UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SPARKBLAST: UTILIZAÇÃO DA FERRAMENTA APACHE SPARK PARA A EXECUÇÃO DO BLAST EM AMBIENTE DISTRIBUÍDO E ESCALÁVEL

MARCELO RODRIGO DE CASTRO

ORIENTADOR: PROF. DR. HERMES SENGER

São Carlos – SP

Fevereiro/2017

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SPARKBLAST: UTILIZAÇÃO DA FERRAMENTA APACHE SPARK PARA A EXECUÇÃO DO BLAST EM AMBIENTE DISTRIBUÍDO E ESCALÁVEL

MARCELO RODRIGO DE CASTRO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas distribuídos e redes de computadores
Orientador: Prof. Dr. Hermes Senger

São Carlos – SP

Fevereiro/2017

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de Dissertação de Mestrado do candidato Marcelo Rodrigo de Castro, realizada em 13/02/2017.

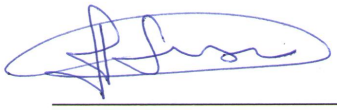
Prof. Dr. Hermes Senger
(UFSCar)

Prof. Dr. Paulo Estevão Cruvinel
(EMBRAPA)

****

Prof. Dr. Fabricio Alves Barbosa da Silva
(FIOCURZ)

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Fabricio Alves Barbosa da Silva, depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Marcelo Rodrigo de Castro.



Prof. Dr. Hermes Senger
Presidente da Comissão Examinadora
(UFSCar)

A meu Deus autor e consumador da minha fé, minha esposa, família e amigos.

AGRADECIMENTOS

Agradeço meu Deus pela força de cada dia. Agradeço a minha esposa e família por acreditarem e incentivarem o meu trabalho. Agradeço ao Prof. Hermes Senger pela orientação, paciência e pela oportunidade em poder fazer este trabalho. Agradeço ao pessoal da FIOCRUZ, em especial o Prof. Fabrício Alves Barbosa da Silva, pela ajuda na definição do problema proposto. Agradeço aos amigos e colegas do DC e GSDR, em especial a turma de 01/2014 pela ajuda nas disciplinas e companheirismo. Agradeço ao IFSULDEMINAS - Campus Muzambinho pelo PIQ e à Google e Microsoft por disponibilizarem um ambiente para poder executar este trabalho.

“Tudo posso Naquele que me fortalece.”

Filipenses 3.13

RESUMO

Com a redução dos custos e evolução dos mecanismos que efetuam o sequenciamento genômico, tem havido um grande aumento na quantidade de dados referentes aos estudos da genômica. O crescimento desses dados tem ocorrido a taxas mais elevadas do que a indústria tem conseguido aumentar o poder dos computadores a cada ano. Para melhor atender à necessidade de processamento e análise de dados em bioinformática faz-se o uso de sistemas paralelos e distribuídos, como por exemplo: Clusters, Grids e Nuvens Computacionais. Contudo, muitas ferramentas, como o BLAST, que fazem o alinhamento entre sequências e banco de dados, não foram desenvolvidas para serem processadas de forma distribuída e escalável. Os atuais *frameworks* Apache Hadoop e Apache Spark permitem a execução de aplicações de forma distribuída e paralela, desde que as aplicações possam ser devidamente adaptadas e paralelizadas. Estudos que permitam melhorar desempenho de aplicações em bioinformática têm se tornado um esforço contínuo. O Spark tem se mostrado uma ferramenta robusta para processamento massivo de dados. Nesta pesquisa de mestrado a ferramenta Apache Spark foi utilizada para dar suporte ao paralelismo da ferramenta BLAST (Basic Local Alingment Search Tool). Experimentos realizados na nuvem Google Cloud e Microsoft Azure demonstram desempenho (speedup) obtido foi similar ou melhor que trabalhos semelhantes já desenvolvidos em Hadoop.

Palavras-chave: BLAST, Apache Spark, Hadoop, Nuvens Computacionais, Sequenciamento genético.

ABSTRACT

With the evolution of next generation sequencing devices, the cost for obtaining genomic data has significantly reduced. With reduced costs for sequencing, the amount of genomic data to be processed has increased exponentially. Such data growth supersedes the rate at which computing power can be increased year after year by the hardware and software evolution. Thus, the higher rate of data growth in bioinformatics raises the need for exploiting more efficient and scalable techniques based on parallel and distributed processing, including platforms like Clusters, and Cloud Computing. BLAST is a widely used tool for genomic sequences alignment, which has native support for multicore-based parallel processing. However, its scalability is limited to a single machine. On the other hand, Cloud computing has emerged as an important technology for supporting rapid and elastic provisioning of large amounts of resources. Current frameworks like Apache Hadoop and Apache Spark provide support for the execution of distributed applications. Such environments provide mechanisms for embedding external applications in order to compose large distributed jobs which can be executed on clusters and cloud platforms. In this work, we used Spark to support the high scalable and efficient parallelization of BLAST (Basic Local Alingment Search Tool) to execute on dozens to hundreds of processing cores on a cloud platform. As result, our prototype has demonstrated better performance and scalability then CloudBLAST, a Hadoop based parallelization of BLAST.

Keywords: BLAST, Apache Spark, Hadoop, Cloud Computing, Genetic sequencing

LISTA DE FIGURAS

2.1 Crescimento número de sequências GenBank 23

2.2 Crescimento número de bases GenBank 23

2.3 Custo mapear genoma vs Lei de Moore 25

2.4 Trecho de um arquivo FASTA contendo parte da proteína *Ehrlichia canis* . . . 27

2.5 Características do *big data*. 32

2.6 Arquitetura *big data*. 33

2.7 Lista e função Lisp 34

2.8 Função em Lisp 34

2.9 Arquitetura HDFS 38

2.10 Fluxo de execução MapReduce no Hadoop 39

2.11 Gerenciador de execução YARN 40

2.12 Hadoop Streaming 41

2.13 Cluster Spark 43

2.14 Regressão Logística em Hadoop e Spark 44

2.15 Divisão tarefas YARN 45

2.16 Spark Pipe 45

2.17 MapReduce no Hadoop 46

2.18 MapReduce no Spark 47

3.1 *Speedup* em relação à execução do *tblastx* na plataforma Biodoop. 51

3.2 Paralelização dos dados para o BLAST 53

3.3 SparkSequ vs SeqPig 56
4.1 Distribuição das sequências gênicas entre os nós 60
4.2 Divisão do arquivo FASTA entre os nós 61
4.3 Funcionamento de um Cluster Spark 61
4.4 Arquitetura Spark na Nuvem Google 63
4.5 Saída BLAST, programa executado em apenas um nó. 67
5.1 Execução do BLAST sobre o Spark - Google 74
5.2 Speedup SparkBLAST vs CloudBlast - Google 75
5.3 Eficiência SparkBlast vs CloudBlast - Google 75
5.4 Execução do BLAST sobre o Spark - Azure 77
5.5 Speedup SparkBLAST vs CloudBlast - Azure 78
5.6 Eficiência SparkBlast vs CloudBlast - Azure 78

LISTA DE TABELAS

2.1 Programas BLAST 29

3.1 Trabalhos correlatos e motivacionais 57

5.1 Tempo execução (Experimento 1 - query.fa - 36MB) - SparkBLAST - Google Cloud 72

5.2 Tempo execução (Experimento 1 - query.fa - 36MB) - SparkBLAST 1 *core*/nó - Google Cloud 72

5.3 Tempo execução (Experimento 1 - query.fa - 36MB) - SparkBLAST 2 *core*/nó - Google Cloud 73

5.4 Execução CloudBLAST vs SparkBLAST (tempo, *speedup* e eficiência) 73

5.5 Tempo de processamento, *speedups* e eficiência (Experimento 2 - buz.fasta - 805MB) - SparkBLAST vs CloudBLAST - Microsoft Azure 73

5.6 Tempo de processamento, *speedups* e eficiência (Experimento 2 - ber.fasta - 11GB) - SparkBLAST vs CloudBLAST - Microsoft Azure 74

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO **14**

1.1 Contexto 14

1.2 Motivação 17

1.3 Objetivos 18

1.4 Organização do Trabalho 18

CAPÍTULO 2 – REFERENCIAL TEÓRICO **20**

2.1 Processamento Genômico 20

2.2 Bioinformática 21

2.2.1 Sequenciamento genômico 21

2.2.2 Crescimento dos dados 24

2.2.3 Alinhamento de sequências 25

2.3 Ferramentas de Bioinformática 26

2.3.1 Arquivo FASTA 27

2.3.2 BLAST 27

2.4 Big Data 31

2.5 Modelo de programação MapReduce 33

2.6 Apache Hadoop 35

2.6.1 Hadoop Common 36

2.6.2 HDFS 37

2.6.3 MapReduce no Hadoop 38
2.6.4 YARN 39
2.6.5 Hadoop Streaming 40
2.7 Apache Spark 42
2.8 Fluxo de processamento Apache Hadoop e Apache Spark 46
2.9 Medidas de desempenho 48
2.10 Considerações finais 48

CAPÍTULO 3 – TRABALHOS CORRELATOS **49**

3.1 Considerações iniciais 49
3.2 Biodoop 49
3.3 CloudBLAST 52
3.4 SparkSeq 55
3.5 Resumo dos trabalhos considerados 56

CAPÍTULO 4 – UTILIZAÇÃO DO SPARK PARA ALINHAMENTO DE SEQUÊNCIAS USANDO O BLAST **58**

4.1 Considerações iniciais 58
4.2 Definição do Trabalho 59
4.2.1 Ambiente de execução 62
4.2.2 Dados a serem processados 63
4.2.3 Pré-processamento 65
4.2.4 Processamento 66
4.2.5 Pós-processamento 69

CAPÍTULO 5 – RESULTADOS E TRABALHOS FUTUROS **70**

5.1 Resultados 70

CAPÍTULO 6 – CONCLUSÃO **80**

6.1 Conclusão 80
6.2 Contribuições 81
6.3 Trabalhos Futuros 82

REFERÊNCIAS **83**

GLOSSÁRIO **88**

Capítulo 1

INTRODUÇÃO

Este capítulo contém a apresentação do trabalho e aborda o contexto, a motivação e os objetivos estabelecidos a serem alcançados ao final da pesquisa e traz também informações sobre a estrutura e a organização deste documento.

1.1 Contexto

Após o trabalho de Watson e Crick sobre a descoberta do material químico do DNA, onde é armazenada a informação gênica (ALBERTS, 2010), surgiram muitos projetos para sequenciamento e estudo do genoma de organismos. Um dos projetos mais complexos foi o Projeto Genoma Humano (PGH), que teve início em outubro de 1990. O objetivo do PGH foi descobrir e sequenciar os genes humanos e torná-los acessíveis para estudos sobre doenças, medicamentos e qualidade de vida.

O NCBI (National Center for Biotechnology Information), fundado em 1988, possui informações genômicas cadastradas e tem várias aplicações que auxiliam na manipulação das mesmas (NCBI, 2015a). Uma das bases de dados criada para tal fim foi a GenBank, banco com informações de alguns sequenciamentos genéticos de proteínas de acesso público (NCBI, 2015a). Muitas organizações, como por exemplo, a FIOCRUZ efetuam estudos e aplicações similares.

Com o cadastramento e disponibilização de informações genômicas, criou-se uma rotina segundo a qual alguns cientistas trabalham com sequenciamento de DNA realizam o armazenamento de informações gênicas. Já outros, trabalham com a investigação dessas bases gênicas, realizando operações rotineiras de recuperação e análise dessas informações. Uma das análises feitas é a similaridade entre uma sequência e outra a fim de descobrir possíveis características entre elas (NCBI, 2015a).

Além do armazenamento, existe a necessidade de análise dessas informações, o que torna indispensável a utilização de ferramentas computacionais eficientes para a interpretação dos resultados obtidos. A Bioinformática nasceu dessa necessidade. Uma nova ciência que envolve a união de diversas linhas de conhecimento - a engenharia de software, a matemática, a estatística, a ciência da computação e a biologia molecular, entre outras.

A tecnologia da informação, uma das linhas de conhecimento da bioinformática, tem um grande desafio: alinhar o conhecimento biológico ao conhecimento sobre a informática para que o trabalho de pesquisa e desenvolvimento de metodologias voltadas ao estudo dos genes seja facilitado. Uma dessas facilidades é a análise de arquivos com sequências gênicas, pois há sistemas capazes de fornecer informação mais detalhada e rápida, facilitando a coleta de dados relevantes(ALTSCHUL, 1990).

Cabe ressaltar que, com a constante redução dos custos e evolução dos mecanismos que efetuam o sequenciamento de DNA, estão sendo produzidas e disponibilizadas quantidades enormes de informações sobre o genoma de indivíduos(NCBI, 2015b). O foco do presente trabalho é mais voltado para o estudo de genomas de bactérias, área na qual a geração e disponibilização de genomas ocorre de forma muito intensa. Tais dados, referentes aos filamentos de DNA produzidos, são destinados aos estudos da genômica e metagenômica.

Com o acesso a tantas informações gênicas surge a necessidade de compreender melhor "o que", "como"e "onde"aplicar os conhecimentos extraídos. Para os pesquisadores em biologia molecular, uma das primeiras e principais características investigadas entre as sequências genômicas é a busca pela similaridade entre dois ou mais filamentos(MILLER, 2004). Isto é identificado através do alinhamento das bases, ou seja, comparação entre uma sequência e outra.

Na Bioinformática, embora recente, a genômica comparativa é o estudo da relação entre genomas de diferentes espécies ou linhagens biológicas, desde estruturas até as suas funções. Um dos métodos para encontrar essas características são os genes homólogos. Pode-se dizer que a detecção de homólogos é um grande campo de pesquisa nessa área de atuação (MILLER, 2004). A busca por homólogos(EDDY et al., 2009)[1] tem um papel fundamental em pesquisas ligadas, por exemplo, à filogenia, que tem por objetivo detectar sequências homólogas entre organismos para o estudo da relação evolutiva. Uma das ferramentas que implementam algoritmos para

[1]Homologia é o estudo biológico das semelhanças entre estruturas de diferentes organismos que possuem a mesma origem ontogenética e filogenética. Tais estruturas podem ou não ter a mesma função. A homologia tem sido uma forte evidência em favor da Teoria da Evolução, pois ela sugere ancestralidade comum entre organismos diferentes possuindo estruturas frequentemente semelhantes com a mesma origem embriológica, lembrando que o desenvolvimento do embrião recapitula parcialmente as origens do organismo, como por exemplo, as nadadeiras ventrais dos peixes e os membros dos mamíferos.

realizar a detecção de homólogos é o BLAST (ALTSCHUL, 1990).

Desenvolvida pelo NCBI, a ferramenta BLAST (Basic Local Alingment Seach Tool) busca alinhar sequências de bases de dados genômicos, apresentando, entre outras características, a homologia (NCBI, 2015a)(ALTSCHUL, 1990). Toda essa demanda pelo processamento de dados na área de Bioinformática surge da necessidade e urgência em estudar os genes, proteínas e suas possíveis características.

Cabe observar que conforme sugere a Lei de Moore (WETTERSTRAND, 2011) o poder computacional dos computadores dobraria a cada 18 meses. Assim, nem sempre o poder computacional acompanha o crescimento das informações produzidas pela bioinformática, visto que as informações são produzidas de forma exponencial diariamente(NCBI, 2015b). Nesse sentido, é com frequência que a execução de determinadas ferramentas computacionais não trazem os resultados em tempo hábil. Por exemplo, em aplicações que foram feitas para serem executadas em máquinas locais e que processam poucas informações por segundo o tempo para processar é moroso. O BLAST, ferramenta que é estudada nesse trabalho, levaria anos para processar base de dados muito grandes(NORDBERG, 2013).

Essa limitação de processamento na área da Bioinformática se deve ao fato de que determinados algoritmos e ferramentas computacionais(LEO; SANTONI; ZANETTI, 2009) não foram preparadas para usar processamento paralelo e, portanto, não é escalável(HASHEM, 2015). Por conseguinte, para melhorar o poder computacional, tais algoritmos devem ser adaptados para utilizar o processamento paralelo e distribuído ou, ainda, serem reescritos.

No entanto, com as tecnologias atuais de paralelismo (MPI, Hadoop, Spark), há casos em que se pode encontrar ferramentas que facilitam a implementação do paralelismo. Além disso, permitam que a execução de determinados tipos de algoritmos sejam adaptados para que sejam processados em um *cluster* com muitas máquinas. Isso acontece a fim de que a execução seja concluída em um tempo razoavelmente mais rápido sem ser necessário reescrever toda a aplicação (PIREDDU; LEO; ZANETTI, 2011).

Uma dessas vantagens de se adaptar algoritmo para que esse seja escalável é projetado e observado em um trabalho denominado CloudBLAST (MATSUNAGA; TSUGAWA; FORTES, 2008). Nesse trabalho é proposto um algoritmo para a execução do BLAST em uma Nuvem Computacional utilizando o Hadoop. O CloudBLAST é a base comparativa desta pesquisa de mestrado.

Neste trabalho, pretende-se utilizar uma adaptação da ferramenta BLAST (executar o BlastP) sobre o *framework*[2] Apache Spark com o propósito de que a execução seja produzida em tempo

[2]Framework é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica.

relativamente mais rápido que aquele obtido quando executado em máquinas locais. Essa abordagem se dará com a ideia da execução da ferramenta BLAST em vários nós sem que seja necessário reescrever o seu algoritmo para ser executado em sistemas distribuídos.

1.2 Motivação

A necessidade de aumentar o poder computacional para execução de tarefas diárias de biólogos e pesquisadores afins tem se tornado uma forte motivação para desenvolvedores de aplicações em Bioinformática.

Para melhor atender à demanda pelo processamento e análise das informações geradas pelos sequenciadores e analisadores de genes, um caminho promissor é o uso de sistemas paralelos para computação de alto desempenho (HPC - High Performance Computing), como por exemplo: Clusters, Grids e Computação em Nuvem. Com estes tipos de sistemas é possível a utilização de diversas máquinas para obtenção de desempenho de tarefas computacionais em menor tempo.

No passado houve iniciativas e trabalhos que utilizaram a ferramenta Apache Hadoop na área de Bioinformática para execução da ferramenta BLAST. Esses trabalhos tiveram como foco a redução do tempo de execução do BLAST, que utilizaram a funcionalidade do Hadoop Streaming. Dos trabalhos estudados, alguns refazem o algoritmo da aplicação e outros apenas abstraem a ideia do paralelismo para a execução em sistemas distribuídos através da divisão do trabalho em vários nós.

Dentre as ferramentas e bibliotecas de computação distribuída e paralela, o Apache Spark tem se mostrado vantajoso e flexível em relação ao modelo de programação MapReduce do Hadoop para alguns tipos de aplicações. Neste contexto, cabe avaliar se as características do Spark são adequadas para ser utilizado como um *cluster* para ferramentas já desenvolvidas como, por exemplo, o BLAST, para obter escalabilidade.

Segundo dados da Organização Mundial de Saúde, o crescente abuso no uso clínico de antibióticos tem tornado os microorganismos mais resistentes aos fármacos disponíveis e desta forma a criação de novos medicamentos é cada vez mais urgente (WHO, 2015). Nesse contexto, a FIOCRUZ tem despendido esforços no estudo de bactérias resistentes à radiação. Apenas para citar exemplos, análises com cerca de 10 bactérias poderiam levar entre 10 e 12 horas executando em um único computador com múltiplos cores de processamento. Já a comparação dos genomas dessas bactérias com metagenomas levava mais de 30 dias em uma única máquina. Além disso, ferramentas de processamento distribuído existentes possuíam limitações de uso

de memória. Assim, a presente pesquisa de mestrado foi desenvolvida com o objetivo de desenvolver ferramentas que permitam reduzir o tempo de processamento quando se aumenta a quantidade de informações a serem processadas e o número de resultados produzidos ao utilizar o BLAST.

Portanto, a motivação para a realização da presente pesquisa de mestrado se resume na busca por alternativas à execução de ferramentas de Bioinformática em tempo hábil, utilizando a computação paralela e distribuída.

1.3 Objetivos

O objetivo principal desta pesquisa é propor uma abordagem computacional, que possibilite a utilização do *framework* Apache Spark para execução da ferramenta BLAST. Para alcançar esse objetivo principal, faz-se necessário atingir os seguintes objetivos específicos:

- Adaptar a execução do *software* BlastP[3] para executar em ambiente distribuído;
- Implementar ferramental de software (scripts) que automatizem a execução eficiente do software BlastP ambiente de nuvem computacional;
- Avaliar o uso de uma infraestrutura de nuvem pública, como a Google Cloud[4], para a execução escalável e eficiente da ferramenta BlastP;
- Comparar o desempenho computacional deste trabalho com outros trabalhos relacionados.

1.4 Organização do Trabalho

O presente trabalho está organizado da seguinte forma: o **Capítulo 1** descreve a introdução, motivação e objetivos desta dissertação; no **Capítulo 2** são apresentados termos de biologia molecular, a importância da bioinformática, ferramenta BLAST e termos computacionais que se constituem a base para o entendimento da pesquisa deste trabalho de mestrado. No **Capítulo 3** são apresentados os trabalhos correlatos à abordagem de criação de uma nova ferramenta desenvolvida em Spark; no **Capítulo 4** é apresentada a definição deste trabalho de mestrado, justificando a realização da pesquisa e exibindo o cronograma de tarefas planejadas e; no **Capítulo 5**

[3] Seção 2.3.1, usaremos o BlastP que trabalha somente com proteínas.
[4] Cloud Google - https://cloud.google.com

são mostrados os resultados obtidos e a discussão da solução desenvolvida, bem como propostas de trabalhos futuros e, por fim o **Capítulo 6** apresenta uma conclusão desta dissertação.

Capítulo 2

REFERENCIAL TEÓRICO

Este capítulo apresenta, de forma resumida, os fundamentos científicos e tecnológicos relacionados à Bioinformática. Além disso apresenta ferramentas que serão utilizadas neste projeto, tais como, Apache Hadoop e Apache Spark.

2.1 Processamento Genômico

Na segunda metade da década de 90, com o surgimento dos sequenciadores automáticos de DNA, houve uma explosão na quantidade de sequências identificadas em diversas espécies como, por exemplo, o Projeto Genoma Humano (PGH) [1] que teve como objetivo sequenciar o gene humano. Os genes de cada indivíduo definem suas características e permitem que se estude sua fisiologia, hereditariedade, morfologia, grau de parentesco com outras espécies, dentre outras informações (GRIFFITHS, 2008).

A genômica comparativa é o estudo da relação entre genomas de diferentes espécies ou linhagens biológicas, desde estruturas às funções dos mesmos. A genética comparativa tem por finalidade comparar genomas com o objetivo de encontrar sequências homólogas através de similaridade. Pode-se dizer que a detecção de homólogos é um grande campo de pesquisa nessa área de atuação (MILLER, 2004). A busca por homólogos tem um papel fundamental em pesquisas ligadas, por exemplo, à filogenia, que tem por objetivo detectar sequências homólogas entre organismos para o estudo da relação evolutiva entre os mesmos. Uma das ferramentas que implementam algoritmos para realizar a detecção de homólogos é o BLAST (ALTSCHUL, 1990), cuja relevância tem crescido juntamente com a grande quantidade de dados. Estes dados são oriundos do sequenciamento produzidos pelos novos sequenciadores de genes (MILLER, 2004).

[1] Projeto Genoma Humano - http://www.ufrgs.br/bioetica/genoma.htm

2.2 Bioinformática

A Bioinformática tem surgido como uma ciência multidisciplinar que busca compreender as funcionalidades biológicas das células, mais especificamente dos genes, com a ajuda de ferramentas computacionais. O profissional desta área deve ser capaz tanto de dominar conhecimentos biológicos (Biologia Molecular), quanto prover aplicações ou ter conhecimento de sistemas computacionais que resolvam problemas intrínsecos à área biológica (TAYLOR, 2010).

Os trabalhos de Bioinformática, de um modo geral, consistem em representar computacionalmente técnicas biomoleculares executadas em laboratórios, seja construindo aplicações ou utilizando ferramentas já existentes. Nessa área há um envolvimento nas tecnologias de armazenamento de dados, recuperação, manipulação e distribuição das informações relacionadas às moléculas de DNA, RNA e proteínas.

A Bioinformática é uma área muito promissora e que vem crescendo muito nos últimos anos. Segundo Pivetta (2013, p. 16),

> ... há pouco mais de uma década quase não havia genomas completos para serem analisados. Hoje faltam programas e mão de obra especializada para dar conta da quantidade de sequências de DNA já depositadas em bases públicas de dados e que saem diariamente de uma nova geração de sequenciadores. Extremamente velozes essas máquinas determinam os pares de bases do material genético, as chamadas letras químicas, a um preço milhares de vezes menores do que no início dos anos 2000, quando chegou ao fim a epopeia de sequenciar o primeiro genoma humano.

Com o constante crescimento na geração de dados em genoma, a bioinformática visa auxiliar os pesquisadores em biologia a obter, melhorar, desenvolver e manipular as informações gênicas. Uma vez que estas informações geradas estão armazenadas em arquivos tabulares e em bancos de dados de biologia molecular e podem ser manipuladas computacionalmente. As ferramentas computacionais servem para coletar, organizar e interpretar os dados experimentais obtidos em laboratórios e inferir resultados que poderão ser comprovados na prática (LUSCOMBE, 2001).

2.2.1 Sequenciamento genômico

Um dos motivos para a utilização da computação paralela, distribuída, computação em nuvem e ferramentas que melhoraram o tempo computacional gasto na execução de trabalhos em Bioinformática é a grande quantidade de informações geradas pelos sequenciadores de genoma.

Segundo (GRIFFITHS, 2008), o sequenciamento genético consiste de métodos bioquímicos

que determinam a sequência em que as bases nitrogenadas aparecem (A, T, C, G), no caso do DNA. Logo, surgiram métodos de sequenciamento de genes, que permitiram a definição da sequência das bases do genoma dos organismos. Na segunda metade da década de 90, com o surgimento dos sequenciadores automáticos de DNA, houve uma explosão na quantidade de sequências identificadas de diversas espécies.

O mapeamento de genomas gera diariamente um volume elevado de informações que são sistematicamente armazenadas em bancos de dados computacionais, que servem de fontes de estudo para biologia, medicina, entre outras áreas (NCBI, 2015a).

Um dos primeiros métodos de armazenamento de sequências genômicas foi a criação e manipulação em formato texto, pois, na época, a principal fonte de informações eram publicações em artigos científicos. Com o aumento do volume de dados, as informações genômicas passaram a ser gerenciadas por meio de Sistemas Gerenciadores de Banco de dados (SGBDs).

Com uma quantidade significativa de dados oriundos de sequenciamentos com armazenamentos em bancos de dados, o próximo passo foi disponibilizá-los de forma pública e organizada, por exemplo, por meio da web. Isso proporcionaria um acesso fácil por parte da comunidade científica e interessados. Assim, os bancos de dados de genoma representam hoje uma ferramenta de suporte indispensável aos biólogos e geneticistas (NCBI, 2015b).

Neles são armazenados dados de sequências genéticas e anotações relacionadas às sequências de DNA disponíveis, como por exemplo definição do gene, autores que descobriram a sequência, onde foi publicado, etc. É um banco de dados público, que tem sua atualização realizada por pesquisadores ao redor do mundo.

O GenBank[2] é um banco de dados de nucleotídeos do NLM/NCBI[3], disponibilizado pelo National Institutes of Health (NIH[4]), armazenando informação sobre sequências nucleotídicas de aproximadamente 260.000 espécies. O GenBank faz parte de uma rede de colaboração juntamente com o European Molecular Biology Laboratory (EMBL[5]) e o DNA DataBank of Japan (DDBJ[6]). Juntos esses três bancos formam a International Nucleotide Sequence Database Collaboration (INSDC[7]), armazenando e trocando informações para reunir as sequências nucleotídicas depositadas nesses bancos e garantir que essas sequências sejam acessadas em todo o mundo. Os três bancos de dados de nucleotídeos estão diariamente trocando informações,

[2]GenBank - http://www.ncbi.nlm.nih.gov/genbank/
[3]NLM/NCBI - http://www.ncbi.nlm.nih.gov/
[4]NIH - https://www.nih.gov/
[5]EMBL - http://www.embl.org/
[6]DDBJ - http://www.ddbj.nig.ac.jp/
[7]INSDC - http://www.insdc.org/

portanto sequências encontradas em um banco também serão encontradas nos outros bancos.

As Figuras 2.1 e 2.2 representam o crescimento das bases e sequências armazenadas no GenBank (NCBI, 2015b) o que reflete o constante crescimento na geração de informações relacionadas à genômica.

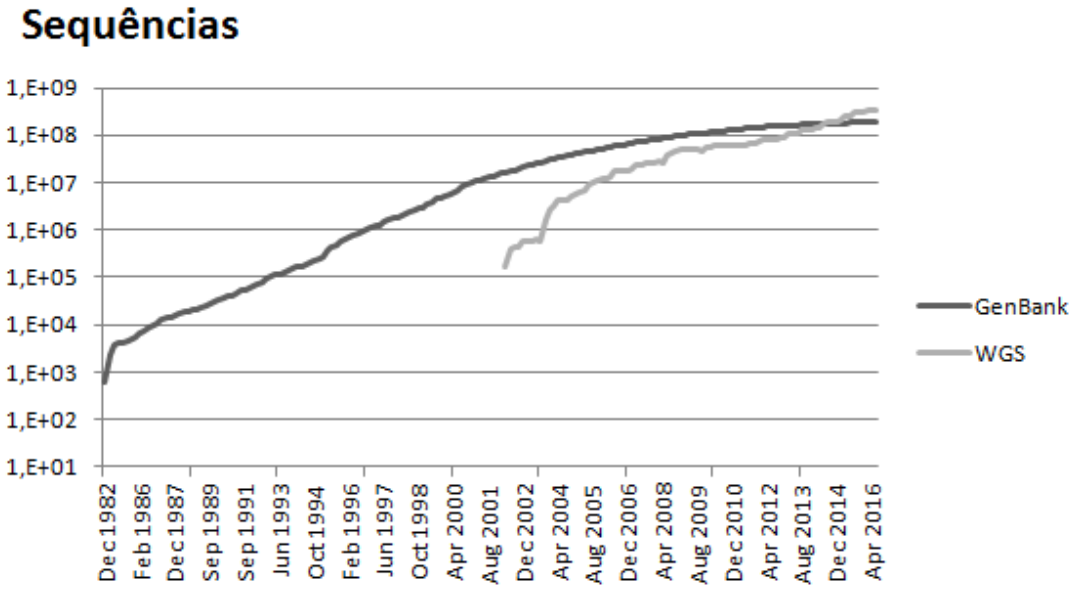


Figura 2.1: Crescimento base de dados GenBank em relação ao número de sequências - Adaptado de: NCBI http://www.ncbi.nlm.nih.gov/ genbank/statistics

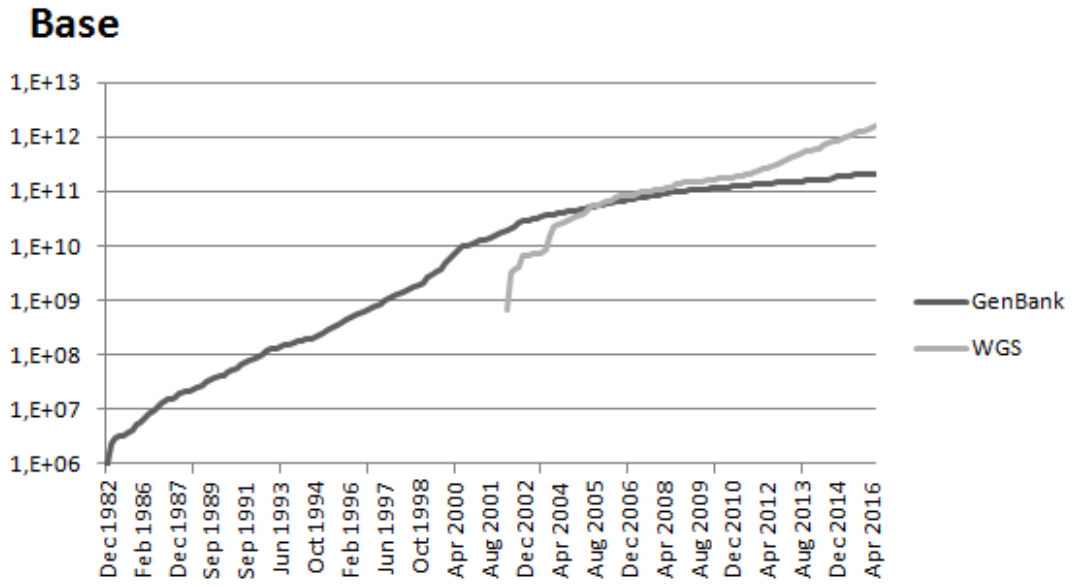


Figura 2.2: Crescimento base de dados GenBank em relação ao número de bases- Adaptado de: NCBI http://www.ncbi.nlm.nih.gov/ genbank/statistics

O Genbank começou a acumular sequências e sua versão 3 em 1982 apresentava 606 sequências nucleotídicas e 680.338 bases. O número de sequências acumuladas cresceu muito

rápido e atualmente são 171 milhões de sequências depositadas. Em 2002, juntamento com o GenBank, começou o acúmulo de sequências geradas nos Whole Genome Shotgun (WGS [8]).

O crescimento dos WGS foi muito rápido e pode-se observar que o crescimento em bases e número de sequências foi muito maior que o crescimento do GenBank. Atualmente o WGS tem mais bases e quase o mesmo número de sequências que o GenBank. Essas duas divisões, GenBank e WGS, apesar de manterem informações do mesmo tipo (sequências nucleotídicas) elas são mantidas separadas.

2.2.2 Crescimento dos dados

Todo esse crescimento das informações gênicas se deve ao surgimento dos Sequenciadores de Nova Geração - NGS (do termo na língua inglesa, Next-Generation Sequencing)(TREANGEN; SALZBERG, 2012). Esses são processos de sequenciamento de DNA que utilizam metodologias diferentes da de Sanger, com o objetivo de acelerar e baixar o custo do processo de sequenciamento. Apesar de se diferenciarem consideravelmente entre si, todos os sequenciadores de NGS se baseiam no processamento paralelo massivo de fragmentos de DNA.

Por exemplo, a eficiência dos novos métodos de sequenciamento aumentou em cerca de 100.000 vezes por década desde que a sequenciamento do genoma humano foi completado. NGS são máquinas que podem sequenciar todo o genoma humano em poucos dias. Essa capacidade tem inspirado uma grande quantidade de novos projetos que visam o sequenciamento do genoma e a descoberta de novas características entre as espécies (TREANGEN; SALZBERG, 2012).

Além das informações genômicas armazenadas em arquivos e bancos de dados, a manipulação, observação e análise desses dados, produzem novas informações que são armazenadas em formato de textos ou em novos bancos de dados. Pode-se descrever algumas dessas informações geradas pela análise gênica: sequência do genoma, mutação, expressão do gene (onde e quando ocorre), sequências de proteínas, localização de proteínas, doenças, entre outras. Para armazenar toda essa base de conhecimento gerada e que seu acesso seja público, organizado e gerenciado faz-se o uso dos bancos de dados (NCBI, 2015a).

Cabe ressaltar que nem sempre o poder computacional acompanha o crescimento das informações, para que possam ser processadas e analisadas em tempo hábil. Conforme é observado

[8] WGS - são conjuntos de genomas incompletos ou cromossomos incompletos de procariontes ou eucariontes que estão, geralmente, sendo sequenciados por uma estratégia em que todo o genoma é separado em pedaços de vários tamanhos que, depois de sequenciados e unidos, utilizando um software especial.

na Figura 2.3[9] o custo para se obter sequências genômicas tem caído muito mais rápido do que o custo para se obter mais poder computacional.

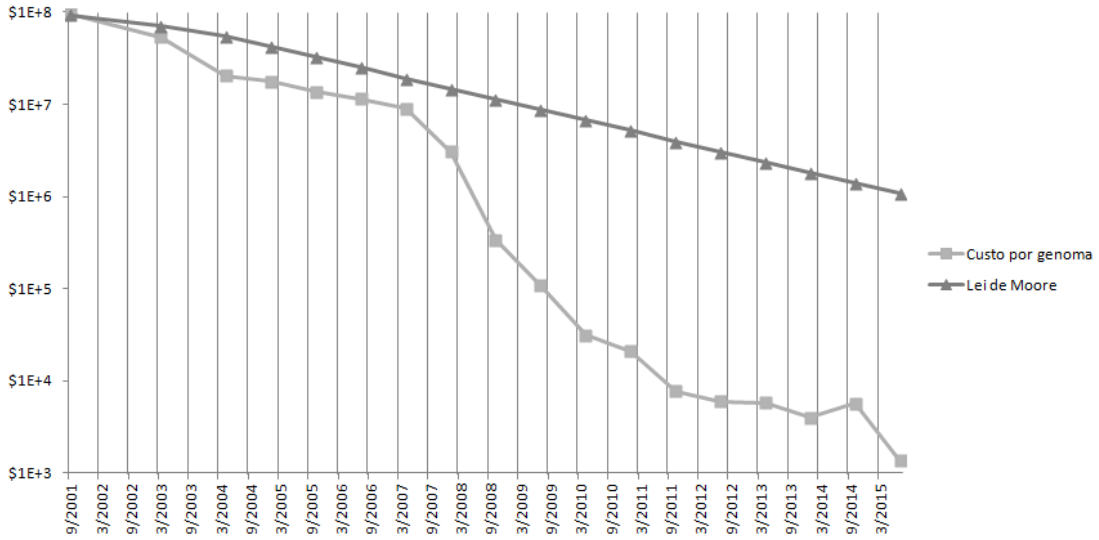


Figura 2.3: Custo para mapear as informações de um genoma e custo computacional para processar a informação produzida - Adaptado de: NIH - https://www.genome.gov/sequencingcosts/.

2.2.3 Alinhamento de sequências

Alinhamento é uma forma de se comparar sequências biológicas (DNA ou proteína) entre si. Ao se estabelecer um alinhamento entre essas sequências, é possível inferir se elas estão evolutivamente relacionadas ou não, como por exemplo, através da homologia(ALBÀ; CASTRESANA, 2007). Uma sequência biológica é representada por meio de uma cadeia de caracteres extraídos de um determinado alfabeto[10].

Existem ferramentas de comparação de sequências biológicas como o BLAST (Basic Local Alignment Search Tool), descrita na **Seção 2.3.1**, que buscam o alinhamento de sequências para a análise de similaridades a fim de inferir possíveis características entre as várias sequências manipuladas.

Assim, comparar duas sequências biológicas é equivalente a comparar duas cadeias de caracteres. Exemplo:

[9] www.genome.gov/sequencingcosts

[10] Alfabetos:
No DNA o alfabeto é composto por A - Adenina, C - Citosina, T - Timina, G - Guanina;
No RNA o alfabeto é composto por A - Adenina, C - Citosina, U - Uracila, G - Guanina;
Nos Códons o alfabeto é composto por A - Alanina, R - Arginina, N - Asparagina, D - Ácido aspártico, C - Cisteína, Q - Glutamina, F - Fenilalanina, L - Leucina, I - Isoleucina, M - Metionina, V - Valina, S - Serina, P - Prolina, T - Treonina, Y - Tirosina, H - Histidina, K - Lisina, E - Ácido glutâmico, W - Triptofano, G - Glicina.

```
SEQUÊNCIA 1     AGTTGGGATCACAGGCTTCT

SEQUÊNCIA 2       AGTTGTCACAGCGTTC
```

Nesse exemplo, as sequências possuem tamanhos diferentes. Por isso, é preciso igualar os tamanhos usando um caractere nulo "-" conhecido como *gap* ou espaço. Com a inserção do caractere nulo obtém-se as "novas" sequências conforme observado abaixo:

```
SEQUÊNCIA 1       AGTTGGGATCACAGGCTTCT
                  |||||   ||||||   |||
SEQUÊNCIA 2       AGTTG---TCACAGGGTTC-
```

Pode-se dizer que o alinhamento em determinados alfabetos é dado pela seguinte definição (KUMAR; FILIPSKI, 2007):

Dadas as sequências,

$$\text{s} = s_1 \ldots s_m$$

e

$$\text{t} = t_1 \ldots t_n,$$

com símbolos pertencentes ao alfabeto A, e com m, n $\geq$ 0, um alinhamento de s e t é um mapeamento de s e t nas sequências s' e t', respectivamente, cujos símbolos pertencem ao alfabeto A' = $A \cup -$, onde o símbolo '–' é chamado de espaço, tal que:

1. | s' | = | t' | = l;

2. a remoção dos espaços de s' e t' leva a s e t, respectivamente;

3. não é permitida a condição s'_i = - = t_i', 1 $\leq$ i $\leq$ l.

Em ferramentas como o BLAST, o alinhamento é local, ou seja, apenas partes das sequências precisam estar alinhadas e não elas completas, como é identificada nos alinhamentos globais.

2.3 Ferramentas de Bioinformática

Com os dados gerados pelos diversos projetos de sequenciamento de genomas, torna-se necessária a utilização de ferramentas computacionais para análise destas informações. As

ferramentas de bioinformática são os algoritmos e softwares projetados para extrair informações significativas da grande massa de dados biológicos (LUSCOMBE, 2001).

A existência de uma grande quantidade de ferramentas disponíveis para análise genômica mostra a necessidade de se atender a diferentes objetivos e a dificuldade que se tem em englobar várias funcionalidades em uma única ferramenta. Além disso, é importante que o resultado da execução das ferramentas seja concluído em tempo hábil e transparente para os pesquisadores (TAYLOR, 2010).

2.3.1 Arquivo FASTA

A pesquisa e manipulação de sequências de genes de DNA ou proteínas faz o uso de arquivos em formatos pré-definidos, para facilitar a manipulação. A ferramenta BLAST, base dessa pesquisa, utiliza os arquivos no formato FASTA ou Multi-FASTA. O formato FASTA tem o caractere $>$ no início da sequência. Nos arquivos Multi-FASTA há várias sequências em um mesmo arquivo e cada uma é identificada pela primeira linha começando com o caractere acima descrito. Na Figura 2.4 é exemplificado um trecho desse tipo de arquivo.

```
>gi|73666633|ref|NC_007354.1|:c2204-1644 Ehrlichia canis str. Jake cromosome,
ATGCTTTTTTACAGGTACATGTTATTCTTATTTTTATTAAAAAATATAAAGAACCTTGTTGTTAGGGTGT
TTTTTCTTGCGTTATTTCTTTATTTTTCTTCCTTTATTTTTATTGCCATGAAGGAAGGAATTAAAGTTGC
TTTTGCATATAAAGATCCTCAAAGTGTTGCTATTTATATTTATAAATACTTAAAGTGGTGTTATTATAAC
TATGATTCACTAACTTATTCTGTATTTTTTAAAGCAGGAGTATCATTAATTATACCTTTTTGGTTTTACT
TGAAGGTTTCTAAAATTAATTGGAATGAATTTTTTAGATGTTTGTTTGAGTATGTGTGTTCTATGTTTAA
AAAGGATAATTATAATAAGGAGAATTTAGAATTTTATAATTATAATGACTTTAACTTTATAGATACTAAT
CATGTTGAAGAAATTGATAAAATAAAAAAGAATGCAATTGCAGAGATAGAGGAATCAATAAATAAGATGG
TTTCAAATATTTTATGTGTGAAAAGTAAAAAGAATAATCAGAATTATGATAATGTGATTGATAGAGACTAA
```

Figura 2.4: Trecho de um arquivo FASTA contendo parte da proteína *Ehrlichia canis*

2.3.2 BLAST

O Basic Local Alignment and Search Tool (BLAST) é uma ferramenta muito poderosa para identificação de sequências em banco de dados que compartilham similaridades com uma sequência consultada (ALTSCHUL, 1990).

O BLAST é utilizado para encontrar regiões de similaridade em alinhamento local entre sequências. O programa compara sequências de nucleotídeos ou sequências de proteínas para calcular a significância estatística dos resultados entre o alinhamento encontrado. Ele pode ser usado para inferir relações funcionais e evolutivas entre sequências de diferentes espécies por meio de informações evidenciadas ao final do resultado de sua execução (VOUZIS; SAHINIDIS,

2011).

Os relatórios e informações gerados pelo BLAST são importantes para quem trabalha com bioinformática, pois possibilita realizar análises de similaridades e depreender novas informações de sequências de DNA, RNA e proteínas. Existe uma linha de pesquisa (NCBI, 2015a) muito importante de pressupostos utilizados em pesquisa biológica que é geralmente seguida pelo uso dos dados produzidos pelo BLAST:

- Genes homólogos: sequência com partes similares;
- Genes ortólogos: sequências que tem a maior similaridade entre múltiplas espécies.

Contudo, é muito importante entender que estas são apenas suposições, e há muitas pesquisas e problemas em que estes pressupostos podem não ser verdadeiros (ALTSCHUL, 1990). No entanto, eles são um ponto de partida razoável para a construção de uma base de conhecimento sobre um determinado gene que serão confirmados em testes *in vitro*[11]. Para o BLAST, algumas definições merecem destaque:

- Sequências similares: sequências que compartilham um número significativo de nucleotídeos ou aminoácidos em sua estrutura. As sequências podem ser semelhantes devido à homologia ou simplesmente por acaso. Quanto maior a semelhança entre sequências, mais provável é que elas sejam homólogas;
- Sequências homólogas: sequências que estão relacionadas através de ancestralidade comum. A homologia é qualitativa, uma vez que duas sequências são ou não relacionadas através de uma ascendência comum. As sequências homólogas podem variar muito em seu nível de similaridade de 100% a 0%;
- Alinhamento local: um alinhamento de sequências que se estende apenas por parte da sequência;
- Alinhamento global: um alinhamento de sequências que se estende ao longo de toda a sequência (de ponta a ponta).

Cada sequência similar identificada através de uma pesquisa é conhecida como *hit* e são acompanhadas de alinhamentos, juntamente com o *score*, e de uma estimativa de significância,

[11] *in vitro* - é uma expressão latina que designa todos os processos biológicos que têm lugar fora dos sistemas vivos, no ambiente controlado e fechado de um laboratório e que são feitos normalmente em recipientes de vidro. Foi popularizada pelas técnicas de reprodução assistida.

denominada de *e-value*. O *e-value* é proporcional à probabilidade de um *hit* com o seu alinhamento ser encontrado ao acaso. Assim, quanto menor o *e-value*, mais significante é o *hit*. Esses e outros parâmetros podem ser alterados quando se executa a aplicação, que é encontrada via Web ou instalada em uma máquina qualquer.

A estratégia usada no BLAST para a determinação dos *hits* é a busca de "sementes", que consiste em pares de sequências muito curtas entre as sequências em estudo, as quais são estendidas em ambos os lados. A extensão prossegue até o alcance dos escores máximos. Nem todas as extensões são investigadas, porque o programa compara os escores destas extensões com um limiar cuidadosamente escolhido pelo algoritmo da aplicação (HIGA, 2001).

O programa BLAST é bem amplo e pode ser considerado uma "família de serviços"(HIGA, 2001). Ele possui algoritmos para análises entre sequências e banco de DNA e proteína dentre outras funcionalidades. As variações do programa BLAST são resumidas na Tabela 2.1 (ROSA, 2006).

Tabela 2.1: Programas BLAST

Programa	**Consulta**	**Banco de Dados**	**Alinhamento**	**Descrição**
BLASTN	DNA	DNA	DNA	Compara uma sequência de nucleotídeos contra um banco de dados de sequências de nucleotídeos.
TBLASTX	DNA	DNA	Proteína	Traduz uma sequência de nucleotídeo para aminoácidos e compara com o banco de proteínas.
BLASTX	DNA	Proteína	Proteína	Compara uma sequência de nucleotídeos contra um banco de dados de sequências de proteínas.
TBLASTN	Proteína	DNA	Proteína	Traduz uma sequência de aminoácidos para nucleotídeo e compara com o banco de dados de genes.
BLASTP	Proteína	Proteína	Proteína	Compara uma sequência de aminoácidos contra um banco de dados de sequências de proteínas.

Esse material apresenta apenas uma pequena explicação do que e como o BLAST funciona, pois o foco deste projeto de mestrado é abstrair a ideia de paralelizar a execução de um dos sub-programas do BLAST entre *n* nós e não reescrever o seu algoritmo. Pois, conforme é

observado em Taylor (2010) a ideia é reutilizar a aplicação sem reescrita de código. Na abordagem utilizada por Taylor (2010) faz-se adaptações para executar determinada aplicação, no caso, o BLAST, para que seja processado em vários nós.

Os programas descritos na Tabela 2.1 são os utilizados na busca pelos alinhamentos nas sequências confrontadas com o banco de informações gênicas. Há, ainda, outras aplicações e funcionalidades que compõem os serviços do BLAST[12], são eles:

- blastdbcheck - Checa a integridade de um banco de dados BLAST;
- blastdbcmd - Retorna sequências ou outras informações de um banco de dados BLAST;
- blastdb_aliastool - Cria aliases, ou seja, nomes amigáveis para um banco de dados BLAST;
- blast_formatter - Formata uma saída do BLAST usando um ID pedido atribuído ou salvando em um arquivo;
- convert2blastmask - Converte máscara de letras minúsculas para um formato que pode ser executado pelo *makeblastdb*;
- deltablast - Procura uma sequência de proteína contra um banco de proteína, usando um algoritmo mais sensitivo;
- dustmasker - Mascara uma região com baixa complexidade para entradas que sejam de nucleotídeos;
- legacy_blast.pl - Converte uma busca feita num BLAST legado para a versão atual do BLAST e o executa;
- makeblastdb - Formata uma entrada FASTA para um banco de dados BLAST;
- makembindex - Indexa um banco de dados de nucleotídeo para ser usado com o *megablast*;
- makeprofiledb - Cria um domínio do banco de dados de uma lista de entrada com as posições pontuadas pelo *score* geradas pelo *psiblast*;
- psiblast - Encontra membros de uma família de proteínas, identifica parentesco com uma sequência de proteínas ou constrói posições específicas de acordo com uma matriz de uma sequência de entrada;

[12]Extraído de: http://www.ncbi.nlm.nih.gov/books/NBK52640/

- rpsblast - Procura uma proteína contra um domínio de banco de dados conservado para identificar domínios de funcionalidades presentes na sequência de entrada;
- rpstblastn - Procura uma sequência de nucleotídeo, traduzindo dinamicamente para todos os primeiros seis frames, contra uma um domínio de banco de dados conservado;
- segmasker - Mascara para regiões de baixa complexidade uma entrada com sequências de proteínas;
- update_blastdb.pl - Downloads banco de dados pré-formatados do NCBI;
- windowmasker - Mascara sequências repetidas de nucleotídeos são encontradas;
- blastclust - Organiza sequências no formato FAST em grupos relacionados;
- copymat - Copia saídas do *psiblast* para entradas do *makemat*;
- formatrpsdb - Formata arquivos com pontuações em um banco de dados *rpsblast*;
- impala - Programa que busca perfis de proteínas, foi substituído pelo *rpsblast*;
- makemat - Converte arquivos do tipo *copymat* para um formato de pontuação, não é mais necessário com as novas saídas geradas pelo *psiblast*
- seedtop - programa de pesquisa padrão.

O *blastp*, algoritmo utilizado neste trabalho é melhor descrito em Oehmen e Nieplocha (2006), Liu, Schmidt e Muller-Wittig (2011) e Jacob (2008).

A ideia de aproveitar o algoritmo do BLAST e adaptá-lo para ser executado de forma distribuída surge da grandiosidade desta gama de aplicações. Por conseguinte, reescrevê-lo de forma completa e correta levaria muito tempo (LEO; SANTONI; ZANETTI, 2009) e é justamente essa abordagem que é seguida nesse trabalho de mestrado, adaptar o *blastp* para ser executado em nuvens computacionais.

2.4 Big Data

Segundo (LANEY, 2001), Big Data está relacionado à velocidade, volume e variedade em que os dados são gerados, o que ocasiona um gargalo no processamento de informações com as tecnologias de processamento local com poucos *cores* e algoritmos sendo executados de

forma sequencial (WETTERSTRAND, 2011). Para (JAGADISH, 2014), há várias questões e desafios envolvidos com o tema *Big Data*, tais como: colaboração humana, privacidade e propriedade dos dados, consistência e completude das informações, interpretação, modelos e análises, integração, agregação e representação dos dados, informação, extração, clareza e aquisição das informações.

A Figura 2.5 descreve os fatores exemplificados por Jagadish (2014) em relação aos diversos desafios relacionados com Big Data. Dentre esses desafios, esta pesquisa está relacionada com um aspecto em particular, a escalabilidade. As áreas hachuradas são as que estão diretamente ligadas à bioinformática e crescimento exponencial das informações.

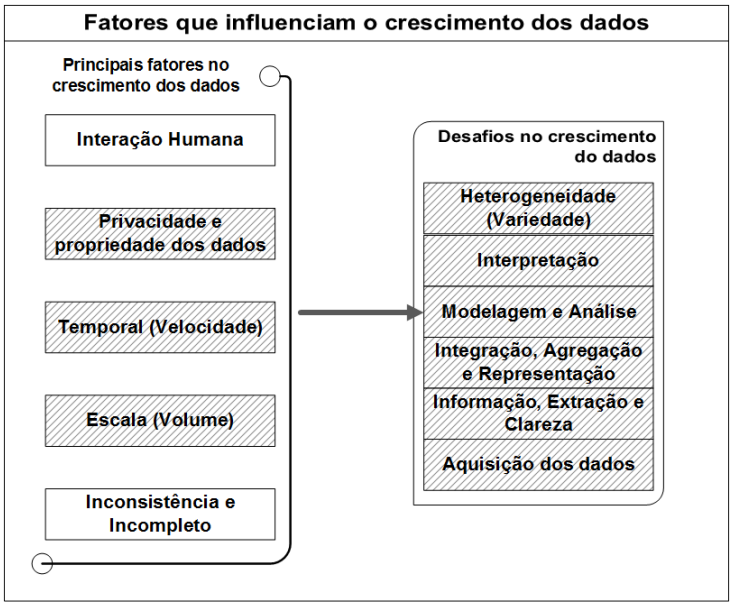


Figura 2.5: Características do *big data*. As características que compõem o crescimento das informações e os principais desafios ao processar essas informações - Adaptado de (JAGADISH, 2014).

Na Figura 2.6 é exemplificado como alguns termos estão ligados ao *big data* e um modelo de como seria a replicação das informações, considerando um sistema de arquivos distribuídos. Os termos que circundam o círculo da Figura 2.6 são aqueles que uma informação precisa ter: estar segura, distribuída, escalável e acessível a qualquer momento são algumas premissas. Para que isso ocorra é necessário ter redundância dessas informações. Sistemas de arquivos distribuídos, como o HDFS (**Seção 2.6.2**) permitem ter essas demandas.

Pode-se dizer que o desafio encontrado em relação ao termo Big Data para os profissionais que fazem uso desses dados é processar, analisar e manter a corretude dessas informações em tempo hábil para cada problema proposto (SONNHAMMER, 2014). De modo geral, observa-se que os dados gerados na Bioinformática apresentam desafios semelhantes aos descritos em (JAGADISH, 2014) e as ferramentas até então desenvolvidas dispendem grande quantidade de

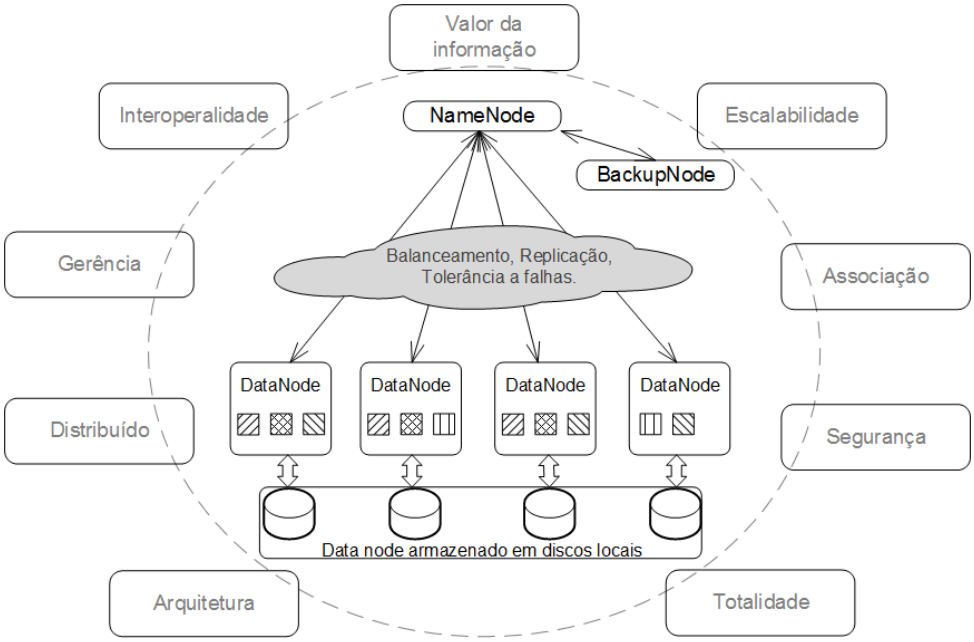


Figura 2.6: Arquitetura *big data*. Desafios em relação à arquitetura de grande quantidade de dados e um modelo abstrato da replicação dessas informações em um sistema distribuído.

tempo na execução dessa massa de dados produzida na genômica(O'DRISCOLL; DAUGELAITE; SLEATOR, 2013). Isso ocorre porque armazenar, analisar e processar de forma eficiente e correta os dados produzidos no sequenciamento e análise genômicos é uma tarefa diária e dispendiosa (PIVETTA, 2013). A nova geração de sequenciadores tem provocado um aumento ainda maior dos dados produzidos (WIEWIORKA, 2014).

Além dos dados produzidos pelo mapeamento e sequenciamento genético (**Seção 2.2.1**) há outros resultados relevantes que se originam dessas informações. Segundo Sonnhammer (2014) o rápido crescimento do sequenciamento genético traz a necessidade de identificar genes homólogos e outras características que tem demonstrado alguns desafios para a bioinformática: processar, analisar e armazenar os padrões encontrados em tempo adequadamente baixo (DAI, 2012).

2.5 Modelo de programação MapReduce

O MapReduce é um modelo de programação funcional e paralela comumente utilizado para processamento de grandes quantidades de dados de forma distribuída que foi apresentado e popularizado pela Google em meados de 2004 (DEAN; GHEMAWAT, 2008).

Este modelo de programação foi inspirado em linguagens funcionais, como Lisp e Haskell. A manipulação e execução dos trabalhos feitos pelo MapReduce utiliza basicamente duas

funções: *map* e *reduce*.

Com essa abordagem de programação é possível a execução de programas em ambientes paralelos, heterogêneos e com tolerância a falhas de forma transparente para o usuário. Assim, arquivos com grande quantidade de dados puderam ser processados em tempo hábil. Outras necessidades para que a execução tivesse a complexidade abstraída do usuário é o controle da distributividade e paralelismo, distribuição dos dados entre os nós e balanceamento do processamento (NGUYEN; SHI; RUDEN, 2011).

O conceito do modelo de programação MapReduce foi proposto inicialmente na linguagem Lisp [13]. Lisp itera sobre uma sequência de dados aplicando sobre cada um deles a função *map*. O resultado do processo é armazenado em uma nova sequência, que é tratada como um dado do tipo lista. A Figura 2.7 ilustra uma lista e uma função simples em Lisp. As funções são anotadas em forma de prefixo, onde o tipo de operação a ser executada antecede uma ou várias listas.

```
Lista: '(5 6 1 -3 -9 7)

Função aplicada sobre a lista (abs - função que retorna o
valor absoluto dos números):

'(abs'(5 6 1 -3 -9 7)) -> (5 6 1 3 9 7)
```

Figura 2.7: Lista e uma função em Lisp. Para cada valor da lista é aplicado a função, retornando uma nova lista com o resultado da execução da função sobre a lista.

A função *reduce* em Lisp combina ou reduz os resultados armazenados em uma lista provenientes das execuções do *map* e guarda o novo resultado em uma outra lista. A Figura 2.8 ilustra uma operação de soma a partir de uma função *reduce* em Lisp. O modelo MapReduce utiliza-se basicamente de duas funções desenvolvidas pelo usuário, Map e Reduce (DEAN; GHEMAWAT, 2008).

```
(reduce #' + '(5 6 1 -3 -9 7)) -> 7
```

Figura 2.8: Função soma aplicada sobre a lista. Retorna um resultado com o valor da função soma da lista.

Na utilização das funções, a computação é dada a partir do processamento de um conjunto de pares de chaves advindos da função Map. Então são criadas chaves intermediárias que, por sua vez, são repassadas à função Reduce para produção dos resultados finais, conforme o seguinte esquema:

[13] Lisp é uma linguagem de programação funcional. Foi projetada por J. McCarthy em 1959. Mais em: https://common-lisp.net/

map: (chave1, valor1) -> list(chave2, valor2)

reduce: (chave2, list(valor2)) -> list(valor3)

Para um dado valor de entrada valor1, e uma chave expressa por chave1 na função Map, são criados valores intermediários dados por chave2 e valor2 que, por sua vez, servem como parâmetros de entrada para a função Reduce, resultando em um valor, neste caso representado por valor3 (DEAN; GHEMAWAT, 2008).

Portanto, o MapReduce é um modelo de programação que permite o processamento de dados massivos em um algoritmo escalável, paralelo e distribuído, geralmente utilizando um *cluster* de computadores. Por exemplo, tanto o Spark quanto o Hadoop fazem uso dessa metodologia e são utilizados em larga escala por grandes corporações, como Facebook, Yahoo!, Google e Twitter, em aplicações Big Data. Essa abordagem, mesmo não sendo fácil de ser utilizada, é muito útil para aplicações que envolvam dados massivos para processamento paralelo ou até mesmo para processamento de qualquer tipo de dado (DEAN; GHEMAWAT, 2008).

2.6 Apache Hadoop

O projeto Apache Hadoop foi criado pelos Laboratórios da Yahoo![14], e mais tarde se tornou um dos projetos mantidos pela Apache Software Foundation[15] como um *framework* de código aberto para programação escalável, computação confiável e distribuída.

A biblioteca de softwares Apache Hadoop permite o processamento distribuído de grandes conjuntos de informações em um *cluster* de computadores que usam os modelos de programação simples, ou seja, não é necessário equipamentos robustos e confiáveis. Isso acontece devido a algumas características na qual pode-se utilizar máquinas simples (poucos *cores* e pouca memória por equipamento). Ele é projetado para escalar a partir de um único servidor para milhares de máquinas, cada uma oferecendo processamento e armazenamento local (WHITE, 2009).

Assim, diferente de muitos Data Centers que têm equipamentos caros e robustos para serem tolerantes a falhas, o Hadoop permite que o hardware possa ou não proporcionar alta disponibilidade para se recuperar de falhas em disco. Essa recuperação de falhas se dá pela replicação dos dados entre os vários nós e *racks*[16] que compõem o *cluster*. A biblioteca do Hadoop em si é concebida para detectar e tratar falhas na camada de aplicação, de modo a fornecer um serviço

[14] Labs Yahoo - http://labs.yahoo.com/

[15] Apache Hadoop - https://hadoop.apache.org/

[16] Rack é uma armação de metal usada para armazenar vários dispositivos de hardware, como servidores, discos rígidos, modems ou outros equipamentos eletrônicos.

altamente disponível no topo de um conjunto de computadores, cada um dos quais pode ser propenso a falhas (Apache Software Foundation,).

Uma das principais características do Hadoop é o WORM (Write Once, Read Many - característica que privilegia aplicações em que os dados são escritos uma única vez e podem ser lidos muitas vezes). Claramente, esse modelo tem foco em aplicações batch, de alta vazão, em que a latência não é importante. O modelo de armazenamento permite que uma elevada largura de banda agregada de acesso a dados seja feita sem a necessidade de mecanismos de sincronização complexos, uma vez que os dados são estáticos(LEO; SANTONI; ZANETTI, 2009), mas há bibliotecas em Hadoop que utilizam dados dinâmicos para serem processados em tempo real como, por exemplo, o Apache Storm[17].

O modelo de programação e execução do Hadoop, o MapReduce (DEAN; GHEMAWAT, 2008), utiliza as funçãoes *map* e *reduce*. Com esse modelo de programação é possível oferecer a paralelização e operação em grandes *clusters* computacionais, nos quais a execução de tarefas e escalonamento de cargas de trabalho entre os nós podem ser realizados por meio da divisão (*map*) e concatenação dos resultados (*reduce*), onde cada nó Hadoop pode atuar como um mestre ou escravo (WHITE, 2009).

A constituição do Hadoop é dada por módulos que compõem sua estrutura:

- Hadoop Common;
- Hadoop Distributed File System (HDFS);
- Hadoop MapReduce;
- Hadoop YARN.

2.6.1 Hadoop Common

O pacote Hadoop Common é considerado a base para funcionamento dessa ferramenta, uma vez que fornece serviços essenciais e processos básicos, tais como abstração do sistema operacional e seu sistema de arquivo. Nele é encontrado os arquivos e *scripts* (Java Archive - JAR) necessários para iniciar, parar, monitorar e manter o Hadoop em execução. Além de fornecer código fonte e documentação, bem como uma seção de contribuição que inclui diferentes projetos da comunidade Hadoop (Apache Software Foundation,).

[17] Apache Storm - http://storm.apache.org/

O Hadoop Common refere-se à coleção de utilitários e bibliotecas comuns que suportam outros módulos do Hadoop. É uma parte essencial ou módulo do *framework*, juntamente com HDFS, YARN e MapReduce. Como todos os outros módulos, Hadoop Common assume que falhas de hardware são comuns e que devem ser tratadas automaticamente no software pelo Hadoop. Hadoop Common também é conhecido como Hadoop Core.

2.6.2 HDFS

Para tratar a característica de tolerância a falhas, o Hadoop utiliza seu próprio sistema de armazenamento de arquivos chamado de Sistema de Arquivos Distribuídos do Hadoop (HDFS - Hadoop Distributed File System). O HDFS é uma implementação *open source* do GFS[18] que dá suporte ao armazenamento de dados para o Hadoop (BORTHAKUR, 2008).

O HDFS é composto por nós interconectados no local onde os arquivos e diretórios residem. Um *cluster* HDFS consiste em um único nó mestre, conhecido como um NameNode, que gerencia o *namespace* do sistema de arquivos e regula o acesso do cliente aos arquivos. Além disso, os nós de dados (DataNodes), conhecidos como nós escravos, armazenam dados como blocos dentro dos arquivos. Um NameNode gerencia operações de *namespace* do sistema de arquivos do tipo abrir, fechar e renomear arquivos e diretórios. Um NameNode também mapeia blocos de dados a nós de dados, os quais gerenciam as solicitações de leitura e gravação dos clientes HDFS. Além disso, criam, excluem e replicam DataNode de acordo com as instruções do nó de nome dominante (MACKEY; SEHRISH; WANG, 2009).

O HDFS replica blocos de arquivos para tolerância a falhas (por padrão são 3 réplicas entre os *clusters*). A replicação é reconfigurável, mas não para um aplicativo individualmente, e sim par o *cluster* inteiro (várias plicações). O NameNode toma todas as decisões referentes à replicação de bloco. Um dos principais objetivos do HDFS é suportar arquivos grandes. O tamanho de um bloco típico do HDFS é 64 MB. Assim, cada arquivo HDFS é composto por um ou mais blocos de 64 MB. O HDFS tenta colocar cada réplica de um bloco em nós de dados separados para que seja possível não perder essas informações caso um ou alguns nós deixem de funcionar. A Figura 2.9 exemplifica um cliente escrevendo um arquivo no HDFS e as partes aqui descritas.

Ainda sobre a Figura 2.9 todo o topo da imagem diz respeito ao que o cliente visualiza: ele faz uma operação de escrita ou leitura como se estivesse escrevendo em um computador convencional. Porém, o que ocorre é a gravação e replicação desses dados em três blocos

[18] GFS - Google File System http://research.google.com/archive/gfs.html

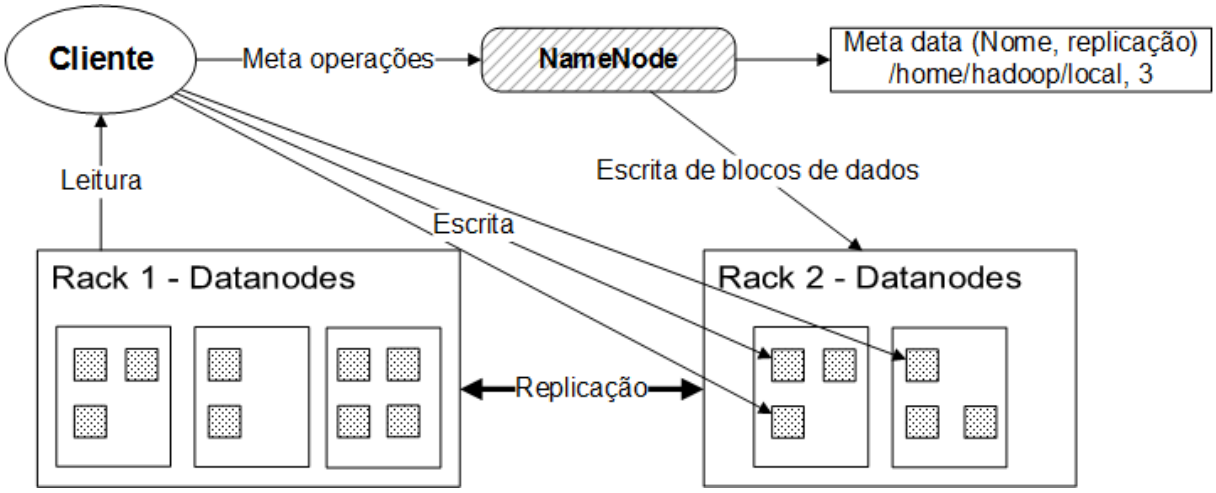


Figura 2.9: Arquitetura HDFS - Adaptado de (HADOOP, 2015b).

diferentes, sendo estas réplicas em nós e *racks* distintos. Por exemplo, quando o cliente escreve um arquivo este é replicado em três réplicas diferentes que por padrão serão alocados em *racks* diferentes. Essa característica é utilizada para prover tolerância a falhas e segurança para evitar que o dado se perca.

2.6.3 MapReduce no Hadoop

Uma aplicação MapReduce em execução no Hadoop recebe seu trabalho dividido entre os nós e os arquivos a serem manipulados pela aplicação residem no sistema de arquivos, HDFS. Em linhas gerais, o Hadoop fragmenta os dados no seu sistema de arquivos quando a função *map* é utilizada e destes fragmentos são geradas tuplas formadas por (chave, valor) produzindo um novo conjunto de chaves e valores intermediários. Em seguida, executa a etapa conhecida como *shuffle* para classificar todos os valores iguais a uma mesma chave para reduzir as tarefas. Por fim, os nós executam a função *reduce* e processam as tuplas geradas pela função *shuffle* produzindo uma tupla única para cada valor e chave correspondentes.

A função *reduce* também se encarrega de escrever as saídas de dados no sistema de arquivos distribuído (Vide Figura 2.9). A Figura 2.10[19] define como é fluxo de trabalho do MapReduce em Hadoop. Quando o cliente inicia a execução de alguma tarefa o nó mestre se encarrega de iniciar e coordenar os processos entre os nós. O Job Tracker envia os trabalhos para nós Task Tracker disponíveis no *cluster*, que usa configurações pré-definidas para manter o mais próximo possível o trabalho dos dados (DEAN; GHEMAWAT, 2008).

Job Tracker sabe qual nó contém os dados e quais outras máquinas estão alocadas mais

[19] https://opensource.com/life/14/8/intro-apache-hadoop-big-data

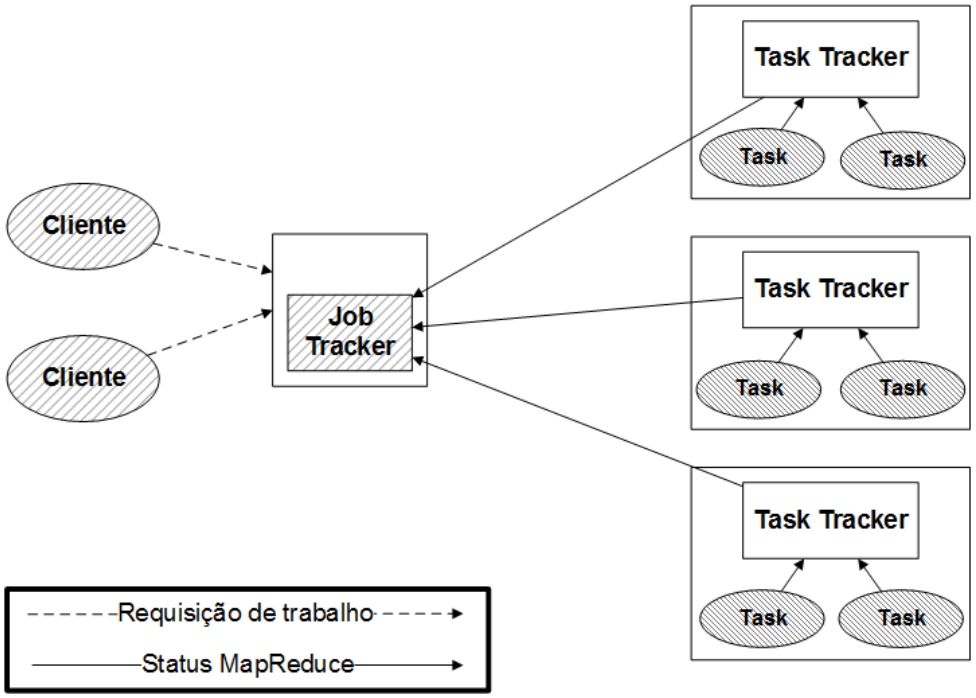


Figura 2.10: Fluxo de execução MapReduce no Hadoop - Adaptado de (HADOOP, 2015c).

próximas. Se o trabalho não pode ser hospedado no nó onde os dados residem, a prioridade é dada a nós no mesmo *rack*. Isso reduz o tráfego de rede entre os equipamentos (LEO; SANTONI; ZANETTI, 2009).

Em cada um dos nós escravos há o Task Tracker. Se o Task Tracker falhar ou o tempo limite expirar o processo é reexecutado. O Task Tracker em cada nó gera um processo de máquina virtual separada para cada sub-tarefa, isso para evitar que o próprio Task Tracker venha a falhar se o trabalho em execução travar a JVM [20]. De tempos em tempos uma confirmação de execução é enviada a partir do TaskT racker ao Job Tracker para verificar seu status (PRATX; XING, 2011).

2.6.4 YARN

O YARN é um escalonador de tarefas onde as funções do JobTracker são repartidas em *deamons* independentes. Uma das funções principais do MapReduce é a de partilhar os dados para as funções de *map* e *reduce*, a outra função é gerenciar as falhas e procurar nós disponíveis para executar a função onde houve falha (VAVILAPALLI, 2013). Para isso o YARN muda um pouco a nomenclatura do nó master e o apelida de Resource Manager (RM) ou Application Master (AM), onde cada função MapReduce é uma aplicação definida pelo nó mestre e o resource manager fica responsável por reordenar os nós no caso de falhas dos nós escravos, NodeManager

[20] JVM - Java Virtual Machine que no Português é Máquina Virtual Java é um programa que carrega e executa os aplicativos Java, convertendo os *bytecodes* em código executável de máquina.

(NM) (WHITE, 2009).

Na Figura 2.11[21] é definido como o YARN coordena o fluxo de execução da aplicação. O YARN foi utilizado porque ele pode ser usado de forma uniforme pelo Spark e pelo Hadoop. De fato, o YARN foi desenvolvido originalmente para a versão 2 do Hadoop. Com o YARN, recursos (por exemplo, *cpu*, memória) podem ser alocados e provisionados para execução de tarefas em um ambiente de computação distribuída. Ele desempenha um papel melhor no gerenciamento da configuração do *cluster* e compartilha dinamicamente os recursos disponíveis, fornecendo suporte para tolerância a falhas, paralelismo inter e intra-nó.

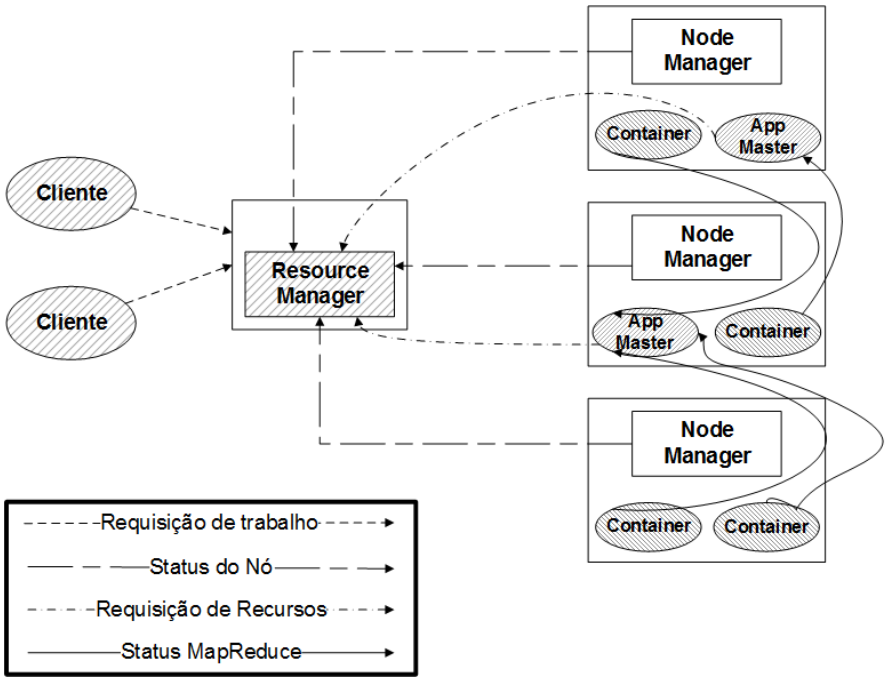


Figura 2.11: Fluxo de execução de um trabalho sobre gerência do YARN - Adaptado de (HADOOP, 2015c).

Em um cluster gerenciado pelo YARN, esse assumirá o papel de dividir os trabalhos entre os nós e gerenciá-los. Nesta dissertação o YARN gerenciará tanto os trabalhos executados pelo Hadoop quanto os do Spark.

2.6.5 Hadoop Streaming

Uma característica dessa ferramenta que vem sendo explorada nos trabalhos citados na **Seção 3** é o Hadoop Streaming que consiste na execução em *batch* de programas, utilizando o esquema do MapReduce (DING, 2011). O *streaming* faz uso de todas as características e funcionalidades do Hadoop e permite que outras aplicações externas sejam reutilizadas durante

[21] https://opensource.com/life/14/8/intro-apache-hadoop-big-data

as fases de *map* e *reduce*.

Dessa forma, o *framework* Hadoop apenas coordena a execução e o fluxo de dados, sendo que as funções executadas são invocações de códigos externos, conforme Figura 2.12. Dessa forma, não é necessário reescrever o código da aplicação inicial para obter escalabilidade, faz-se apenas uma adaptação na forma de execução e o *streaming* fará o papel de paralelizar. O que acontece no *streaming* é a execução da aplicação em cada nó com os dados contidos e enviados a cada um deles.

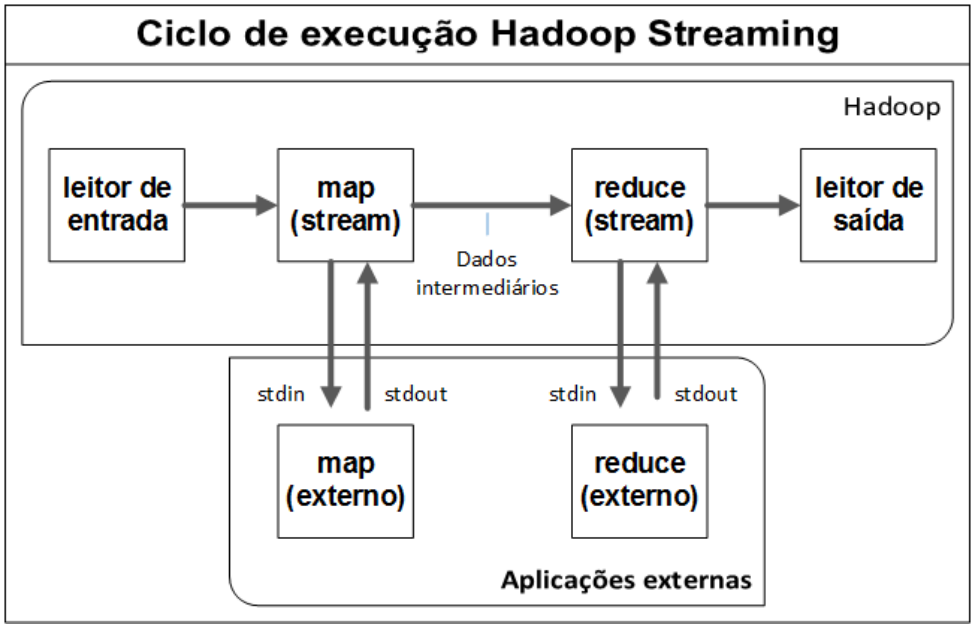


Figura 2.12: Ciclo de execução do Hadoop Streaming - Adaptado de (HADOOP, 2015a)

Muitas ferramentas que trabalham com o paradigma MapReduce para processamento de dados em bioinformática são construídas sobre a ferramenta Apache Hadoop: o Biodoop (LEO; SANTONI; ZANETTI, 2009), CloudBLAST (MATSUNAGA; TSUGAWA; FORTES, 2008), CloudAligner (NGUYEN; SHI; RUDEN, 2011), HBlast (O'DRISCOLL, 2015), Hadoop-BAM (NIEMENMAA, 2012) e um resumo sobre essas funcionalidades (TAYLOR, 2010);(O'DRISCOLL; DAUGELAITE; SLEATOR, 2013).

Na **Seção 3** será estudado dois desses trabalhos: o Biodoop e o CloudBLAST. O Apache Hadoop tem se mostrado promissor na área de bioinformática, pois trabalha com processamento de grandes volumes de dados de forma escalável e rápida e será tomado como base de comparação na execução do Apache Spark para medição dos tempos gastos.

2.7 Apache Spark

Com a evolução das tecnologias computacionais e novos desafios encontrados, surgiu uma nova ferramenta que utiliza a paralelização e execução em memória de aplicações, o Apache Spark (SHORO; SOOMRO, 2015).

O Apache Spark teve seu início como um projeto de pesquisa no laboratório AMP Lab da Universidade de Berkeley[22] em 2009. Em pouco tempo tornou-se um projeto de código aberto e recebeu um grande apoio da comunidade de software livre, o que ocasionou sua inclusão no projeto Apache Incubator em Junho de 2013. Consequentemente, no ano seguinte foi elevado ao nível dos projetos principais da organização Apache Software Foundation devido à sua importância, robustez e foco na área de Big Data (ZAHARIA, 2015).

O Apache Spark tem a característica de ter o processamento mais rápido em algumas aplicações do que outras ferramentas de paralelismo, como por exemplo, o Apache Hadoop, visto que sua execução é em memória e tem seu foco na reutilização de dados já contidos nela. O Spark permite que se trabalhe em um ambiente local, com apenas uma máquina ou em um cluster, como por exemplo o Apache Mesos ou Hadoop YARN (Figura 2.13) e pode funcionar em conjunto com o Hadoop.

Com a característica dos RDDs (Resilient Distribuited Dataset) de reconstrução das falhas, que será visto abaixo, pode-se ter um *cluster* de computadores genéricos[23] (GOPALANI; ARORA, 2015).O RDD é uma abstração de coleção de dados distribuídos. Um RDD pode ser criado a partir de formatos de arquivos distribuídos como, por exemplo, HDFS, por transformação de outros RDDs, paralelização de dados em que será distribuído entre os nós do *cluster* e por modificação na persistência de uma coleção em que um objeto pode ser mantido em cache ou salvo num sistema de arquivos para reutilização futura (GOPALANI; ARORA, 2015).

O suporte a falhas que é uma característica importante do Spark é fornecido pelo RDD. Este provém da hierarquia entre os RDDs (*lineage* - linhagem, ou seja, sequência de como se deu a produção do conjunto de dados). Cada nova partição RDD criada tem informações de como elas foram produzidas; se uma falha ocorre em uma determinada fase do RDD, ele é capaz de consultar os *logs* produzidos em sua *lineage* e reconstruir a estrutura, partindo a execução de onde ocorreu a falha (ZAHARIA, 2015).

Com a implementação Standalone (Figura 2.13(a)) pode-se alocar recursos estaticamente

[22] AMPLab - UC Berkeley - https://amplab.cs.berkeley.edu

[23] Não precisa ser parte de um grande *mainframe* ou com equipamentos caros, pode ser executado em equipamentos comuns não confiáveis, tais como um aglomerado de computadores pessoais.

em todos ou em um subconjunto de máquinas em um *cluster* Hadoop fazendo uso do sistema de arquivos distribuídos. O usuário pode, em seguida, executar trabalhos em Spark com os seus dados armazenados no HDFS. Sua simplicidade faz com que esta implantação seja a escolha para muitos usuários que usam a versão antiga do Hadoop (1.x).

Na implementação sobre o YARN (Figura 2.13(b)) usuários Hadoop que já implementaram ou pretendem implantar Hadoop YARN podem simplesmente executar Spark no YARN sem qualquer pré-instalação ou acesso administrativo necessário. Isso permite que os usuários integrem facilmente Spark em seu ecossistema de aplicações Hadoop e aproveitar as funcionalidades do Spark, bem como de outros componentes que funcionam sobre Spark.

Já na execução do Spark em MapReduece (MR) (2.13(c)) para os usuários do Hadoop que não estão executando YARN ainda, além da implantação Standalone, uma opção é usar SIMR para lançar trabalhos em Spark dentro do MapReduce. Com SIMR, os usuários podem começar a experimentar Spark e usar um *shell* instantaneamente após colocá-lo no *cluster*. Isso reduz enormemente a barreira de implantação e permite que muitos usem Spark. Pois nessa forma de utilização apenas se executa um *shell* no Hadoop com a opção do Spark.

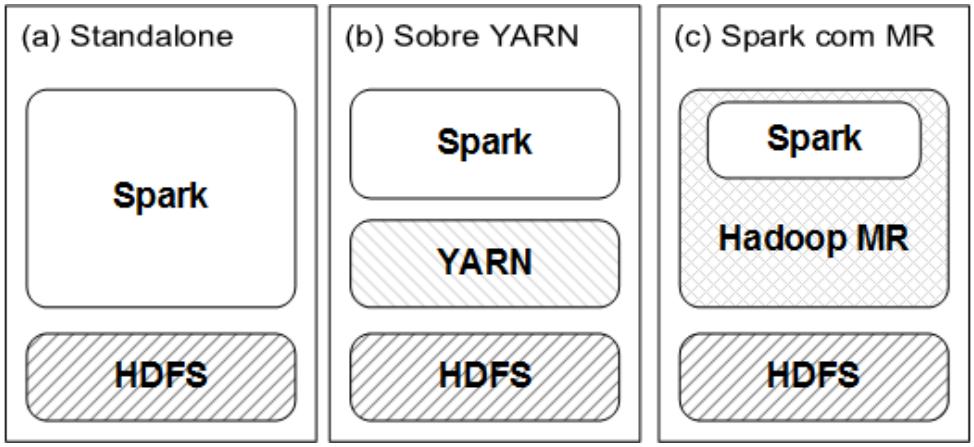


Figura 2.13: Arquitetura de instalação e funcionamento de um *cluster* Apache Spark.

APIs[24] em Java, Scala e Python podem ser utilizadas pelo Spark, facilitando aos desenvolvedores que trabalham com várias linguagens de programação a criação de códigos e escritas de algoritmos. Além disso há bibliotecas específicas para trabalhar com SQL (Spark SQL), aprendizado de máquina (MLlib), processamento de grafos (GraphX), *streaming* de dados (Spark Streaming) (ZAHARIA, 2015) e algumas bibliotecas desenvolvidas por terceiros pré-definidas (SHORO; SOOMRO, 2015).

[24] API é um conjunto de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma baseado na Web. A sigla API refere-se ao termo em inglês Application Programming Interface - Interface de Programação de Aplicativos.

Conforme citado, o Spark tem a execução mais rápida que o Hadoop em algumas aplicações e algoritmos porque possui algumas características em seu framework para a execução de suas aplicações em memória. Quando se tem a aplicação executando em memória, como o algoritmo da regressão logística[25] as aplicações tem se mostrado mais rápidas. Há outras características que ajudam nesse ganho: cache de memória e RDD.

A Figura 2.14[26] exemplifica a velocidade entre processar 100GB de dados em um cluster com 100 nós em Hadoop e Spark. O ganho de desempenho se deve ao fato de o Spark trabalhar com os dados contidos na memória dos nós, para este tipo de algoritmo (APACHE..., 2015).

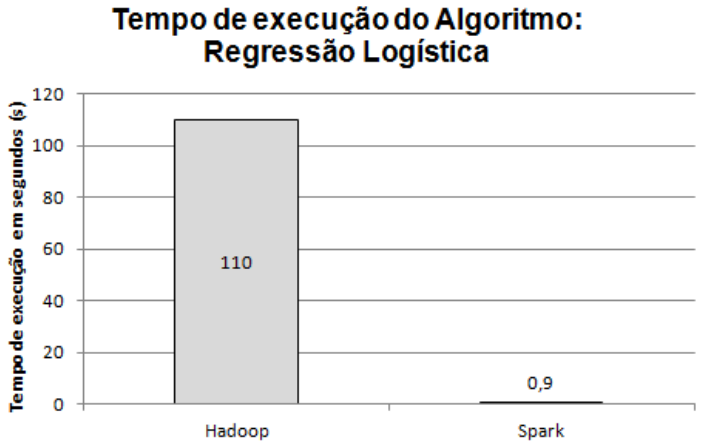


Figura 2.14: Comparação dos tempos de execução do algoritmo de Regressão Logística em Hadoop e Spark.

Quando se trabalha em um *cluster* é necessário que se tenha algum gerenciador da aplicação entre os nós para melhor aproveitamento dos recursos disponíveis. O Spark permite que o *cluster* construído com o YARN gerencie as aplicações Spark. Por conseguinte, o YARN funcionará sendo o driver[27] da aplicação (Figura 2.15), bastando que se instale o Spark compilado para ser executado sobre ele.

Neste trabalho, será utilizada a abstração mostrada na Figura 2.14.b onde é possível trabalhar com o Hadoop e Spark juntos[28], permitindo que o YARN seja o gerenciador dos trabalhos processados em ambos os *frameworks*.

Assim como no framework Hadoop há o Hadoop Streaming (**Seção 2.5**), no Spark há o Spark Pipe que tem a funcionalidade semelhante ao *streaming* do Hadoop, de permitir a execução de aplicações externas previamente desenvolvidas.

[25] Regressão logística - https://pt.wikipedia.org/wiki/Regressão_logística
[26] Apache Spark - http://spark.apache.org
[27] Driver program - direciona os trabalhos e dados aos nós (Workers - Map) e retorna o resultado ao programa principal (Reduce)
[28] Databricks - https://databricks.com/blog/2014/01/21/spark-and-hadoop.html

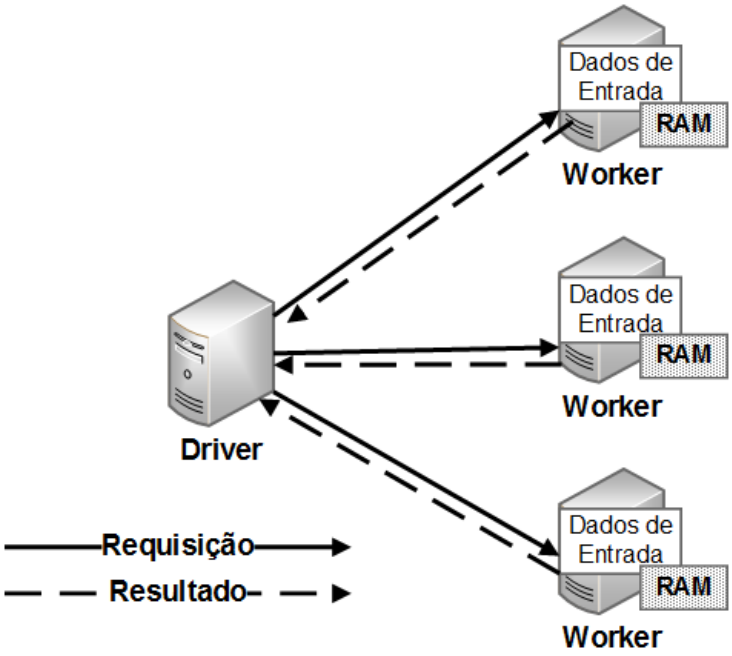


Figura 2.15: Divisão dos trabalhos (driver) entre os nós (workers) - Fonte: (APACHE..., 2015)

Por exemplo, há situações, na análise de dados, que precisa-se usar uma biblioteca externa que não pode ser escrita ou é muito trabalhosa de feita usando Java/Scala. Nesse caso, o operador *pipe* nos permite enviar os dados de RDD para o aplicativo externo (Figura 2.16) a fim de que seja processado (ZAHARIA, 2015). A similaridade entre o método de execução do Hadoop Streaming e Spark Pipe acontece porque em ambos os casos o trabalho é subdividido entre os *n* nós do *cluster*, dando a ideia de várias funções *maps*.

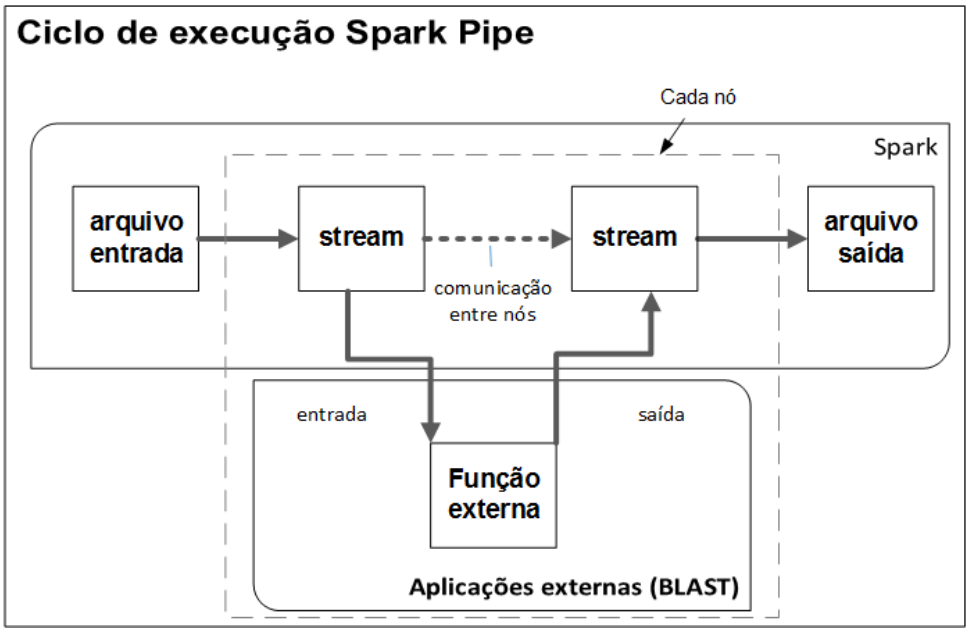


Figura 2.16: Fluxo de execução de aplicativos/bibliotecas externas no Spark Pipe

O *pipe* no Spark também pode ser utilizado na execução de *batches* de programas para que estes sejam executados em um *cluster*, ao invés de reescrever a aplicação para ser processada em vários nós. Nesse método de programação, cada nó executará a aplicação com uma parte dos dados a serem processados e retornará os resultados ao nó executor (*driver*). A característica do paralelismo do *pipe* e sua funcionalidade será explorada neste projeto de mestrado.

2.8 Fluxo de processamento Apache Hadoop e Apache Spark

Devido às constantes mudanças e evoluções no processamento de dados o Spark tem se mostrado uma ferramenta fácil de usar, com bom desempenho em execução para determinados tipos de códigos. Além disso, pode trabalhar em conjunto com o Hadoop, dentre de outras características descritas na **Seção 2.7**.

O desempenho e funcionamento do Spark é consideravelmente diferente daquele definido pelo MapReduce, mas também é dependente de restrições de paralelismo, os tipos de problemas em contexto e os recursos disponíveis (GOPALANI; ARORA, 2015). Nesta seção será descrito como é o fluxo de processamento de uma aplicação em Spark, sendo demostrado a utilização constante da memória do nó, o que a torna mais rápida para determinados tipos de algoritmos.

Na Figura 2.17 é demonstrado um diagrama de como seria o fluxo dos dados entre as funções *map* e *reduce*, o disco rígido e a memória de cada nó no Hadoop. Na Fase 1 cada função *map* emite saídas na forma de pares (chave, valor), esses pares, na próxima fase são saídas que ficam numa memória circular em *buffer*; quando essa memória circular está com sua capacidade quase completa os dados são divididos e alocados em disco (Fase 3). Na última fase da função *map* (Fase 4) os dados são combinados pela função nativa do Hadoop, a *sort*. Esses dados são então combinados em um único arquivo que serão classificados e divididos para a função *reduce*, fazendo uso de todos os *n Reducers* (PINTO, 2015).

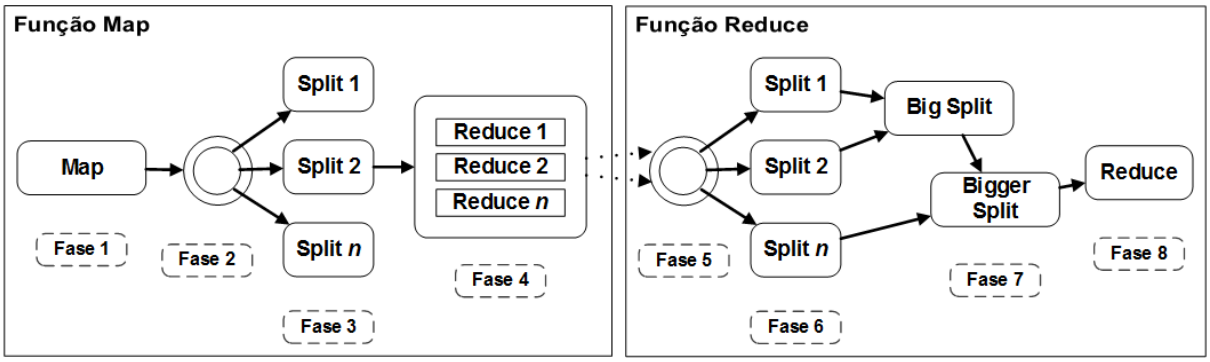


Figura 2.17: Fluxo de execução do paradigma MapReduce no Hadoop - Adaptado de (GOPALANI; ARORA, 2015)

Nesta etapa a função *reduce*(Fase 5) recebe os arquivos intermediários gerados pelos *mappers* e são alocados na memória. Similar à Fase 3 do *map*, caso os dados não consigam ser alocados na memória serão colocados em disco; novamente é invocada funções para organizar e classificar os dados e colocar em arquivos intermediários (Fase 7) e por fim os dados são enviados à função *reduce* para processar as informações.

Após analisar o fluxo de execução do MapReduce no Hadoop, cabe uma comparação com a forma de trabalho dessa mesma operação caso seja executada no Spark. A Figura 2.18 exemplifica como são as fases entre as funções *map* e *reduce* (DAVIDSON; OR, 2013).

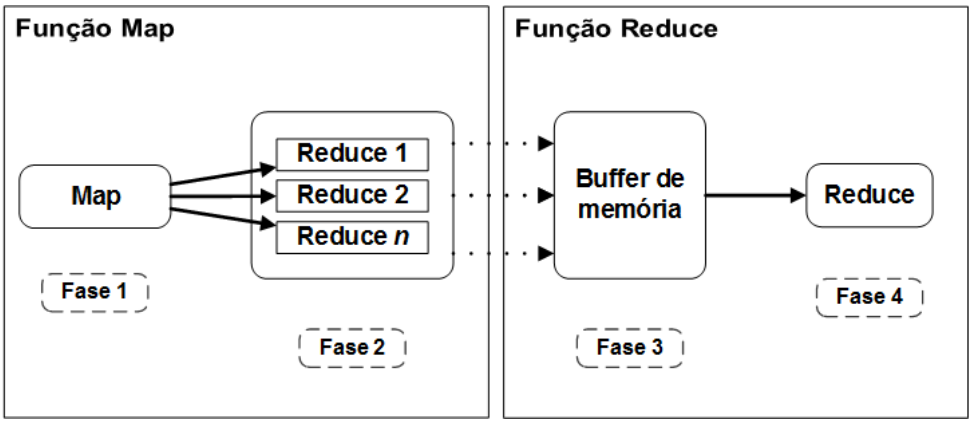


Figura 2.18: Fluxo de execução do paradigma MapReduce no Spark - Adaptado de (GOPALANI; ARORA, 2015)

Os caminhos percorridos pelos dados no Spark são mais abstratos, uma vez que faz uso de configurações já fornecidas pelo sistema operacional (SO) em que a aplicação reside (GOPALANI; ARORA, 2015). Ao observar a Figura 2.18, na Fase 1 os dados são emitidos como saídas na forma de pares. Esses dados são recebidos na Fase 2 e são armazenados na memória cache do SO; diferente da função *merge* do Hadoop, os dados não são mesclados ou particionados, com a diferença de que todas as saídas produzidas pela função *map* de cada *core* são organizadas em um mesmo arquivo intermediário; além de que cada função *map* emite tantos arquivos quanto o número de *reducers* disponíveis.

A função *reduce* é invocada com os dados sendo enviados para a memória (Fase 3) e executados por cada função em questão, que ao término escreve os resultados em arquivos (SPARK..., 2016).

A partir dessas considerações foi feita a análise entre o Hadoop Streaming e Spark Pipe. O Spark Pipe, demonstrou ser similar ou melhor nos resultados descritos na **Seção 5**.

2.9 Medidas de desempenho

Para saber se um *cluster* teve o desempenho esperado duas medidas são importantes: *speedup* e eficiência. Ele é definido como a relação entre o tempo gasto para executar uma tarefa com um único processador e o tempo gasto com *n* processadores, ou seja, *speedup* é uma medida do ganho em tempo. Assim, quanto mais a equação se aproxima de *P*, sendo *P* o número de processadores, melhor é o *speedup*. A fórmula para encontrá-lo é:

```
s = t1/tn
```

Já a eficiência diz à relação entre os resultados obtidos e os recursos empregados. Existem diversos tipos de eficiência, que se aplicam a áreas diferentes do conhecimento. Para este trabalho considera-se que a eficiência é a relação do *speedup* obtido pela quantidade de nós utilizados na execução de determinada tarefa. A fórmula empregada para encontrar a eficiência é:

```
e = s/num_nós
```

Dada as fórmulas acima e um sistema perfeito *s* seria igual a *P* (*s* = *P*) e *e* seria igual a 1 (*e* = 1).

2.10 Considerações finais

Neste capítulo foi explicado o papel da biologia molecular e da bioinformática. Posteriormente, foi apresentado o banco de dados GenBank que armazena informações provenientes de sequenciamentos de DNA.

A seguir foi exemplificado como e para que servem os alinhamentos de genes. Em seguida, foi apresentada a ferramenta BLAST que faz a comparação de genomas através do alinhamento de sequências em relação a banco de dados de DNA e proteínas.

Por fim, foram destacados alguns termos de informática e *frameworks* que fazem uso da computação distribuída e paralela para execução em tempo hábil de programas em bioinformática.

Capítulo 3

TRABALHOS CORRELATOS

3.1 Considerações iniciais

Nesta seção são apresentado alguns trabalhos correlatos que serviram de base para esse trabalho de mestrado. Os primeiros trabalhos desta seção mostram a execução da ferramenta BLAST sobre o Apache Hadoop, onde é utilizado a biblioteca de *Streaming* para melhorar a escalabilidade e desempenho da aplicação em sistemas distribuídos.

Os trabalhos subsequentes servem para mostrar a importância, funcionalidade e poder computacional do Apache Hadoop, além disso é feita um abordagem comparativa quanto à execução de tarefas e aplicações em memória do Apache Spark.

Os trabalhos estudados em relação ao Spark têm o intuito de ilustrar sua forma de execução em relação ao Hadoop e é a motivação para executar esse trabalho de mestrado. Nenhum outro trabalho que aborda a execução do BLAST sobre o Spark utilizando a biblioteca *pipe* foi publicado até o momento.

3.2 Biodoop

As atuais aplicações em bioinformática requerem o processamento de grandes quantidades de dados e alto poder computacional. Para atender a essa demanda é necessário ter maneiras simples de implementar a computação paralela e distribuída. MapReduce e Hadoop, com sua implementação de código aberto, têm se mostrado adequados para a implementação de paralelismo em aplicações de bioinformática. Desenvolvido com base no Hadoop, foi implementado um ambiente também de código aberto denominado Biodoop (LEO; SANTONI; ZANETTI, 2009).

O Biodoop é um conjunto de ferramentas para a biologia computacional destinado para

aplicações distribuídas cujas tarefas exigem poder computacional e uso intensivo de dados. Esse conjunto de ferramentas é constituído por um componente central (*core*) que inclui um conjunto de módulos de uso geral como, por exemplo, o Biodoop-Blast, além de algoritmos específicos da aplicação.

Algumas das atuais aplicações em bioinformática utilizam o alinhamento de sequências e manipulação de registros de alinhamento em bases genômicas para inferir características gênicas. O Biodoop foi implementado com base em três outras ferramentas previamente desenvolvidas em relação ao alinhamento de sequências que são: o BLAST (ALTSCHUL, 1990), o GSEA (LEO; SANTONI; ZANETTI, 2009) e o GRAMMAR (LEO; SANTONI; ZANETTI, 2009).

As principais aplicações do Biodoop são executadas sobre API Pydoop (LEO; ZANETTI, 2010) para Hadoop e foram construídas para ser escaláveis para o número de nós disponíveis e para a quantidade de dados a serem processados. Isso torna o Biodoop bem adequado para o processamento de grandes conjuntos de informações gênicas.

Tipicamente, as aplicações de bioinformática são caracterizadas por grandes quantidades de dados que não são modificados durante a execução do trabalho, o que as torna bem adequadas pelo modelo de programação MapReduce do Hadoop. Porém, ao invés de utilizar a API Java padrão do Hadoop, foi utilizada a biblioteca Hadoop Streaming, já que essa biblioteca permite ao usuário criar e executar trabalhos com bibliotecas e aplicações já prontas, por meio da invocação da tarefa *map* e *reduce*.

No Hadoop, valores de entrada de *streaming* são tratados na forma de linhas de texto e permitem a leitura de um ou mais arquivos. Isso permite que cada linha ou várias linhas de entrada (o delimitador da função *map* define onde será a divisão) podem ser enviadas a vários nós para serem processados.

Conforme visto na Seção 2.3.1, o BLAST é uma ferramenta de alinhamento de sequências genômicas. No Biodoop, foi desenvolvido um módulo na lingaugem de programação Python para BLAST envolvendo as partes relevantes do conjunto de ferramentas NCBI. A linguagem de programação Python foi utilizada para construir um script que gera os Mappers que serão executados no *streaming* do Hadoop. Neste tipo de execução só há a etapa *map*, visto que o próprio executor concatena as saídas de cada nó, exibindo o resultado final.

No desenvolvimento do Biodoop, foi implementado um ambiente de execução numa nuvem privada na qual foi utilizado testes com 5 a 69 nós, com intervalo de configuração a cada 8 nós. Cada nó possuía um total de 2 processadores *dual core* - AMD Opteron 2218 (4 *cores*), totalizando 276 cores no *cluster*. Dos 64 nós, 23 possuíam 16GB de memória RAM e os

outros nós, 8GB. Para os testes do BLAST, essas foram as configurações utilizadas, para as demais aplicações novas configurações de nós foram definidas. Como o foco deste mestrado é o BLAST as outras configurações serão suprimidas.

Para a execução de cada teste, foi utilizada uma base de dados "nt"do NCBI [1] que totalizou 7.348.665 de sequências com um total de 24GB de informações. Essa base serviu de referência para a execução do *tblastx* [2] com o *e-value* de 10^{-50} com um arquivo que continha 10 sequências.

Dos resultados descritos na Figura 3.1 foram executados 3 testes para aferir uma média na execução da aplicação e comprovar os tempos gastos.

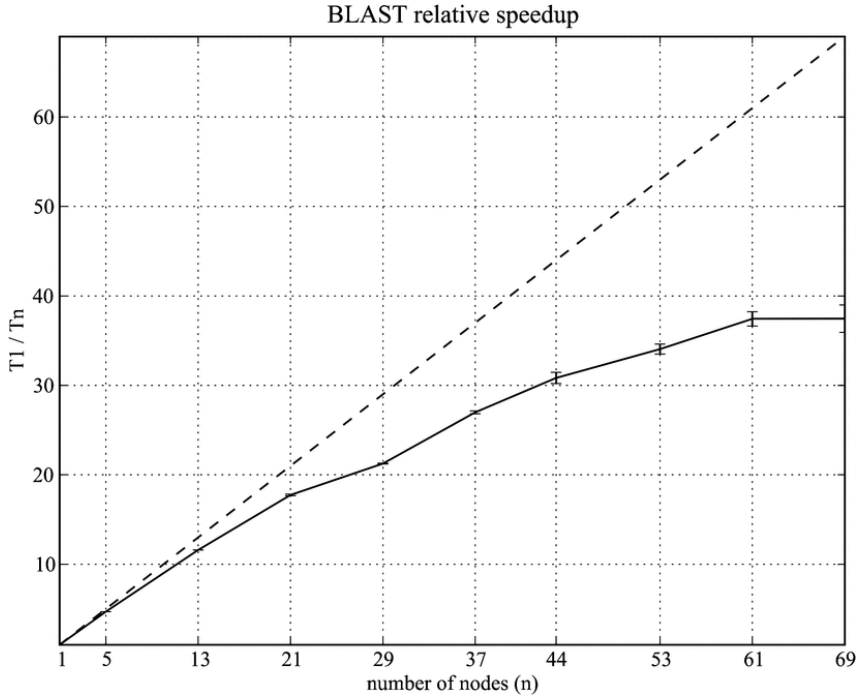


Figura 3.1: *Speedup* em relação à execução do *tblastx* na plataforma Biodoop. Linhas tracejadas representam o *speedup* teórico e linha com tracejados mais próximos definem o *speedup* obtido. - Fonte: (LEO; SANTONI; ZANETTI, 2009)

O ganho de performance definido na aplicação é demonstrado na Figura 3.1. Observa-se na figura que conforme o número de nós aumenta, o *speedup*[3] decresce.

A redução de performance quando se aumenta a quantidade de nós a serem utilizados no processamento da aplicação, mais especificamente a partir do nó 21, refere-se ao tempo gasto do Hadoop em iniciar os nós, distribuir os dados e executar o trabalho. Enfatizando o pouco uso

[1] BLAST databases. Available: ftp://ftp.ncbi.nih.gov/ blast/db/FASTA

[2] tblastx é um dos vários sub-programas que estão contidos na aplicação BLAST - https://blast.ncbi.nlm.nih.gov/Blast.cgi?PROGRAM=tblastx&PAGE_TYPE=BlastSearch&LINK_LOC=blasthome

[3] Speedup é um termo em inglês que é definido como uma métrica para a melhoria do desempenho em relação ao tempo para executar uma tarefa - https://en.wikipedia.org/wiki/Speedup

que o Hadoop faz da reutilização de dados já contidos na memória, ocasionando grande uso de leitura em disco e comunicação entre os nós.

3.3 CloudBLAST

Desenvolvido pelo grupo de Computação Avançada e Laboratório de Sistemas da Informação do Departamento de Elétrica e Engenharia da Computação da Universidade da Flórida, o CloudBLAST teve como foco demonstrar que não seria necessário reescrever algoritmos, códigos e aplicações para que eles pudessem ser escaláveis e que pudessem ser executados sobre uma infraestrutura de nuvem computacional.

Como base de estudos, o grupo de pesquisa da Universidade da Flórida trabalhou com uma ferramenta de análise genômica bastante popular que busca a similaridade entre duas ou mais sequências, o BLAST.

Para tentar contornar o problema em relação ao tempo de execução do BLAST em grandes massas de dados, foi explorado o poder computacional do Hadoop, similar ao Biodoop-Blast (LEO; SANTONI; ZANETTI, 2009). Esta abordagem visava obter ganhos de performance através da escalabilidade e paralelismo de um *cluster* com a utilização do Hadoop. A característica funcional estudada no Hadoop foi o *streaming*. Nesta abstração, os dados a serem processados devem estar contidos na memória, ou seja, a base de informações não deve ser maior que a memória disponível em cada nó e distribuídos entre os *n* nós do *cluster*.

A diferença entre o Biodoop-Blast e o CloudBLAST se dá na divisão das sequências entre os nós. No primeiro a divisão era feita estaticamente antes da execução do BLAST; no segundo, a divisão das sequências é gerenciada pelo nó mestre da aplicação Hadoop que tem o papel de coordenar a execução da aplicação.

Como o *streaming* do Hadoop é executado na memória, surgiu um fator limitante desta aplicação: o banco de dados que será confrontado pelo BLAST, deveria caber na memória. Dos métodos de particionamento das sequências e base de dados apresentados na Figura 3.2 o CloudBLAST fez uso da estratégia (a).

Assim, para cada sequência inicial (S) que foi confrontada com a base de dados (D) obteve-se um resultado (R). O que ocorre na Figura 3.2.a é que para cada sequência de entrada (S) esta é particionada em *i* partições, de acordo com o número de nós do *cluster*, sendo que cada nó processa o subconjunto de sequências enviada a ele, confrontando com a base de dados (D) e produz sub-resultados que serão concatenados no final do processo. A Figura 3.2 b ilustra

outra estratégia no qual tanto a sequência de entrada (S) quanto a base (D) são particionadas e as partições são distribuídas entre os nós. A primeira estratégia é mais simples e foi utilizada nos testes realizados em CloudBLAST e no presente trabalho.

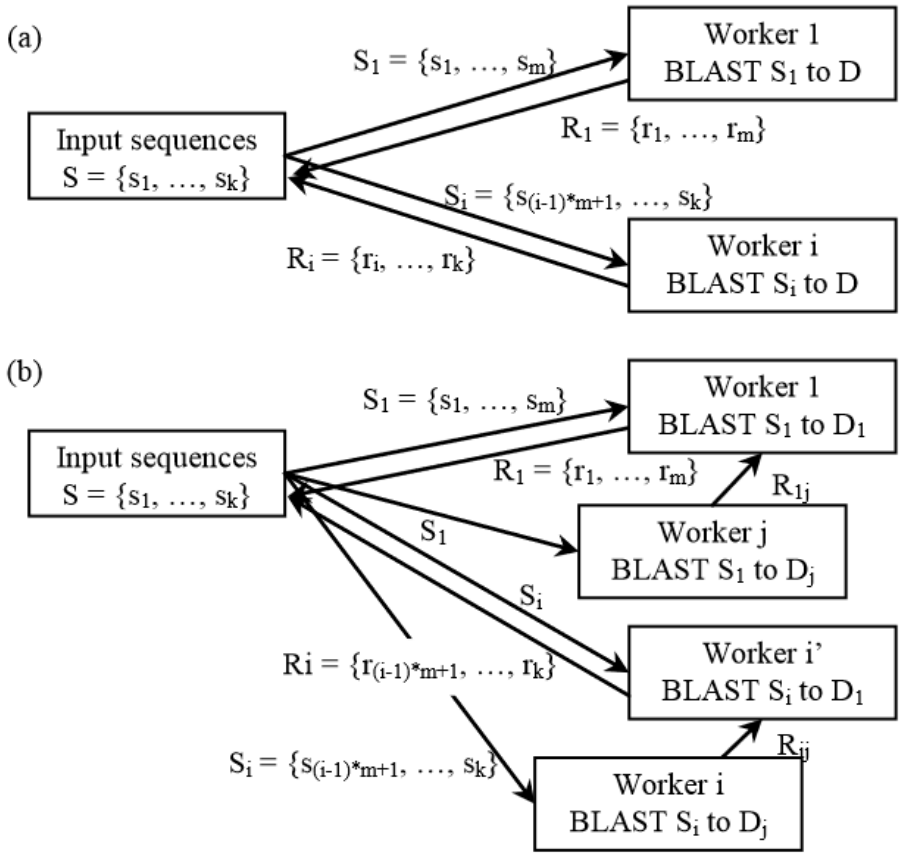


Figura 3.2: Paralelização dos dados para o BLAST: (a) Consultas divididas (S) entre os nós e banco de dados (D) contido na memória; (b) Consultas (S) e banco de dados (D) divididos entre os nós. - Fonte: (MATSUNAGA; TSUGAWA; FORTES, 2008)

Deve-se atentar que o modelo de programação MapReduce segue o método de funcionamento em que cada nó executa partes do trabalho e que o resultado de cada tarefa feito por cada um dos nós são gravados em um sistema de arquivos distribuídos como, por exemplo, o HDFS.

Para aferir o poder computacional do BLAST sendo executado em um ambiente Hadoop, os autores utilizaram uma Nuvem Privada em dois locais, na qual fez-se uso de ferramentas computacionais para que a execução fosse transparente aos usuários que executam o BLAST.

A nuvem foi criada em dois locais distintos: uma nuvem na Universidade da Flórida (24 nós com processador *dual core* e 3,5GB de memória RAM) e outra na Universidade de Chicago (12 nós com processador *dual core* e 2,5GB de memória RAM). Para cada experimento foi alocado no máximo 12 nós em uma nuvem e 12 nós em outra, somente no experimento com 64 processadores utilizou-se 12 nós em uma nuvem e 24 nós em outra.

No trabalho, os experimentos foram realizados com dois arquivos de sequências, que são as entradas que serviram de teste para a aplicação. Cada arquivo continha 960 sequências, variando em quantidade de nucleotídeos para cada sequência (sequências curtas e longas). Além de uma base de dados, que na época totalizava 3.5GB de sequências de proteínas ("nr"[4]), a serem analisadas.

A execução da ferramenta proposta pelo CloudBLAST teve como meio de comparação alguns dados e configurações em relação ao equipamento alocado e à base de dados utilizada, além de outra ferramenta:

- Execução local, MPIBlast(DARLING; CAREY; FENG, 2003) e com o CloudBLAST (Hadoop);
- Em uma única nuvem e em duas nuvens em conjunto;
- Máquinas físicas alocadas e máquinas virtuais;
- Base de dados única e particionada em trinta partes.

O trabalho demonstrou que tanto para máquinas virtuais quanto físicas e a comunicação em uma ou duas nuvens computacionais a performance foi a mesma, uma vez que toda a interconexão dos nós foi feita como se estivesse em uma rede local. Em relação ao particionamento da base em trinta fragmentos, também não houve grande variação no tempo de execução.

Na comparação entre a execução local, MPIBlast e CloudBLAST, o último se mostrou relativamente mais rápido, chegando a ter um *speedup* de 57 vezes contra um de 52,4 vezes do MPIBlast.

Em todas as configurações, a execução foi com os dados acima descritos e foi mostrado o quanto a integração entre Hadoop e BLAST foi promissora. Nesse sentido, demonstrou-se que não foi preciso reescrever códigos para obter um ganho de desempenho relativamente adequado na execução de sequências de nucleotídeos entre vários nós.

A pesquisa relatada no CloudBLAST teve como princípio a abordagem eficiente para a execução de aplicações em bioinformática sem a reescrita de algoritmo. A base para validar a demonstração de alto desempenho se deu através do desenvolvimento de uma ferramenta para a execução do BLAST sobre o Hadoop Streaming em duas nuvens privadas.

[4]Não redundante proteínas - http://www.ebi.ac.uk/trembl/

3.4 SparkSeq

Com o desenvolvimento do framework Spark (ZAHARIA, 2015), conforme visto na **Seção 2.7**, algumas ferramentas começaram a ser adaptadas e/ou desenvolvidas para utilizarem seu método de programação paralela.

Dentro do conceito de NGS[5], há a necessidade de alinhar e produzir sequências através dos pares de bases gerados pelos sequenciadores genômicos. O Sparkseq foi desenvolvido dentro do contexto de alinhamento de arquivos do tipo BAM [6] em genômica para estudar o Apache Spark e explorar as suas funcionalidades.

O Sparkseq é uma ferramenta para analisar sequências de RNA e DNA com precisão de alinhamento de nucleotídeos. Isso desenvolvido na computação em nuvens para permitir um fácil escalonamento e tolerância a falhas (WIEWIORKA, 2014). O desenvolvimento do SparkSeq utilizou a combinação de Pircard SAM JDK com Hadoop-BAM (NIEMENMAA, 2012) para mostrar a operacionalidade do modelo MapReduce sobre a análises de sequenciamento (WIEWIORKA, 2014).

A abordagem do algoritmo opera através de alinhamentos contendo vários arquivos ao mesmo tempo. A operação feita pela ferramenta se divide em três categorias: filtragem de pares, sumarização das características genômicas e análises estatísticas. Baseado no poder de escalabilidade do Spark (ZAHARIA, 2015), o número de arquivos para análise poderia crescer e bastaria que novos nós fossem adicionados ao *cluster* para que o tempo de execução permanecesse mensurável.

O trabalho SparkSeq (Spark) teve como premissa de comparação o SeqPig (Hadoop). Aqui é citado estes trabalhos porque o foco deste projeto de mestrado é comparar a viabilidade de utilizar o Spark em determinados algoritmos de bioinformática, que já tiveram trabalhos similares no Hadoop.

O SparkSeq foi comparado com o SeqPig (SCHUMACHER, 2013) conforme visto na Figura 3.7 com o propósito de demonstrar a velocidade e a escalabilidade entre as ferramentas citadas. Observa-se que SparkSeq supera SeqPig em termos de velocidade (8.4-9.15 vezes) e que encontrar o alinhamento completo para os nucleotídeos mensurável.

Os testes realizados em SparkSeq puderam comprovar a usabilidade baseada em Apache Spark/Hadoop-BAM para análise de arquivos de alinhamento de sequenciamento de alto desempenho. A eficiência é comparável com normalização (Samtools) em uma máquina local e

[5]NGS - Next-Generation Sequencing - nova geração de sequenciadores
[6]BAM - binary alignment/map

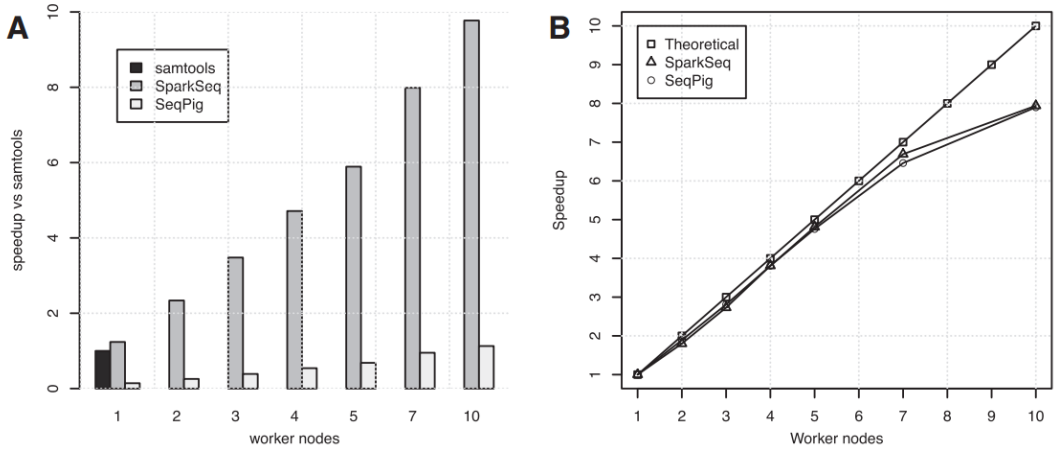


Figura 3.3: Comparação de velocidade (A) e escalabilidade (B) SparkSeq e SeqPig - Fonte: (WIEWIORKA, 2014)

oferece o valor acrescentado de escalabilidade horizontal. Implementações similares podem ser feitas para outros tipos de análises e outros tipos de arquivos de alinhamento genômico.

Isso mostra que, para os investigadores em biologia, o benefício será a aceleração de análises longas e melhorando a precisão de nucleotídeos de resultados em amostras múltiplas em tempo hábil. Também permite buscas para novos fenômenos em genômica com a análise de muitas amostras em paralelo, bem como otimizar análises padrão, executando-os muitas vezes, ajustando seus parâmetros de uma forma interativa (WIEWIORKA, 2014).

3.5 Resumo dos trabalhos considerados

Na tabela 3.1 são elencados os trabalhos que foram utilizados na composição desta seção e alguns outros que serviram como base e motivação para a escrita deste trabalho de mestrado, abordada na **Seção 4**.

Tabela 3.1: Trabalhos correlatos e motivacionais

Ferramenta	**Descrição**	**Referência**
Bioinformática e nuvens computacionais	Os desafios da Bioinformática ao utilizar nuvens computacionais	(DAI, 2012)
CloudAligner	Aplicação baseada no MapReduce para mapear leituras curtas geradas pelos novos sequenciadores (NGS).	(NGUYEN; SHI; RUDEN, 2011)
CloudBLAST	BLAST escalável em nuvens computacionais e executada sobre a plataforma Hadoop.	(MATSUNAGA; TSUGAWA; FORTES, 2008)
HadoopBAM	Biblioteca para manipulação escalável para alinhamento de dados para NGS.	(NIEMENMAA, 2012)
GPU-BLAST	Versão mais rápida do BLAST usando GPU.	(VOUZIS; SAHINIDIS, 2011)
MPIBlast	Utilização da biblioteca MPI para execução paralela do BLAST	(DARLING; CAREY; FENG, 2003)
Biodoop - BLAST	Utilização de nuvens computacionais para BLAST escalável para alinhamento de sequências	(LEO; SANTONI; ZANETTI, 2009)
HBlast	Particionamento de sequências de entrada e base de dados para processamento do BLAST escalável	(O'DRISCOLL, 2015)
Spark	Utilização do Apache Spark para processamento de dados.	(ZAHARIA, 2010)
SparkSeq	Aplicação baseada no Apache Spark para manipulação de alinhamento de sequências em NGS.	(WIEWIORKA, 2014)
ADAM biblioteca	Uma biblioteca em Apache Spark para análises de sequências em vários formatos de arquivos.	(MASSIE, 2013)
Galaxy Cloud	Cloud-scale Galaxy para larga escala de análise de dados.	(AFGAN, 2010)

Capítulo 4

UTILIZAÇÃO DO SPARK PARA ALINHAMENTO DE SEQUÊNCIAS USANDO O BLAST

4.1 Considerações iniciais

Uma das características da ferramenta BLAST é encontrar alinhamentos entre uma ou várias sequências de DNA ou proteínas em um banco de dados com informações gênicas. O alinhamento das sequências tem como objetivo buscar a similaridade entre as sequências estudadas. A essa característica encontrada entre os filamentos dá-se o nome de homologia que é uma das formas de se deduzir o grau de parentesco entre as espécies.

Com o constante crescimento das sequências e bases de genes, tem sido necessário adaptar ou desenvolver novas aplicações com o BLAST que tenham melhor desempenho computacional. Vale notar que o BLAST possui uma implementação originalmente sequencial, com desempenho limitado ao processamento do equipamento em que ele está instalado.

A computação paralela e distribuída visa reduzir o tempo de execução de aplicações que gastariam tempo excessivo para a análise de determinados dados. Muitas ferramentas e bibliotecas são disponibilizadas com a finalidade de acelerar o processamento das aplicações, como, por exemplo, o MPI (Message Passing Interface - Passagem de mensagem por interface), GRID (Grid Computing - Computação em Grades) e Nuvens Computacionais. Contudo, a paralelização de aplicações requer que os algoritmos e aplicações já existentes sejam reescritos, exigindo grande esforço para reprogramá-los e testá-los para que produzam resultados corretos, igual às aplicações originais.

Uma ferramenta da computação paralela, distribuída e escalável que tem ajudado na adaptação das aplicações para serem executadas de forma distribuída e escalável é o Hadoop Strea-

ming. Para executar algum serviço do BLAST, como o *blastp*, não necessariamente é preciso reescrever o algoritmo. Com o Hadoop Streaming é possível reutilizar determinados tipos de componentes e aplicações pré-existentes que são invocados pelas tarefas *map* ou *reduce*. Alguns tipos de aplicações que podem ser utilizadas pelo Streaming do Hadoop são aquelas em que a instância a ser executada recebe parte do trabalho e as saídas não influenciam uma na outra.

Por exemplo, para o caso do BLAST, cada tarefa *map* pode invocar uma instância do BLAST para atuar sobre um subconjunto específico de dados. Dessa forma, o *framework* Hadoop é utilizado para gerenciar a execução global da aplicação, disparando as várias pequenas tarefas paralelas, gerenciando os dados e a sincronização entre essas tarefas.

Da mesma forma que o Hadoop Streaming, o Spark Pipe permite que alguns tipos de aplicações externas sejam invocadas e executadas. Deste modo que o Spark Pipe atua como um *driver* ou principal orquestrador que controla a execução. O que acontece é a invocação de bibliotecas e aplicações externas ao Spark que serão executadas por cada nó. Nesse caso, cada nó executará uma instância do BLAST com subconjuntos e dados locais.

Ao estudar o método execução do Hadoo Streaming e Spark Pipe para processamento do BLAST notou-se que não havia trabalhos que utilizassem o Pipe do Spark para a execução distribuída da ferramenta BLAST. Portanto, neste trabalho será considerado o Apache Spark Pipe que terá como base de comparação o Hadoop Streaming para medir a escalabilidade, eficiência e *speedup*. Ambos os *frameworks* são utilizados para execução distribuída sobre uma nuvem privada como, por exemplo, a Nuvem Google (Google Cloud Platform).

No decorrer deste capítulo é descrito a aplicação desenvolvida neste trabalho e é mostrado os passos a serem realizados para atingir o objetivo final que é a utilização de uma nova ferramenta para "paralelizar"a utilização do BLAST, o SparkBLAST.

4.2 Definição do Trabalho

O objetivo deste trabalho consiste no desenvolvimento de uma implementação paralela, escalável e distribuída para a execução da ferramenta BLAST utilizando a característica de *pipes* no Apache Spark.

A maioria dos trabalhos estudados em relação à paralelização do BLAST consiste na adaptação de algoritmos baseada no modelo Apache Hadoop utilizando a biblioteca para *Streaming* ou reescrita através das bibliotecas MPI. Nota-se que os autores do CloudBLAST abordam técnicas

distintas de particionamento nas etapas de divisão de tarefas e dados (**Seção 3.3, Figura 3.2**). Neste trabalho, é adotado a estratégia de particionamento segundo a qual o banco de dados, contendo sequências de bactérias (D) está replicado em cada nó e apenas as consultas (S) são mapeadas (subdivididas) nos nós para execução, como pode ser observado na Figura 4.1.

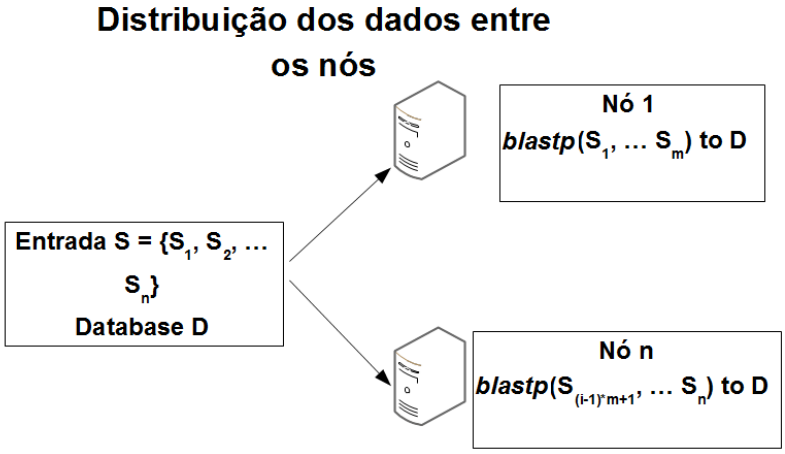


Figura 4.1: Distribuição das sequências gênicas entre os nós

A Figura 4.1 apresenta como é feita a distribuição dos trabalhos e informações em cada nó. Cada um dos *n* nós terá o aplicativo BLAST instalado e receberá uma parte das sequências (S) a serem processadas e o banco de dados (D) que deverá estar replicado nos *n* nós e, consequentemente, na memória de cada um deles. Assim, as sequências de entradas serão particionadas em *i* subconjuntos, os quais serão processados por um *worker* (nó) e os resultados serão organizados no final por ordem de execução.

Nos testes a serem executados considera-se como entrada um arquivo *fasta* ou *multifasta* de sequências S, essas serão particionadas entre os *n* nós. O método de particionamento das sequências é definido na Figura 4.2 e será discutido na seção **Pré-Processamento**.

A execução da aplicação pode ser gerenciada pelo *driver*[1] do Spark que, no caso, é chamado de nó mestre, conforme Figura 4.3 [2] ou pelo YARN. O *driver*, neste tipo de execução, é executado pelo nó mestre que tem a função de receber os comandos do usuário executados em uma máquina cliente e coordenar a execução da aplicação através do código fonte escrito na linguagem de programação Scala[3] e disponibilizado como código aberto[4].

[1]Driver - divide os trabalhos entre os Workers (nós) e recebe os pré-resultados que serão gravados no GCS (Google Cloud Storage)

[2]Apache Spark - https://spark.apache.org/docs/1.4.0/cluster-overview.html

[3]The Scala Programming Language - http://www.scala-lang.org/

[4]SparkBLAST - https://github.com/Ufscar-Fiocruz-Ifsul/spark-blast2.0

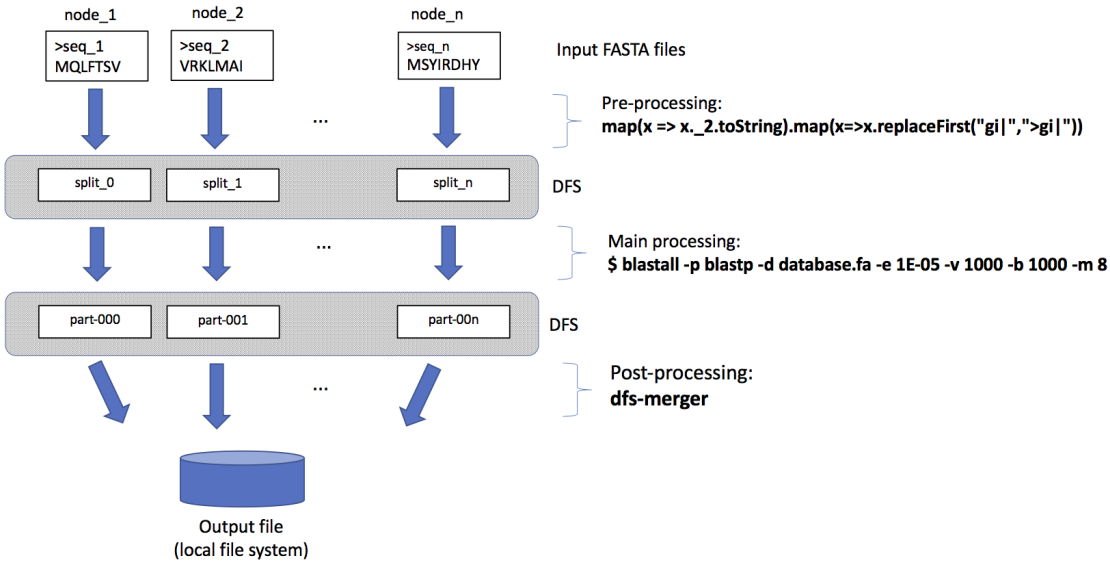


Figura 4.2: Divisão do arquivo FASTA entre os *n* nós.

A entrada consiste em um arquivo contendo sequências de bactérias colhidas em uma pesquisa de doutorado conduzida pela FIOCRUZ[5], gravada no formato *fasta*. A base, que será confrontada com essas sequências, será o próprio arquivo contendo as mesmas sequências e compilado para ter o formato de arquivo utilizado no BLAST (utiliza-se o *makeblastdb*).

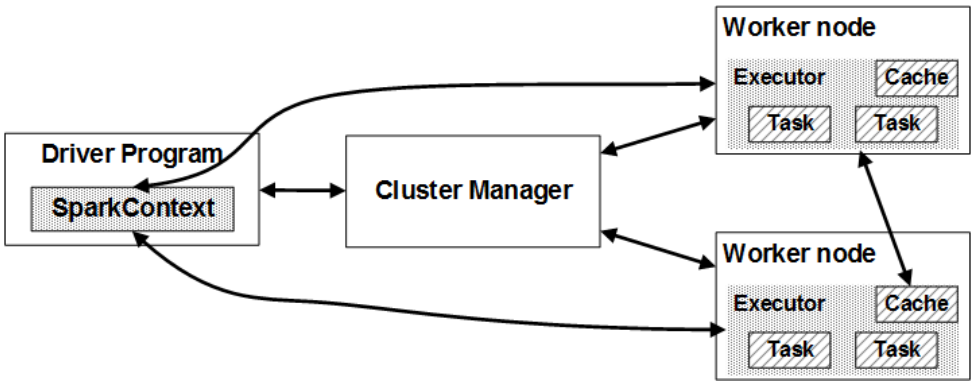


Figura 4.3: Funcionamento do cluster com o Spark: o nó mestre coordena o fluxo das informações entre nós escravos, o YARN pode assumir esse papel. - Adaptado de: (APACHE..., 2015)

O processamento é o próprio BLAST sendo executado em cada nó através da invocação por tarefas do *pipe*. No pós-processamento (**Seção 5**) as saídas de cada nó serão organizadas para

[5]FIOCRUZ - Fundação Oswaldo Cruz http://portal.fiocruz.br/pt-br

se mostrar o resultado da execução e os tempos gastos no processamento total da aplicação. A medição do tempo de processamento será utilizada para comparar com CloudBLAST, trabalho desenvolvido em Hadoop. Nas próximas seções, serão apresentadas em maior nível de detalhes as etapas dos processos propostos.

Assim, a utilização da ferramenta BLAST com o Spark em uma nuvem computacional (Nuvem Google), tem como entrada sequências genômicas (no caso um arquivo com genes de bactérias). O processamento (execução do BLAST sobre a ferramenta Apache Spark (*pipe*)) e saída dos dados armazenados em um DFS (Distributed File System - Sistemas de arquivos distribuídos).

4.2.1 Ambiente de execução

Para demonstrar a viabilidade da ferramenta desenvolvida foi utilizada duas nuves privadas, a Cloud Google[6] e Microsoft Azure[7]. Foi utilizado um total de 65 nós alocados na primeira e 66 na segunda, respeitando as configurações pré-definidas em cada nuvem. Sendo, em ambos os casos, um nó usado para disparar os trabalhos, fazendo o papel do cliente e os outros nós utilizados para o processamento, sendo um desses, escolhidos pelo gerenciador do *cluster* para atuar como mestre e os outros atuando como nós trabalhadores.

Para se entender melhor o funcionamento da arquitetura do Apache Spark na Nuvem Google, esta é ilustrada na Figura 4.4 a mesma abstração pode ser exemplicada para a Azure, mas lembrando que esta utiliza o esquema de *blobs*[8] no sistema de arquivos distribuídos.

Os nós na Google Cloud foram alocados em 4 locais e 13 zonas diferentes. Foram alocados nós na Ásia (Asia East [3 zonas], Europa (Europe West[3 zonas]), Centro dos Estados Unidos (Us Central[4 zonas]) e Leste dos Estados Unidos (Us East[3 zonas]), essa configuração foi utilizada visto os limites da conta cedida pela Google e para mostrar a escalabilidade da alocação de nós na nuvem em locais distintos. Na Azure todos os núcleos foram provisionados em uma mesma zona (Centro Norte dos EUA).

Para os experimentos, foi configurado o Spark 1.6.1 para executar o SparkBLAST em ambos os ambientes de nuvem. Para executar o CloudBLAST, foi utilizado o Hadoop 2.4.1 na Google Cloud e o Hadoop 2.5.2 na Microsoft Azure. Em ambos os casos, foi configurado e utilizado o YARN como o planejador de recursos, uma vez que os experimentos se concentram no desempenho.

[6]Google Cloud Computing - https://cloud.google.com/
[7]Microsoft Azure - https://azure.microsoft.com/pt-br/
[8]Blobs - https://azure.microsoft.com/pt-br/services/storage/blobs/

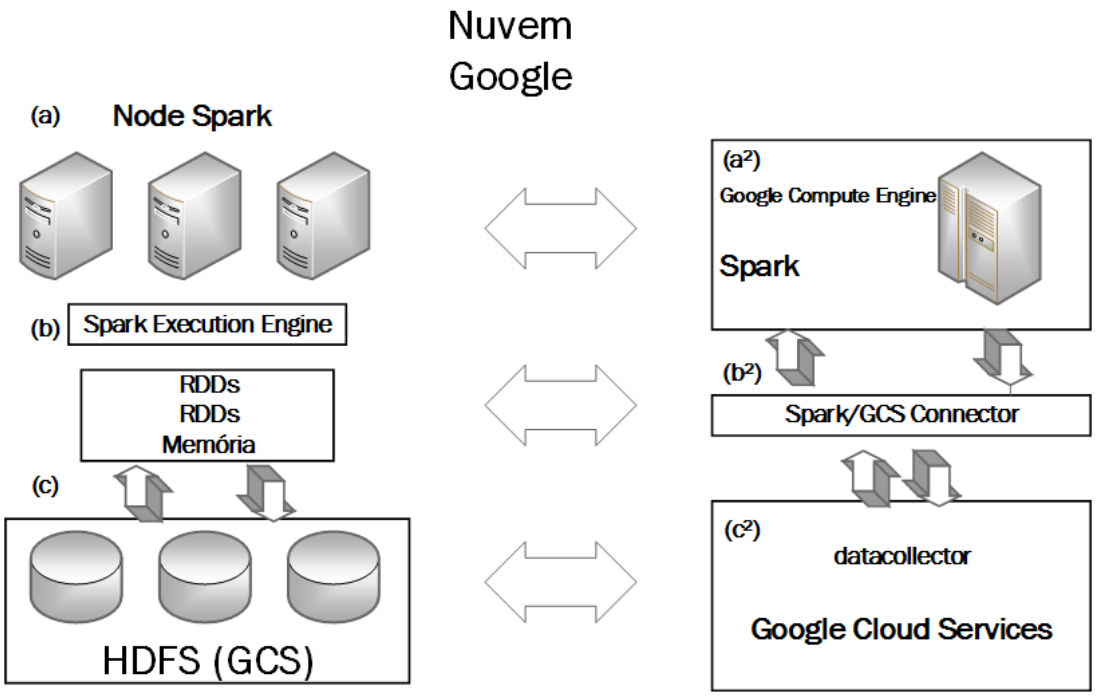


Figura 4.4: Arquitetura Spark na Nuvem Google: o conjunto de nós Spark são partes da infraestrutura utilizada na Nuvem Google. Cada nó possui o conjunto de hardware necessário para funcionamento (CPU, memória, disco rígido, etc.), assim, os RDDs são manipulados em cada nó e o conjunto de informações geradas são salvas no sistema de arquivos distribuídos da Google (GCS - Google Cloud Storage). A comunicação entre os nós e o GCS é feita através de um *plugin* nativo da nuvem, o GCS Connector. Esse conjunto se comunica baseado na estrutura de serviços da Google Cloud Services.

Os testes executados e os tempos medidos na Google tiveram a utilização de: 1, 2, 4, 8, 16, 32 e 64 nós, tanto no CloudBLAST, quanto no SparkBLAST. Cada nó teve como configuração de hardware o modelo cedido pela Google e seguiu a seguinte definição: nó n1-standard-2 com 2 vCPUs, 7,5 GB de memória e a Plataforma de CPU Intel Ivy Bridge[9].

Para a execução e medidas de desempenho da Azure foram utilizados 1, 3, 7, 15, 31 e 63 nós. Sendo as máquinas dos nós trabalhadores do tipo A2 e as do mestre do tipo A4[10].

4.2.2 Dados a serem processados

A fim de aferir a funcionalidade da ferramenta tomou-se como informações dados gerados por um projeto da FIOCRUZ[11]. Este trabalho foi originalmente inspirado e aplicado em um estudo de resistência a radionuclídeos. Podem ser utilizadas sequências genômicas de vários micro-organismos resistentes à radiação para a genômica comparativa para inferir

[9]Tipo máquinas Google Cloud - https://cloud.google.com/compute/docs/machine-types
[10]Tipo máquinas Microsoft Azure - https://azure.microsoft.com/pt-br/pricing/details/virtual-machines/linux/
[11]Portal da FIOCRUZ - http://portal.fiocruz.br/pt-br

as semelhanças e diferenças entre essas espécies. A inferência de homologia é importante para identificar genes compartilhados por diferentes espécies e, como consequência, genes específicos de espécies podem ser inferidos. Dois experimentos são considerados neste trabalho. Os dados de entrada para o experimento 1 (Google Cloud) foram compostos por 11 sequências de proteínas do genoma bacteriano, 10 destas são resistentes à radiação (*Kineococcus*- NC_011883.1, *Desulfovibrio vulgaris* - NC_002937.3, *Rhodobacter sphaeroides* - NC_009429.1, *Desulfovibrio desulfuricans Escherichia coli* - NC_000913.3, *Deinococcus radiodurans* - NC_001263.1, *Desulfovibrio fructosivorans* - NZ_AECZ01000069.1, *Shewanella oneidensis* - NC_004349.1, *Geobacter* - NC_002939.5, *Deinococcs geothermalis* - NC_008010.2, *Geobacter metallireducens* - NC_007517.1) para o processamento de *Reciprocal-Best-Hit* (RBH)(LI; STOECKERT; ROOS, 2003);(BROWN; JURISICA, 2005).

Para o experimento 2 (Microsoft Azure), a consulta de entrada foi composta por 10 bactérias resistentes à radiação. (Isto é, todas as espécies listadas acima, menos a *E. coli*). Este experimento baseado em similaridade consistiu na pesquisa de potenciais homólogos proteicos de 10 genomas resistentes à radiação em 2 conjuntos de dados de metagenómica marinha.

Cada conjunto de dados de entrada foi concatenado em um único arquivo de entrada multifasta chamado query1.fa (Experimento 1) e query2.fa (Experimento 2). Os arquivos query1.fa e query2.fa tinham 91.108 e 86.968 sequências e um tamanho total de 36.7MB e 35MB, respectivamente. Foram utilizados dois conjuntos de dados metagenómicos alvo obtidos a partir da base de dados MG-RAST [12] no Experimento 2: (i) Sargaso Sea (Bermuda), coordenadas: 32.17, -64.5, 11GB, 61255, 260 proteínas (Ber.fasta) e (ii) João Fernandinho (Búzios, Brasil), coordena: -22,738705, -41874604, 805MB, 4795,626 proteínas (Buz.fasta).

As bases de dados que foram confrontadas com essas sequências foram compiladas com as seguintes premissas (comando disponível com a instalação do BLASTALL):

```
makeblastdb -dbtype prot -in database.fa -parse_seqids
```

Os testes executados no CloudBLAST e no SparkBLAST invocaram o *blastp* e tiveram como parâmetro os seguintes comandos (i - Experimento 1; ii - Experimento 2). Observe que o parâmetro *-i*, que é a entrada a ser processada, foi omitido, pois esse valor é passado pela função *map* no SparkBLAST:

```
(i)  /local/bin/blastall -p blastp -d /local/database.fa -e 1E-05
-v 1000 -b 1000 -m 8
```

[12] http://metagenomics.anl.gov/

```
(ii)  /local/bin/blastall -p blastp -d /local/database.fa -e 1E-05
-v 100 -b 100 -m 8
```

Os parâmetros da linha de execução do BLASTP tem as seguintes definições:

- -p: programa do BLAST a ser executado (blastp);
- -i: arquivo de entrada;
- -d: banco de dados a ser comparado;
- -e: valor esperado;
- -v: número de sequências do banco de dados para mostrar nas descrições de uma linha;
- -b: número de sequências do banco de dados para mostrar no alinhamento;
- -m: tipo de saída (8 é tabular).

4.2.3 Pré-processamento

Nesta primeira etapa da aplicação acontece a preparação dos dados, ou seja, as consultas contidas no arquivo *multifasta* serão quebradas e divididas entre os *n* nós. Este processo requer atenção pois cada consulta (linha contendo as informações de cada uma das *s* sequências) pode ter um tamanho diferente e, caso ocorra apenas a divisão simplista do arquivo (quebra de linha), pode-se ocasionar erros conforme pode ser observado na Figura 4.2. Esses erros acontecem devido a fase *map* utilizar como padrão a divisão do arquivo pelo tamanho de *bytes* entre os nós e a quebra de linha.

A representação da Figura 4.2 diz respeito à forma como as sequências serão distribuídas entre os nós. Poderia ocorrer erros quanto ao resultado em relação ao conteúdo de cada sequência (Figura 4.2 - primeiro retângulo). Para evitar esses possíveis problemas, a divisão das sequências entre os nós foi feita tomando como base o caractere $>$[13], que é o delimitador de cada sequência. Isso permite que cada nova sub-consulta tenha sua estrutura correta ao serem enviadas a cada BLAST, instalado em cada um dos nós do *cluster*.

Os dados de entrada utilizados neste trabalho são os informados na **Seção 4.2.2**, que são um arquivo com informações de bactérias, que serão confrontadas com uma base de dados e será dividido entre os *n* nós.

[13] $>$ - Símbolo que precede cada consulta no arquivo fasta ou multifasta

Ressalta-se que a divisão não é estática, ou seja, o arquivo inicial não foi quebrado em *n* arquivos menores, mas é enviada uma consulta a cada um dos *n* nós e após o término de uma consulta é enviada uma nova consulta de acordo com o término do processamento em cada nó e assim sucessivamente até o final da leitura do arquivo de entrada.

O papel do *driver* nesta parte do trabalho é enviar as sub-consultas aos nós. Cabe ressaltar que o tamanho do arquivo de entrada não impacta de forma negativa na execução. Visto que esse arquivo estará na máquina cliente e o gerenciador da aplicação irá enviar as sub-consultas aos nós. Assim, reduzindo o ônus quanto à comunicação entre os nós e a não necessidade de distribuir a entrada completa a todos. As linhas de código abaixo foram utilizadas para definir essa implementação, essa linha define qual será o delimitador entre as linhas do arquivo de entrada:

```
conf.set("textinputformat.record.delimiter", ">")
```

A função *map* abaixo divide o arquivo no local especificado pelo delimitador e configura a nova sequência a ser enviada a cada nó. Isso deve ser feito visto que a divisão cria uma nova *string* que apaga o delimitador $>$ e, caso, essa fosse enviada para ser executada no BLAST ocorreria um erro nos resultados produzidos.

```
map(x =$>$ x._2.toString).map(x=>x.replace("gi|",">gi|"))
```

Após a leitura inicial do arquivo de entrada começa-se o processamento.

4.2.4 Processamento

Após o pré-processamento, tem-se como resultado várias entradas que serão usadas nos nós, para serem confrontadas com o banco de dados no processamento do BLAST. Esta parte da execução da aplicação é a que consome mais tempo no processamento. Cada sub-entrada e banco serão processados em cada nó. A execução do algoritmo BLAST não precisa ser investigada a fundo, visto que o foco deste projeto é a execução de aplicações no *pipe* do Apache Spark sem ser necessário reescrever a aplicação.

Nesta fase o que acontece é a execução do BLAST em cada nó, gerando as saídas locais que serão enviadas ao sistema de arquivos distribuídos (GCS), coordenados pelo (*driver*) da aplicação. A execução do BLAST teve os seguintes parâmetros, que foram passados através de argumentos para a aplicação CloudBLAST e SparkBLAST:

```
/local/bin/blastall -p blastp -d /local/database.fa -e 1E-05 -v 1000
-b 1000 -m 8
```

ou

```
/local/bin/blastall -p blastp -d /local/database.fa -e 1E-05 -v 100
-b 100 -m 8
```

A invocação de aplicações externas ao Spark é feita pelo método *pipe* e passada como parâmetro à aplicação ou biblioteca a ser executada. A paralelização se dá nesse momento, pois o *pipe* irá invocar cada um dos *n* nós para executar o BLAST.

O código abaixo é o que define o processo de execução do BLAST com os parâmetros em cada nó do *cluster*:

```
pipe(script).saveAsTextFile("saída")
```

onde, *script* é o parâmetro a ser executado, no caso a linha de comando do BLAST e *saveAsTextFile* o método nativo do Spark que gravará os resultados no DFS, que no caso deste projeto é o GCS ou *blobs*.

Portanto, cada nó funcionará como um *Mapper*, produzindo saídas similares como ao mostrado na Figura 4.5[14] e as saídas geradas são armazenadas no sistema de arquivos distribuídos e poderão ser manipuladas pelo cliente no final do processamento.

```
sshuser@hn0-cluste:~$ /usr/bin/blastall -p blastp -i query.fa -d database.fa.ORIGINAL -e 1E-05 -v 1000 -b 1000 -m 8

gi|499685313|ref|WP_011366047.1|  gi|499685313|ref|WP_011366047.1|  100.00  220  0  0  1  220  1  220  6e-147  413
gi|499685313|ref|WP_011366047.1|  gi|39996278|ref|NP_952229.1|  85.45  220  32  0  1  220  1  220  2e-126  361
gi|499685313|ref|WP_011366047.1|  gi|499244296|ref|WP_010941836.1|  85.45  220  32  0  1  220  1  220  2e-126  361
gi|499685313|ref|WP_011366047.1|  gi|504364443|ref|WP_014551545.1|  85.00  220  33  0  1  220  1  220  2e-126  361
gi|490649799|ref|WP_004514794.1|  gi|490649799|ref|WP_004514794.1|  100.00  225  0  0  1  225  1  225  2e-167  465
gi|490649799|ref|WP_004514794.1|  gi|499685327|ref|WP_011366061.1|  100.00  225  0  0  1  225  1  225  3e-167  466
gi|490649799|ref|WP_004514794.1|  gi|39996675|ref|NP_952626.1|  40.71  226  132  2  1  225  1  225  2e-50  168
gi|490649799|ref|WP_004514794.1|  gi|499244680|ref|WP_010942220.1|  40.71  226  132  2  1  225  1  225  2e-50  168
gi|490648693|ref|WP_004513688.1|  gi|490648693|ref|WP_004513688.1|  100.00  638  0  0  1  638  1  638  0.0  1219
gi|490648693|ref|WP_004513688.1|  gi|39995144|ref|NP_951095.1|  92.95  638  43  2  1  638  1  636  0.0  1143
gi|490648693|ref|WP_004513688.1|  gi|499243171|ref|WP_010940711.1|  92.95  638  43  2  1  638  1  636  0.0  1143
gi|490648693|ref|WP_004513688.1|  gi|46579224|ref|YP_010032.1|  69.97  636  188  3  1  634  1  635  0.0  856
gi|490648693|ref|WP_004513688.1|  gi|499240572|ref|WP_010938112.1|  69.97  636  188  3  1  634  1  635  0.0  856
gi|490648693|ref|WP_004513688.1|  gi|492839889|ref|WP_005993843.1|  68.59  640  193  2  1  634  1  638  0.0  845
gi|490648693|ref|WP_004513688.1|  gi|506426738|ref|WP_015946457.1|  69.48  639  189  3  1  634  1  638  0.0  840
gi|490648693|ref|WP_004513688.1|  gi|504089973|ref|WP_014323967.1|  71.33  600  171  1  1  600  1  599  0.0  840
gi|490648693|ref|WP_004513688.1|  gi|506419437|ref|WP_015939156.1|  68.11  646  185  3  1  634  1  637  0.0  833
gi|490648693|ref|WP_004513688.1|  gi|550904542|ref|WP_022657731.1|  69.28  638  191  2  1  634  1  637  0.0  830
gi|490648693|ref|WP_004513688.1|  gi|518843853|ref|WP_019999743.1|  68.89  646  187  5  1  634  1  644  0.0  827
gi|490648693|ref|WP_004513688.1|  gi|753983809|ref|WP_041669812.1|  67.03  640  204  4  1  638  1  635  0.0  819
gi|490648693|ref|WP_004513688.1|  gi|500167484|ref|WP_011841910.1|  67.03  640  204  4  1  638  1  635  0.0  818
gi|490648693|ref|WP_004513688.1|  gi|77464754|ref|YP_354258.1|  66.88  640  205  4  1  638  1  635  0.0  817
gi|490648693|ref|WP_004513688.1|  gi|500249372|ref|WP_011909714.1|  66.88  640  205  4  1  638  1  635  0.0  815
gi|490648693|ref|WP_004513688.1|  gi|24372709|ref|NP_716751.1|  66.10  643  209  4  1  638  1  639  0.0  813
gi|490648693|ref|WP_004513688.1|  gi|499383800|ref|WP_011071367.1|  66.10  643  209  4  1  638  1  639  0.0  813
gi|490648693|ref|WP_004513688.1|  gi|16128008|ref|NP_414555.1|  65.89  642  210  4  1  637  1  638  0.0  794
gi|490648693|ref|WP_004513688.1|  gi|501033966|ref|WP_012086009.1|  58.02  605  223  5  1  598  1  581  0.0  654
gi|490648693|ref|WP_004513688.1|  gi|15805168|ref|NP_293855.1|  57.86  541  221  4  1  537  1  538  0.0  620
gi|490648693|ref|WP_004513688.1|  gi|499189237|ref|WP_010886777.1|  57.86  541  221  4  1  537  1  538  0.0  620
```

Figura 4.5: Saída BLAST, programa executado em apenas um nó.

[14]How to use BLAST - http://www.ncbi.nlm.nih.gov/books/NBK52640/

Após o término da execução do trabalho iniciado pelo *driver* tem-se as várias saídas gravadas no GCS. Para concatenar todas as partes em apenas um arquivo, em ambos os trabalhos, CloudBLAST e SparkBLAST usa-se o comando *getmerge*, nativo do Hadoop:

```
hadoop dfs -getmerge gs://hadoop-spark-fiocruz/saída saída-local
```

Para medir a escalabilidade, eficiência e *speedup*, o SparkBLAST foi executado de forma a utilizar 1 *core* por nó e 2 *cores* por nó no Experimento 1 e 4 *cores* no Experimento 2. Além disso foram comparados os resultados com o CloudBLAST. Para cada tipo de experimento (CloudBLAST, SparkBLAST 1 core/nó, SparkBLAST 2 cores/nó e CloudBLAST e SparkBLAST 4 *core*/nó) foram rodados 6 testes, com o objetivo de retirar a média entre eles e confirmar que o tempo medido não sofre grande variação entre as várias execuções **Seção 5**.

Em relação ao SparkBLAST e CloudBLAST, a única diferença na execução dos trabalhos é a forma como os processos são invocados e executados. Para o SparkBLAST teve-se que criar um pequeno trecho de código para execução (arquivo disponibilizado no GitHub[15]), já para o CloudBLAST utilizou o código fonte fornecido pelos desenvolvedores da aplicação. Nas linhas subsequentes é mostrado como cada trabalho foi executado:

- SparkBLAST:

```
spark-submit --executor-memory {qtd_memory} --driver-memory {qtd_memory}
--num-executors {num} --executor-cores 1 --driver-cores {num} --class Sp
arkBLAST /local/codigo.jar {qtd_core x num_nós} "/local/bin/blastall -p
blastp -d /local/DB/database.fa -e 1E-05 -v 1000 -b 1000 -m 8" {entrada}
{saída}
```

- CloudBLAST:

```
hadoop jar /local/hadoop-streaming-2.7.1.jar -libjars ./StreamPatternRe
cordReader.jar -input query.fa -output hadoop-output -mapper "/local/bi
n/blastall -p blastp -d /local/DB/database.fa -e 1E-05 -v 1000 -b 1000
-m 8" -numReduceTasks 0 -inputreader "org.apache.hadoop.streaming.Strea
mPatternRecordReader,begin=>" -cmdenv BLASTDB=/local/db -jobconf mapred
uce.job.maps="qtd_core x num_nós"
```

O resultado final dessas execuções são pequenos arquivos gerados por cada nó que serão concatenados em um único arquivo através do comando *getmerge*, nativo do Hadoop. O arquivo final com os resultados tem um tamanho aproximado de 610 MB.

[15] SparkBLAST - https://github.com/Ufscar-Fiocruz-Ifsul/spark-blast2.0

4.2.5 Pós-processamento

O pós-processamento corresponde à etapa final na qual é feita medição dos tempos de execução desta ferramenta em relação às ferramentas citadas na seção de Trabalhos Correlatos, mais especificamente, o CloudBLAST.

Essa medição é discutida na **Seção 5 - Resultados.**

Capítulo 5

RESULTADOS E TRABALHOS FUTUROS

5.1 Resultados

Nesta seção são descritos os resultados obtidos com a execução do BLAST nos ambientes configurados na Google Cloud e Microsoft Azure exemplificados na **Seção 4**. Estes resultados foram colhidos tanto para a aplicação desenvolvida em Hadoop, quanto para a desenvolvida e descrita nesta dissertação, feita em Spark.

Nos ambientes de testes e cenários definidos ao longo deste trabalho, teve-se como meta medir a escalabilidade, eficiência e *speedup* do SparkBLAST comparado ao CloudBLAST (BLAST sobre o Hadoop). Além disso, poder demonstrar que o Apache Hadoop e Apache Spark podem trabalhar em conjunto e que o Spark pode ser uma alternativa para a execução de determinados tipos de aplicações. Ainda é investigado a robustez e facilidade em escrever algoritmos no Spark e demonstrado que essa aplicação é transparente para a execução em relação ao BLAST. Ou seja, o cliente não precisa conhecer as ferramentas aqui apresentadas, apenas saber utilizar o BLAST.

O SparkBLAST foi executado de forma a utilizar 1 *core* por nó e 2 *cores* por nó, além disso foram comparados os resultados com o CloudBLAST (2 *cores* por nó) na nuvem Google e 4 *cores* por nó na nuvem Azure em ambas as ferramentas. Foram executados 2 experimentos um em cada nuvem, para cada um destes foram executados 6 medidas de tempo com a mesma entrada e parâmetros (estes definidos na **Seção 4**). O objetivo desses testes foi de retirar a média entre eles e confirmar que o tempo medido não sofre grande variação entre as diversas execuções.

Os tempos foram medidos em segundos (s). Na Google houve uma comparação em relação à quantidade de nós utilizados por tempo de execução de cada tarefa e na Azure essa relação foi

referente à quantidade de núcleos (*core*) por tarefa executada.

As saídas produzidas pela aplicação em cada um dos testes fornecem o resultado e os tempos de início e término da execução. Esses dados são referentes ao tempo necessário ao processamento da aplicação e são usados para medir o *speedup* e eficiência do SparkBLAST e do CloudBLAST. Nas tabelas e imagens subsequentes os tempos de execução foram mensurados com o propósito de aferir os tempos médios para processar as informações, cada um dos testes teve o tempo medido em segundos (s).

Para todas as tabelas, a primeira linha representa a quantidade de nós alocados para a medição do tempo de execução. Para a quantidade total de nós disponíveis foram executados 7 configurações diferentes em relação à escalabilidade, variando entre 1 e 64 nós na Google e 6 configurações na Azure, variando entre 1 e 63 nós. Em cada configuração de conjuntos de nós, conforme proposto, foram feitas 6 medições de tempo para retirar a média gasta na execução.

Dos experimentos executados nota-se que ao acrescentar mais nós ao *cluster* o tempo de execução para ambas as ferramentas foi decrescendo. Porém, conforme é esperado, o tempo não decresce na proporção de nós alocados por tarefa, há uma perca. Isso porque o mesmo trabalho foi dividido entre outros nós e a divisão de tarefas, comunicação entre os nós e latência da rede fazem com que o tempo aumente. Nos parágrafos que se seguem é feita uma discussão para cada uma das Tabelas que detalham os resultados obtidos.

As Tabelas 5.1, 5.2, 5.3 e 5.4 são relacionadas ao Experimento 1 conduzido na Google e as Tabelas 5.5 e 5.6 dizem respeito ao Experimento 2 e os tempos foram colhidos da execução no ambiente da Azure.

Na Tabela 5.1 é demonstrado os tempos gastos pelo CloudBLAST para processar as informações. O tempo médio para o processamento em apenas 1 nó foi de 30.547,29 segundos e com 64 nós o processamento foi concluído em uma média de 825,62 segundos. Pode ser observado que com o aumento de nós, devido à comunicação entre eles o *speedup* e eficiência caem também. O *speedup* variou de 1,61 e eficiência de 0,80 para 2 nós até 37 vezes para 64 nós com uma eficiência de 0,58.

Nas Tabelas 5.2 e 5.3 é demonstrado os tempos gastos pelo SparkBLAST 1 *core* e SparkBLAST 2 *core*.

Para o processamento do SparkBLAST com a utilização de apenas 1 *core* para processamento o tempo médio foi de 36.690,20 segundos e com 2 *core* decresceu para 28.983,44 segundos. Ao acrescentar mais nós para executar o mesmo trabalho o tempo foi reduzido e as informações foram processadas em 898,02 e 693,30 segundos com 64 nós, para máquinas

Tabela 5.1: Tempo execução (Experimento 1 - query.fa - 36MB) - SparkBLAST - Google Cloud

# nó	**1**	**2**	**4**	**8**	**16**	**32**	**64**
T1	29.921,40	19.018,00	11.324,00	6.204,00	2.866,00	1.680,00	794,00
T2	30.256,23	18.550,25	13.799,23	5.779,21	2.959,65	1.828,23	900,00
T3	31.016,85	19.221,81	12.580,32	5.700,52	3.004,52	1.597,00	815,21
T4	31.350,25	19.102,68	10.489,53	5.850,02	2.961,23	1.806,25	842,30
T5	30.726,89	18.981,32	12.721,23	5.780,34	2.990,81	1.780,32	799,21
T6	30.012,14	19.118,72	11.820,85	5.900,64	3.008,15	1.753,23	802,98
Média	30.547,29	18.998,80	12.122,53	5.869,12	2.965,06	1.740,84	825,62
Desvio	576,25	235,28	1.164,02	177,70	52,79	87,20	40,32
Desvio/ Média	1,89%	1,24%	9,60%	3,03%	1,78%	5,01%	4,88%
Speedup	1.00	1.61	2.52	5.20	10.30	17.55	37.00
Eficiência	1.00	0.80	0.63	0.65	0.64	0.55	0.58

utilizando 1 e 2 *core* respectivamente.

O *speedup* obtido variou de 1,93 para 2 nós a até 40,86 com 64 nós com uma eficiência de 0,97 a 0,64 na execução com 1 *core* por equipamento. Já para o processamento com 2 *core* por máquina o *speedup* de 1,99 para 2 nós e 41,81 vezes para 64 nós com uma eficiência de praticamente 1 a 0,65, para 2 e 64 nós respectivamente.

Tabela 5.2: Tempo execução (Experimento 1 - query.fa - 36MB) - SparkBLAST 1 *core*/nó - Google Cloud

# nó	**1**	**2**	**4**	**8**	**16**	**32**	**64**
T1	36.106,86	18.845,23	10.189,11	5.556,22	3.129,20	1.716,10	905,21
T2	36.510,12	19.120,32	10.199,85	5.540,15	3.115,12	1.730,58	899,84
T3	36.720,86	18.952,15	10.170,23	5.560,88	3.140,01	1.790,96	894,76
T4	38.120,25	18.998,06	10.200,01	5.543,62	3.120,58	1.694,69	900,42
T5	36.230,56	19.112,23	10.178,76	5.552,10	3.122,15	1.701,55	897,65
T6	36.452,53	18.880,11	10.183,61	5.565,11	3.127,58	1.710,68	890,25
Média	36.690,20	18.984,68	10.186,93	5.553,01	3.125,77	1.724,09	898,02
Desvio	733,00	115,14	11,83	9,73	8,62	35,01	5,14
Desvio/ Média	2,00%	0,61%	0,12%	0,18%	0,28%	2,03%	0,57%
Speedup	1	1.93	3.60	6.61	11.74	21.28	40.86
Eficiência	1	0.97	0.90	0.83	0.73	0.67	0.64

Na Tabela 5.4 é mostrado um resumo dos tempos médios gastos para execução do BLAST para o Spark e o Hadoop (resumo das Tabelas 5.1, 5.2 e 5.3, discutidos acima) e na Figura 5.1 é exibido um gráfico com esses resultados nos cenários descritos.

Nas Tabelas 5.5 e 5.6 é demonstrado os tempos gastos pelo SparkBLAST e CloudBLAST ao serem executados na plataforma Azure.

Tabela 5.3: Tempo execução (Experimento 1 - query.fa - 36MB) - SparkBLAST 2 *core*/nó - Google Cloud

# nó	**1**	**2**	**4**	**8**	**16**	**32**	**64**
T1	28.915,52	14.500,86	7.935,45	4.287,85	2.249,94	1.260,12	695,23
T2	29.002,21	14.520,23	7.945,10	4.290,12	2.230,26	1.259,28	690,04
T3	29.001,89	14.515,35	7.950,01	4.283,56	2.255,04	1.260,10	701,50
T4	28.989,52	14.557,51	7.942,20	4.282,21	2.242,63	1.259,52	710,11
T5	28.990,32	14.580,01	7.940,80	4.310,12	2.249,26	1.259,82	680,80
T6	29.001,15	14.520,23	7.950,12	4.295,56	2.251,08	1.262,15	682,10
Média	28.983,44	14.532,37	7.943,95	4.291,57	2.246,37	1.260,17	693,30
Desvio	33,78	29,93	5,68	10,27	8,85	1,03	11,37
Desvio/ Média	0,12%	0,21%	0,07%	0,24%	0,39%	0,08%	1,64%
Speedup	1.00	1.99	3.65	6.75	12.90	23.00	41,81
Eficiência	1.00	1.00	0.91	0.84	0.81	0.72	0.65

Tabela 5.4: Execução CloudBLAST vs SparkBLAST (tempo, *speedup* e eficiência)

# nó	**1**	**2**	**4**	**8**	**16**	**32**	**64**
SparkBLAST 1 *core*	36,690.20	18,984.68	10,186.93	5,553.01	3,125.77	1,724.09	898.02
Speedup	1	1.93	3.60	6.61	11.74	21.28	40.86
Eficiência	1	0.97	0.90	0.83	0.73	0.67	0.64
SparkBLAST 2 *core*	28983.44	14532.37	7943.95	4291.57	2246.37	1260.17	693.30
Speedup	1.00	1.99	3.65	6.75	12.90	23.00	41,81
Eficiência	1.00	1.00	0.91	0.84	0.81	0.72	0.65
CloudBLAST 2 *core*	30547.29	18998.80	12122.53	5869.12	2965.06	1740.84	825.62
Speedup	1.00	1.61	2.52	5.20	10.30	17.55	37.00
Eficiência	1.00	0.80	0.63	0.65	0.64	0.55	0.58

Tabela 5.5: Tempo de processamento, *speedups* e eficiência (Experimento 2 - buz.fasta - 805MB) - SparkBLAST vs CloudBLAST - Microsoft Azure

# nó	**1**	**3**	**7**	**15**	**31**	**63**
SparkBLAST	143.228,95	47.031,62	24.850,51	11.692,45	6.041,64	3.138,64
Speedup	3,83	11,67	22,09	46,95	90,86	174,89
Efficiency	0,96	0,97	0,79	0,78	0,73	0,69
CloudBLAST	148.512,95	47.950,05	26.858,71	11.951,11	6.993,52	3.879,06
Speedup	3,7	11,45	20,44	45,93	78,49	141,51
Efficiency	0,92	0,95	0,73	0,77	0,63	0,56

Na medição de tempo, pode ser observado que para todos os testes mensurados, quando se usa todo o poder computacional dos equipamentos, o Spark se mostrou superior ao Hadoop. Até mesmo quando se utilizou apenas um *core* por nó, dos dois disponíveis, o Spark se mostrou

Tabela 5.6: Tempo de processamento, *speedups* e eficiência (Experimento 2 - ber.fasta - 11GB) - SparkBLAST vs CloudBLAST - Microsoft Azure

# nó	1	3	7	15	31	63
SparkBLAST	2.678.902,06	859.687,13	458.759,75	224.869,12	110.222,98	56.200,21
Speedup	3,73	11,61	21,76	44,4	90,57	177,64
Efficiency	0,93	0,97	0,78	0,74	0,73	0,7
CloudBLAST	-	-	-	-	-	-
Speedup	-	-	-	-	-	-
Efficiency	-	-	-	-	-	-

promissor.

Para uma execução local com uma máquina configurada com apenas um processador de um *core* levaria em torno de 10 horas de processamento para executar o Experimento 1. Esse tempo foi decrescido para apenas, aproximadamente, 11 minutos de processamento com 64 nós. O que mostra que ambas as ferramentas, a desenvolvida em Spark, similar à estudada em Hadoop, é escalável.

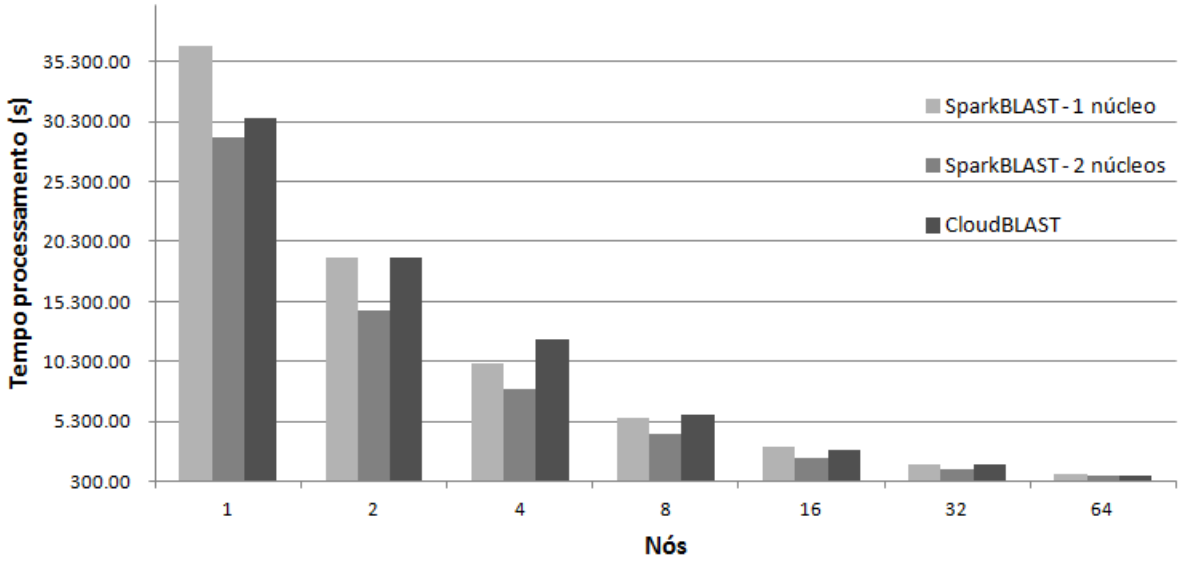


Figura 5.1: Execução do BLAST sobre o Spark comparado com o CloudBLAST - Google

Além disso, pode-se comprovar que o *speedup* obtido pela execução do Spark Pipe 2 *core* é relativamente melhor que os demais resultados obtidos pela execução do Spark em apenas 1 *core* e do CloudBLAST, ressaltando o aproveitamento da configuração do *hardware* disponível. Pois, no CloudBLAST houve um *speedup* de 37 vezes, no SparkBLAST 1 *core* 40,86 vezes e com o SparkBLAST 2 *core* um *speedup* de 41,81 vezes.

Esse ganho em relação ao tempo gasto no processamento e *speedup* diz respeito à utilização e reaproveitamento dos dados em memória e do uso feito por todos os *cores* do processador no

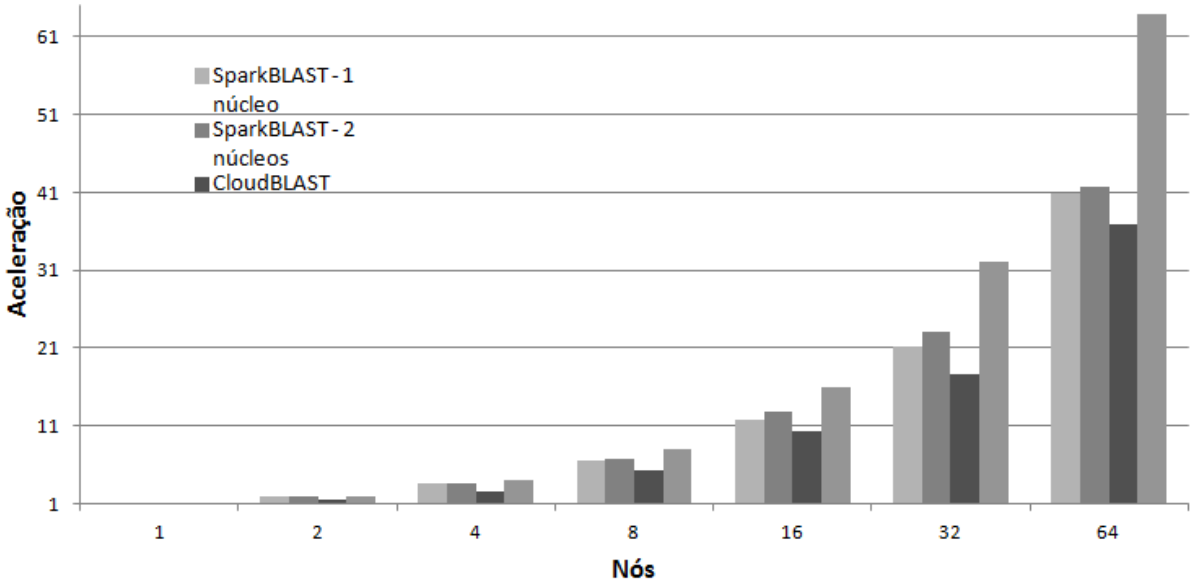


Figura 5.2: Speedup obtido na execução do BLAST sobre o Spark e a comparação com o CloudBLAST - Google

SparkBLAST.

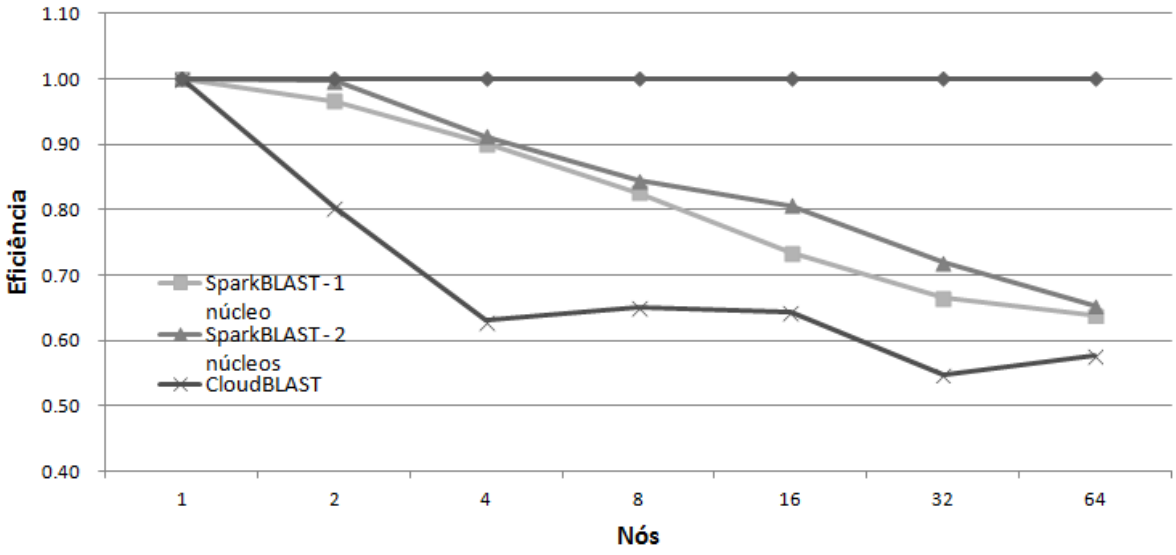


Figura 5.3: Eficiência obtida na execução do BLAST sobre o Spark e a comparação com o CloudBLAST - Google

Por ter se obtido um *speedup* superior do Spark em relação ao Hadoop, a ferramenta também se mostra mais eficiente (Figura 5.3).

Durante toda a etapa de processamento cada tarefa individual produz um pequeno arquivo de saída. Durante a fase de pós-processamento, é mesclado todos esses pequenos arquivos em um único arquivo de saída final com o *getmerge*. O Experimento 1 produziu um arquivo de saída final de 610MB e o Experimento 2 dois arquivos, um arquivo com 92MB (Buz.fasta)

e o outro com 441MB (Ber.fasta). Todos os dados de saída são gravados no DFS, isto é, no armazenamento do Google Cloud Storage ou no Blob do Microsoft Azure.

Assim, o Experimento 1 engloba execuções de $2 \times 7 \times 6 = 84$ no total, o que exigiu mais de 350 horas (relógio de parede) para executar. Como uma estimativa da quantidade de recursos computacionais necessários, esta experiência consumiu 2.420 horas de vCPU para ser executada no Google Cloud. O Experimento 2 abrangiu $2 \times 2 \times 7 \times 6 = 168$ execuções no total, o que exigiu mais de 8.118 horas (relógio de parede) para executar. Uma estimativa sobre a quantidade de recursos computacionais, esta experiência consumiu mais de 139.595 vCPU-horas para executar no Azure Cloud.

Os tempos médios de execução são apresentados na Figura 5.1. O SparkBLAST obteve um *speedup* máximo (que é a razão entre o tempo de execução da linha de base de um nó durante o tempo de execução para a execução paralela) de 41,78, reduzindo o tempo de execução de 28.983 segundos em um único nó para 693 segundos em 64 nós. No mesmo cenário, CloudBLAST atingiu a velocidade de 37 reduzindo o tempo de execução de 30.547 segundos para 825 segundos em 64 nós. Para este conjunto de execuções, tanto o SparkBLAST quanto o CloudBLAST usaram 2 vCPUs por nó para a execução de tarefas. O *speedup* é apresentado na Figura 5.2. Conforme mostrado, o SparkBLAST apresenta melhor escalabilidade do que CloudBLAST.

Para a plataforma Microsoft Azure assim como na Google Cloud, o SparkBLAST supera o CloudBlast em todos os cenários. Ambos os conjuntos de dados (Buz.fasta e Ber.fasta) foram processados e os resultados são apresentados nas Figuras 5.4 apresentando o tempo de processamento. A Figura 5.5 apresenta o *speedup* e a Figura 5.6 a eficiência. Os resultados também são apresentados na Tabela 5.5 (Buz.fasta) e Tabela 5.6 (Ber.fasta). Vale ressaltar que o maior conjunto de dados (ber.fasta - 11 GB) foi maior do que a memória disponível nós de computação. Por esse motivo, CloudBLAST não pôde processar o conjunto de dados ber.fasta, enquanto SparkBLAST não tem essa limitação. Também vale a pena mencionar que maiores velocidades foram alcançadas no Microsoft Azure quando comparado ao Google Cloud. Isso pode ser parcialmente explicado pelo fato de que todos os nós de computação alocados no Microsoft Azure são colocados no mesmo local, enquanto os nós de computação no Google Cloud foram distribuídos em 4 locais diferentes.

Neste trabalho, foi investigado a paralelização de algoritmos de alinhamento de seqUências através de uma abordagem que emprega computação em nuvem para o provisionamento dinâmico de grandes quantidades de recursos computacionais e Apache Spark como a estrutura de coordenação para a paralelização da aplicação. O SparkBLAST, uma paralelização escalável de algoritmos

de alinhamento de sequências foi apresentado e avaliado. O operador *pipe* do Apache Spark e sua abstração principal RDD (*resilient distribution dataset*) são usados para realizar pesquisas escalonáveis de alinhamento de proteínas invocando BLASTP como uma biblioteca de aplicativos externa. Os experimentos acima descritos demonstraram que o sistema baseado em Spark supera um sistema de última geração implementado no Hadoop em termos de tempos de aceleração e execução.

Para a plataforma Microsoft Azure, todas as medidas são melhores no SparkBLAST quando comparadas com a correspondente execução do CloudBLAST para o conjunto de dados Buz.fasta (805 MB). Vale ressaltar que a diferença de aceleração em favor de SparkBLAST aumenta com o número de nós de computação, o que destaca a escalabilidade aprimorada do SparkBLAST sobre o CloudBLAST. Como mencionado nesta seção, não foi possível processar o maior conjunto de dados Ber.fasta (11 GB) usando CloudBLAST devido à limitação de memória principal do nó de computação. Essa restrição não afeta o SparkBLAST, que pode processar conjuntos de dados mesmo quando eles são maiores que a memória principal disponível em nós de computação (Tempo processamento - Figura 5.4, Aceleração - Figura 5.5 e Eficiência - Figura 5.6). No caso do Spark, cada processo invocado por uma tarefa (cada núcleo é associado a uma tarefa) pode usar o mesmo RDD quando o banco de dados não se encaixa na memória, a reutilização do conteúdo de memória e a implementação da memória circular (GOPALANI; ARORA, 2015).

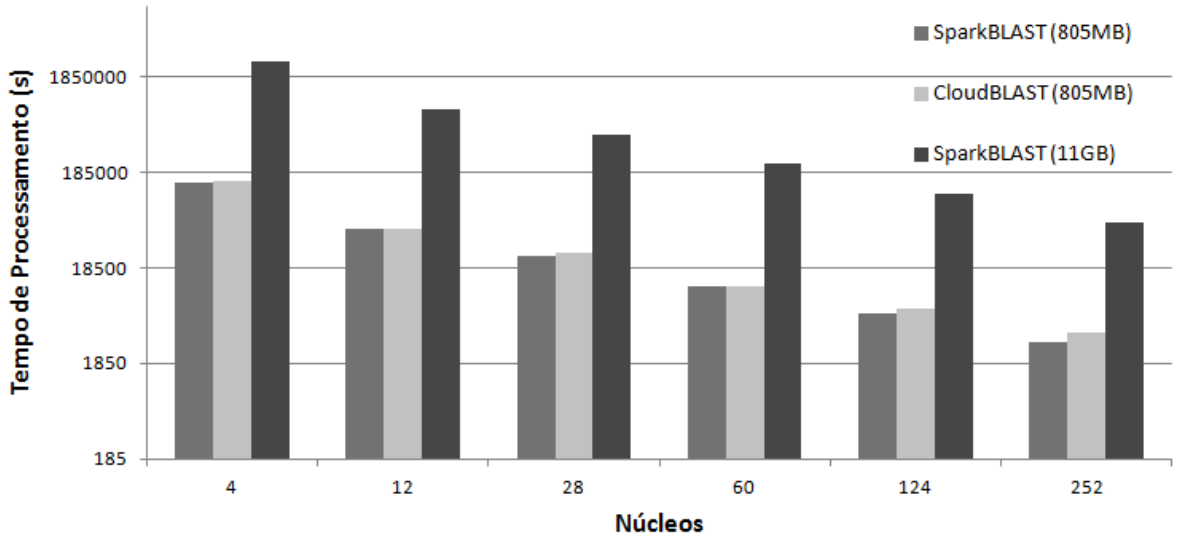


Figura 5.4: Execução do BLAST sobre o Spark comparado com o CloudBLAST - Azure

Assim, a principal razão por trás do desempenho do SparkBLAST quando comparado aos sistemas baseados em Hadoop são as operações em memória e sua abstração RDD relacionada. O número reduzido de operações de E/S de disco pelo SparkBLAST resulta em uma melhoria

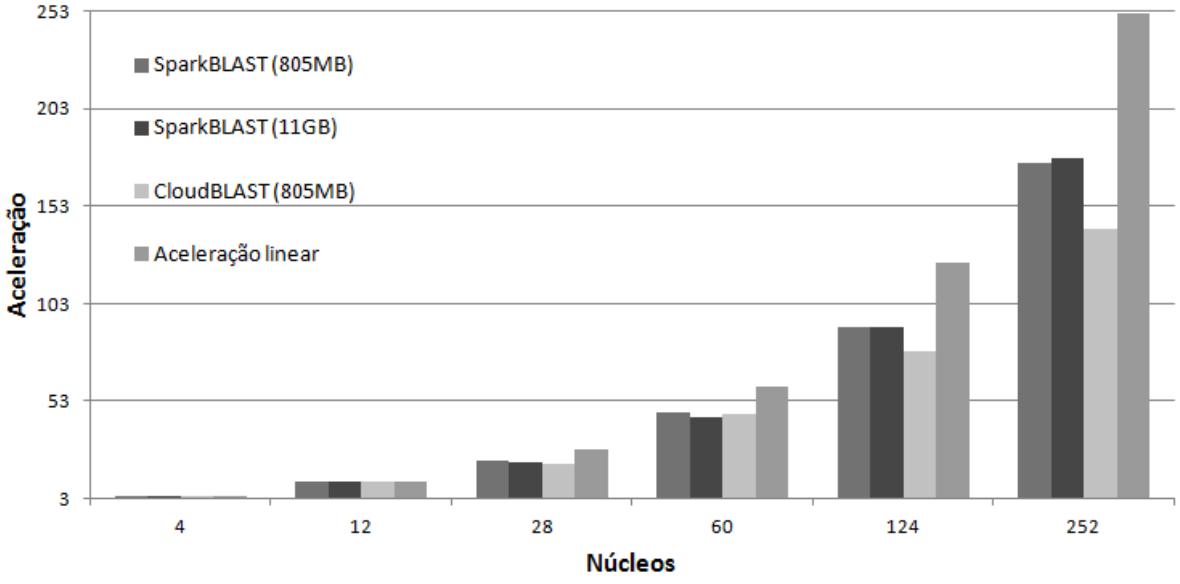


Figura 5.5: Speedup obtido na execução do BLAST sobre o Spark e a comparação com o CloudBLAST - Azure

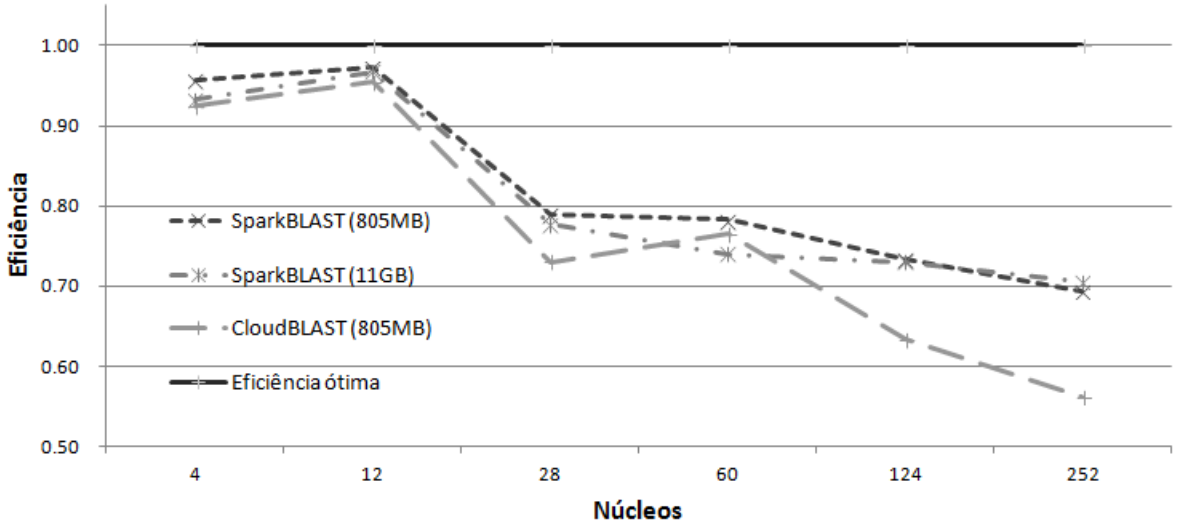


Figura 5.6: Eficiência obtida na execução do BLAST sobre o Spark e a comparação com o CloudBLAST - Azure

significativa no desempenho geral quando comparado com a implementação do Hadoop. É claro que as operações em memória disponíveis no SparkBLAST desempenham um papel importante tanto nas melhorias de *speedup* e eficiência e, como consequência, também na escalabilidade do sistema.

Ainda sobre a limitação do CloudBLAST e a não limitação do SparkBLAST para dados maiores que a memória têm-se que para cada função *map*, a memória principal é alocada em um nó. Se o processo exceder a quantidade de memória disponível em um nó, a função *map* fa-

lha (como ocorreu quando houve a tentativa em processar o conjunto de dados ber.fasta usando CloudBLAST) ou o tempo de execução aumenta significativamente (O'DRISCOLL, 2015). No caso do Spark, cada processo invocado por uma tarefa (cada núcleo é associado a uma tarefa) pode usar o RDD mesmo quando o banco de dados não se encaixa na memória, a reutilização do conteúdo de memória e a implementação da memória circular (GOPALANI; ARORA, 2015). Com essa abordagem de reutilização de dados, o Spark pode processar tarefas mesmo quando os dados são maiores do que a memória disponível em um nó. Na página inicial do projeto Spark (http://spark.apache.org/) é possível observar como a persistência do RDD funciona quando os dados não se encaixam na memória: os RDDs são armazenados como objetos Java desserializados na JVM. Se o RDD não se encaixa na memória, algumas partições não serão armazenadas em cache e serão recalculadas no momento em que forem necessárias (ZAHARIA, 2010). Este é o caso padrão. No artigo "Comparando Apache Spark e Map Reduce com Análise de Desempenho usando K-Means"(GOPALANI; ARORA, 2015) os autores apresentam vários cenários onde Spark pode ser melhor que Hadoop devido a uma alocação de memória mais eficiente. Portanto, as razões para escolher Spark são: cache de dados na memória, o que é benéfico no caso de algoritmos iterativos; e, consulta de dados interativa, onde um aplicativo carrega dados na memória principal de um cluster para várias consultas consecutivas.

Em relação ao Experimento 1 e inferência RBH, mostramos que nossos resultados SparkBLAST podem ser pós-processados para inferir genes compartilhados, gerando, então, valor agregado para a análise de similaridade. Isso também significa que RBH experiências usando SparkBLAST são potencialmente escaláveis para muitos mais genomas, e pode mesmo ser usado como parte de outro software de inferência de homologia baseado em Blast, como OrthoMCL (LI; STOECKERT; ROOS, 2003). Considerando a Experiência 2, os resultados indicam que 1,27 % das proteínas metagenômicas das Bermudas e 1,4 % das proteínas metagenômicas de Búzios representam homólogos potenciais às 10 bactérias resistentes à radiação, e até onde é sabido não foram publicados estudos relacionados até o momento. Esses homólogos potenciais serão investigados em outro estudo.

Portanto, é demonstrado que para aplicações como o BLAST, onde o trabalho pode ser dividido entre vários nós e parte dos resultados gerados não interferem nos demais, a execução sobre um *cluster* com o *framework* Spark é viável, escalável e reduz o tempo para o processamento de dados.

Capítulo 6

CONCLUSÃO

6.1 Conclusão

Com o passar dos anos têm-se produzido uma grande quantidade de informações relacionadas à Bioinformática, principalmente em relação à genômica. Esse crescimento se deve aos atuais mecanismos de sequenciamento genético e aos estudos desenvolvidos a partir de uma ou várias sequências gênicas. Muitos aplicativos previamente desenvolvidos para processar tais informações foram feitos para serem executados em apenas uma máquina ou com apenas um *core* por equipamento. Por outro lado, a computação em nuvem tem colocado a disposição de pesquisadores grandes quantidades de recursos, que podem ser alocados rapidamente e com baixo custo se comparado ao custo de recursos próprios. Na busca de alternativas para melhorar o desempenho quanto ao tempo de processamento muitas aplicações tem sido reescritas utilizado alguma tecnologia de processamento paralelo e distribuído, tais como: MPI, OpenMP, CUDA, entre outras..

No entanto, reescrever aplicações é uma tarefa muitas vezes excessivamente dispendiosa e lenta. Em alguns casos, tem sido possível adaptar a aplicação para ser executada em vários nós (*cluster*) tem sido uma alternativa para esse problema. Este é o caso da ferramenta BLAST, utilizada nesta pesquisa. Os *frameworks* Hadoop e Spark, que utilizam processamento distribuído oferecem mecanismos (Hadoop streaming e Spark pipes) que facilitam a execução distribuída de ferramentas legadas como o BLAST. Ambos os *frameworks* permitem a distribuição do trabalho inicial entre os vários nós de um sistema distribuído, como um *cluster* local ou na nuvem, com grande escalabilidade e eficiência.

Para melhorar a escalabilidade e reduzir o tempo de processamento do BLAST, alguns trabalhos anteriores utilizaram o Hadoop para executar o BLAST de forma distribuída sobre as

bases de dados e sequências particionadas, como é o caso do CloudBLAST e Biodoop. Já o mpiBLAST reescreveu a aplicação para executar sobre a biblioteca MPI.

Nesta pesquisa de mestrado foi investigado como o Spark trabalha com o processamento da aplicação em vários nós. O SparkBLAST foi projetado sobre a ferramenta Spark porque esta se mostrou bastante escalável e eficiente no uso de memória. Além disso, na época em que a pesquisa foi iniciada não havia implementações do BLAST sobre o Spark.

Foram feitos experimentos comparando o SparkBLAST com o CloudBLAST, este último feito sobre o Hadoop. O SparkBLAST se mostrou mais escalável em ambiente de nuvem, além de permitir a execução com bases de dados maiores que o a memória das máquinas, o que foi um avanço significativo. Vale destacar que não foi necessário modificar o código fonte do BLAST, o que permite atualizar o SparkBLAST a cada nova versão do BLAST de forma trivial.

6.2 Contribuições

A presente pesquisa gerou as seguintes contribuições:

- A ferramenta BLAST foi adaptada para executar de forma distribuída sobre o ambiente Spark em cluster ou na nuvem. O código desenvolvido foi disponibilizado publicamente no endereço https://github.com/sparkblastproject/v2. Até a data de início desta pesquisa, nenhum outro trabalho havia utilizado o Spark para a paralelização do BLAST.
- Foi apresentada uma comparação entre o CloudBLAST (baseado no Hadoop) e o SparkBLAST (baseado no Spark) executando em ambiente de nuvem. Os experimentos demonstraram o uso das ferramentas em duas plataformas de nuvem, tanto da Google quanto Microsoft Azure.
- Foi submetido um artigo para a BMC Bioinformatics. Os autores deste artigo são os envolvidos nessa dissertação e os seguintes pesquisadores da FIOCRUZ: Catherine Tostes, Alberto Davila e Fabrício Silva. O título do trabalho submetido é: SparkBLAST: Scalable BLAST processing using in-memory operations. O artigo foi aceito para publicação.

viável

6.3 Trabalhos Futuros

Uma das características do Spark que ainda não foi investigada é o *cache* de RDDs. Esse método permite que dados já processados por alguma função do Spark seja mantida na memória e seja reaproveitada por futuras execuções.

Uma outra investigação que deve ser abordada em trabalhos futuros é a divisão da base de dados entre os nós, tendo a base e a sequência divididas e não somente as sequências distribuídas entre os nós.

REFERÊNCIAS

(2015). Apache spark - spark. url: http://spark.apache.org/. Accessed: 2015-09-23.

(2016). Spark job flow. url: https://databricks-training.s3.amazonaws.com/slides/advanced-spark-training.pdf. Accessed: 2016-06-23.

Afgan, E., Baker, D., Coraor, N., Chapman, B., Nekrutenko, A., and Taylor, J. (2010). Galaxy cloudman: delivering cloud compute clusters. *BMC bioinformatics*, 11(12):1.

Albà, M. M. and Castresana, J. (2007). On homology searches by protein blast and the characterization of the age of genes. *BMC evolutionary biology*, 7(1):53.

ALBERTS, B., BRAY, D., HOPKIN, K., JOHNSON, A., LEWIS, J., RAFF, M., ROBERTS, K., and WALTER, P. (2010). *Fundamentos da Biologia Molecular da Célula.* Editora Artmed, 5ª Edição.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410.

Apache Software Foundation. Hadoop.

Borthakur, D. (2008). Hdfs architecture guide. *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, page 39.

Brown, K. R. and Jurisica, I. (2005). Online predicted human interaction database. *Bioinformatics*, 21(9):2076–2082.

Dai, L., Gao, X., Guo, Y., Xiao, J., and Zhang, Z. (2012). Bioinformatics clouds for big data manipulation. *Biology Direct*, 7(1):43+.

Darling, A., Carey, L., and Feng, W.-c. (2003). The design, implementation, and evaluation of mpiblast. *Proceedings of ClusterWorld*, 2003:13–15.

Davidson, A. and Or, A. (2013). Optimizing shuffle performance in spark. *University of California, Berkeley-Department of Electrical Engineering and Computer Sciences, Tech. Rep*.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Ding, M., Zheng, L., Lu, Y., Li, L., Guo, S., and Guo, M. (2011). More convenient more overhead: the performance evaluation of hadoop streaming. In *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*, pages 307–313. ACM.

Eddy, S. R. et al. (2009). A new generation of homology search tools based on probabilistic inference. In *Genome Inform*, volume 23, pages 205–211.

Gopalani, S. and Arora, R. (2015). Comparing apache spark and map reduce with performance analysis using k-means. *International Journal of Computer Applications*, 113(1).

Griffiths, A., Wessler, S.R.and Lewontin, R., Gelbart, W., Suzuki, D., and Miller, J. (2008). *Introdução à Genética.* Editora Guanabara, 9ª Edição.

Hadoop, A. (2015a). Hadoop Streaming apache hadoop.

Hadoop, A. (2015b). HDFS Architecture apache hadoop.

Hadoop, A. (2015c). YARN Architecture apache hadoop.

Hashem, I. A., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., and Ullah Khan, S. (2015). The rise of " big data" on cloud computing: Review and open research issues. *Information Systems*, 47:98–115.

Higa, R. H. (2001). Entendendo e interpretando os parâmetros utilizados por blast. *Embrapa Informática Agropecuária - Instruções Técnicas*.

Jacob, A., Lancaster, J., Buhler, J., Harris, B., and Chamberlain, R. D. (2008). Mercury blastp: Accelerating protein sequence alignment. *ACM Trans. Reconfigurable Technol. Syst.*, 1(2):9:1–9:44.

Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., and Shahabi, C. (2014). Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94.

Kumar, S. and Filipski, A. (2007). Multiple sequence alignment: in pursuit of homologous dna positions. *Genome research*, 17(2):127–135.

Laney, D. (2001). 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group.

Leo, S., Santoni, F., and Zanetti, G. (2009). Biodoop: bioinformatics on hadoop. In *2009 International Conference on Parallel Processing Workshops*, pages 415–422. IEEE.

Leo, S. and Zanetti, G. (2010). Pydoop: A python mapreduce and hdfs api for hadoop. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 819–825. ACM.

Li, L., Stoeckert, C. J., and Roos, D. S. (2003). OrthoMCL: identification of ortholog groups for eukaryotic genomes. *Genome research*, 13(9):2178–2189.

Liu, W., Schmidt, B., and Muller-Wittig, W. (2011). Cuda-blastp: Accelerating blastp on cuda-enabled graphics hardware. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 8(6):1678–1684.

Luscombe, N. M., Greenbaum, D., Gerstein, M., et al. (2001). What is bioinformatics? a proposed definition and overview of the field. *Methods of information in medicine*, 40(4):346–358.

Mackey, G., Sehrish, S., and Wang, J. (2009). Improving metadata management for small files in hdfs. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–4. IEEE.

Massie, M., Nothaft, F., Hartl, C., Kozanitis, C., Schumacher, A., Joseph, A. D., and Patterson, D. A. (2013). Adam: Genomics formats and processing patterns for cloud scale computing. Technical Report UCB/EECS-2013-207, EECS Department, University of California, Berkeley.

Matsunaga, A., Tsugawa, M., and Fortes, J. (2008). CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications. *eScience, IEEE International Conference on*, 0:222–229.

Miller, W., Makova, K. D., Nekrutenko, A., and Hardison, R. C. (2004). Comparative genomics. *Annu. Rev. Genomics Hum. Genet.*, 5:15–56.

NCBI (2015a). National center for biotechnology information. http://www.ncbi.nlm.nih.gov.

NCBI, G. (2015b). Growth of genbank. http://www.ncbi.nlm.nih.gov/genbank/statistics.

Nguyen, T., Shi, W., and Ruden, D. (2011). CloudAligner: A fast and full-featured MapReduce based tool for sequence mapping. *BMC Research Notes*, 4(1):171+.

Niemenmaa, M., Kallio, A., Schumacher, A., Klemelä, P., Korpelainen, E., and Heljanko, K. (2012). Hadoop-BAM: Directly manipulating next generation sequencing data in the cloud. *Bioinformatics*, 28(6):876–877.

Nordberg, H., Bhatia, K., Wang, K., and Wang, Z. (2013). BioPig: a Hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics*, 29(23):3014–3019.

O'Driscoll, A., Belogrudov, V., Carroll, J., Kropp, K., Walsh, P., Ghazal, P., and Sleator, R. D. (2015). HBLAST: Parallelised sequence similarity – A Hadoop MapReducable basic local alignment search tool. *Journal of Biomedical Informatics*, 54(0):58–64.

O'Driscoll, A., Daugelaite, J., and Sleator, R. D. (2013). 'Big data', Hadoop and cloud computing in genomics. *Journal of Biomedical Informatics*, 46(5):774–781.

Oehmen, C. and Nieplocha, J. (2006). Scalablast: A scalable implementation of blast for high-performance data-intensive bioinformatics analysis. *IEEE Transactions on Parallel and Distributed Systems*, 17(8):740–749.

Pinto, P. M. T. L. N. (2015). *Uma Avaliação de Desempenho dos Ambientes de Programação Paralela Hadoop e Spark*. PhD thesis, CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS.

Pireddu, L., Leo, S., and Zanetti, G. (2011). Seal: a distributed short read mapping and duplicate removal tool. *Bioinformatics*, pages 2159–2160.

Pivetta, M. (2013). Mais bits a serviço do dna. a era dos genomas comparáveis. Revista FAPESP.

Pratx, G. and Xing, L. (2011). Monte carlo simulation of photon migration in a cloud computing environment with mapreduce. *Journal of Biomedical Optics*, 16(12):125003–125003–9.

Rosa, J. O. M. (2006). *Um Estudo de Compactaçãoo de Dados para Biossequências*. PhD thesis, PUC-Rio.

Schumacher, A., Pireddu, L., Niemenmaa, M., Kallio, A., Korpelainen, E., Zanetti, G., and Heljanko, K. (2013). Seqpig: simple and scalable scripting for large sequencing data sets in hadoop. *Bioinformatics*.

Shoro, A. G. and Soomro, T. R. (2015). Big data analysis: Apache spark perspective. *Global Journal of Computer Science and Technology*, 15(1).

Sonnhammer, E. L. L., Gabaldón, T., Sousa da Silva, A. W., Martin, M., Robinson-Rechavi, M., Boeckmann, B., Thomas, P. D., Dessimoz, C., and the Quest for Orthologs consortium (2014). Big data and other challenges in the quest for orthologs. *Bioinformatics*, 30(21):2993–2998.

Taylor, R. (2010). An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics*, 11(Suppl 12):S1+.

Treangen, T. J. and Salzberg, S. L. (2012). Repetitive dna and next-generation sequencing: computational challenges and solutions. *Nature Reviews Genetics*, 13(1):36–46.

Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al. (2013). Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM.

Vouzis, P. D. and Sahinidis, N. V. (2011). GPU-BLAST: using graphics processors to accelerate protein sequence alignment. *Bioinformatics*, 27(2):182–188.

Wetterstrand, K. (2011). DNA Sequencing Costs: Data from the NHGRI Large-Scale Genome Sequencing Program. http://www.genome.gov/sequencingcosts/.

White, T. (2009). *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition.

WHO (2015). World health organization - antimicrobial resistance: global report on surveillance. http://www.who.int/en/.

Wiewiorka, M. S., Messina, A., Pacholewska, A., Maffioletti, S., Gawrysiak, P., and Okoniewski, M. J. (2014). SparkSeq: fast, scalable, cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster Computing with Working Sets. In *HotCloud'10: Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, page 10, Berkeley, CA, USA. USENIX Association.

Zaharia, M., Hamstra, M., Karau, H., Konwinski, A., and Wendell, P. (2015). *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, Incorporated, 22 de fev de 2015 - 276 p.

GLOSSÁRIO

BLAST – *Basic Local Alingment Search Tool*

DDBJ – *DNA Data Bank of Japan*

DFS – *Distributed File System*

DNA – *Deoxyribonucleic Acid*

EMBL – *European Molecular Biology Laboratory*

FASTA – *Formato de arquivo baseado em texto para representar tanto sequencias genômicas, no qual as sequências são representadas usando códigos de uma única letra*

GCS – *Google Cloud Storage*

GPU – *Graphic Processing Unity*

GS – *Google Storage*

GenBank – *Is the NIH genetic sequence database, an annotated collection of all publicly available DNA sequences*

HDFS – *Hadoop Distributed File System*

HPC – *High Performance Computing*

MPI – *Message Passing Interface*

NCBI – *National Center for Biotechnology Information*

NIH – *National Institutes of Health - US*

PGH – *Projeto Genoma Humano*

RDD – *Resilient Distributed Dataset*

RNA – *Ribonucleic Acid*

SGBD – *Sistema Gerenciador de Banco de Dados*

WGS – *Whole-genome Shotgun*

WORM – *Write Once, Read More*