4

Framework Hadoop em Plataformas de Cloud e Cluster Computing

Raffael B. Schemmer – raffael.schemmer@inf.ufrgs.br[1],
Julio C. s. Anjos – julio.c.s.anjos@gmail.com[2],
André L. Tibola – altibola@gmail.com[3],
Junior F. Barros – junior.figueiredo.barros@gmail.com[4],
Cláudio F. R. Geyer – geyer@inf.ufrgs.br[5]

Resumo:

Este minicurso apresenta o framework Hadoop sob uma perspectiva teórica e aplicada. Inicialmente são explicados detalhes da arquitetura do sistema de arquivos distribuído (HDFS) e do modelo de programação MapReduce. Em seguida, o minicurso apresenta o processo de instalação e configuração do Hadoop para o cluster GradeP e para a *cloud* Azure. Por fim, um conjunto de aplicações reais implementadas sob o Hadoop são apresentadas de forma prática e aplicada.

[1]Raffael Bottoli Schemmer é Bacharel e Mestre em Ciência da Computação pela URI Campus de Santo Ângelo (2010/I) e PUC/RS (2012/II) tendo como temas de pesquisa exploração de paralelismo em de arquiteturas VLIW e projeto, geração e avaliação de tráfego em redes intrachip não síncronas. Membro do GPPD na UFRGS e trabalha voluntariamente em um projeto de pesquisa que explora o uso do Hadoop em aplicações de bioinformática.

[2]Júlio César Santos dos Anjos é Doutorando em Ciência da Computação pela Universidade Federal do Rio Grande do Sul - UFRGS/RS. Mestre em Ciência da Computação, pela UFRGS/RS, em 04/2012. Graduado em Eng. Elétrica pela PUC/RS - Pontifícia Universidade Católica do Rio Grande do Sul, em 01/1991. Atualmente é membro do Avalon Team, INRIA/LIP - Laboratoire de l'Informatique du Parallélisme - ENS Lyon, França.

[3]André Luiz Atibola é Engenheiro da Computação pela FURG, mestrando do Programa de Pós-Graduação em Informática da UFRGS e analista em desenvolvimento de sistemas no SERPRO. Atua na aplicação e adequação de frameworks de Big Data para ambientes voluntários com foco na verificação de resultados e autenticidade.

[4]Júnior Figueiredo Barros é Graduando de Engenharia de Computação na UFRGS, atualmente atua no Grupo de Processamento Paralelo e Distribuído - GPPD na UFRGS trabalhando com Computação Intensiva de Dados (BigData), Biotecnologia e adaptação/desenvolvimento sobre o framework Apache Hadoop (MapReduce).

[5]Claudio Fernando Resin Geyer tem Pos-Doutorado em Informática pela Université de Grenoble I (Scientifique et Medicale - Joseph Fourier) - França, em 1996. Doutorado em Informática pela Université de Grenoble I, em 1991. Mestre em Ciência da Computação pela UFRGS/RS, em 1986. Graduado em Engenharia Mecânica pela UFRGS/RS, em 1978. Professor Associado do Instituto de Informática, UFRGS.

4.1. Introdução e Objetivos

Indústria como academia pesquisam há várias décadas por tecnologias e dispositivos computacionais capazes de endereçar as novas demandas causadas pelo surgimento de diferentes tipos de serviços. Na atualidade, os computadores mais adequados às novas tecnologias são as máquinas distribuídas, muitas delas existentes pela demanda causada por sistemas de rede distribuídos [Sterling 2001]. Estes equipamentos combinam recursos computacionais discretos como circuitos de processamento e armazenamento através do uso de uma rede de interconexão. Tais recursos são programados em nível de infraestrutura de maneira que na visão do usuário o equipamento opere de forma transparente, como sendo apenas um único computador. Tecnologias como *cluster* e *cloud computing* compartilham deste conceito de computador. Através da combinação de recursos, possibilita que o usuário adquira uma grande capacidade de processamento [Mell and Grance 2011].

Em um sentido paralelo e semelhante ao esforço despendido para a criação de dispositivos de computação e máquinas distribuídas, plataformas e sistemas computacionais são concebidos para serem utilizados por estes modelos de máquinas em específico. O Hadoop através de uma série de componentes e bibliotecas de programação permite a construção de sistemas capazes de processar arquivos na ordem dos Petabytes de informação em equipamentos distribuídos que operam na ordem de 8000 computadores [White 2012].

Este minicurso tem como objetivo apresentar o framework Hadoop tanto sob uma perspectiva teórica onde a arquitetura dos componentes que formam suas fundações, sendo eles o sistema de arquivos distribuído (HDFS) e o modelo de programação MapReduce, sejam apresentados e estudados. O Hadoop é também apresentado sob um perspectiva prática através de aplicações reais. O processo de instalação e configuração do Hadoop em uma máquina distribuída é também um dos pré-requisitos necessários para o entendimento da tecnologia. Este minicurso pretende também endereçar esta atividade através do uso de duas implementações distintas: um *cluster* dedicado ao minicurso, sendo ele a GradeP do grupo GPPD da UFRGS, e um conjunto de máquinas virtuais na *cloud* Microsoft Azure. O processo de obtenção, instalação e configuração do Hadoop será uma das atividades realizadas por este minicurso.

O restante deste documento está dividido como segue. A seção 4.2. apresenta o Hadoop sob um viés teórico tratando da arquitetura do sistema de arquivos distribuído e do modelo de programação. A seção 4.3. apresenta as etapas e atividades necessárias na obtenção, instalação e configuração do Hadoop para o *cluster* GradeP e para um conjunto de 12 máquinas virtuais em nível de infraestrutura na *cloud* Azure. A seção 4.4. apresenta o Hadoop através de duas aplicações reais. A seção 4.5. descreve as conclusões e apontamentos realizados pelo minicurso.

4.2. Conceitos e Tecnologias Relacionadas

Esta seção apresenta o framework Hadoop sob uma perspectiva teórica e incremental. Inicialmente é feita uma revisão conceitual dos componentes do Hadoop. Em seguida é explicado a arquitetura do sistema de arquivos distribuído HDFS e do modelo de programação MapReduce [White 2012].

4.2.1. O Framework Hadoop

O Hadoop resume-se a um framework ou conjunto de bibliotecas que permitem o armazenamento e processamento de dados de forma distribuída. O Hadoop foi originalmente escrito em 2002 por Doug Cutting sob o nome Nutch [White 2012]. O objetivo inicial visava a implementação de um motor de buscas de páginas web. Nos anos de 2003 e 2004, a Google apresenta a arquitetura do sistema de arquivos distribuído (GFS) [Ghemawat et al. 2003] e o modelo de programação MapReduce utilizados em seu motor de busca [Dean and Ghemawat 2008].

A arquitetura original do Nutch, no que refere-se ao sistema de arquivos distribuído e ao modelo de programação, é fortemente influenciada pela arquitetura do motor de busca da Google [White 2012]. Em 2006, o Nutch passa oficialmente a se chamar Hadoop, quando a Yahoo contrata Doug Cutting para trabalhar em um projeto de integração do Hadoop como motor de buscas da Yahoo. Em 2007, o Hadoop torna-se um projeto de código fonte aberto da Apache [Apache 2013c].

A versão Apache do Hadoop é apenas uma das inúmeras outras versões ou plataformas Hadoop existentes no mercado. O Hadoop é comercializado como produto por empresas como MapR, Cloudera, HortonWorks dentre muitas outras. Cada fabricante procura disponibilizar além dos componentes básicos do Hadoop, suportados pela versão oficial da Apache, uma plataforma completa para o armazenamento e a consulta dos dados estruturados e não estruturados. Este trabalho faz uso do Hadoop 1.2.1 [Apache 2013c]. A Figura 1 descreve os principais componentes do Hadoop suportado pela versão 1.2.1 [White 2012]. Este trabalho irá realizar uma revisão de conceitos referentes as funcionalidades e arquitetura dos principais componentes do Hadoop sendo eles o sistema de arquivos distribuído (HDFS) e o modelo MapReduce.

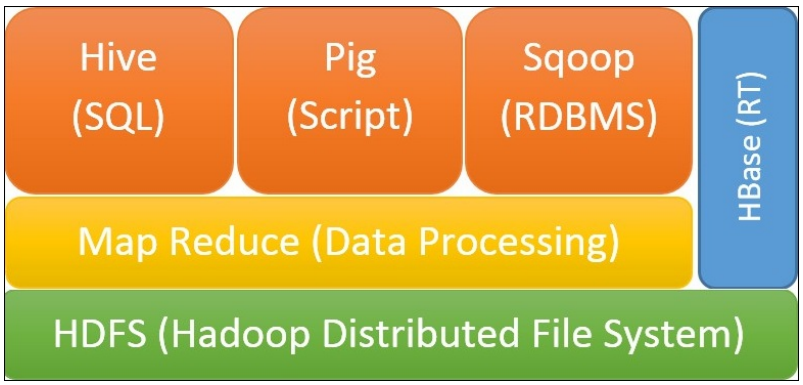

Figura 4.1: A arquitetura de componentes do framework Hadoop..

4.2.2. O Modelo de Programação MapReduce

O MapReduce é um modelo de computação intensiva em dados proposto em 2004, por Jeffrey Dean e Sanjay Ghemawat, para simplificar a construção de índices reversos na análise de pesquisas na web [Dean and Ghemawat 2008]. Os algoritmos utilizados no MapReduce permitem o tratamento de dados intensivos em um sistema de arquivos distribuído, a abstração do paralelismo de tarefas e o controle de falhas.

O funcionamento do MapReduce, em uma abstração em alto nível, pode ser comparado a uma linguagem de consulta SQL. A Figura 2 [Chen and Schlosser 2008] exem-

plifica uma aplicação *word count* para a contagem de palavras de uma determinada entrada de dados. Os dados da entrada são selecionados por uma tarefa *Map* que gera um conjunto de tuplas (chave_in, valor_in). As tuplas são então ordenadas para, na saída, serem agrupadas em tuplas (chave_out, valor_out) por uma tarefa *Reduce*.

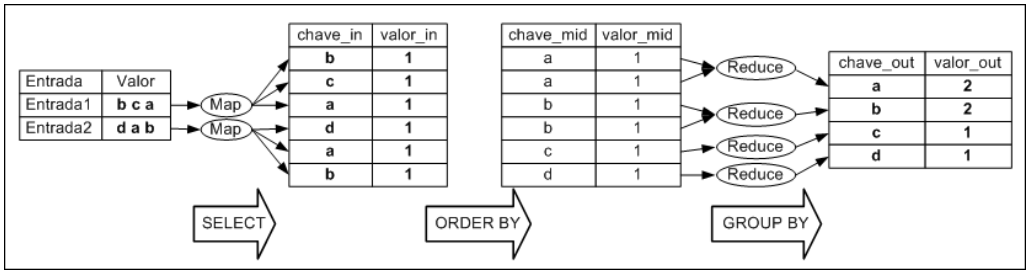

Figura 4.2: Estrutura de programação do modelo MapReduce..

No exemplo, a contagem dos valores das ocorrências das chaves iguais é somada na tarefa de Reduce. Assim, abstraindo-se as funções de *Map* e *Reduce*, pode-se comparar a execução do processamento como um fluxo de uma consulta na linguagem SQL [White 2012]. A representação da consulta ao banco de dados é exemplificada através das funções representadas nos comandos a seguir:
SELECT chave_in **FROM** entrada **ORDER BY** chave_in **CREC**
SELECT COUNT(*), chave_mid **AS** chave_out **FROM** chave_mid **GROUP BY** chave_mid **CREC**

Entretanto, diferentemente de uma consulta em banco de dados, o MapReduce executa esta busca nos dados em centenas ou milhares de máquinas simultaneamente, abstraindo o paralelismo das tarefas, através da declaração de duas funções, como apresentado em [Dean and Ghemawat 2008] no exemplo (pseudocódigo 1) para a contagem de palavras ou *word count*.

O objetivo do *word count* é contar a quantidade de ocorrências de cada palavra em um documento. Cada chamada da função *Map* recebe como valor uma linha de texto do documento e como chave o número desta linha. Para cada palavra encontrada na linha recebida a função emite um par (chave, valor), onde a chave é a palavra em si e o valor é a constante 1 (um). A função *Reduce*, então, recebe como entrada uma palavra (chave) e um iterador para todos os valores emitidos pela função *Map*, associados com a palavra em questão. No final é emitido um par (chave, valor), contendo em cada palavra o total de sua ocorrência [Dean and Ghemawat 2010].

A Figura 3 exemplifica o funcionamento do MapReduce para uma operação *word count*. As funções de *Map* e *Reduce* são aplicadas sobre os dados, e.g. 1 TB de dados. Os dados contêm o número de carros de uma concessionária. O objetivo é encontrar o somatório total do número de carros existentes na concessionária.

Quando uma tarefa for submetida por um programador, um processo *Map* executa as linhas da entrada de dados e emite um valor 1 a cada nome de carro encontrado. Na fase de *Shuffle*, os dados intermediários produzidos no *Map* são ordenados por um *sort* e após as chaves intermediárias são enviadas para serem consumidas pelas tarefas *Reduce* na próxima fase. Finalmente, após as máquinas receberem todos os dados intermediários executa-se um *merge* para preparar os dados para a execução da tarefa *Reduce*. A tarefa *Reduce* soma os valores das chaves semelhantes e emite um resultado.

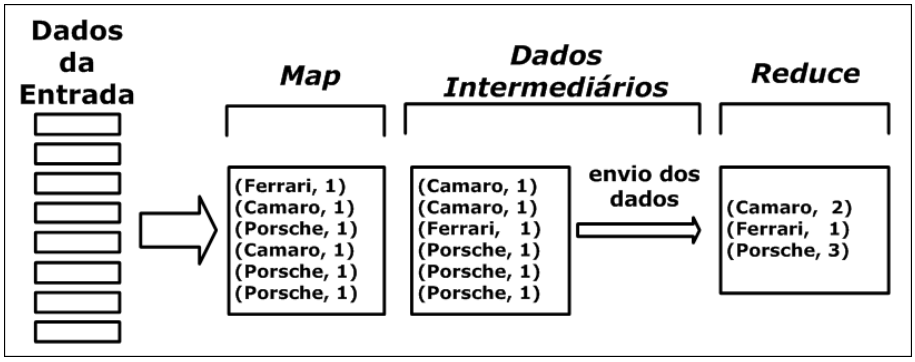

Figura 4.3: Exemplo do fluxo de dados do *word count*..

O MapReduce relaciona-se com um banco de dados pela lógica de consulta dos dados. Porém, diferentemente de uma consulta SQL, o MapReduce tem uma escalabilidade linear. Isto ocorre porque podem ser feitas diversas consultas simultaneamente distribuídas entre várias tarefas. Os dados podem não ter qualquer estrutura interna, serem estruturados ou semiestruturados. Arquivos longos de todos os tipos são bem aceitos para serem analisados neste contexto, sendo desmembrados em tamanhos menores e manipulados por diversas máquinas [White 2012].

O MapReduce abstrai a complexidade do paralelismo das aplicações. O conceito computacional simples da manipulação de dados, através de uma função de tuplas (chave, valor), esconde do programador a grande complexidade da distribuição e do gerenciamento de dados. A complexidade associada aos dados se deve ao grande volume de dados, à dispersão desses dados espalhados em centenas ou milhares de máquinas e à necessidade da computação ser realizada com intervalos de tempos cada vez menores [Dean and Ghemawat 2010].

A arquitetura do MapReduce é constituída de uma máquina mestre que gerencia as demais máquinas escravas. A estrutura do MapReduce é composta de diversos módulos como: um escalonador, mecanismos de *split* de dados da entrada, sistema de arquivos distribuídos, *sort/merge* de dados e sistemas de monitoramento e gerenciamento. O mecanismo de comunicação é baseado no envio de mensagens de *heartbeat* que informa o status das máquinas escravas ao mestre.

A entrada de dados é dividida em pedaços menores chamados de *chunks*. O tamanho é um parâmetro fornecido pelo programador, normalmente de 64 MB, embora o valor dependa da tecnologia dos discos e da taxa de transferência de dados da memória para o disco [White 2012]. Os dados são divididos e após distribuídos em um sistema de arquivos que mantém um mecanismo de replicação dos dados.

A Figura 4, adaptada de [White 2012], apresenta o modelo do fluxo de dados do MapReduce em três fases distintas. As fases de *Map* e *Reduce* são acessíveis ao programador. A terceira, chamada de *Shuffle*, é criada pelo sistema durante sua execução. A tarefa *Map*, quando executa uma função de mapeamento, gera dados intermediários que são armazenados na máquina local. A tarefa *Reduce* recebe estes dados intermediários e executa uma função de redução que agrupa as chaves iguais. O *Shuffle* é constituído de dois processos: um na máquina que processa a tarefa *Map* onde executa-se um *sort* (ordenação de chaves) e serialização dos dados. O segundo ocorre após os dados intermediários serem enviados para a máquina que executará a tarefa de *Reduce*. Nesta máquina é aplicado um *merge* nos dados recebidos das tarefas *Map* com o objetivo de

agrupar adequadamente as chaves intermediárias antes de se executar a função *Reduce* [White 2012].

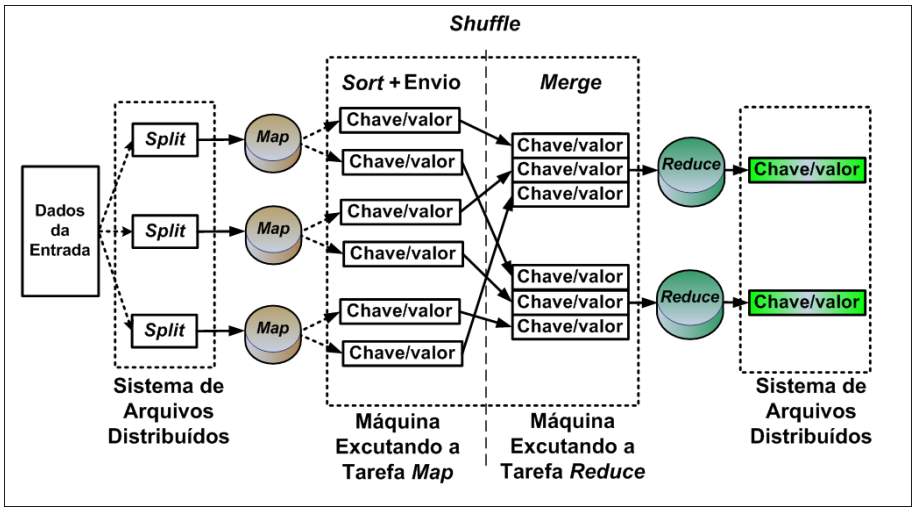

Figura 4.4: Representação das etapas de *Map*, *Reduce* e *Shuffle* do MapReduce..

A implementação do MapReduce particiona cada tarefa em um conjunto de subtarefas menores de tamanhos iguais, com características *Bag-of-Tasks* (tarefas sem dependências). Um cliente submete um *job* com as definições das funções *Map* e *Reduce* que, então, será dividido em várias tarefas ou *tasks* (*Map* e *Reduce*) no mestre. O mestre designa as tarefas às máquinas que irão executar cada etapa do processamento. Os dados intermediários produzidos no *Map* são armazenados temporariamente no disco local. O modelo de execução cria uma barreira computacional, que permite sincronizar a execução de tarefas entre produtor e consumidor. Uma tarefa *Reduce* não inicia seu processamento enquanto todas as tarefas *Map* não tiverem terminado.

Uma função *Hash* é aplicada sobre os dados intermediários produzidos no *Map*, para determinar quais das chaves irão compor as tarefas de *Reduce* a serem executadas. As chaves iguais de todas as tarefas *Map* são transferidas, no processo de *Shuffle*, para uma mesma máquina processar a tarefa de *Reduce*. Após uma função de redução ser aplicada nestes dados intermediários é emitido um novo resultado. O resultado em formato de tuplas é então armazenado no sistema de arquivos distribuído para ser disponibilizado ao cliente que submeteu o *job*.

4.2.3. A arquitetura do Hadoop e do Sistema de Arquivos Distribuído (HDFS)

O Hadoop possui uma arquitetura de serviços distribuídos que oferece suporte a execução de aplicações MapReduce. Em uma visão geral, a arquitetura do Hadoop pode ser interpretada como sendo do tipo mestre/escravo. O mestre possui dois serviços sendo eles o *namenode* e o *jobtracker*. Os escravos possuem cada um deles dois serviços sendo eles o *datanode* e o *tasktracker* [White 2012].

O *namenode* presente no mestre possui como responsabilidade manter a organização e o gerenciamento dos dados que são armazenados nos computadores escravos. Os dados são organizados no *namenode* usando uma estrutura do tipo *metadata*

[White 2012]. O Hadoop possui como objetivo garantir tolerância a falhas através de um mecanismo de replicação de arquivos. A Figura 5 ilustra como exemplo um conjunto de três arquivos que possuem 3 réplicas cada um, dispostas em diferentes computadores do sistema distribuído. É responsabilidade do *namenode* garantir que o sistema de arquivos esteja integro em caso de falhas. Uma vez que exista uma falha no sistema distribuído, o *namenode* deverá criar novas réplicas dos arquivos. As réplicas são criadas para aumentar a tolerância do sistema de arquivos caso novos eventos de falha venham a acontecer [Apache 2013a].

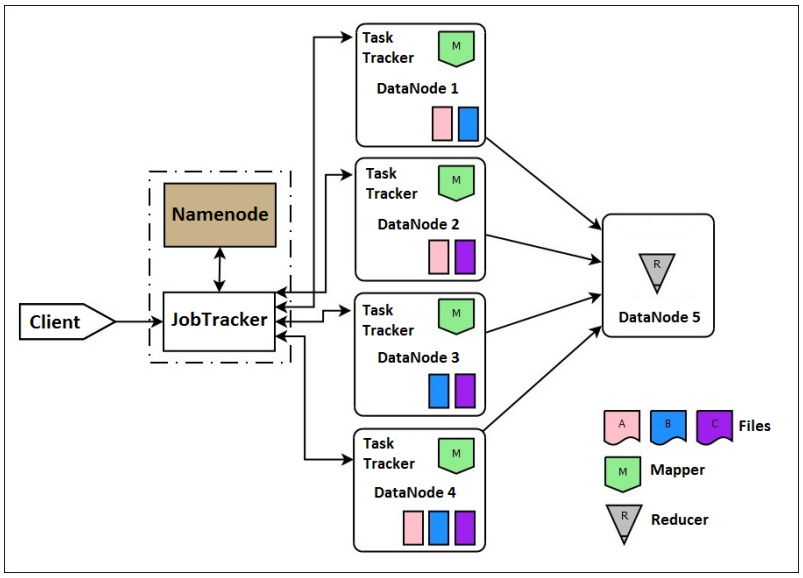

Figura 4.5: Arquitetura mestre/escravo dos serviços de dados e de processos do Hadoop [Menon 2013]..

O *datanode* presente em cada um dos computadores escravos do sistema distribuído possui como responsabilidade armazenar os dados e atender as requisições do *namenode*. O Hadoop especifica políticas e protocolos de maneira que os computadores escravos que executam o *datanode* realizam operações cooperativas, coletivas e autônomas entre si. Estas tecnologias possibilitam que o Hadoop seja escalável sob um sistema largamente distribuído [White 2012].

A Figura 5 [Menon 2013] ilustra os componentes *jobtracker* e *tasktracker* responsáveis respectivamente pela coordenação e pelo processamento de aplicações MapReduce. O *jobtracker* localiza-se no computador mestre do Hadoop, sendo o responsável pelo escalonamento das tarefas e pela coordenação da execução [White 2012]. O serviço *tasktracker* presente em cada um dos computadores escravos do sistema distribuído é o responsável pela execução das tarefas *Map* e *Reduce*, escalonadas pelo *jobtracker*. O mecanismo de gerenciamento utilizado pelo *jobtracker* é baseado no envio de mensagens de *heartbeat* pelos componentes *tasktracker* presentes nos escravos, que informam o seu *status* ao mestre. Esta mensagem permite que o mestre identifique o nível de ocupação dos computadores e ajuste dinamicamente o escalonamento de tarefas entre os computadores escravos do sistema distribuído [Apache 2013a].

4.3. Configuração e Instalação do Hadoop em Máquinas Distribuídas

Esta seção apresenta a instalação e configuração do Hadoop em duas máquinas distribuídas. Este trabalho utiliza como máquinas distribuídas um *cluster* dedicado chamado GradeP e um conjunto de máquinas virtuais disponível na infraestrutura de *cloud* Microsoft Azure.

4.3.1. O cluster GradeP e o Hadoop

Um *cluster* nada mais é do que um conjunto de computadores fracamente acoplados interconectados através de uma rede. A literatura apresenta diferentes classificações quanto aos tipos de *cluster*, que variam conforme suas características como por exemplo a topologia da rede ou a organização da hierarquia de memória do *cluster*. Este trabalho faz uso de um *cluster* do tipo Beowulf chamado GradeP [Sterling 2001]. Este *cluster* foi inicialmente proposto no ano de 2012, sendo até os dias atuais, mantido e atualizado pelo GPPD da UFRGS [GPPD 2015].

O *cluster* GradeP é formado por 30 computadores empilhados em dois racks (201 e 205). Destes 30 computadores, 12 deles estão dedicados e configurados para execução do Hadoop em modo exclusivo. Das 12 máquinas 11 delas são idênticas e possuem processador Intel Pentium IV HT 2.8GHz 32 bits, 2.5GBytes de memória RAM DDR e 1TB de HDD Sata [GPPD 2015]. Uma das máquinas, sendo esta a máquina responsável pela execução do *namenode* e do *jobtracker*, ou o computador coordenador do Hadoop, possui processador Intel Core i5 3GHz, 8GBytes de memória RAM DDR3 e 1TB de HDD Sata. Estes 12 computadores estão diretamente interconectados por uma rede dedicada Ethernet Gigabit. A Figura 6 ilustra um dos racks da GradeP, sendo este o responsável por suportar as 12 máquinas que permitem a execução do Hadoop sob o modo de agendamento exclusivo.

A GradeP opera logicamente com o sistema operacional Rocks *sandboa* 6.1.1 32 bits, baseada na distribuição Linux Cent OS 6.5 kernel 2.6 [Rocks 2014]. O Rocks opera sobre uma arquitetura do tipo mestre/escravo. O sistema operacional é instalado em um computador, sendo este o mestre, gerenciador do *cluster*. Os demais computadores escravos do *cluster* são instalados e configurados automaticamente pelo mestre do *cluster*. No processo de instalação do Rocks é possível instalar automaticamente serviços de rede para uma máquina distribuída do tipo *cluster*, como um sistema de arquivos de rede (NFS) e um mecanismo de acesso seguro do tipo (SSH). Estas ferramentas em específico são fundamentais para acesso e compartilhamento dos dados entre os computadores que formam o *cluster*. O Java 7 requerido pelo Hadoop também é por padrão instalado automaticamente pelo Rocks.

O Hadoop necessita que o computador mestre que irá executar o *namenode* e o *jobtracker* tenha livre acesso aos demais computadores escravos do *cluster*. O acesso é feito pelo serviço de SSH. Além disso, o NFS quando instalado no *cluster*, facilita o processo de instalação do Hadoop. Uma vez que a configuração da instalação do Hadoop esteja concluída, ela deve ser copiada para cada computador escravo que compõe o *cluster*. Este trabalho assume que o Rocks já esteja instalado no *cluster* GradeP, juntamente com os serviços de SSH e Java para o conjunto de 12 máquinas dedicadas da GradeP para execução do Hadoop em modo exclusivo. Este exemplo assume também que o usuário te-

Figura 4.6: Rack 201 do *cluster* GradeP. Permite a execução do Hadoop em modo de agendamento exclusivo [GPPD 2015]..

nha acesso a uma conta cadastrada no *cluster* GradeP. A seguir, será explicado o processo de instalação e configuração do Hadoop na GradeP.

O acesso ao computador central da GradeP deverá ser feito via sessão SSH através do comando *ssh erad2015@gradep.inf.ufrgs.br*. O processo de instalação do Hadoop inicia-se pela cópia do Hadoop 1.2.1 do servidor da Apache [Apache 2013c]. Em seguida o pacote do Hadoop deverá ser descompactado no diretório do usuário sob o nome hadoop. No total 4 arquivos deverão ser configurados. Três deles do tipo XML estão presentes no diretório (/hadoop/conf).O primeiro arquivo chamado de *core-site.xml* define o endereço do computador que irá executar o *namenode* [Apache 2013c]. Do conjunto de 12 máquinas da GradeP alocadas para executar o Hadoop, o computador *compute-0-0* é quem irá executar o *namenode*. O usuário deverá definir um campo XML de nome *fs.defaultFS* com valor *hdfs://compute-0-0:8000*. Ele indica que o *namenode* no *cluster* estará acessível e operacional no computador *compute-0-0* através da porta de rede 8000.

O usuário deverá criar um diretório de nome *hdfs* com permissões de leitura e escrita 750 internamente ao diretório do Hadoop. Este será o diretório onde o HDFS irá armazenar seus arquivos. É necessário que o usuário defina em *core-site.xml* um campo XML de nome *hadoop.tmp.dir* contendo como valor o caminho absoluto onde reside a pasta do HDFS [Apache 2013c]. No cenário da GradeP este caminho estará presente em (/hadoop/hdfs). O segundo arquivo XML chamado de *mapred-site.xml* define para o conjunto de 12 máquinas da GradeP alocadas para executar o Hadoop qual computador irá executar o *jobtracker*. No cenário da GradeP o computador *compute-0-0* é quem irá executar o *jobtracker*. O usuário deverá definir um campo XML de nome *mapred.job.tracker*

com valor *compute-0-0:8010*. Ele indica que o *jobtracker* no *cluster* estará acessível e operacional no computador *compute-0-0* através da porta de rede 8010.

O terceiro arquivo XML chamado *hdfs-site.xml* define o número de réplicas que deverão ser feitas para os arquivos armazenados no HDFS [Apache 2013c]. Os arquivos são persistidos em fragmentos menores chamados de blocos. As réplicas são feitas sob os blocos que constituem os arquivos. A replicação é necessária no sentido de tornar o Hadoop que irá executar na GradeP tolerante a falhas. É necessário que o usuário defina no arquivo *hdfs-site.xml* um campo XML de nome *dfs.replication* com valor maior ou igual a um. No cenário do *cluster* GradeP será utilizado valor 3. Este valor refere-se ao número de réplicas que serão feitas para cada bloco que compõe um arquivo.

É necessário ainda configurar o script *hadoop-env.sh* presente no mesmo diretório dos arquivos XML descritos anteriormente. O usuário deverá definir em *hadoop-env.sh* as variáveis de ambiente JAVA_HOME e HADOOP_PREFIX que devem possuir o diretório de instalação do Java 7 e do Hadoop [Apache 2013c]. No cenário da GradeP a variável JAVA_HOME deverá apontar para o diretório $/usr/java/latest/bin$ sendo este o caminho onde o Java 7 é instalado automaticamente pelo Rocks. A variável HADOOP_PREFIX deverá receber o caminho $/hadoop$ sendo este o diretório de instalação do Hadoop na GradeP.

Uma vez configurados os arquivos do Hadoop, o framework Hadoop deverá ser copiado para cada um dos 12 computadores que compõem o *cluster*. A GradeP possui uma aplicação chamada *tentakel* que possibilita que comandos sejam executados remotamente em outros computadores. Além disso, o *tentakel* permite que um conjunto arbitrário de computadores estejam vinculados em uma lista. Por padrão, o *tentakel* quando chamado executa o comando passado por parâmetro a todos os computadores presentes na lista. O *tentakel* configurado na GradeP possui em sua lista os 12 computadores que formam o *cluster* do Hadoop. O usuário deverá realizar a cópia automática do Hadoop para cada um dos 12 computadores do *cluster* através do comando *tentakel scp -r gradep:/home/erad2015/hadoop /hadoop*. Este comando copia o Hadoop configurado para o diretório $/hadoop$ dos 12 computadores que formam o *cluster* Hadoop da GradeP.

Uma última e importante configuração a ser realizada, refere-se à modificação do arquivo *slaves* presente na pasta do Hadoop $/hadoop$ do computador *compute-0-0*. Este arquivo define quais serão os computadores escravos do *cluster* Hadoop e serão responsáveis pela execução dos serviços *datanode* e *tasktracker*, responsáveis respectivamente pelo armazenamento dos arquivos do HDFS e pela execução dos processos *Map* e *Reduce*. O usuário deverá acessar via SSH o computador *compute-0-0* e adicionar ao arquivo *slaves* presente no diretório $/hadoop$ os 11 computadores que compõem o *cluster*. O nome de rede de cada computador deverá ser escrito em cada linha deste arquivo, a começar por *compute-0-1* até *compute-0-11*. O *compute-0-0* será o computador mestre responsável pela execução dos serviços *namenode* e *jobtracker*.

Antes de iniciar o Hadoop é necessário formatar o sistema de arquivos distribuído (HDFS), operação realizada apenas uma vez, durante a instalação. Todos os comandos de operação do Hadoop deverão ser executados sob o computador *compute-0-0* uma vez que este será o mestre ou o computador coordenador. Para formatar o HDFS o usuário deverá navegar até o diretório $/hadoop/bin$ e disparar o comando *hdfs namenode –format* [Apache 2013c]. Uma vez feito o HDFS estará formatado. O usuário então deverá navegar até o diretório $/hadoop/sbin$ onde o script *start-all.sh* deverá ser executado. Este script inicializa o Hadoop, o que significa que os serviços de *namenode* e *jobtracker* serão inicializados no computador *compute-0-0*. Da mesma forma, cada um dos 11 computado-

res escravos que formam o *cluster* terão seus serviços de *datanode* e *tasktracker* também inicializados.

A GradeP possui o serviço Ganglia, instalado nativamente pelo Rocks, que permite a monitoração dos recursos do *cluster* através de um servidor web [Rocks 2014]. O serviço pode ser acessado de forma distribuída por um navegador web. O Ganglia oferece diferentes formas de visualização do uso dos recursos que formam o *cluster*. A Figura 7 [Amazon 2009] ilustra a interface principal do Ganglia, acessível no cenário da GradeP pelo endereço de rede *gradep.inf.ufrgs.br/ganglia*. De maneira semelhante ao Ganglia o Hadoop que roda na GradeP permite a monitoração do *namenode* e do *jobtracker* que executam no computador *compute-0-0* através de serviços web. Estes serviços estão acessíveis através dos endereços *gradep.inf.ufrgs.br:8000* para o *namenode* e *gradep.inf.ufrgs.br:8010* para o *jobtracker*.

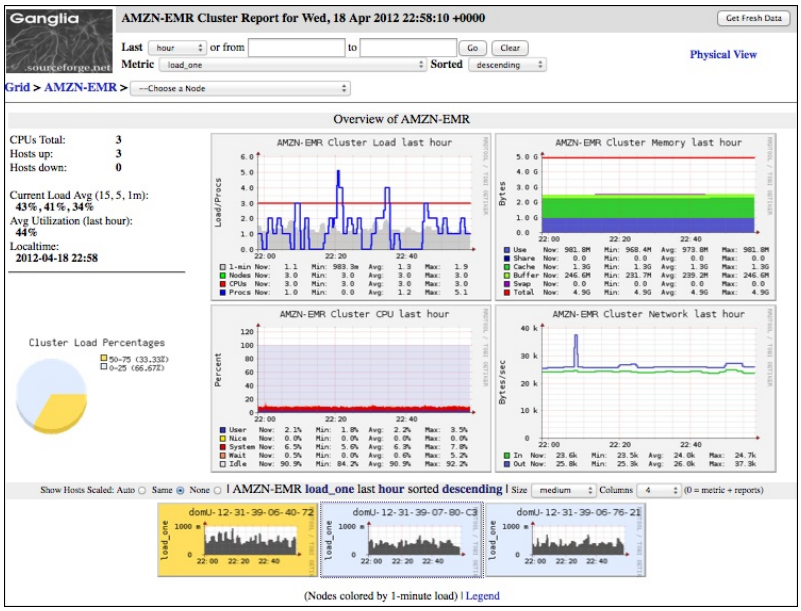

Figura 4.7: Sistema de monitoramento Ganglia para um *cluster* baseado na distribuição Rocks *cluster* [Amazon 2009]..

4.3.2. A Cloud Azure e o Hadoop

A Azure constitui-se de uma infraestrutura do tipo datacenter, que permite a criação e implementação de serviços de *cloud computing* em nível de plataforma ou infraestrutura [Microsoft 2013]. Este trabalho faz uso da Azure em nível de infraestrutura onde um conjunto de máquinas virtuais são instanciadas e configuradas para execução do Hadoop [Microsoft 2015b]. O processo de criação de instâncias de máquinas virtuais é feito de forma gráfica e intuitiva através do portal de gerenciamento da Azure. O portal permite durante a criação de cada máquina virtual que o usuário selecione o tipo de sistema operacional desejado além de um nome de usuário e senha padrão para a máquina virtual [Microsoft 2015a].

Este trabalho propõe uma configuração de *cloud* composta por um total de 12 máquinas virtuais do tipo A1 [Microsoft 2015b]. Para cada tipo de máquina virtual disponível, um número de recursos como processamento, memória ou armazenamento é

oferecido. Conforme maior for o número de recursos disponibilizados pela máquina virtual, maior será seu custo operacional, cobrado em reais por hora utilizada. Cada máquina A1 possui um núcleo de processamento de um processador Intel Xeon E5-2660 2.2 GHz, 1.75 GB de memória RAM e 70 GB de HDD [Microsoft 2015b]. No atual momento da escrita deste material, o custo de cada máquina virtual do tipo A1 é de $0,252$ centavos de real por hora utilizada [Microsoft 2015b]. O conjunto de 12 máquinas alocadas para este trabalho possui um custo total de R$3,02 por hora utilizada.

Em cada uma das 12 máquinas será instalado o sistema operacional Linux Ubuntu 14.04 LTS 64 bits [Microsoft 2015a]. A Figura 8 ilustra o processo de criação de uma das máquinas virtuais propostas por este trabalho. Conforme comentado, o processo de instanciação e criação das máquinas é gráfico e intuitivo. A instalação do sistema operacional é feita totalmente de forma automática pela Azure [Microsoft 2013]. Cada máquina possui por padrão um endereço IP associado a um nome de rede. Toda máquina virtual possui um servidor de SSH configurado, que permite que esta esteja acessível externamente ao datacenter da Azure. Neste trabalho foram instanciadas 12 máquinas virtuais seguindo a mesma nomenclatura utilizada pelos computadores da GradeP. Para facilitar a gerência, optou-se por utilizar um mesmo nome de usuário e senha da GradeP em todas as máquinas virtuais da Azure.

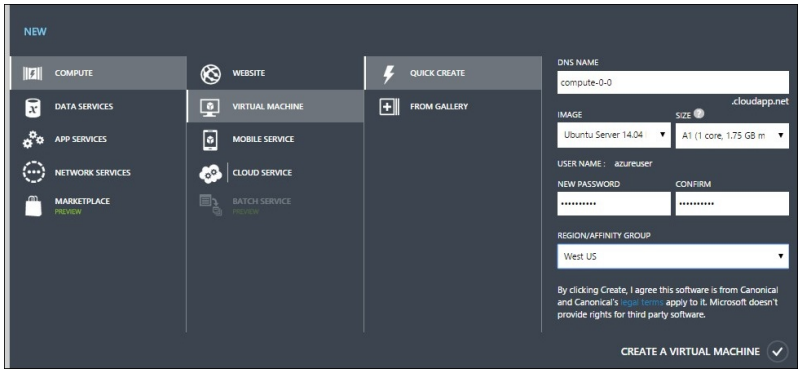

Figura 4.8: Criação de uma máquina virtual na *cloud* Microsoft Azure através do portal de gerenciamento da Azure..

Afim de facilitar a gerência e o acesso externo do conjunto de 12 máquinas que compõem a *cloud* alvo, será criado um serviço de rede que permite associar cada uma das máquinas da *cloud* a um único domínio de rede. O conjunto de todas as máquinas será representado pelo domínio de rede *erad2015hadoop*. Na Azure, todo domínio é acrescido do sufixo *.cloudapp.net*. Por conseqüente, o endereço absoluto do domínio criado é *erad2015hadoop.cloudapp.net*. Este trabalho faz uso de apenas o serviço de rede SSH para acesso externo das 12 máquinas virtuais que compõem a *cloud*. A *cloud* configurada para este trabalho pode ser acessada através do domínio de rede *erad2015hadoop.cloudapp.net* via acesso SSH.

O processo de instalação e configuração do Hadoop na Azure é semelhante ao realizado no *cluster* GradeP. A Azure se diferencia da GradeP uma vez que cada máquina virtual possui como sistema operacional uma instalação Linux Ubuntu versão 14.04 LTS. Também, por não possuir, a priori instalados e configurados conforme demanda o Hadoop, os serviços de Java 7 e o acesso SSH. O computador *compute-0-0* tem como responsabilidade a execução dos serviços referentes ao mestre do Hadoop devendo possuir acesso irrestrito via SSH aos demais computadores escravos da *cloud*. O processo que autoriza

o mestre para acessar as demais máquinas virtuais da *cloud* é feito através do uso dos comandos *ssh-keygen* e *ssh-copyid*. O primeiro deles cria uma chave SSH do mestre e o segundo realiza uma cópia da chave SSH para os computadores escravos. Uma vez realizado este comando o mestre passará a ter acesso ilimitado e irrestrito aos computadores escravos.

O programa *pssh* deverá ser instalado na máquina virtual mestre *compute-0-0* via gerenciador de repositório *apt-get*. Ele será utilizado no lugar do comando *tentakel* da GradeP para execução de comandos remotos nas máquinas virtuais da *cloud*. O usuário deverá criar um arquivo contendo o nome de rede das máquinas virtuais a começar por *compute-0-0* até a máquina *compute-0-11*. Este arquivo deverá ser passado por parâmetro durante a chamada do *pssh*. Através do *pssh* o Open JDK 7 deverá ser instalado nas máquinas virtuais via gerenciador de repositório *apt-get*. O diretório padrão do Open JDK é $/usr/bin$, que deve ser atribuído à variável de ambiente JAVA_HOME no script *hadoop-env.sh*. A cópia do Hadoop aos demais computadores deverá ser feita com o uso do *pssh* e não do *tentakel* como no caso da GradeP.

A Azure possui um portal de gerenciamento que permite a monitoração dos recursos referentes às máquinas virtuais, acessível externamente à *cloud* através de um navegador web. A Figura 9 ilustra o portal de gerenciamento da Azure para uma máquina virtual em execução. De maneira semelhante ao Ganglia o Hadoop que roda nas máquinas virtuais definidas na Azure permite a monitoração do *namenode* e do *jobtracker* que executam no computador *compute-0-0*. Estes serviços estão acessíveis via endereços de rede *erad2015hadoop.cloudapp.net:8000* para o *namenode* e *erad2015hadoop.cloudapp.net:8010* para o *jobtracker*.

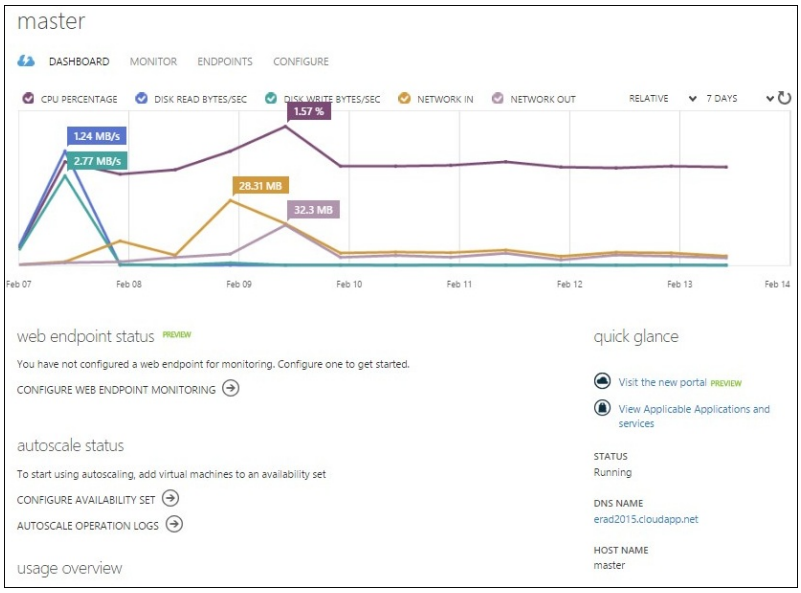

Figura 4.9: Sistema de monitoramento do portal de gerenciamento da *cloud* Microsoft Azure para uma máquina virtual..

4.4. Aplicação do Hadoop em Problemas Reais

Esta seção apresenta o Hadoop sob uma perspectiva prática e aplicada. Duas aplicações reais são propostas como estudo de caso. A primeira aplicação implementa um mecanismo que realiza a busca e a contagem de palavras em um conjunto oportuno de livros clássicos. A segunda aplicação faz uso do modelo MapReduce e implementa um mecanismo que realiza o alinhamento de sequências de DNA humano com objetivo de encontrar mutações em genes de amostras de pacientes.

4.4.1. Uso do Hadoop como Contador de Palavras

A contagem de palavras ou *wordcount* usando MapReduce é uma aplicação que explica didaticamente o funcionamento do Hadoop [Chen and Schlosser 2008]. Ainda, é um tipo de aplicação que consegue demonstrar a capacidade e escalabilidade do processamento distribuído suportado pelo Hadoop. A aplicação tem por objetivo realizar a contagem de palavras de um conjunto arbitrário de dados. O Hadoop possui como principal característica a capacidade de realizar o processamento de grandes quantidades de informação. A literatura apresenta exemplos práticos do Hadoop em que o mesmo é colocado para processar dados (contagem de palavras) da ordem dos Petabytes de informação [Chen and Schlosser 2008].

A arquitetura do código fonte da aplicação de contagem de palavras divide-se em cinco etapas, conforme ilustra a Figura 10 [Apache 2013b]. Na etapa (1) o usuário deverá manualmente realizar a cópia das palavras para o sistema de arquivos distribuído (HDFS). Em seguida, a aplicação MapReduce é então executada, em um total de 4 etapas. A etapa (2) executa tarefas do tipo *Map*, que realizam a leitura dos blocos de palavras do sistema de arquivos distribuído (HDFS) e geram um conjunto de dados do tipo (chave, valor). A chave refere-se à palavra e o valor que a palavra referida foi encontrada. Na etapa (3) é realizado o *Shuffle* que através de uma função do tipo *hash* calcula o *Reduce* destino de cada palavra processada pelo *Map*. Na etapa (4) o *Reduce* realiza a contagem do total de ocorrências para cada palavra. O *Reduce* gera como saída um par de elementos do tipo (chave, valor). A chave indica a palavra e o valor, a quantidade de ocorrências da palavra durante a execução do *Reduce*. Na etapa (5) os dados gerados são escritos novamente no sistema de arquivos distribuído (HDFS).

A operação do Hadoop para execução da aplicação de contagem de palavras é equivalente, tanto se tratando do *cluster* GradeP como da *cloud* Azure. Os comandos detalhados a seguir se aplicam para ambas as máquinas sem exceções. O mecanismo de acesso para ambas as máquinas deve ser feito via SSH. O endereço de rede para acesso ao computador principal do *cluster* GradeP é *gradep.inf.ufrgs.br*. A *cloud* Azure configurada exclusivamente para este minicurso é acessível através do endereço de rede *erad2015hadoop.cloudapp.net*. A partir do computador *compute-0-0* que deve ser acessado via SSH o Hadoop deverá estar em execução e o HDFS deverá estar formatado, conforme detalhado pela seção 3.1.

Inicialmente os dados devem ser copiados para um diretório do sistema de arquivos local do *compute-0-0* através do comandos de rede *wget*. Este trabalho irá utilizar como dados de entrada para executar o programa contador de palavras alguns livros clássicos da bibliografia de Shakespeare [Gutenberg 2011]. O sistema de arquivos distribuído (HDFS) deverá possuir um diretório para receber os dados. Para criar um

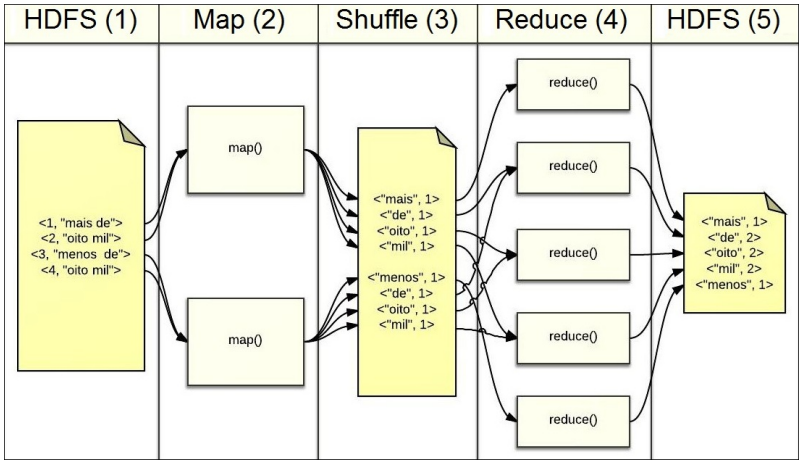

Figura 4.10: Arquitetura da aplicação *wordcount* que realiza a contagem de palavras [Apache 2013b]..

diretório o usuário deverá utilizar o comando *hadoop dfs –mkdir directory*. Os dados referentes aos livros de Shakespeare devem ser carregados para o diretório criado anteriormente no HDFS através do comando *hadoop dfs -copyFromLocal data directory* [Apache 2013c]. O usuário também deverá criar um diretório no HDFS onde os dados referentes à contagem das palavras deverão ser escritos pelo Hadoop.

O Hadoop por padrão possui um pacote com códigos Java MapReduce exemplos, sendo o *wordcount* um destes códigos. Para executar o código *wordcount* presente no pacote de exemplo o usuário deverá executar o comando *hadoop jar hadoop-examples-1.2.1.jar wordcount input output*. O resultado referente à contagem das palavras dos livros de Shakespeare presente no diretório *input* será escrito no diretório *output* do HDFS. Os dados presentes no diretório *output* poderão ser visualizados através do comando *hadoop dfs –cat /user/erad2015/output/part-r-00000* [Apache 2013c].

4.4.2. Sequenciamento e Busca de Mutações Genéticas em DNA Humano

O grupo GPPD da UFRGS em parceria com o Hospital de Clínicas de Porto Alegre (HCPA) possui um projeto de pesquisa e cooperação com objetivo de criar ferramentas computacionais capazes de realizar o processamento dos dados gerados por equipamentos de sequenciamento de segunda geração (NGS), como por exemplo o IonTorrent [Ion-Life 2015]. Em [Filho 2013] o autor Bruno Filho propôs como trabalho de graduação uma aplicação do tipo MapReduce para a realização de análise e sequenciamento de DNA humano.

Inicialmente amostras de pacientes sequenciadas pelo equipamento IonTorrent são armazenadas no sistema de arquivos distribuído (HDFS). A Figura 11 ilustra a arquitetura da aplicação MapReduce. Cada amostra irá possuir para um respectivo paciente um conjunto de um ou mais genes. A aplicação MapReduce realiza na atividade *Map* a busca no sistema de arquivos distribuído (HDFS) por genes. O *Map* realiza o sequenciamento dos genes lidos. O sequenciamento é feito com base na comparação de cada um dos nucleotídeos que formam o gene com sua referência. Cada computador do sistema distribuído irá possuir em sua memória local uma referência de cada gene consultado

[Filho 2013]. Esta referência é obtida originalmente do Genoma humano. O processo de sequenciamento usa um contador de códon que aponta para a posição de um nucleotídeo em específico onde foi encontrada uma variante, ou diferença em relação a referência. Para cada variante encontrada é gerada uma (chave, valor) sendo a chave o nucleotídeo e o valor a posição do códon onde foi encontrada a variante [Filho 2013].

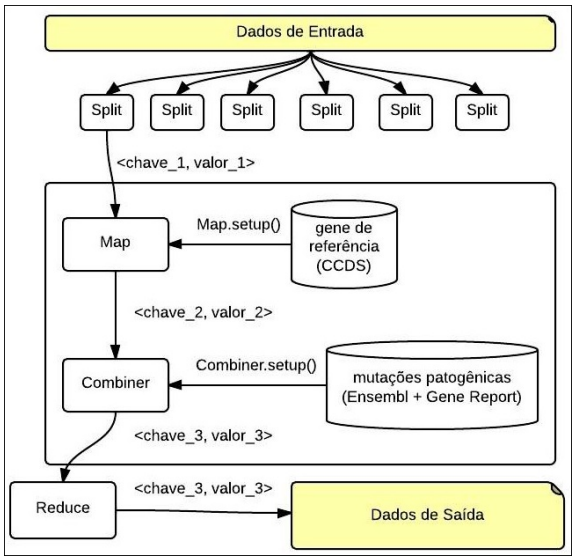

Figura 4.11: Arquitetura da aplicação MapReduce que realiza o sequenciamento (*Map*) e a busca por mutações (*Reduce*) em amostras de pacientes [Filho 2013]..

Cada variante gerada pelo *Map* é processada por uma tarefa *Reduce*. O *Reduce* têm como função consultar um conjunto de bases públicas como o Ensembl e o Gene Report e reportar se para uma variante encontrada existe na base consultada uma mutação associada [Filho 2013]. Cada computador do sistema distribuído irá possuir em sua memória local os dados referentes as mutações encontradas nas bases públicas de mutações para cada gene de interesse a ser sequenciado e consultado. Uma vez que o *Reduce* encontrar para um gene uma mutação em uma variante detectada pelo *Map*, ele deve escrever na saída do sistema de arquivos distribuído (HDFS) o tipo de patologia presente na mutação [Filho 2013]. A patologia indica o tipo de doença que o paciente em análise pode vir a ser propenso a desenvolver. Caso não for encontrada mutação para uma variante detectada, o *Reduce* deve apenas escrever no HDFS que encontrou durante o sequenciamento uma variante na amostra do paciente, mas que durante a busca nas bases públicas não encontrou mutação semelhante [Filho 2013].

A operação do Hadoop para execução da aplicação referente ao sequenciamento de DNA humano é equivalente, tanto se tratando do *cluster* GradeP como da *cloud* Azure. Os comandos detalhados a seguir se aplicam para ambas as máquinas sem exceções. Antes da execução do código MapReduce é necessário preparar os dados a serem escritos no HDFS. Estes dados referem-se aos genes das amostras dos pacientes, o Genoma de cada gene e os dados das bases públicas de mutações para cada gene. A aplicação possui três scripts Python responsáveis pela realização destas atividades. O usuário deve apenas configurar algumas variáveis de ambiente necessárias para execução dos scripts Python [Filho 2013].

Em seguida, o usuário deve escrever os dados gerados pelos scripts Python no sistema de arquivos distribuídos. O usuário deverá criar diretórios para cada um dos

respectivos dados no HDFS através do comando *hadoop dfs –mkdir directory*. Em seguida os dados devem ser carregados ao HDFS através do comando *hadoop dfs - copyFromLocal data directory*. Um diretório chamado *output* deverá ser criado pelo usuário. Os dados da saída da aplicação deverão ser escritos neste diretório. A execução do código MapReduce é feito com o comando *hadoop jar diseaseapp.jar org.Hadoop. DiseaseApplication input output* [Filho 2013]. O resultado referente à análise dos genes será escrito no diretório *output* do HDFS. Os dados presentes no diretório *output* poderão ser visualizados através do seguinte comando *hadoop dfs –cat /user/erad2015/output/part-r-00000*.

4.5. Conclusões

Este minicurso teve por objetivo apresentar o framework Hadoop sob uma perspectiva teórica e aplicada. O trabalho detalha a arquitetura de componentes do Hadoop, o sistema de arquivos distribuído (HDFS) e o modelo de programação MapReduce. Além disso, ele apresenta o processo de obtenção, instalação e configuração do Hadoop para o *cluster* GradeP e para a *cloud* Azure.

Com relação à aplicação prática do Hadoop são apresentadas duas aplicações com ênfase no processamento de dados reais, escritos no sistema de arquivos distribuído do Hadoop. O minicurso detalha também, questões específicas referentes ao código fonte das aplicações e ao funcionamento interno do Hadoop. Por fim, é demonstrado o processo de configuração e execução das aplicações para cada uma das máquinas distribuídas utilizadas.

Agradecimentos

Os autores deste minicurso agradecem ao GPPD da UFRGS por ceder o uso do *cluster* GradeP para realização da instalação, configuração e execução do Hadoop para os estudos de caso utilizados por este trabalho. Da mesma forma, agradecem também a Microsoft por disponibilizar o uso da *cloud* Azure para realização da instalação, configuração e execução dos estudos de caso utilizados por este trabalho.

4.6. Bibliografia

[Amazon 2009] Amazon (2009). Monitor Performance with Ganglia. Amazon Web Services, Inc.

[Apache 2013a] Apache (2013a). HDFS Architecture Guide. Dhruba Borthakur.

[Apache 2013b] Apache (2013b). MapReduce Tutorial - (WordCount v1.0). Apache.

[Apache 2013c] Apache (2013c). Overview of Apache Hadoop. Apache.

[Chen and Schlosser 2008] Chen, S. and Schlosser, S. W. (2008). Map-Reduce Meets Wider Varieties of Applications. Technical Report IRP-TR-08-05, Intel Research Pittsburgh.

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

[Dean and Ghemawat 2010] Dean, J. and Ghemawat, S. (2010). MapReduce: A Flexible Data Processing Tool. *Commun. ACM*, 53(1):72–77.

[Filho 2013] Filho, B. (2013). Aplicação do MapReduce na Análise de Mutações Genéticas de Pacientes.

[Ghemawat et al. 2003] Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA. ACM.

[GPPD 2015] GPPD (2015). Especificação e Documentação da GradeP. Grupo de Processamento Paralelo e Distribuído - GPPD/UFRGS.

[Gutenberg 2011] Gutenberg (2011). EBook of The Complete Works of William Shakespeare. Project Gutenberg.

[Ion-Life 2015] Ion-Life (2015). Ion Personal Genome Machine (PGM) System. Life Technologies.

[Mell and Grance 2011] Mell, P. M. and Grance, T. (2011). Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States.

[Menon 2013] Menon (2013). Introducing Hadoop - Part II. Rohit Menon.

[Microsoft 2013] Microsoft (2013). Microsoft Azure for Research Overview. Microsoft Corporation.

[Microsoft 2015a] Microsoft (2015a). Create a Virtual Machine Running Linux. Kathy Davies.

[Microsoft 2015b] Microsoft (2015b). Virtual Machine and Cloud Service Sizes for Azure. Microsoft Corporation.

[Rocks 2014] Rocks (2014). Rocks Base Users Guide - (6.1.1 Edition). University of California.

[Sterling 2001] Sterling, T. (2001). *Beowulf Cluster Computing with Linux*, volume 1. The MIT Press., 1rd edition.

[White 2012] White, T. (2012). *Hadoop - The Definitive Guide*, volume 1. OReilly Media, Inc., 3rd edition.