulo 2 são apresentados conceitos referentes à Genética Médica, ao problema que deseja-se solucionar (análise do efeito de mutações em sequências genéticas de pacientes) e a algumas tecnologias relacionadas ao mesmo. No capítulo 3 é apresentado uma visão geral do modelo de programação MR e da plataforma Hadoop. Em seguida, o capítulo 4 é dedicado à descrição detalhada da solução proposta e do protótipo desenvolvido. Por fim, no capítulo 5, são apresentadas análises dos resultados obtidos, conclusões e trabalhos futuros a serem considerados.

2 GENÉTICA MÉDICA

Neste capítulo serão introduzidos os conceitos que servirão de base para a compreensão da Genética e suas aplicações no âmbito da medicina. Também serão expostas tecnologias atualmente utilizadas nessa área e os problemas que vem sendo enfrentados para que, desse modo, seja proposto o desenvolvimento de uma nova solução para um problema específico (análise do efeito de mutações em sequências genéticas de pacientes).

2.1 Genética

A Genética é uma disciplina da Biologia que envolve a ciência dos genes, da hereditariedade e da variação em organismos vivos (GRIFFITHS et al., 2000). Ela diz respeito ao processo de herança de pais para filhos, incluindo a estrutura molecular e a função dos genes, o comportamento do gene no contexto de uma célula ou organismo e a distribuição, variação e alteração do gene em populações.

Esta área de pesquisa envolve vários outros conceitos – relacionados com a composição e organização da estrutura dos genes, suas funções, etc. – que serão comentados a seguir.

2.1.1 Nucleotídeos

Nucleotídeos são moléculas biológicas que compõe a estrutura dos Ácidos Nucléicos (ALBERTS et al., 2002). Como visto na Figura 2.1, eles são compostos por uma base Purina (Adenina ou Guanina) ou por uma base Pirimidina (Timina, Citosina ou Uracil).

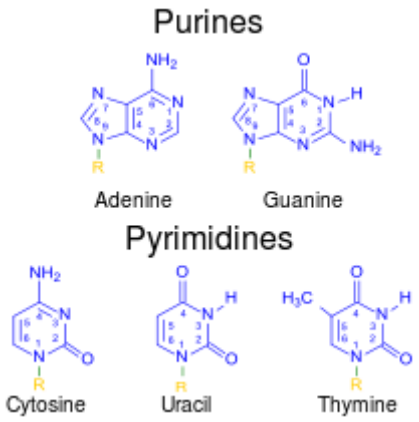

Figura 2.1: Composição base dos nucleotídeos (THIEMAN et al., 2008).

2.1.2 Ácidos Nucléicos (DNA e RNA)

Ácidos Nucléicos são grandes moléculas biológicas, essenciais para todas as formas de vida conhecidas. Eles incluem *DNA* ou *RNA* (do inglês *Ribonucleic Acid*) e são compostos por longas sequências de nucleotídeos. A Figura 2.2 mostra a estrutura, em forma de hélice, adotada tanto pelo DNA quanto pelo RNA, além das bases de nucleotídeos que os compõe. A diferença básica estrutural entre os dois está na quantia de hélices (o DNA possui duas e o RNA uma) e na troca do uso da Timina por Uracil. A sequência de nucleotídeos, formada ao longo dessas hélices, é que define a informação genética codificada pelo organismo em questão.

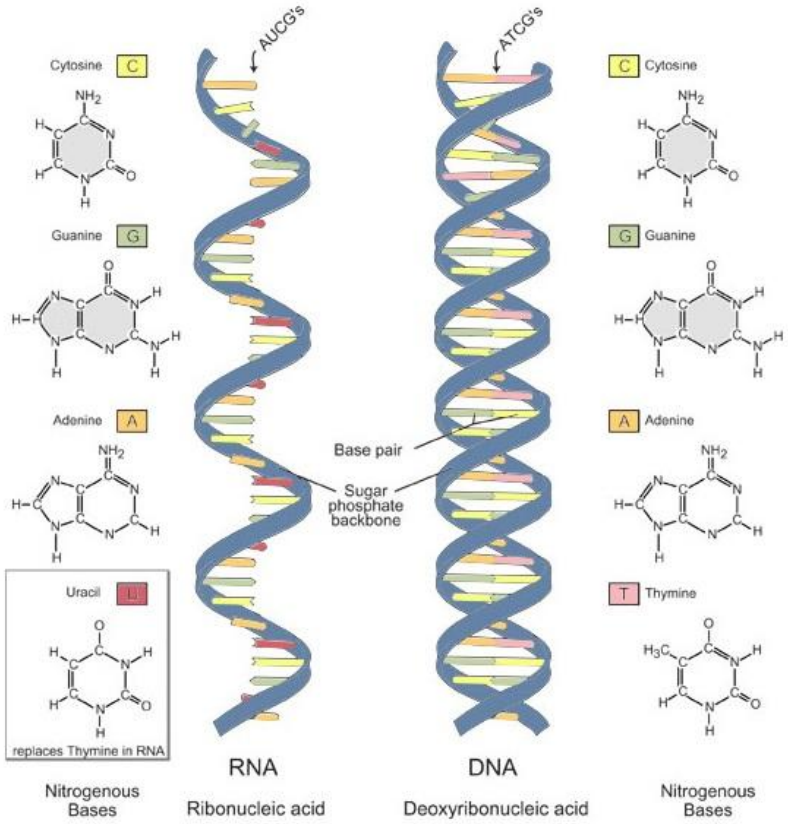

Figura 2.2.: Estrutura e composição dos Ácidos Nucléicos (THIEMAN et al., 2008).

Através do uso do conjunto de regras definidas pelo código genético, as informações genéticas, providas por sequências de DNA através do mRNA (RNA mensageiro), são traduzidas para uma sequência de aminoácidos, gerando, desse modo, uma proteína (THIEMAN et al., 2008). Nesse processo, como visto na Figura 2.3, cada tripla de nucleotídeos é chamada de códon e cada códon representa um aminoácido. Porém, é importante salientar que somente as regiões genéticas definidas como codificantes são consideradas no processo de tradução para proteínas.

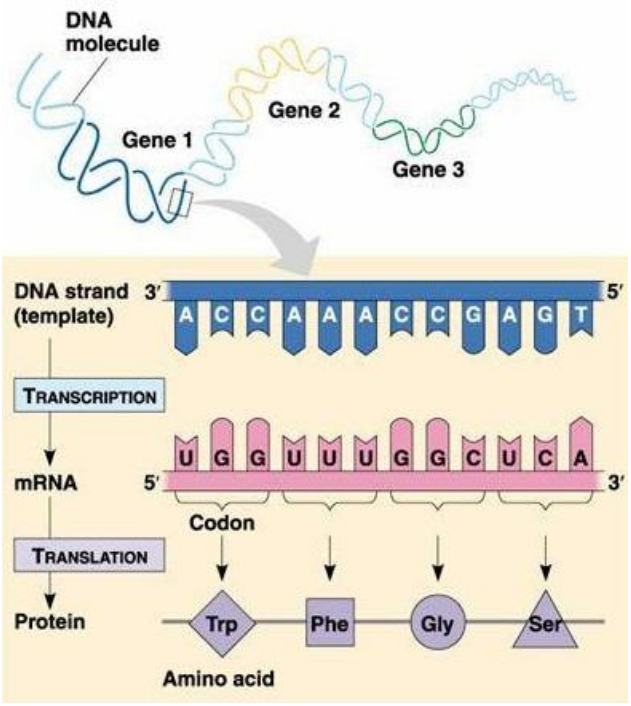

Figura 2.3: Dogma central na Biologia Molecular (THIEMAN et al., 2008).

2.1.3 O Genoma Humano e sua organização

O genoma equivale à totalidade da informação hereditária de um organismo. O genoma humano consiste em uma quantidade grande de DNA, dividida em estruturas organizadas chamadas de cromossomos. Os cromossomos equivalem a um único pedaço enrolado de DNA, contendo muitos genes (unidades basicas físicas e funcionais de herança (PEARSON, 2006)), elementos regulatórios e outras sequências de nucleotídeos. Como pode ser visto na Figura 2.4, o genoma humano contem 23 pares de cromossomos e o número estimado de genes que o compõe é de, aproximadamente, 32 mil. O número total de bases do DNA humano gira em torno de 3,6 bilhões.

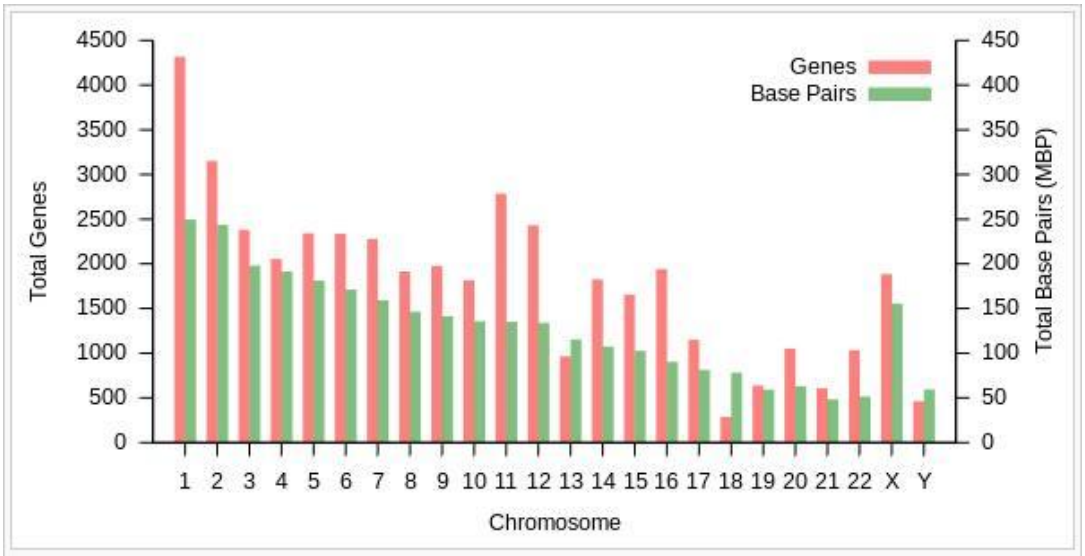

Figura 2.4: Quantia de genes nos cromossomos humanos (NUSSBAUM et al., 2007)

2.1.4 Mutação

Uma mutação é definida como uma alteração da sequência de nucleotídeos do genoma de um organismo. Mutações ocorrem devido a danos irreparáveis sofridos pelo genoma, a erros no processo de replicação ou a inserção/deleção de fragmentos de DNA por elementos genéticos móveis. Vários estudos, como o definido por (SAWYER et al., 2007), sugerem que, se uma mutação altera a proteína que é produzida por um gene, provavelmente o resultado será prejudicial para o organismo. Também é importante notar que, como destacado na Figura 2.5, algumas mutações de nucleotídeos não modificam o aminoácido gerado (ou seja, mesmo que uma mutação ocorra, a proteína produzida continua a mesma).

First Position	Second Position				Third Position
	T	C	A	G	
T	PHE	SER	TYR	CYS	T
	PHE	SER	TYR	CYS	C
	LEU	SER	stop	stop	A
	LEU	SER	stop	TRP	G
C	LEU	PRO	HIS	ARG	T
	LEU	PRO	HIS	ARG	C
	LEU	PRO	GLN	ARG	A
	LEU	PRO	GLN	ARG	G
A	ILE	THR	ASN	SER	T
	ILE	THR	ASN	SER	C
	ILE	THR	LYS	ARG	A
	MET	THR	LYS	ARG	G
G	VAL	ALA	ASP	GLY	T
	VAL	ALA	ASP	GLY	C
	VAL	ALA	GLU	GLY	A
	VAL	ALA	GLU	GLY	G

Figura 2.5: Relação do códon com seu aminoácido (THIEMAN et al., 2008).

Segundo (NUSSBAUM et al., 2007), existem três categorias de mutações: mutações que afetam o número de cromossomos da célula (chamadas de mutações genômicas), mutações que alteram a estrutura de cromossomos específicos (mutações cromossômicas), e mutações que alteram genes individualmente (mutações gênicas). Já que o foco deste trabalho está na análise de mutações gênicas, a análise do efeito das outras mutações no estado de saúde de pacientes será considerada em trabalho futuro.

2.2 Genética Clínica e Tecnologias Atuais

A genética clínica preocupa-se com o diagnóstico e controle dos aspectos médicos, sociais e psicológicos de doenças hereditárias. É fundamental que se faça um diagnóstico correto e se ofereça um tratamento adequado para a pessoa afetada, a fim da melhor compreensão da natureza e das consequências da doença em questão.

Visando ajudar o processo de diagnóstico de doenças, causadas por mutações genéticas específicas, BDs foram criados. O OMIM (do inglês *Online Mendelian Inheritance in Man*) é um exemplo de BD onde são mantidos catálogos de todas as doenças conhecidas com um componente genético, que - quando possível - os liga aos

genes relevantes do genoma humano e fornece referências para novas pesquisas e ferramentas de análise genômica de um gene catalogado (HAMOSH et al., 2005).

Outras BDs também merecem citação neste trabalho. Entre elas estão dbSNP (uma BD que mantém diversas informações sobre as mais diversas mutações gênicas específicas), PubMed (uma BD que mantém textos de cunho analítico médico, além de referências a descrições de doenças), Ensembl (uma BD que referencia o dbSNP e fornece informações adicionais sobre as mutações), CCDS (uma BD que mantém informações sobre a composição da área codificante de diversos genes) e a mais recente GeneReport (uma BD que mantém informações somente de mutações que sejam consideradas patogênicas, ou seja, diretamente relacionadas a doenças).

Com o advento de máquinas sequenciadoras de DNA, torna-se possível a análise da sequência de pacientes. Atualmente, algumas delas possibilitam tanto o sequenciamento de um conjunto específico de genes quanto o do genoma, além de tornar possível o sequenciamento da amostra de diversos pacientes ao mesmo tempo. Portanto, assim como foi ressaltado no capítulo anterior, tais máquinas acabam gerando um grande volume de dados e, desse modo, a existência de aplicações que analisem esses dados de forma automática faz-se necessária.

Neste âmbito, este trabalho propõe o desenvolvimento de um aplicativo que auxilie no diagnóstico clínico de pacientes, através da analíse automatizada de tais dados, utilizando como base de conhecimento as informações obtidas dessas BDs. Visto que essa aplicação provavelmente aplica-se ao contexto de computação intensiva em dados (ANJOS et al., 2013), decidiu-se utilizar uma implementação do paradigma proposto pelo MapReduce (DEAN; GHEMAWAT, 2004). Para a adoção do framework MapReduce, utilizou-se a implementação do sistema Hadoop.

2.3 Considerações Finais

Este capítulo proveu noções básicas sobre o campo da Genética e sua contextualização com algumas tecnologias disponíveis na área da Medicina com foco no diagnóstico clínico de pacientes. Além disso, o objetivo da aplicação apresentada neste trabalho foi devidamente esclarecido.

O próximo capítulo detalha os conceitos envolvidos com o paradigma de computação utilizado no aplicativo desenvolvido e apresenta a arquitetura do framework que será utilizado.

3 MAPREDUCE E HADOOP

Neste capítulo serão apresentadas as tecnologias que servirão como base para a execução do aplicativo proposto neste trabalho. O funcionamento de cada uma delas é exposto de maneira detalhada, a fim de permitir a melhor compreensão por parte do leitor.

3.1 MapReduce

O MapReduce é um modelo de programação paralela que foi criado pela Google para o processamento distribuído de grandes volumes de dados. Um dos seus objetivos é facilitar o desenvolvimento de aplicativos distribuídos que se enquadrem nesse perfil. Para tanto, seu modelo inspira-se em duas primitivas chamadas *Map* e *Reduce*, oriundas de linguagens funcionais como Lisp e Scheme, por exemplo.

Através do uso desse modelo em sistemas distribuídos de arquivos em *clusters*, com operações *Map* e *Reduce* definidas pelos programadores, computações são paralelizadas e distribuídas facilmente sobre a arquitetura utilizada. Originalmente proposto em (DEAN; GHEMAWAT, 2008), tal modelo surgiu devido ao fato de que, na maioria das computações, era necessário mapear fragmentos dos dados de entrada a uma chave identificadora para então processar todos os fragmentos que compartilhassem a mesma chave.

Todo o trabalho de distribuição do sistema (considerando problemas de comunicação, tolerância a falhas, concorrência, etc) acaba sendo abstraído pelo framework do MapReduce. Por isso, a única tarefa com que o programador precisa se preocupar é a implementação dessas duas funções, indicando qual a computação que será efetuada por elas. Os dados de entrada e saída dessas funções são definidos por pares <*chave*, *valor*>. Como a estrutura destes elementos depende da aplicação que será executada, também cabe ao desenvolvedor se encarregar da definição destas propriedades.

Para que o uso desse modelo fique claro para o leitor, demonstra-se na Figura 3.1 um exemplo de uso do MapReduce, no qual especifica-se uma aplicação onde o objetivo é a contagem do número de ocorrências de cada palavra em um documento. Neste pseudocódigo, cada chamada da função *Map* recebe como *chave* o número de uma linha do arquivo e como *valor* o conteúdo desta linha. A cada palavra encontrada no *valor* da linha em questão, a função emite outro par <*chave*, *valor*> onde a *chave* é a palavra em si e o *valor* é a constante 1 (um). Já a função *Reduce* recebe como entrada um par <*chave*, *valor*> onde a *chave* é uma palavra e o *valor* é um iterador sobre todos os valores associados com a palavra em questão, emitidos por funções *Map*. Finalmente, todos os valores são somados pela função *Reduce* e um par <*chave*, *valor*> é emitido,

contendo como *chave* a palavra em questão e como *valor* o número total de suas ocorrências no arquivo de entrada.

```
1   function map(Integer chave, String valor):
2     // chave: número da linha no arquivo.
3     // valor: conteúdo da linha.
4     conteudo_linha = valor.split()
5     for each palavra in conteudo_linha:
6       emit (palavra, 1)
7
8   function reduce(String chave, Iterator valores):
9     // chave: palavra emitida por uma função Map
10    // valores: uma lista de valores emitidos para a chave
11    num_ocorrencias = 0
12    for each numero in valores:
13      num_ocorrencias += 1
14    emit (chave, num_ocorrencias)
```

Figura 3.1: Pseudocódigo de programação no MapReduce

A seguir, na Figura 3.2, apresenta-se um fluxo de execução e dados da aplicação citada anteriormente. Como entrada é utilizado um arquivo texto com apenas quatro linhas, cujos conteúdos são “mais de”, “oito mil”, “menos de” e “oito mil”.

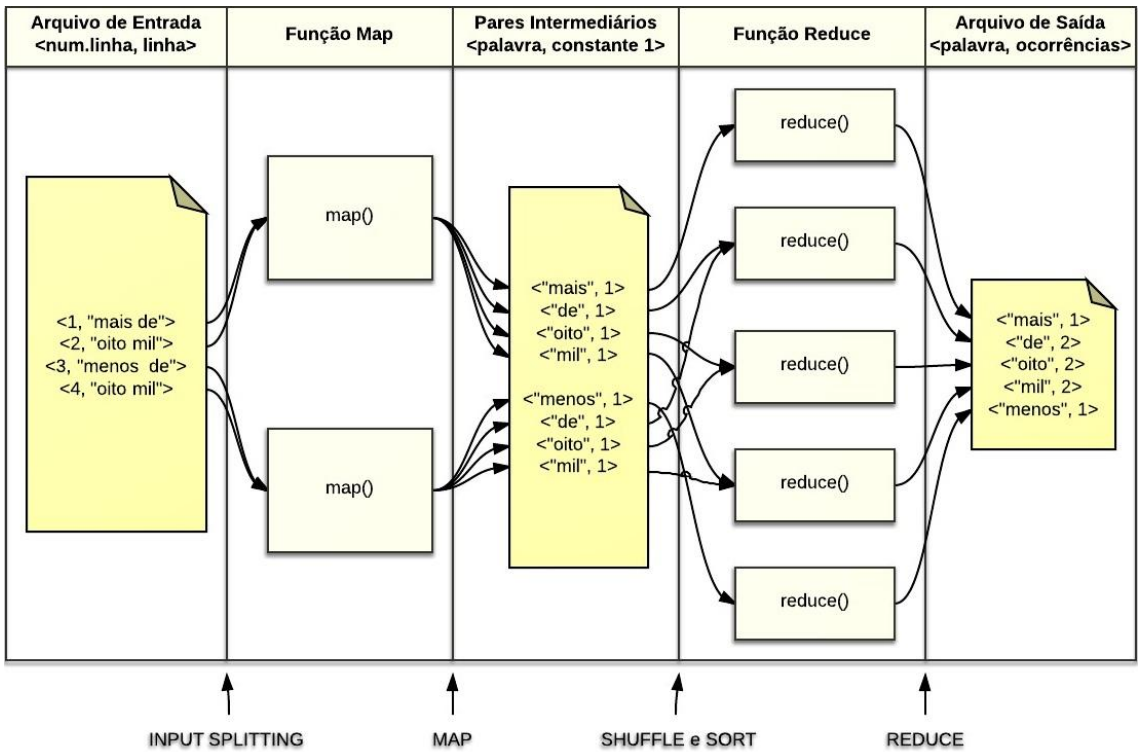

Figura 3.2: Fluxo simplificado de execução e dados da aplicação *WordCount*.

Neste exemplo, a cada duas linhas de texto do arquivo de entrada é feita uma chamada à função *Map*, gerando um total de duas funções desse tipo. Considerando o pseudocódigo anterior e o exemplo em questão, cada função *Map* gera quatro pares <*chave*, *valor*> intermediários, um para cada palavra encontrada nas linhas de texto recebidas pela função. Na fase indicada como *Shuffle e Sort*, todos os pares intermediários que estão associados a uma mesma chave (neste caso, uma palavra do documento de entrada) são passados para uma chamada da função *Reduce*. Portanto, devido à existência de cinco palavras distintas (“mais”, “de”, “oito”, “mil” e “menos”), cinco funções *Reduce* são executadas, sendo que cada uma delas retorna a soma de todos os valores presentes na lista de pares recebidos. Finalmente, os pares gerados na fase *Reduce* são armazenados em um arquivo de saída.

3.1.1 Arquitetura do MapReduce

O modelo MR proposto por (DEAN; GHEMAWAT, 2008) foi desenvolvido para ser utilizado em grandes clusters do Google, compostos por computadores de baixo a médio custo, interligados por rede. Neste contexto, estas máquinas serão referenciadas pela palavra *nós(*ou no caso de uma única máquina, *nó)* neste trabalho. Tal modelo considera a existência de dois tipos de nós básicos na sua composição: *Master* (em português, mestre) e *Worker* (em português, trabalhador).

O nó mestre funciona basicamente como o gerenciador de tarefas desse ambiente. Ele atende a requisições de execução feitas pelos usuários (denominadas *jobs* em inglês), cria tarefas (do inglês *tasks*) e delega-as aos nós trabalhadores. Estes são, então, encarregados de executar as tarefas, fazendo o uso das funções *Map* ou *Reduce* definidas pelo usuário. Portanto, nota-se que tal modelo implementa uma típica arquitetura mestre-escravo (DUBREUIL; GAGNÉ; PARIZEAU, 2006).

A arquitetura faz o uso de um sistema de arquivos distribuído no qual são armazenados os dados que serão utilizados como entrada para a execução dos *jobs* pelos *workers*. A fim de evitar a transferência excessiva de dados entre os nós componentes do sistema, os *workers* também são considerados como integrantes desse sistema de arquivos. Outro fator que também contribui para este quesito é o uso de funções do tipo combine, por proverem um mecanismo de redução dos dados gerados pelas funções *Map*.

O nó mestre também fica encarregado pela manuteção e controle dos metadados do sistema de arquivos utilizado, incluindo o espaço de nomes (namespace), o mapeamento de arquivos a *chunks* (blocos do sistema de arquivos que representam “pedaços” de arquivos) e a localização dos *chunks*. Já os trabalhadores possuem a tarefa de armazenar tais dados em formato de *chunks*. Esta característica é fundamental para o bom desempenho do sistema.

Existe uma forte ligação entre o *DFS* (do inglês *Distributed File System*) e a implementação do *MapReduce*. Tal fato ocorre pois os *workers* são também *chunkservers* (nós que armazenam dados dos *DFS*). Desse modo, quando o escalonador utilizado pelo *MapReduce* atribui uma tarefa *Map* a um *worker*, ele tenta fazê-lo para um nó que já possua, em sua unidade de armazenamento local, uma réplica dos *chunks* que devem ser processados, evitando, assim, a transferência de informações pela rede. Caso esse processo não possa ser realizado, o escalonador tenta passar a tarefa para o nó que estiver mais próximo do *chunk server* que possui tais dados (DEAN;

GHEMAWAT, 2008), a fim de não sobrecarregar a rede com a transferência de arquivos. Na subseção 3.2.3 serão apresentados mais detalhes sobre o *DFS* desenvolvido e utilizado pelo Hadoop.

3.1.2 Funcionamento do MapReduce

Além das duas funções básicas (mapeamento e redução) executadas pelo framework MapReduce, outras etapas mereçem citação. O funcionamento de quatro delas, conhecidas como *Input Splitting*, *Shuffle*, *Sort* e *Combine*, sera descrito a seguir devido à importância que têm nesse contexto.

É fundamental que o conceito de distribuição de dados entre os nós esteja claro. Neste processo (executado pelo nó *master*) os dados de entrada são divididos em blocos de tamanho fixo (no Hadoop, o valor *default* é de 64MB) e então são distribuídos entre os *workers*. Após essa operação, a submissão de *jobs* pode ser feita ao *master*.

Na fase definida como *Input splitting* (WHITE, 2010), a divisão *lógica* dos blocos de dados é feita nos nós do tipo *worker*. Utilizando a implementação do Hadoop como exemplo, cada intervalo de dados gerado nesse processo é chamado de *Input Split* (WHITE, 2010), estruturas utilizadas como entrada pelas tarefas de mapeamento. Eles são calculados em tempo de execução, pelo *Input Format* (WHITE, 2010) definido na aplicação que foi submetida ao *master*. Normalmente o tamanho desses intervalos varia de 16MB a 64MB (possuem relação com o tamanho dos blocos do *DFS* em questão). Uma relação entre o tamanho e a organização dos blocos e dos splits, utilizados pelo sistema de arquivos distribuídos do Hadoop, pode ser vista na Figura 3.3. Nota-se que cada bloco pode gerar um ou mais splits.

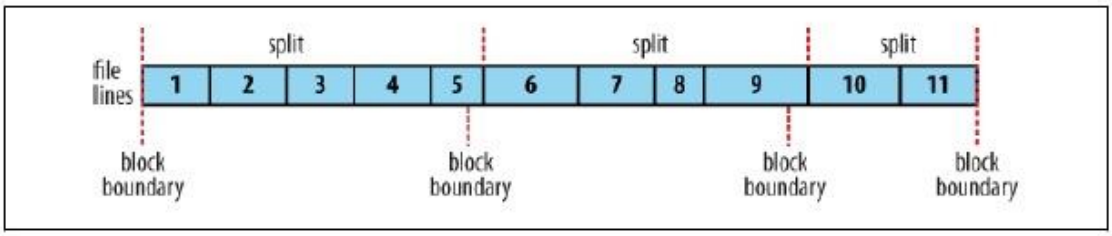

Figura 3.3: Organização dos blocos e splits no Hadoop (WHITE, 2010)

Assim que essa divisão estiver pronta, estruturas conhecidas como *Readers* (WHITE, 2010) irão processar os *Input Splits*, gerando diversos pares <chave, valor> que serão repassados diretamente, como entrada, para a tarefa de mapeamento que foi definida pelo usuário. Normalmente as implementações do *MapReduce* permitem que os desenvolvedores especifiquem seus próprios *Readers* ou utilizem os existentes.

A partir do momento que as tarefas de mapeamento começarem a produzir pares intermediários, a etapa de *Shuffle* (WHITE, 2010) começa. Nela ocorre primeiramente a criação de partições, geralmente uma para cada chave distinta que é gerada pelas funções *Map* e em seguida é feita a ordenação interna dessas partições, a fim de facilitar o posterior processamento por parte das funções *Reduce*.

Devido ao fato de que o mapeamento dos dados de entrada é, na maioria das vezes, executado em diversos *workers*, cabe ao redutor o processo de cópia e fundição das

muitas partições a ele destinadas, expalhadas pela rede. Essas funcionalidades ocorrem na fase de *Sort* (WHITE, 2010). O fluxo de dados nas fases de *Shuffle* e *Sort* é demonstrado pela Figura 3.4.

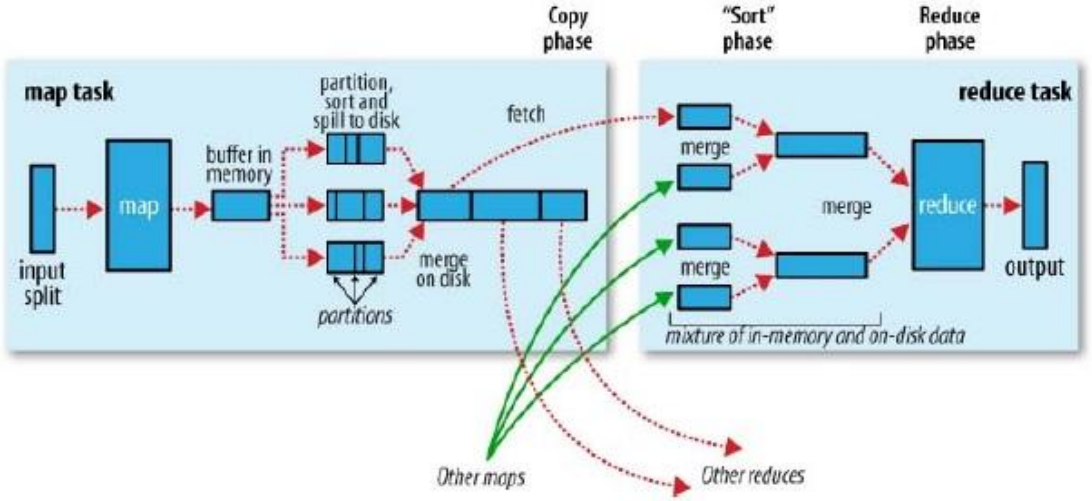

Figura 3.4: Fases de *Shuffle* e *Sort* no MapReduce (WHITE, 2010)

Visando o ganho de desempenho por parte de algumas aplicações, o uso da função *Combine* é proposto (porém não obrigatório). Seu uso ocorre nos *workers*, após a fase de ordenamento do *Shuffle*, e basicamente pré-processa os dados que serão posteriormente reduzidos, gerando assim, menor comunicação entre os nós que executam funções *Map* e os que executam funções *Reduce*. A seguir, o uso do *Combine* pela aplicação citada anteriormente é demonstrado pela Figura 3.5.

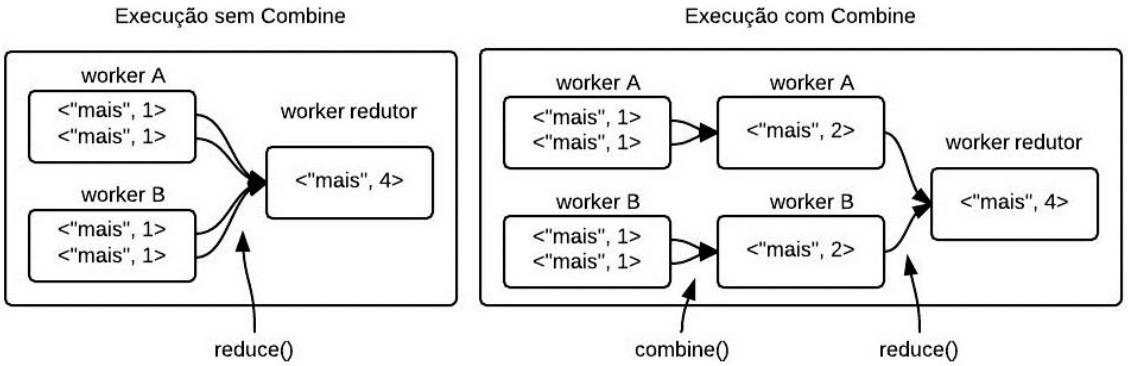

Figura 3.5: Fluxo de dados da função *Combine*

No exemplo de execução citado na figura acima, dois workers geraram dois pares intermediários, cada um, no formato <”mais”, 1>. Nota-se que no caso em que não se utiliza a função *Combine*, 4 pares são transferidos para o *worker* redutor enquanto que na presença do *Combine*, esse número é diminuído pela metade. Além disso, a carga de processamento exigida na função de *Reduce* será menor.

3.2 Hadoop

Ao contrário do framework do Google, que foi desenvolvido em C++, a implementação do Hadoop MapReduce foi feita em Java. No anexo A, detalha-se o modo como a aplicação *Wordcount* é programada em Hadoop. Na sua impementação, algumas nomenclaturas diferem das especificadas pelo framework MR. Como exemplo, citam-se as definições de *master* e *worker* identificados, respectivamente, como *JobTracker* e *TaskTracker* no Hadoop.

3.2.1 Escalonamento e Execução de Tarefas

A arquitetura do Hadoop MapReduce, assim como o framework MR proposto por (DEAN; GHEMAWAT, 2008), é baseada no modelo cliente-servidor onde os nós *worker* se comunicam, através de *heartbeats*, com o nó *master*. Desse modo, os trabalhadores enviam informações sobre seu estado atual ao nó mestre e, caso o trabalhador esteja disponível para executar tarefas, o nó mestre lhe atribui uma nova tarefa de acordo com a seguinte lógica.

- Caso ainda reste uma tarefa *Map* para ser finalizada, o nó mestre procura, na seguinte ordem, por uma tarefa

    1) que processe dados contidos localmente no *worker* (considerando que tais nós integram o HDFS).

    2) que processe dados de outro nó (neste caso a transferência de dados pela rede faz-se necessária).

    3) especulativa (caso das tarefas lentas, ainda em execução).

- Caso contrário, o nó mestre escalona as tarefas de redução. Vale notar que na redução não existe o conceito de dados locais, visto que a entrada para tais tarefas consiste dos resultados das tarefas *Map*, os quais se encontram distribuídos entre os nós do *cluster*.

Como descrito em (DEAN; GHEMAWAT, 2008), nós que degradam a performance do framework MapReduce acabam causando perda de desempenho por parte do sistema. Visto que as tarefas executadas por eles atrasam o processamento, principalmente na etapa final dos *jobs*, introduziu-se o conceito de *Backup Tasks*. Essas tarefas funcionam como cópias das tarefas que estão em andamento no final de operações de mapeamento e de redução. Desse modo, caso uma das cópias de uma tarefa seja finalizada com sucesso, as demais são encerradas.

No Hadoop, a implementação do mecanismo *Backup Tasks* também é feita, mas é chamada de execução especulativa (WHITE, 2010). Com o objetivo de identificar nós lentos, cada nó envia sinais de *heartbeat* contendo informações de seu progresso, em cerrtos períodos de tempo, para o nó mestre e ele faz o re-escalonamento de tarefas especulativas.

3.2.2 HDFS

Além da implementação MR proposta pelo projeto Hadoop, ele provê um sistema de arquivos distribuído como alternativa ao GFS. Tal solução, mais conhecida como HDFS (Hadoop Distributed File System), oferece um ambiente apto para rodar em hardware de prateleira e tolerante a falhas (HADOOP, 2013b). Portanto, o HDFS é o sistema de arquivos padrão utilizado pelo Hadoop.

Na Figura 3.6, a arquitetura do HDFS é demonstrada (HADOOP, 2013c). Basicamente o HDFS possui um nó mestre (denominado *NameNode*) responsável por atender requisições dos usuários e gerenciar a localização dos dados, distribuído-os entre os vários nós trabalhadores (*DataNodes*) os quais, por sua vez, armazenam os dados e transmitem-os caso usuários façam requisições de acesso aos dados.

Como citado em (WHITE, 2010), algumas propriedades são importantes para se obter o melhor desempenho do HDFS. Uma delas refere-se ao fato de que o sistema de arquivos procura sempre manter os dados igualmente distribuídos entre os *DataNodes*, redistribuindo os blocos de dados caso necesssário. Outra que influencia bastante no desempenho do MapReduce, é mais conhecida como *Rack Awareness*. Através dessa propriedade, o HDFS consegue identificar os *DataNodes* que pertencem ao mesmo rack e, por isso, ele consegue fazer a distribuição de réplicas de forma mais inteligente.

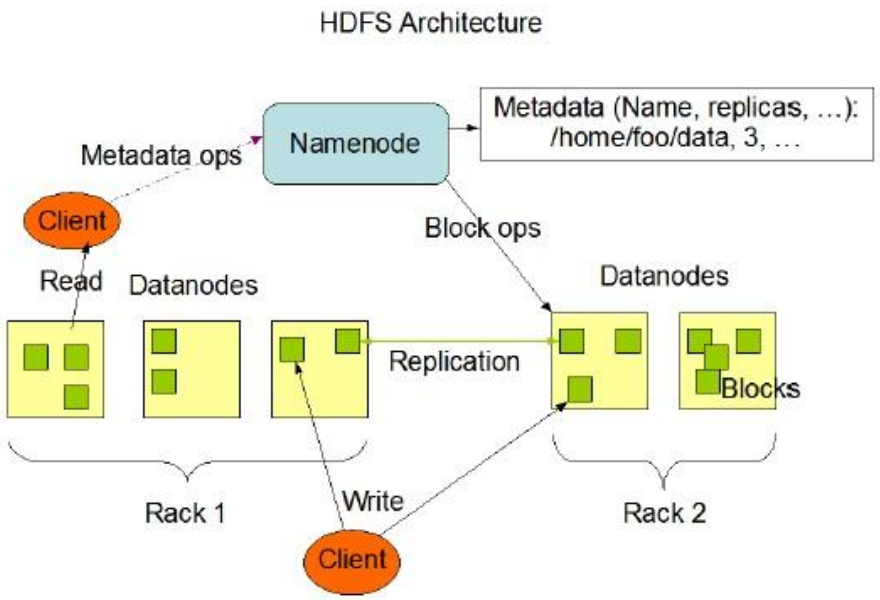

Figura 3.6: Arquitetura do HDFS (HADOOP, 2013b)

3.3 Considerações Finais

Neste capítulo, a composição e funcionamento das tecnologias envolvidas no framework MR foram detalhadamente apresentadas. Além disso, demonstrou-se que a implementação do Hadoop MapReduce possibilita o uso de uma solução similar à proposta por (DEAN; GHEMAWAT, 2008).

No próximo capítulo será demonstrada a metodologia utilizada no desenvolvimento da aplicação MR anteriormente proposta e o protótipo resultante.

4 METODOLOGIA E PROTOTIPAÇÃO

Neste capítulo serão apresentadas as decisões tomadas no desenvolvimento da aplicação proposta neste trabalho, como esta tarefa será realizada, além dos recursos utilizados neste processo e no ambiente de testes.

4.1 Descrição do Protótipo

Atualmente nota-se a inexistência de aplicativos, de diagnóstico clínico, que automatizem a análise genética de pacientes e que, ao mesmo tempo, ofereçam uma solução escalável, de acordo com o volume de dados gerados pelas novas máquinas sequenciadoras. Portanto, devido à existência de frameworks como o proposto pelo Hadoop, decidiu-se pelo desenvolvimento de um aplicativo MR que supra essas duas necessidades.

O método anteriormente utilizado, pelos pesquisadores do HCPA, para o diagnóstico clínico de pacientes era feito de forma manual utilizando tabelas. Primeiramente eles obtiam os dados genéticos de cada paciente, oriundos de sequenciadores, e inseriam-os de forma manual em uma tabela. Após a inserção correta desses dados, funções da tabela os processavam e geravam uma lista de mutações, relacionadas a doenças previamente conhecidas. Tal processo é considerado extremamente desgastante e rudimentar pois seus usuários são obrigados a inserir tais dados em tabelas, de forma coerente, e também precisam manter a lista de mutações conhecidas atualizada.

A medida que o volume de dados gerados pelos sequenciadores aumenta, o uso dessa solução torna-se humanamente inviável. O protótipo implementado só requer que o usuário disponibilize os dados dos pacientes, em um formato conhecido, e execute-o. Ele extrai informações de BDs com relações do tipo mutação-doença e utiliza-as no processamento das sequências genéticas dos pacientes. A versão atual do aplicativo contem, aproximadamente, 260 linhas de código e devido a esse fato, ela não foi adicionada como apêndice neste trabalho.

Além disso, para que o objetivo deste trabalho fosse alcançado com sucesso, tornou-se necessário o estudo tanto do formato dos dados de saída dos sequenciadores quanto da composição e organização das BDs com informações sobre doenças genéticas. Para melhor esclarecer a metodologia utilizada e o protótipo desenvolvido, o aplicativo será apresentado em três etapas e, posteriormente, serão abordados aspectos do ambiente de desenvolvimento e de testes.

4.1.1 1ª Etapa: Análise e Tratamento da Entrada

A quantia de máquinas sequenciadoras existentes é consideravelmente grande, porém nem todas se baseiam na mesma tecnologia e, de fato, a maioria acaba gerando formatos dos mais diversos. Outro detalhe que difere umas das demais é a possibilidade de se sequenciar somente um conjunto específico de genes em vez do genoma completo, além de que a saída já é automaticamente alinhada. Nessa categoria entram alguns dos sequenciadores IonTorrent (IONTORRENT, 2013b), os quais são o foco neste trabalho.

Devido a atual inacessibilidade aos dados gerados pelas máquinas sequenciadoras que deseja-se abordar, pesquisadores do HCPA forneceram dados coerentes que servirão como entrada para os testes da aplicação MR. Porém, como tais dados foram disponibilizados em tabelas e não sabe-se ao certo a formatação de dados utilizada pelos sequenciadores, foram desenvolvidos scripts para a conversão desses dados para um formato similar ao FASTA (FASTA, 2013).

A figura 4.1 mostra o formato dos dados de entrada da aplicação, onde cada linha começa com o símbolo >, seguido pelo nome identificador do gene que foi sequenciado (ABCB11), pelo identificador do paciente (p1), e pela sequência de nucleotídeos sequenciada pela máquina.

```
>ABCB11_p1 atg tct gnc tca gta att ctt cga
>ABCB11_p2 atg tct gac tca gta att ctt cga
```

Figura 4.1: Formato da entrada do aplicativo

Compreende-se que as sequências de nucleotídeos são referentes aos exons (áreas de DNA codificante) e, por isso, estão organizadas separadamente em blocos de três letras, cada um representando um códon, para facilitar a posterior leitura dos dados por parte da aplicação. As posíveis mutações que serão encontradas nessas entradas de dados podem ser vistas na Tabela 4.1. De acordo com alguns pesquisadores do HCPA, as mutações do tipo n podem ser, em um primeiro momento, desconsideradas na análise pois indicam que o sequenciador não foi capaz de sequenciar corretamente o nucleotídeo na posição em questão.

Tabela 4.1: Possíveis mutações encontradas nos dados de entrada.

Simbolo	Bases Representadas
w	a ou t
s	c ou g
m	a ou c
k	g ou t
r	a ou g
y	c ou t
n	a, c, g ou t

Fonte: Adaptado de (NC-IUB, 2013).

Após a conversão dos dados fornecidos pelos pesquisadores do HCPA através do uso de scripts, notou-se que o tamanho do arquivo gerado (50MB) não era comparável ao grande volume de dados gerados pelas máquinas sequenciadoras atuais. Portanto, para fins de teste do aplicativo com grandes volumes de dados, a solução adotada inicialmente baseia-se na replicação desse arquivo.

4.1.2 2ª Etapa: Análise e Tratamento das Bases de Dados

Um dos problemas que os sistemas atuais de diagnóstico clínico enfrentam é a disperção de informação. No caso da análise genética, informações que relacionem mutações gênicas diretamente a doenças ou a significados clínicos patogênicos, estão distribuídas em diversas BDs, sendo algumas até mesmo privadas.

Seguindo esse contexto, para que a aplicação MR funcione corretamente ela necessita de basicamente dois tipos de informação. A primeira faz referência aos genes que estão sendo recebidos como entrada. Como a intenção do aplicativo é procurar por mutações nas entradas recebidas, ele precisa obter dados sobre a sequência correta do gene que será analisado, para que então a comparação entre tal sequência e a sequência do paciente possa ser feita. Essa informação foi obtida da BD de um projeto que é mantido pelo NCBI (HOEPPNER, 2012) chamado CCDS (PRUITT et al., 2009).

O segundo tipo de informação está relacionado com os significados clínicos das mutações. Após a detecção de uma mutação é necessário verificar se ela é uma provável causadora de uma doença ou patogenia. Neste caso, tais dados foram obtidos de duas BDs. A principal, chamada GeneReport (GENEREPORT, 2013), é mantida pelo NCBI e contém diversas referências sobre as mutações de interesse clínico, em cada um dos genes conhecidos. A segunda, chamada Ensembl (KINSELLA et al., 2013), contém scores, como o SIFT (PAVLIDIS, 2008) e o Polyphen (ADZHUBEI et al., 2013), pré-calculados para a maioria das mutações SNP conhecidas.

A estrutura exposta na Figura 4.2 mostra quais BDs foram utilizados pela aplicação. Visando um melhor desempenho do aplicativo, foram desenvolvidos scripts para que somente dados realmente utilizados pela aplicação fossem extraídos dessas BDs.

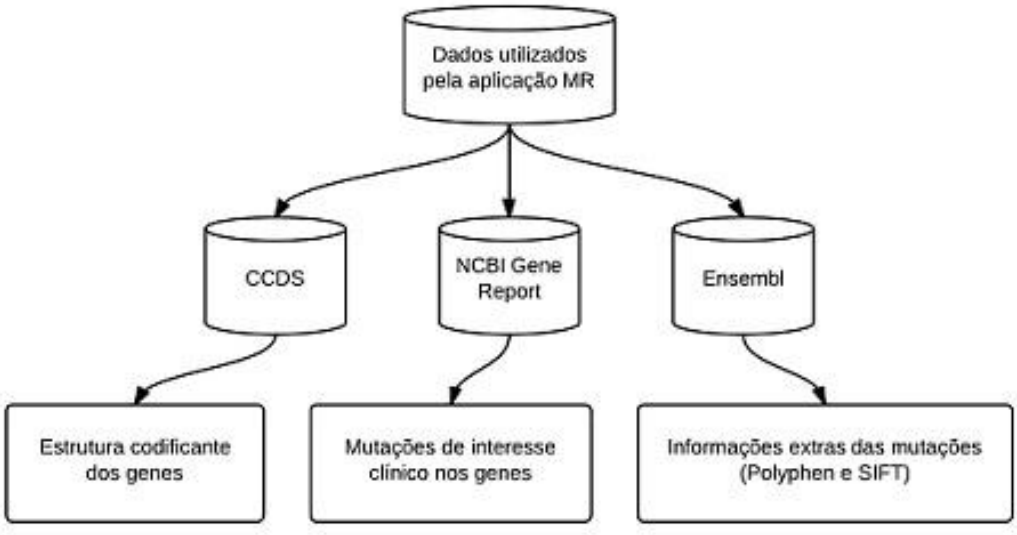

Figura 4.2: Bases de Dados utilizadas pelo aplicativo MR

4.1.3 3ª Etapa: Modelagem do Aplicativo MR

A seguir, na Figura 4.3, é demonstrado o modelo da aplicação MR proposta neste trabalho. Considera-se que os dados que servirão como entrada para o aplicativo seguirão o formato descrito na subseção 4.1.1 e estarão presentes no Sistema de Arquivos Distribuídos do Hadoop (HDFS) do ambiente de execução do aplicativo.

Desse modo, quando da inicialização do Job pelo Hadoop, os nós da infraestrutura utilizada criam *Splits* (de acordo com o aplicativo) sobre seus dados de entrada locais e repassam-os, no formato <*chave_1*, *valor_1*>, para o *Map*. Porém, antes que a função *Map* seja executada, as informações sobre a estrutura dos genes são carregadas em memória, visando melhor desempenho.

Assim que a função *Map* encontra uma mutação em um dos pares <*chave*, *valor*> recebidos, ela repassa tal <*chave*, *valor*>, com algumas modificações, para o Combiner. Assim como no caso do *Map*, o *Combiner* também carrega informações em memória, mas sobre as mutações patogênicas que foram encontradas nas BDs citadas na seção 4.3. Caso ele encontre a mutação na BD mantida em memória, essas informações são passadas para o *Reduce* que se encarregará de salvá-las em um arquivo de sáida no HDFS.

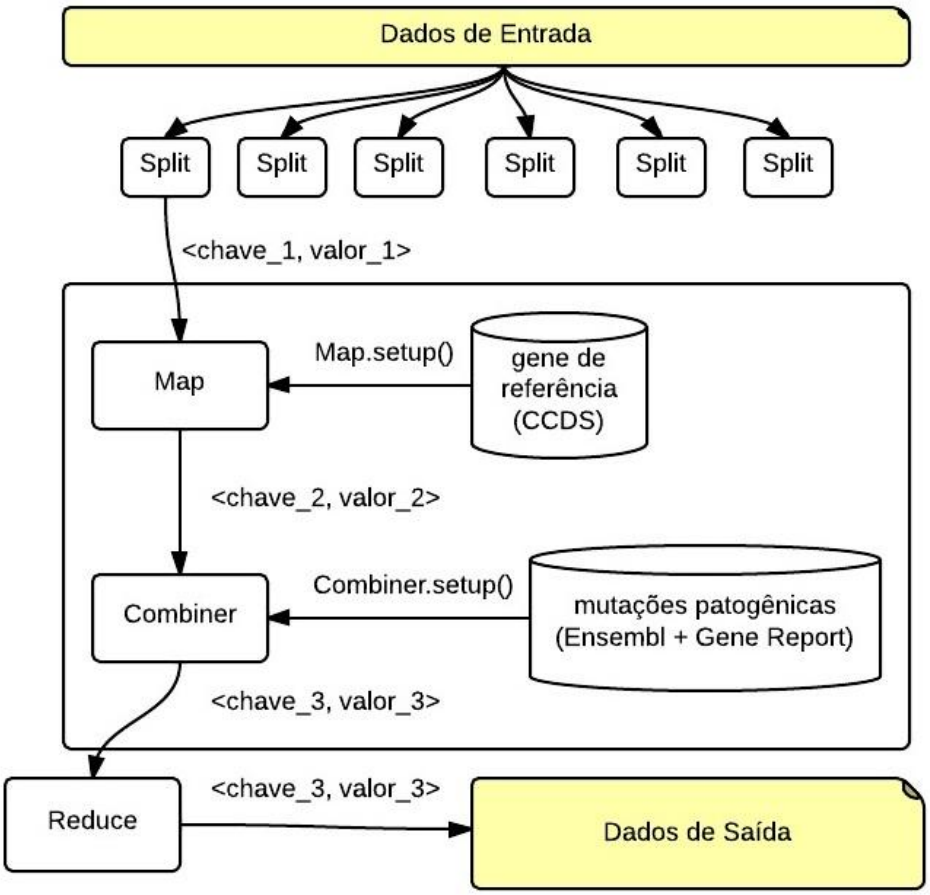

Figura 4.3: Modelo do aplicativo MR desenvolvido

Um dos problemas encontrados é que, seguindo esse modelo, precisa-se fazer com que as BDs sejam carregadas dentro de algumas funções (*Map* e *Combiner*) antes que

elas começem o processamento das suas entradas (pares <*chave*, *valor*>). O Hadoop disponibiliza uma solução específica para este propósito, chamada *DistributedCache* (WHITE, 2010), a qual permite que arquivos pequenos sejam transferidos, para todos os nós do tipo *worker*, a fim de tornar possível o acesso a tais arquivos em tempo de inicialização por parte dos *workers*. Visto que as informações dos BDs utilizadas pelo aplicativo foram reduzidas, após a execução dos passos de filtragem/organização por meio de scripts, o uso da *DistributedCache* fez-se necessário.

Como descrito no Capítulo 3, a comunicação entre *Map* e *Reduce* demanda a transferência de dados pela rede interna da infraestrutura utilizada. Por isso e devido ao fato de que nem todas mutações serão encontradas na BD de mutações patogênicas usada pela aplicação, optou-se pelo desenvolvimento do *Combiner* ao invés de sua implementação dentro do *Reduce*.

Para auxiliar na compreenção da estrutura dos pares <*chave*, *valor*> utilizadas no aplicativo, um exemplo de execução pode ser visto na Figura 4.4. O primeiro par (recebido como entrada pelo *Map*) contem como *chave* o *offset* da linha no arquivo e como *valor* o conteúdo da linha. O segundo par (saída do *Map*/entrada do *Combiner*) contem como *chave* o identificador do gene e como *valor* uma expressão que contem o identificador do paciente, número e posição do códon onde a mutação ocorreu, sequência de referência e sequência do paciente. Os outros pares seguem o modelo proposto pelo segundo par, porém com a adição de referências clínicas da mutação no final do *valor*.

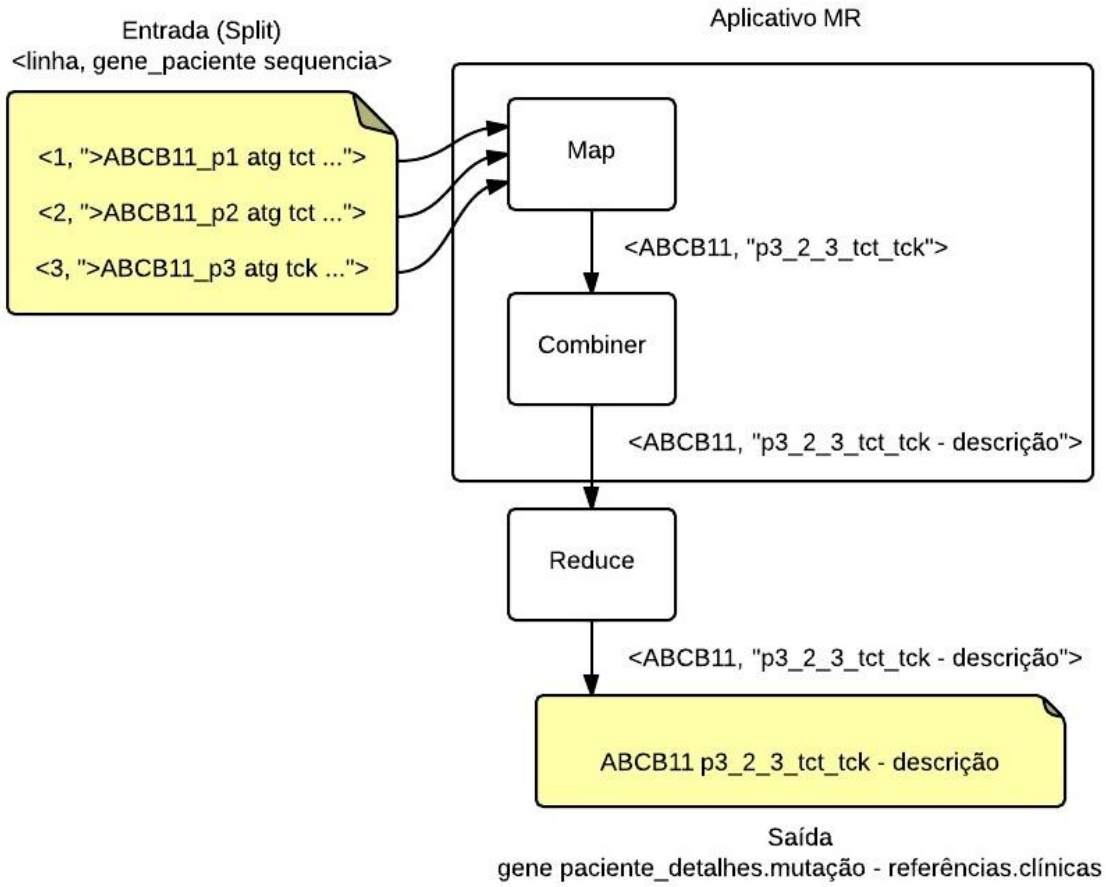

Figura 4.4: Exemplo de execução do aplicativo

Um exemplo de uma saída real do protótipo é demonstrado pela Figura 4.5. Para melhor visualização, cada mutação foi descrita em 4 linhas. Na primeira encontra-se o identificador do gene no qual a mutação ocorre, seguido pelo identificador do paciente e informações sobre a mutação e a posição onde ela ocorre neste gene. As linhas restantes são referentes aos dados utilizados, pelo *Combiner*, no BD de mutações patogênicas do aplicativo. Novamente, a segunda linha indica a posição da mutação e qual o aminoácido resultante. A terceira e quarta linha representam referências a BDs externas, vinculadas à descrição de doenças ou mutações como, por exemplo, o OMIM, o PubMed e o dbSNP. Além disso, elas contêm informações de score (SIFT e PolyPhen) e de frequência (MAF) sobre a mutação em questão.

```
ABCB11  p52_297_2_gag_grg
DB data: 297 2 A>G Glu-Gly
11568372 pathogenic germline
http://omim.org/entry/603201#0002
ATP8B1  p8_952_2_cga_cra
DB data: 952 2 G>A Arg-Gln
12968116 5.6e-2 polyphen 0.994 sift 0.25
http://www.ncbi.nlm.nih.gov/pubmed?Term=22001757
```

Figura 4.5: Exemplo de sáida do aplicativo

4.2 Ambiente de Desenvolvimento e de Testes

A maior parte do desenvolvimento do aplicativo se deu com o uso de duas linguagens de programação: Python e Java. Python foi muito utilizado na criação de scripts, para o ajuste da formatação dos dados de entrada do aplicativo MR e para o processo de filtragem das BDs de mutações com significado clínico. Devido ao uso do Hadoop como framework para a execução da aplicação, Java foi utilizado no desenvolvimento do aplicativo MR.

Atualmente, o GPPD possui um cluster com a versão 1.0.4 do Hadoop instalada e funcionando. Por isso, este foi o ambiente de testes escolhido para a aplicação. O *cluster* é composto por 20 nós, sendo um deles utilizado como *master* pelo Hadoop e o restante como possíveis *slaves*. A configuração dos nós é variada e, por isso, a execução de tarefas especulativas acaba aliviando o desempenho do sistema. Considerando as configurações de *hardware* utilizadas pelo cluster e demonstradas pela Tabela 4.1, 5 nós *slave* contêm a especificação *slave_básico*, 14 nós contêm a especificação *slave_intermediário* e 1 nó contem a especificação *master*. Além disso, todos os nós utilizam o mesmo sistema operacional (Linux 2.6.18-238.19.1.el5 (x86)). Outros detalhes sobre os tipos de testes, que foram executados no *cluster*, serão abordados no próximo capítulo.

Especificação	*CPUs*	*Memória (RAM)*	*Disco Local*
slave_básico	1 x 2.79 GHz	1.98 GB	~387 GB
slave_intermediário	2 x 2.79 GHz	1.97 GB	~968 GB
master	4 x 3.10 GHz	3.56 GB	~968 GB

Tabela 4.1: Configuração do ambiente de testes.

Considerando que testes tiveram que ser feitos, seja variando a configuração do Hadoop, a quantia de nós *slave* utilizados ou o volume de dados utilizados pela aplicação MR, alguns scripts, apresentados no Anexo B, foram desenvolvidos utilizando *Shell Script*.

Uma vez conectado ao *cluster*, faz-se a verificação/restauração das configurações que serão utilizadas pelo Hadoop (exemplo: definição do fator de replicação utilizado, definição dos nós que serão *slaves*/*workers*, etc) através do seguinte comando.

Posteriormente é feita a inicialização do Hadoop no *cluster* e então os arquivos de entrada e os referentes às BDs utilizadas são copiados para o HDFS. Assim que a cópia é feita, o código da aplicação MR é compilado e transferido para um arquivo de extensão jar. Por último, indica-se ao Hadoop a localização do arquivo jar da aplicação e da pasta de entrada e de saída de dados no HDFS e ele começa a execução do *job*.

4.3 Considerações Finais

Este capítulo apresentou o modelo MR proposto pela aplicação, os recursos utilizados na sua implementação e o ambiente no qual os testes especificados foram feitos. No próximo capítulo os resultados serão apresentados e as devidas conclusões serão apresentadas.

5 RESULTADOS

Neste capítulo serão apresentados os resultados, obtidos através de testes de execução da aplicação MR, desenvolvida neste trabalho, no ambiente de testes exposto anteriormente.

5.1 Validação da Aplicação MR

A validação da aplicação MR foi realizada através da comparação entre execuções da mesma no cluster do GPPD, além de comparações entre a saída gerada pelo aplicativo e a saída gerada manualmente por pesquisadores do HCPA.

5.1.1 Tempo de Execução

Os testes executados no cluster do GPPD foram propostos com base na variação dos seguintes atributos:

1. Número de nós trabalhadores utilizados (1, 2, 3, 4, 5, 10, 15 e 19)
2. Tamanho do arquivo de entrada (1GB, 2GB, 5GB, 10GB e 20GB)

A combinação da variação desses atributos gerou 40 casos de teste e cada um deles foi executado 5 vezes. Logo, foram executados um total de 200 testes no cluster. Em todos eles, a configuração do Hadoop utilizou um *chuncksize* no valor de 64MB e um fator de replicação de dados igual a 3. Os nós escolhidos para a execução dos testes foram selecionados de forma randômica. A seguir são apresentados gráficos indicando o desempenho da aplicação no ambiente de testes.

A Figura 5.1 respresenta o tempo de execução do protótipo, de acordo com a quantia de nós trabalhadores utilizados e com o tamanho de entrada. A Figura 5.2 representa o ganho de *SpeedUp* do aplicativo quando comparado com a execução do algoritmo sequencial implementado. Considera-se que o tempo de execução do algoritmo sequencial é equivalente ao do aplicativo desenvolvido, utilizando somente um nó trabalhador do cluster.

A partir da análise dos gráficos dessas duas figuras, é notável o desempenho ganho pelo protótipo com a simples adição de nós trabalhadores à configuração do Hadoop. O ganho de *SpeedUp* máximo ocorre quando 19 nós trabalhadores são utilizados com uma entrada de 20GB. Nesse caso, o algoritmo sequencial leva aproximadamente cinco mil segundos para processar a entrada, enquanto que a execução do algoritmo MR leva quatrocentos segundos (ou seja, *SpeedUp* de 12,4).

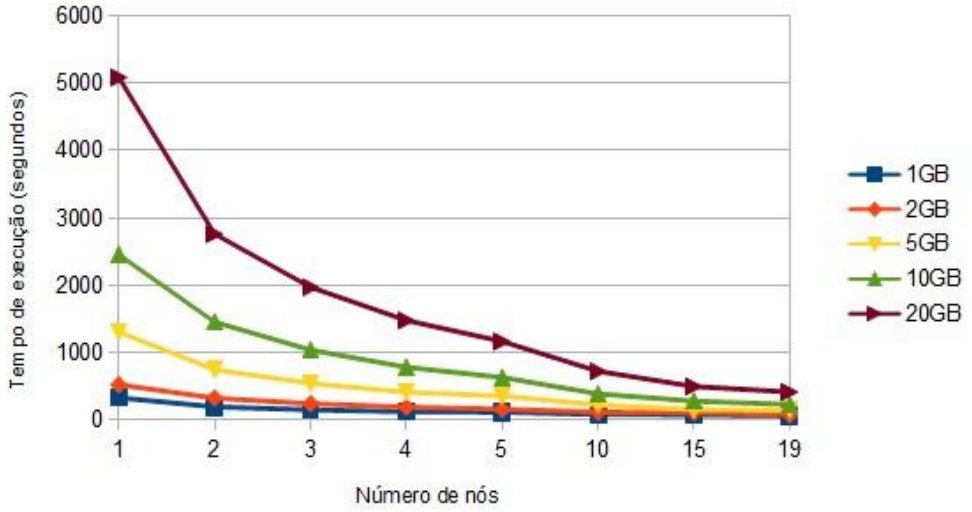

Figura 5.1: Gráfico de comparação (Tempo vs. Número de nós)

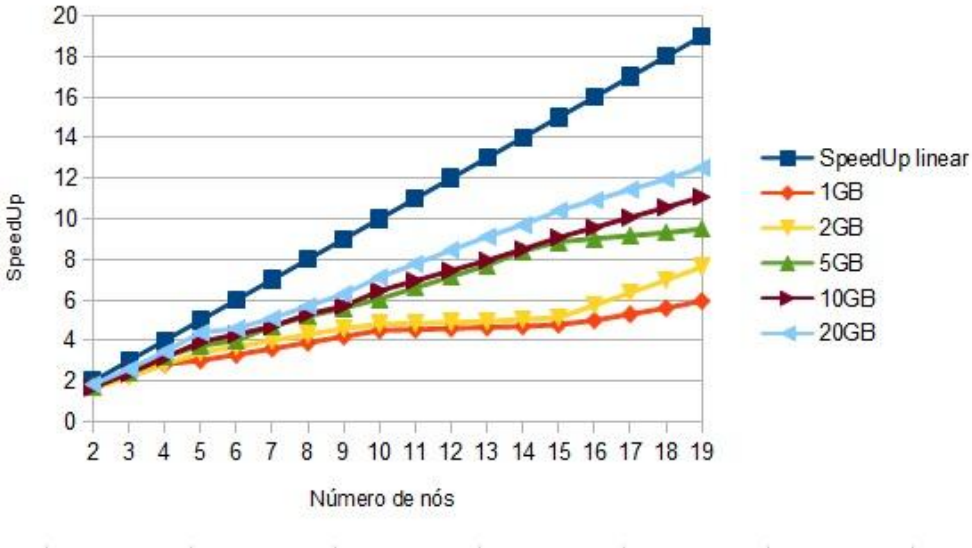

Figura 5.2: Gráfico de comparação (SpeedUp vs. Número de nós)

Portanto, uma das vantagens obtidas com o uso do MapReduce, aplicado ao problema do diagnóstico clínico de pacientes, é que tal tecnologia possibilitou o desenvolvimento de uma solução escalável. Em suma, o desempenho do protótipo aumenta à medida que o tamanho da entrada aumenta. Devido ao grande volume de dados gerado por máquinas sequenciadoras, este foi um dos objetivos propostos neste trabalho.

5.1.2 Saída de Dados

Além de comparações que considerem o fator tempo na execução do aplicativo, é necessário que a saída gerada pelo mesmo seja comparada com a saída esperada pelos pesquisadores do HCPA. Após algumas verificações, notou-se que algumas mutações

presentes na saída esperada não estavam presentes na saída gerada pelo protótipo. Este fato ocorreu devido a falta de informações sobre tal mutação no BD utilizado.

Portanto, foi decidido que toda e qualquer mutação que não for encontrada na BD, deve ser apresentada na saída, porém, informando ao usuário da inexistência de informações adicionais. A seguir, um exemplo deste caso é demonstrado na Figura 5.3. Nota-se que a primeira mutação não foi encontrada pelo aplicativo, porém está nas anotações criadas pelos pesquisadores. Considerando como trabalho futuro, é fundamental que seja desenvolvido uma funcionalidade para o aplicativo, a qual permita ao pesquisador, de modo simples e prático, adicionar/modificar mutações e seus respectivos detalhes clínicos ao BD utilizado.

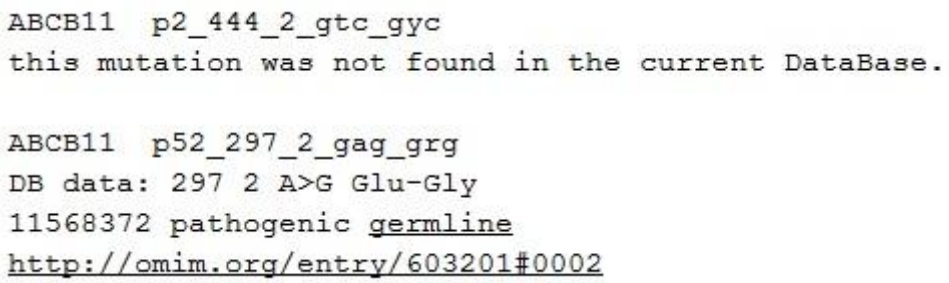

Figura 5.3: Exemplo de saída do aplicativo

Quando comparada com o método anteriormente utilizado, por pesquisadores do HCPA, para o diagnóstico clínico de pacientes, fica claro que outra vantagem consequente da aplicação MR desenvolvida se da na automatização desse processo. Com o uso do protótipo atual, não será necessário que os pesquisadores manipulem os dados gerados pelos sequenciadores, transferindo-os manualmente para tabelas. Este também foi um dos objetivos propostos neste trabalho.

5.2 Considerações Finais

Os resultados obtidos na validação do protótipo desenvolvido, confirmam a existência de meios para a obtenção de um melhor desempenho em aplicações que involvam a análise dos grandes volumes de dados gerados por máquinas sequenciadoras. Também demonstrou-se que os objetivos, definidos inicialmente, foram alcançados pelo protótipo desenvolvido.

De fato, todas as mutações obtidas pelos pesquisadores do HCPA, foram encontradas pelo aplicativo. Porém, nem todas foram relacionadas à saída final devido a inesistência das mesmas nas BDs utilizadas, internamente, pelo protótipo. Devido a inexistência de sistemas semelhantes, poucas comparações foram apresentadas neste trabalho.

6 CONCLUSÃO

Neste trabalho foi apresentado o desenvolvimento de uma nova solução para o problema de análise do grande volume de dados gerados por sequenciadores de DNA. Através do uso da tecnologia implementada pelo Hadoop, comprovou-se que é possível desenvolver aplicações paralelas e distribuídas, baseadas em MapReduce, para a área de análise genética.

A partir da pesquisa que foi realizada e dos resultados obtidos pelo protótipo criado, torna-se possível citar alguns trabalhos futuros de grande importância para área: desenvolver aplicações semelhantes em outros contextos como, por exemplo, utilizando MPI em máquinas de alto desempenho, utilizando novas tecnologias de computação intensiva em dados como o Spark ou utilizando tecnologias em Cloud; desenvolver uma UI (do inglês *User Interface*) adequada para o protótipo; estudar a aplicação de técnicas de mineração de dados sobre BDs e sistemas que possuam informações de relevância para a aplicação (de fato, nem todas relações mutação-doença foram consideradas no BD do protótipo); otimizar o protótipo através da remodelagem da sua estrutura de funções MR e do uso de práticas de programação de acordo, por exemplo, com (XIE et al., 2010)

Portanto, o estudo realizado proporciona uma introdução a conceitos fundamentais da genética e do MapReduce, levantando questões interessantes sobre as adaptações às quais o modelo MR esteve sujeito ao decorrer do trabalho. Desse modo, espera-se que tenha sido atingido o objetivo do fomento à pesquisa da tecnologia abordada, visto que sua aplicação pode se tornar muito útil em diversas áreas como a de Biotecnologia.

REFERÊNCIAS

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. **Commun. ACM**, New York, NY, USA, v.51, n.1, p.107–113, 2008.

DEAN, J.; GHEMAWAT, S. MapReduce: a flexible data processing tool. **Commun. ACM**, New York, NY, USA, v.53, n.1, p.72–77, 2010.

DUBREUIL, M.; GAGNÉ, C.; PARIZEAU, M. Analysis of a master-slave architecture for distributed evolutionary computations. **Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on**, [S.l.], v.36, n.1, p.229–235, 2006.

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The Google file system. In: SOSP '03: PROCEEDINGS OF THE NINETEENTH ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 2003, New York, NY, USA. **Anais. . .** ACM, 2003. p.29–43.

ZAHARIA, M.; KONWINSKI, A.; JOSEPH, A. D.; KATZ, R. H.; STOICA, I. **Improving MapReduce Performance in Heterogeneous Environments**. [S.l.: s.n.], 2008. (UCB/EECS-2008-99).

WHITE, T. **Hadoop**: the definitive guide (2nd ed.). [S.l.]: O'Reilly Media, Inc., 2010.

WATSON, J. D. **Molecular Biology of the Gene** (6th ed.). [S.l.:s.n] 2007.

SANGER, F. **DNA sequencing with chain-terminating inhibitors**, [S.l.:s.n] 1977.

THIEMAN, W.J.; PALLADINO, M.A. **Introduction to Biotechnology**. [S.l.]: Pearson/Benjamin Cummings, 2008.

GRIFFITHS, W. M.; MILLER, J. H.; SUZUKI, D. T.; LEWONTIN, R. C.; GELBART. **An Introduction to Genetic Analysis** (7th ed.). New York :[s.n], 2000.

ALBERTS, B.; JOHNSON, A.; LEWIS, J.; RAFF, M.; ROBERTS, K.; WLATER, P. **Molecular Biology of the Cell** (4th ed.). [S.l.]: Garland Science, 2002.

NUSSBAUM, R. L.; MCINNES, R. R.; WILLARD, H. F. **Thompson & Thompson: Genetics in Medicine** (7th ed.). [S.l.]: Saunders Elsevier, 2007

DOS ANJOS, J. C. S.; IZURIETA, I. M. C.; TIBOLA, A. L.; GEYER, C. F. R. O 4º Paradigma e a Computação Intensiva em Dados. **ERAD 2013**, Porto Alegre, fev. 2013.

PEARSON, H. Genetics: what is a gene? **Nature 441** [S.l.:s.n], p. 398-401, mai. 2006.

SAWYER, S. A.; PARSCH, J.; ZHANG, Z.; HARTL, D. L. Prevalence of positive selection among nearly neutral amino acid replacements in Drosophila. **Proc. Natl. Acad. Sci. U.S.A. 104** [S.l.:s.n], abr. 2007.

HAMOSH, A.; SCOTT, A.; AMBERGER, J.; BOCCHINI, C.; MCKUSICK, V. Online Mendelian Inheritance in Man (OMIM), a knowledgebase of human genes and genetic disorders. **Nucleic Acids Research 33** [S.l.:s.n], jan. 2005.

XIE, J. X. J. et al. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. **Parallel amp Distributed Processing Workshops and Phd Forum IPDPSW 2010 IEEE International Symposium on**, v. 400, p. 1–9, 2010.

KINSELLA, R. J. et al. Ensembl BioMarts: a hub for data retrieval across taxonomic space. **Database the journal of biological databases and curation**, v. 2011, p. 9, 2011.

HOEPPNER, M. A. NCBI Bookshelf: books and documents in life sciences and health care. **Nucleic acids research**, p. gks1279–, 2012.

PAVLIDIS, T. An Evaluation of the Scale Invariant Feature Transform (SIFT). **An Evaluation of SIFT**, 2008.

ADZHUBEI, I.; JORDAN, D. M.; SUNYAEV, S. R. Predicting functional effect of human missense mutations using PolyPhen-2. **Current protocols in human genetics editorial board Jonathan L Haines et al**, v. Chapter 7, p. Unit7.20, 2013.

PRUITT, K. D. et al. The consensus coding sequence (CCDS) project: Identifying a common protein-coding gene set for the human and mouse genomes. **Genome research**, v. 19, p. 1316–23, 2009.

NHGRI. **National Human Genome Research Institute Website.** Disponível em: <http://www.genome.gov>. Acesso em: maio 2013.

NC-IUB. **Nomenclature for Incompletely Specified Bases in Nucleic Acid Sequences.** Disponível em: <http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html>. Acesso em: junho 2013.

FASTA. **Descrição do formato FASTA.** Disponível em: <http://zhanglab.ccmb.med.umich.edu/FASTA>. Acesso em: junho 2013.

GENE REPORT. **Gene Report - Uma Base de Dados de mutações patogênicas.** Disponível em: <ftp://ftp.ncbi.nlm.nih.gov/snp/organisms/human_9606/gene_report>. Acesso em: maio 2013.

IONTORRENT. **IonTorrent Website - Specifications Table.** Disponível em: <http://blueseq.com/knowledgebank/sequencing-platforms/life-technologies-ion-torrent>. Acesso em: maio 2013.

IONTORRENT. **Sequenciadores IonTorrent - Ion Proton System.** Disponível em: <http://products.invitrogen.com/ivgn/product/4476610>. Acesso em: maio 2013.

APACHE. **Welcome! - The Apache Software Foundation**. Disponível em: <http://www.apache.org>. Acesso em: julho 2013.

HADOOP. **Welcome to Apache Hadoop!** Disponível em: <http://hadoop.apache.org>. Acesso em: julho 2013.

HADOOP. **HDFS Architecture.** Disponível em: <http://hadoop.apache.org/docs/r1.0.4/hdfs_design.html>. Acesso em: julho 2013.

HADOOP. **HDFS - Hadoop Wiki.** Disponível em: <http://wiki.apache.org/hadoop/HDFS>. Acesso em: julho 2013.

HADOOP. **Map/Reduce Tutorial.** Disponível em: <http://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.html>. Acesso em: junho 2013.

ANEXO A EXEMPLO DE PROGRAMA NO HADOOP

O código fonte deste anexo apresenta um exemplo de implementação da aplicação para contagem de ocorrências de palavras em arquivos texto, utilizando o Hadoop MapReduce. Fonte: (HADOOP, 2013d).

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

  public static class Map extends MapReduceBase
   implements Mapper<LongWritable, Text, Text, IntWritable> {
      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();

      public void map(LongWritable key, Text value,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
        throws IOException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
              word.set(tokenizer.nextToken());
              output.collect(word, one);
            }
      }
   }

   public static class Reduce extends MapReduceBase
```

```
    implements Reducer<Text, IntWritable, Text, IntWritable> {
      public void reduce(Text key, Iterator<IntWritable> values,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
        throws IOException {
            int sum = 0;
            while (values.hasNext()) {
              sum += values.next().get();
            }
            output.collect(key, new IntWritable(sum));
      }
   }

   public static void main(String[] args) throws Exception {
     JobConf conf = new JobConf(WordCount.class);
     conf.setJobName("wordcount");

     conf.setOutputKeyClass(Text.class);
     conf.setOutputValueClass(IntWritable.class);

     conf.setMapperClass(Map.class);
     conf.setCombinerClass(Reduce.class);
     conf.setReducerClass(Reduce.class);

     conf.setInputFormat(TextInputFormat.class);
     conf.setOutputFormat(TextOutputFormat.class);

     FileInputFormat.setInputPaths(conf, new Path(args[0]));
     FileOutputFormat.setOutputPath(conf, new Path(args[1]));

     JobClient.runJob(conf);
   }
}
```

ANEXO B SCRIPTS UTILIZADOS NO PROTÓTIPO

O código fonte deste anexo apresenta os scripts implementados para a execução do protótipo, utilizando o Hadoop MapReduce no ambiente de testes deste trabalho.

- Script de restauração das configurações do Hadoop

```
cp /home/brfilho/HCPA_App/hadoop-original/* /etc/hadoop/
```

- Script de inicialização do Hadoop e cópia dos arquivos de entrada

```
hadoop namenode -format
/usr/sbin/start-dfs.sh
/usr/sbin/start-mapred.sh

hadoop dfs -mkdir generef
hadoop dfs -copyFromLocal /home/brfilho/HCPA_App/generef/*.* generef

hadoop dfs -mkdir db
hadoop dfs -copyFromLocal /home/brfilho/HCPA_App/db/*.* db

hadoop dfs -mkdir input
hadoop dfs -copyFromLocal /home/brfilho/HCPA_App/input/*.* input
```

- Script de compilação do código do protótipo

```
javac -cp `hadoop classpath` DiseaseApplication.java -d App_classes
jar -cvf diseaseapp.jar -C diseaseApp_classes/ .
```

- Script de execução do *job*

```
hadoop jar diseaseapp.jar org.Hadoop. DiseaseApplication input output
```