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

Modelo para Estimar Performance de um Cluster Hadoop

José Benedito de Souza Brito

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Pós-graduação em Computação Aplicada

Orientadora
Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo

Brasília
2014

Ficha catalográfica elaborada pela Biblioteca Central da Universidade de Brasília. Acervo 1017062.

Brito, José Benedito de Souza.
B862m Modelo para estimar performance de um Cluster Hadoop / José Benedito de Souza Brito. -- 2014.
xvii, 97 f. : il. ; 30 cm.

Dissertação (mestrado) - Universidade de Brasília, Instituto de Ciências Exatas, Departamento de Ciência da Computação, 2014.
Inclui bibliografia.
Orientação: Aletéia Patrícia Favacho de Araújo.

1. Cluster (Sistema de computador). 2. Computação de alto desempenho. 3. Processamento paralelo (Computadores). I. Paumgartten, Aletéia Patrícia Favacho de Araújo Von. II. Título.

CDU 004.272:004.75

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

Modelo para Estimar Performance de um Cluster Hadoop

José Benedito de Souza Brito

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Pós-graduação em Computação Aplicada

Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo (Orientadora)
CIC/UnB

Prof.ª Dr.ª Maria Clicia Stelling de Castro
IME/UERJ

Prof.ª Dr.ª Maristela Terto de Holanda
CIC/UnB

Prof. Dr. Marcelo Ladeira
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 09 de julho de 2014

Dedicatória

Dedicado à toda minha Família, pelo carinho, paciência e capacidade de me trazerem paz no transcorrer deste Mestrado.

Agradecimentos

Agradeço a todo o corpo docente do Mestrado Profissional em Computação Aplicada, em especial à Profª Drª Aletéia Patrícia Favacho de Araújo pelo esforço e dedicação na realização deste trabalho.

Resumo

O volume, a variedade e a velocidade dos dados apresenta um grande desafio para extrair informações úteis em tempo hábil, sem gerar grandes impactos nos demais processamentos existentes nas organizações, impulsionando a utilização de *clusters* para armazenamento e processamento, e a utilização de computação em nuvem. Este cenário é propício para o *Hadoop*, um *framework open source* escalável e eficiente, para a execução de cargas de trabalho sobre *Big Data*. Com o advento da computação em nuvem um *cluster* com o *framework Hadoop* pode ser alocado em minutos, todavia, garantir que o *Hadoop* tenha um desempenho satisfatório para realizar seus processamentos apresenta vários desafios, como as necessidades de ajustes das configurações do *Hadoop* às cargas de trabalho, alocar um *cluster* apenas com os recursos necessários para realizar determinados processamentos e definir os recursos necessários para realizar um processamento em um intervalo de tempo conhecido. Neste trabalho, foi proposta uma abordagem que busca otimizar o *framework Hadoop* para determinada carga de trabalho e estimar os recursos computacionais necessário para realizar um processamento em determinado intervalo de tempo. A abordagem proposta é baseada na coleta de informações, base de regras para ajustes de configurações do *Hadoop*, de acordo com a carga de trabalho, e simulações. A simplicidade e leveza do modelo permite que a solução seja adotada como um facilitador para superar os desafios apresentados pelo *Big Data*, e facilitar a definição inicial de um cluster para o *Hadoop*, mesmo por usuários com pouca experiência em TI. O modelo proposto trabalha com o *MapReduce* para definir os principais parâmetros de configuração e determinar recursos computacionais dos *hosts* do *cluster* para atender aos requisitos desejados de tempo de execução para determinada carga de trabalho.

Palavras-chave: Clusters, Modelo para Performance, Big Data, Hadoop e MapReduce

Abstract

The volume, variety and velocity of data presents a great challenge to extracting useful information in a timely manner, without causing impacts on other existing processes in organizations, promoting the use of clusters for storage and processing, and the use of cloud computing. This a good scenario for the Hadoop an open source framework scalable and efficient for running workloads on Big Data. With the advent of cloud computing one cluster with Hadoop framework can be allocated in minutes, however, ensure that the Hadoop has a good performance to accomplish their processing has several challenges, such as needs tweaking the settings of Hadoop for their workloads, allocate a cluster with the necessary resources to perform certain processes and define the resources required to perform processing in a known time interval. In this work, an approach that seeks to optimize the Hadoop for a given workload and estimate the computational resources required to realize a processing in a given time interval was proposed. The approach is based on collecting information, based rules for adjusting Hadoop settings for certain workload and simulations. The simplicity and lightness of the model allows the solution be adopted how a facilitator to overcome the challenges presented by Big Data, and facilitate the use of the Hadoop, even by users with little IT experience. The proposed model works with the MapReduce to define the main configuration parameters and determine the computational resources of nodes of cluster, to meet the desired runtime for a given workload requirements.

Keywords: Clusters, Performance Model, Big Data, Hadoop and MapReduce

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Equações

Lista de Abreviaturas e Siglas

Amazon EMR *Amazon Elastic MapReduce*. 5, 39, 65–67, 70, 71, 74, 75, 78

AWS *Amazon Web Services*. xii, 5, 39, 65, 66, 68, 70, 72, 75, 78

CPU *Central Process Unit*. 3, 30, 31, 35, 40–44, 47, 48, 50, 51, 53–56, 58, 59, 61, 64, 65, 69, 71, 72, 74, 77, 78

EMR *Amazon Elastic MapReduce*. 33

HCE$_m$ Modelo Estimador de *Cluster Hadoop*. ix, 5, 38, 40, 41, 44, 47, 64–66, 70–72, 74, 75, 87

HD *Hard Disc*. 3, 31, 40, 41, 47, 50, 54–56, 59, 65, 71, 72, 77

HDFS *Hadoop Distributed Filesystem*. 15

IaaS *Infrastructure-as-a-service*. 2, 11, 40

IBM *International Business Machines*, empresa de alta tecnologia. 2, 35

JCDF *Job-Centric Data Flow*. 34

MIPS *Million Instructions Per Second*. 2

MRBS *MapReduce Benchmark Suite*. 35

OLTP *Online Transaction Processing*. 1, 2, 16

PaaS *Plataform as a Service*. 11

RAM *Random Access Memory*. 3, 31, 32, 35, 40, 41, 47–51, 53–55, 65, 71, 72, 77

RCC *Relative Computacional Complexity*. 42

SaaS *Software as a Service*. 11

SO Sistema Operacional. 65

TI Tecnologia da Informação. vi, 2, 4, 38, 75

VM *Virtual Machines*. 30, 31

Capítulo 1

Introdução

O grande volume, a variedade e a velocidade dos dados, fenômeno conhecido como *big data* [89], continua sendo um grande desafio às empresas e à comunidade científica, mas também é uma fonte de pesquisa, de inovação, de diferencial competitivo e de aumento de produtividade, ou seja, uma fonte de oportunidades para os negócios e para a comunidade científica.

Nos ambientes, corporativos e científicos, existem cada vez mais dados, muitos dos quais semiestruturados ou não estruturados. Além disso, o ritmo do mercado competitivo, a pressão por redução de custos e por aumento de resultados geram uma crescente demanda para transformar essa avalanche de dados em informações úteis, no menor intervalo de tempo possível [28].

Para extrair informações úteis desses grandes volumes de dados, em tempo hábil e sem impactar o processamento de transações *on-line* OLTP (*Online Transaction Processing*) [73], usualmente utiliza-se o processamento *batch*, que é um tipo de execução de tarefas que ocorre sem a interação com o usuário final. Além disso, o processamento *batch* pode ser agendado para ser realizado em determinados horários, quando existirem recursos computacionais disponíveis para suportar esses processamentos [36].

Para *Ebbers et al.* (2011)[11], grandes empresas investem e confiam na plataforma computacional de *mainframes*, que são computadores que podem suportar milhares de aplicativos e dispositivos de entrada/saída para servir simultaneamente milhares de usuários [11], para o processamento de suas aplicações de tempo real (OLTP), principalmente em virtude da taxa anual de disponibilidade que é de 99,999%. Naturalmente grande parte dos processamentos *batch* também são executados nos *mainframes*, porém o custo é bastante elevado.

A IBM (*International Business Machines*) [54], uma das principais fornecedoras de *mainframes*, utiliza MIPS (*Million Instructions Per Second*) que é uma medida geral de desempenho de processadores, para definir a quantidade de trabalho realizada por um *mainframe*, e desta forma, medir o custo da computação. Assim, empresas consumidoras pagam um valor mensal de acordo com a quantidade de MIPS utilizadas nos *mainframes*.

Desta forma, executar o processamento em *batch* de grandes arquivos nos *mainframes* consome preciosos recursos computacionais e muitos MIPS. Como consequência, geram um alto custo financeiro e podem degradar o desempenho das aplicações de tempo real (OLTP), quando competem pelos mesmos recursos computacionais.

Neste contexto, parte desse processamento poderia migrar para uma plataforma de baixo custo como um *cluster*, que é um agrupamento de computadores visando o aumento de desempenho de aplicações distribuídas e paralelas [9], e utilizar um *framework open source* que possua um bom desempenho. Além disso, um *cluster* de computadores é uma solução rápida, simples e flexível de implementar para as organizações que possuem uma nuvem privada [57].

É notório que processar grandes bases de dados não é uma exclusividade de grandes empresas. O nicho das pequenas e médias empresas também possuem demandas, mas, em muitos casos essas organizações não possuem uma infraestrutura de TI (Tecnologia da Informação) adequada para realizarem esses processamentos. Casos semelhantes acontecem com algumas instituições de pesquisa. Nesses casos, essas organizações podem optar por contratar um serviço de computação em nuvem (IaaS - *Infrastructure-as-a-service*) [57], para realizar o processamento dos seus respectivos dados.

Para *Herotodos et al.* [32], com o advento das plataformas de computação em nuvem pode-se provisionar um *cluster* de qualquer tamanho nessa infraestrutura dentro de alguns minutos, processar suas bases de dados e pagar proporcionalmente apenas pelos recursos utilizados. Além disso, as plataformas de computação em nuvem permitem que usuários sem conhecimentos técnicos em TI façam o provisionamento do *cluster*, sem o suporte de administradores de sistemas.

Considerando os cenários acima expostos, o *framework Hadoop* é uma excelente solução para ser utilizado nos *clusters*, uma vez que permite um processamento escalável, confiável e distribuído de grandes conjuntos de dados através de *clusters* de computadores, e utilizando modelos de programação simples [14]. Além disso, o *Hadoop* é um projeto de software *open-source*, que pode ser adquirido sem custos, sendo mantido e disponibilizado pela *Apache* [22], uma organização reconhecida pelo mercado que promove a pesquisa e o desenvolvimento de projetos *open-source* desde 1999.

Entretanto, em se tratando de grandes organizações que possuem nuvens privadas, migrar processamentos de plataformas já consolidadas, como a plataforma de *mainframes*, apresentam custos com o desenvolvimento de novos softwares ou adaptações nos softwares existente. Além disso, muitas vezes não é possível determinar se os recursos disponíveis na nuvem privada poderão realizar o processamento nas janelas de tempo disponíveis.

Conforme *Herotodos et al.* [32], plataformas de nuvem fazem do *framework Hadoop* uma proposta atraente para as pequenas e médias organizações que precisam processar grandes conjuntos de dados, mas não possuem os recursos computacionais e humanos das grandes empresas. No entanto, é primordial saber qual a infraestrutura necessária, ou seja, a quantidade de nós e suas respectivas configurações (CPU, RAM, HD), deverá ser contratada para realizar o processamento em intervalos de tempo que atendam as necessidades dessas organizações.

Considerando as grandes, as médias ou as pequenas organizações, fica claro que um dos grandes desafios dos cenários acima é prever o tamanho do *cluster* necessário para realizar o processamento, para evitar custos com subalocação, quando são alocados recursos insuficientes para realizar o processamentos no tempos necessários, ou com superalocação, onde são alocados mais recursos que o necessário, e ficam ociosos.

Neste cenário, um *cluster* com o *framework Hadoop* é uma solução para as grandes, médias e pequenas organizações realizarem o processamento de grandes bases de dados. Todavia, utilizando nuvens privadas ou contratando serviços de computação em nuvem de um provedor, essas organizações serão frequentemente confrontadas com os desafios de definir o tamanho do *cluster*, bem como as configurações básicas (CPU, RAM, HD) dos nós desse *cluster*.

Assim sendo, este é o foco deste trabalho, que busca apresentar um modelo capaz de predizer o tamanho aproximado de um *cluster* para o *framework Hadoop*, para que ele seja capaz de executar um *job* e realizar determinada carga de processamento em um intervalo aproximado de tempo.

1.1 Justificativa

A cada dia as organizações possuem mais dados, logo extrair informações úteis e em tempo hábil torna-se mais difícil. São necessários mais investimentos em TI, seja para aumentar a capacidade de processamento, seja para adquirir novas soluções de softwares. Além disso, considerando especificamente o mercado de trabalho, existe a necessidade de melhoria contínua da eficiência e da redução de despesas no uso de recursos de TI, objetivos que podem ser alcançados por meio de soluções alternativas que não comprometam a qualidade das operações de TI.

Neste sentido, diversos pesquisadores [30, 32, 49, 50, 52, 74, 79, 88] apresentam o *cluster* com o *framework Hadoop* como uma solução para o processamento de grandes bases de dados. Todavia, para que o *Hadoop* seja eficiente é necessário fazer ajustes em seus parâmetros de configurações e dimensionar o *cluster* de acordo com a carga de trabalho a ser processada. Todavia, essas tarefas de configurações não são triviais.

Além disso, foi observado que existem diversas pesquisas visando otimizar o desempenho do *cluster Hadoop* e há outras que buscam prever o tempo médio do processamento em virtude de alterações na carga de trabalho a ser processada. No entanto, há poucos trabalhos que auxiliam no problema de definição inicial do tamanho do *cluster*.

Dessa forma, a principal justificativa deste projeto é auxiliar na problemática da definição inicial do tamanho de um *cluster* para a utilização do *framework Hadoop*, com foco em ambientes virtualizados, para o processamento *batch* de grandes bases de dados. No contexto desta pesquisa, grandes bases de dados são aquelas que possuem de centenas de *gigabytes* a alguns *terabytes* de tamanho.

Outra justificativa é a redução de custos de TI, sem comprometer a qualidade e tempestividade dos processamentos, pois o modelo proposto pode ser utilizado para definir quais os processamentos podem migrar de uma plataforma centralizada para um *cluster* utilizando o *framework Hadoop* com os mesmos índices de desempenho, fornecendo uma estratégia alternativa ao processamento *batch* de grandes arquivos.

Assim, este trabalho objetiva definir um modelo que seja capaz de estimar o tamanho aproximado de um *cluster* para o *framework Hadoop* executar um *job* e processar determinada carga de trabalho em dado intervalo de tempo.

Por fim, a realização deste trabalho justifica-se, entre outros motivos, pela necessidade de mecanismos que auxiliem as organizações a processarem bases de dados cada vez maiores, em face do crescente volume de dados, vivenciado em nossa sociedade moderna [9], em unidades de tempo cada vez menores e com recursos limitados. E o processamento distribuído, disponibilizado pelo *cluster Hadoop*, é capaz de superar as limitações encontradas nas arquiteturas computacionais centralizadas, como o processamento local [9] e os problemas de armazenamento e análise em um único disco rígido ou agrupamento [79], com alto desempenho e a custo reduzido.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é propor um modelo que seja capaz de estimar o tamanho aproximado de um *cluster* para o *framework Hadoop* executar um *job* e processar determinada carga de trabalho em dado intervalo de tempo, considerando a utilização de ambientes virtualizados.

1.2.2 Objetivos Específicos

São objetivos específicos deste trabalho:

- Analisar a viabilidade e o desempenho do *Hadoop* em ambiente virtualizado;
- Determinar e utilizar a ferramenta de *benchmark* que é mais adequada para realizar testes com o modelo proposto;
- Determinar um modelo para estimar o tempo de execução das *tasks* de um *job Hadoop MapReduce*;
- Propor um modelo de otimização leve para *jobs Hadoop MapReduce*;
- Implementar *job Hadoop MapReduce* e realizar testes com o modelo proposto utilizando um ambiente real e uma base de dados com 1 TB.

1.3 Organização do Trabalho

Além deste capítulo, este trabalho apresenta mais 5 capítulos, estruturados conforme a seguir. O Capítulo 2 apresenta um estudo contextualizado sobre computação distribuída, focando nas plataformas necessárias ao presente trabalho. O Capítulo 3 apresenta um estudo base para o *framework Hadoop*, a arquitetura do sistema de arquivos distribuídos do *Hadoop* e explanações sobre o *pipeline* de execução do *Hadoop MapReduce*. O Capítulo 4 apresenta uma discussão sobre as principais pesquisas relacionadas com este trabalho. O Capítulo 5 apresenta o *Modelo Estimador de Cluster Hadoop*, como uma solução para a problemática de definição inicial do tamanho de um *cluster*. O Capítulo 6 analisa o desempenho do HCE_m utilizando o serviço Amazon EMR da plataforma AWS. O Capítulo 7 conclui este trabalho, apresenta as principais contribuições e cita alguns trabalhos futuros. O Anexo A apresenta uma base de regras proposta para otimizar uma *task* (*map/reduce*). O Anexo B apresenta a codificação do *job MapReduce* utilizado para testar o modelo proposto neste trabalho.

Capítulo 2

Computação Distribuída

Neste capítulo são apresentados os conceitos de sistemas distribuídos, visando contextualizar o leitor com a base de conhecimentos aplicados para o desenvolvimento deste trabalho. Para isso, são apresentados conceitos de *grid*, de *clusters* e de computação em nuvem, que estão relacionados à pesquisa realizada neste trabalho.

2.1 Técnicas de Processamento de Dados

A sociedade atual está fortemente sedimentada no uso da informática, das redes de computadores, da internet e das redes sociais. Há diversos sistemas de computação inseridos em todos os segmentos da sociedade, desde a infraestrutura básica, como transporte, energia, comunicações, segurança, educação e saúde, a nichos econômicos específicos, como o comércio em geral, pesquisa e desenvolvimento, dentre outros.

Assim, é possível vislumbrar o mundo como uma complexa e multidimensional rede de computadores interligados, que tende a ser ainda mais interligado com o advento da internet das coisas [47], onde espera-se que a internet seja como a eletricidade e esteja em todas as coisas que são utilizadas no cotidiano das pessoas.

Dessa forma, impulsionadas pelo uso de computação intensiva, a cada dia as organizações geram e armazenam cada vez mais dados relacionados aos negócios, em um volume crescente e sem perspectivas de mudanças nesse cenário, que gera a necessidade de armazenamento e de processamentos de alta capacidade e larga escala, e, requerem a adoção de técnicas de processamento de dados eficientes e de sistemas de alto desempenho para atender as demandas de processamento.

Uma opção para melhorar o desempenho dos sistemas é a utilização de técnicas de "divisão e conquista", onde os problemas são separados em partes menores, visando a execução paralela dos processos e a redução do nível de complexidade dos sistemas, características presentes nos sistemas paralelos e nos sistemas distribuídos.

2.2 Computação Paralela

A utilização de múltiplos processadores foi a solução encontrada para suprir a necessidade de melhorar o desempenho dos sistemas, quando os limites físicos impostos pela tecnologia atual, começaram a impedir o aumento de desempenho contínuos dos processadores. A computação paralela permitiu aos sistemas a execução de tarefas de forma concorrente, onde cada processador pode executar uma tarefa [8].

Neste tipo de arquitetura ocorre o compartilhamento de vários recursos, como por exemplo a memória principal, o que contribui para um melhor desempenho. Além disso, os sistemas paralelos aumentam a segurança das operações, pois na falha de um dos processadores, os demais continuarão em execução, apesar do sistema ficar com um menor poder de processamento.

A computação paralela pode ser utilizada em todos os servidores atuais, inclusive nos servidores virtualizados, isso é importante, pois os servidores monoprocessados não atendem mais as demandas atuais de processamento.

2.3 Computação Distribuída

De forma resumida, pode-se definir a computação distribuída como um conjunto de máquinas conectadas por uma rede de comunicação funcionando como um único sistema. É uma linha de pesquisa clássica no mundo da computação abordada por importantes pesquisadores [8, 9, 72, 76, 78].

Para *Tanenbaum et al.* [78], um sistema distribuído é uma coleção de computadores independentes visto como um único sistema pelos usuários. Segundo *Coulouris et al.* [8], nos sistemas distribuídos os componentes de hardware ou software, localizados em computadores em rede, se comunicam e coordenam suas ações apenas pela passagem de mensagens.

Conforme *Tanenbaum et al.* [78], esses sistemas são formados por complexas partes de componentes de software que, por definição, são dispersos através de múltiplas máquinas, que podem ser heterogêneas. Desta forma, a implementação da comunicação é bastante complexa e são empregados protocolos de comunicação para que esses dispositivos heterogêneos possam se comunicar.

Segundo *Coulouris et al.* [8], os sistemas distribuídos devem tratar de problemas como a concorrência entre componentes, a falta de um relógio global, a heterogeneidade dos componentes, a tolerância a falhas, a segurança, o balanceamento de cargas, a estabilidade e a disponibilidade. Além disso, o principal motivo para a construção e a utilização desses sistemas são a sua capacidade de compartilhamento de recursos de processamento, memória e armazenamento, que podem reduzir o alto custo computacional e melhorar a disponibilidade e confiabilidade das informações e dos serviços contidos no sistema.

Dessa forma, é possível construir sistemas distribuídos robustos, eficientes e de baixo custo, se comparado a um modelo que utiliza máquinas robustas e com grande capacidade de processamento como no caso dos *mainframes* [11], a partir da conexão de vários servidores comuns, conhecidos como *commodities*.

Conforme *Coulouris et al.* [8], os sistemas distribuídos possuem uma camada denominada *middleware*, conforme ilustrado na Figura 2.1, que provê transparência ao sistema, fornecendo abstração de todas as informações a respeito da estrutura de um ambiente distribuído. Desta forma, mesmo um conjunto heterogêneo de máquinas pode compor um único sistema, independente das questões de localização, de hardware ou de software.

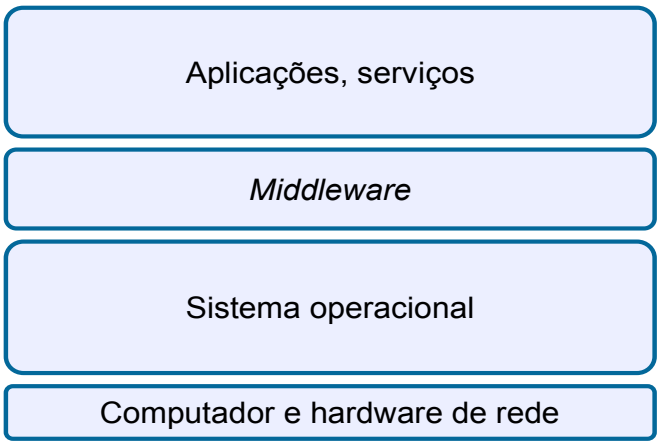

Figura 2.1: Camadas dos Sistemas Distribuídos, adaptado de [8].

2.3.1 Classificação dos Sistemas Distribuídos

Conforme *Stanoevska e Wosniak* [76], os sistemas distribuídos podem ser classificados de acordo com o escopo dos recursos que ele compartilha, sendo divididos em sistemas de *grid* e sistemas de *cluster*.

Para *Tanenbaum et al.* [78], *grids* são sistemas distribuídos construídos como uma federação de sistemas de computador, onde cada sistema pode estar geograficamente disperso e ser muito diferente em relação ao hardware, ao software e à tecnologia de rede.

Segundo *Coulouris et al.* [8], o termo *grid* também é utilizado para referir-se ao *middleware* que é projetado para permitir o compartilhamento de recursos, como arquivos, computadores, software, dados e sensores em uma escala muito grande. Esses recursos são compartilhados, normalmente, por grupos de usuários em diferentes organizações que estão colaborando na solução de problemas que exigem grande número de computadores para resolvê-los. Além disso, é necessário gerenciamento para coordenar a utilização dos recursos e garantir que os clientes obtenham os serviços que necessitam.

Na literatura, é possível encontrar várias definições de *cluster* ao longo do tempo. Por exemplo, para *Dantas* [9] *clusters* computacionais podem ser uma agregação de computadores para a execução de aplicações específicas de uma organização. Segundo *Tanenbaum et al.* [78], *cluster* é uma coleção de computadores similares, executando o mesmo sistema operacional e ligados por meio de uma rede local de alta velocidade. Conforme *Coulouris et al.* [8], *cluster* é um conjunto de computadores interconectados que cooperam estreitamente para fornecer uma capacidade única e integrada de computação de alto desempenho.

2.3.2 Arquitetura de Sistemas Distribuídos

Os sistemas distribuídos podem ser organizados de acordo com o posicionamento e o relacionamento de seus componentes. Os principais exemplos desta arquitetura são o modelo *cliente-servidor* e o *peer-to-peer* [8]. O modelo *cliente-servidor* é predominante, baseado em protocolo de requisição/resposta entre um cliente e um servidor para o compartilhamento de recursos. No modelo *peer-to-peer* todos os componentes realizam funções semelhantes na exploração dos recursos dos computadores [78].

Essas formas clássicas de organizar sistemas distribuídos apresentam problemas relacionados à troca de mensagens entre os seus componentes, relativos a eventuais atrasos nas redes de computadores. Uma vez que não é viável a utilização de um relógio global, foram utilizados métodos síncronos e assíncronos no sistema de tratamento das mensagens. Os sistemas síncronos utilizam limites conhecidos para o tempo de execução dos processos e para o tempo de entrega de mensagens, enquanto nos sistemas assíncronos não existe essa necessidade [76].

Computação em Nuvens

Segundo *Chen e Bairagi* [5], computação em nuvem é um modelo de arquitetura de sistema baseado na *internet* ou em uma *intranet* corporativa, que possui uma complexa infraestrutura técnica de interconexão de computadores.

Para *Coulouris et al.* [8], a computação em nuvem é um conjunto de aplicações baseadas na internet, que fornecem armazenamento e serviços de computação suficientes para suportar a maioria das necessidades dos usuários, permitindo-lhes, em grande parte ou totalmente, dispensar o uso de armazenamento de dados e softwares de aplicações locais. Além disso, promove uma visão de tudo como um serviço de infraestrutura física ou virtual através de softwares.

Segundo *Sabharwal e Wali* [65], a computação em nuvem é um modelo que permite conveniente, acesso sob demanda da rede para um grupo compartilhado de recursos computacionais configuráveis (como redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados com o mínimo esforço de gerenciamento ou interação com o provedor de serviços.

Mather et al. [57] apresenta uma definição de computação em nuvem baseada em cinco atributos:

- Compartilhamento de recursos: computação em nuvem é baseado em um modelo de negócios em que os recursos são compartilhados em nível de rede, de *hosts* e de aplicações;
- Escalabilidade massiva: computação em nuvem fornece a capacidade de escala a dezenas de milhares de sistemas, bem como a capacidade para massivamente escalar largura de banda e espaço de armazenamento;
- Elasticidade: os usuários podem rapidamente aumentar ou diminuir seus recursos de computação, conforme necessário, bem como liberar recursos para outros usos, quando eles não são mais necessários;
- Pagar conforme a utilização: os usuários pagam apenas pelos recursos que eles realmente usam e apenas pelo tempo que precisar deles;
- Autoaprovisionamento de recursos: os próprios usuários podem aproviosionar e alterar os recursos utilizados em uma nuvem.

Para *Sabharwal e Wali* [65], as nuvens podem ser classificadas, conforme seu modelo de implantação, de quatro formas distintas:

- Nuvem Pública: disponibilizada aos clientes por um fornecedor de serviços;
- Nuvem Privada: toda a infraestrutura é dedicada a apenas uma organização, essa nuvem é restrita à intranet dessa organização e não há compartilhamento com outras organizações;

- Nuvem Comunitária: é compartilhada por diversas organizações sendo concebida especificamente para satisfazer as exigências de uma determinada comunidade ou tipo de negócio;
- Nuvem Híbrida: formada por uma combinação de dois ou mais tipos de nuvens.

Além disso, vários autores [55, 63, 65] definem três modelos de serviço para as nuvens, conforme resumido abaixo, e ilustrado na Figura 2.2:

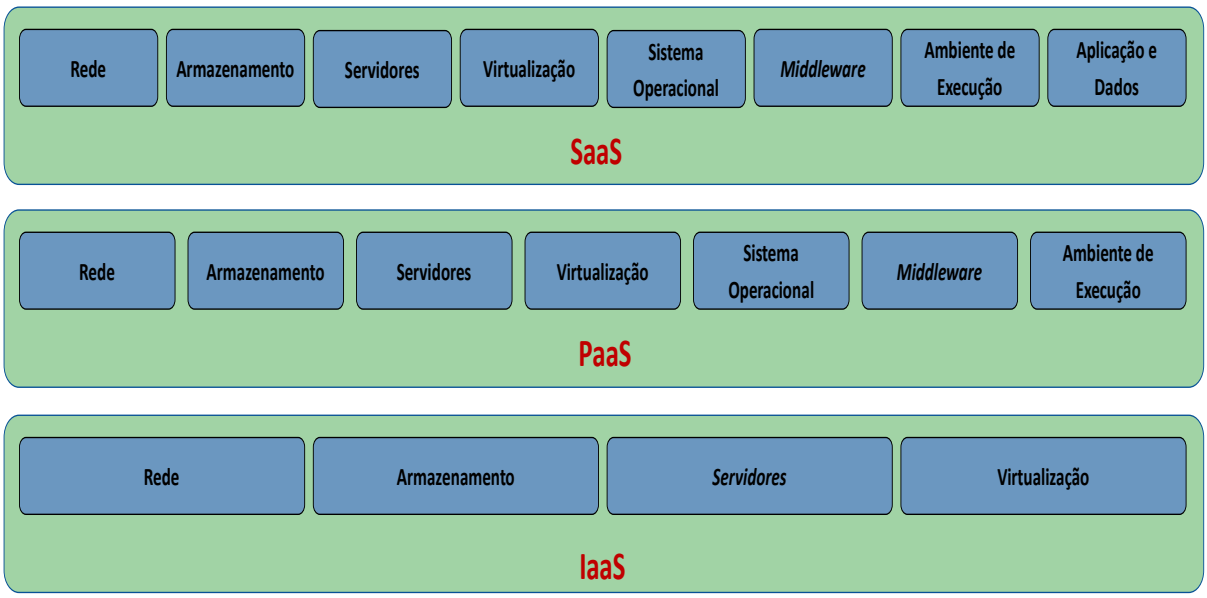

Figura 2.2: Modelos de Serviços Disponibilizados na Computação em Nuvem, adaptado de [65].

- IaaS (*Infrastructure as a Service*): neste modelo o provedor de serviços gere os centros de dados, equipamentos de redes, hardware, virtualização e camadas de automação para fornecer os recursos para os clientes. Os consumidores não gerenciam e nem controlam a infraestrutura de nuvem, mas possui controle sobre o sistema operacional, os aplicativos e os dados implantados;
- PaaS (*Plataform as a Service*): neste modelo é fornecido uma plataforma de software aos clientes para a construção e a implantação de aplicativos. Os consumidores não gerenciam e nem controlam a infraestrutura de nuvem, incluindo rede, servidores, sistemas operacionais, armazenamento ou a plataforma de aplicativos, mas possuem controle sobre os aplicativos implementados. PaaS fornece um meio poderoso para que os desenvolvedores de software criem aplicativos rapidamente, sem se preocuparem com os elementos de infraestrutura;

- SaaS (*Software as a Service*): neste modelo o provedor de serviços fornece aos consumidores aplicativos prontos para uso, rodando sob uma infraestrutura de nuvem. O consumidor não precisa se preocupar com infraestrutura básica como rede, servidores, sistemas operacionais, armazenamento, ou mesmo capacidades de aplicativos individuais.

Um ponto forte da computação em nuvem é a facilidade do uso dessa plataforma, onde tudo está acessível por meio de navegadores de internet ou emuladores de terminais, e pode ser acessado de qualquer lugar. Além disso, a computação em nuvem leva a computação para um novo formato de negócio, onde o cliente paga apenas pelos recursos utilizados, o chamado modelo *pay-per-use*.

2.4 Considerações Finais

Neste capítulo foram apresentadas algumas das principais características de sistemas distribuídos, necessárias ao contexto deste trabalho.

As principais definições de *sistemas distribuídos* sugerem que ele pode ser utilizado para melhorar o desempenho de tarefas de computação. Neste contexto, é importante a utilização de sistemas robustos e eficientes que se beneficiem dessa arquitetura, como o *framework Hadoop* que permite o processamento distribuído de grandes conjuntos de dados, por meio de *clusters* de computadores, assunto objeto no próximo capítulo.

Capítulo 3

Framework Hadoop

Neste capítulo são apresentados conceitos e aspectos importantes da arquitetura do *framework Hadoop*, que foi utilizado na solução proposta deste trabalho. Inicialmente foram abordadas as características mais relevantes dos dois principais subprojetos do *Hadoop*, o sistema de arquivos distribuídos (HDFS) e o modelo de processamento paralelo (*MapReduce*). Além disso, é abordado o fluxo de execução de um *job Hadoop MapReduce*.

3.1 Definições

No Mundo atual, altamente informatizado e conectado, a cada dia que passa as organizações geram e armazenam cada vez mais dados relacionados aos seus negócios, em um nível sem precedentes, gerando assim a necessidade de armazenamento e de processamento de alta capacidade.

De uma forma geral, a maioria das organizações que utiliza TI para apoiar suas atividades, possue grandes bases de dados sobre seus negócios, geradas ao longo dos anos. Analisar de forma eficaz esses dados permite uma melhor compreensão dos negócios em si, dos clientes, e, por consequência, é um diferencial para a alavancar novos negócios. Além disso, as organizações possuem a necessidade de gerir, de forma eficaz, seus recursos computacionais, aperfeiçoar procedimentos e reduzir custos, sem prejuízo do crescente número de interações de negócios e de suporte em TI. Em outra vertente, as instituições acadêmicas também possuem enormes bases de dados utilizadas nas pesquisas e desenvolvimento, e também necessitam de armazenamento e processamento de alta capacidade.

Em ambos os casos, a computação distribuída, através das plataformas de *clusters* de computadores, pode fornecer uma estrutura apropriada para atender as necessidades de armazenamento e de processamento das organizações e instituições acadêmicas.

As principais definições de *cluster* sugerem que ele seja utilizado para melhorar o desempenho de tarefas de computação. Além disso, geralmente, a computação em *cluster* é utilizada para executar programas paralelos, onde um único programa pode ser executado simultaneamente, em várias máquinas [78]. Neste cenário, o *framework Hadoop* é uma excelente plataforma de software a ser utilizada em um *cluster*.

O *Hadoop* é um projeto *open-source* da *Apache Foundation* [22] que visa proporcionar computação distribuída escalável e confiável. Conforme a Apache [14], o *Hadoop* é um *framework* que permite o processamento distribuído de grandes conjuntos de dados, por meio de *clusters* de computadores usando modelos de programação simples, podendo funcionar em um único servidor ou em até milhares de máquinas, cada uma disponibilizando armazenamento e processamento computacional local. Além disso, o *Hadoop* foi concebido para detectar e tratar falhas na camada de aplicação, fornecendo um serviço altamente disponível sob um conjunto de computadores, todos propensos a falhas [14].

O *framework Hadoop* é um projeto com alta maturidade e utilizado por grandes organizações, tais como a Amazon [1], o Facebook [37], o Google [38], a IBM [54], a Intel [6], a Oracle [7], o Yahoo [42] e muitas outras. Ele foi, inicialmente, projetado para usar servidores com hardware comum, reduzindo bastante o custo de construção do *cluster*. É uma tecnologia de código aberto desenvolvida através da linguagem de programação *Java* [60], o que permite sua instalação em muitos sistemas operacionais, sendo fácil de obter, distribuir e modificar.

Assim a maturidade, a utilização eficaz, o baixo custo e o fácil acesso tornam o *framework Hadoop* uma solução potencialmente estável para o armazenamento e de processamento de bases de dados de larga escala.

As características e as propriedades do *framework Hadoop* motivaram a sua escolha para esta pesquisa, que busca auxiliar no problema de dimensionamento inicial de um *cluster* executando o *Hadoop*, que neste trabalho foi denominado de *cluster Hadoop*, para realizar determinado processamento em um dado intervalo de tempo.

O *Hadoop* possui vários projetos relacionados, todos mantidos pela *Apache*, que fornecem diversos recursos para computação distribuída e processamentos de dados em larga escala. A seguir são apresentados alguns dos principais projetos.

- *Ambari*^{TM}: Uma ferramenta web para provisionamento, gerenciamento e monitoramento de *clusters Hadoop*, com suporte para a maioria dos projetos *Hadoop* [19].
- *Avro*^{TM}: Um sistema de serialização de dados [18].
- *Cassandra*^{TM}: Um gerenciador de bando de dados escalável, que possui múltiplos nós mestres e portanto não apresenta um ponto de falha único [20].

- *Chukwa*^{TM}: Um sistema de coleta de dados para o monitoramento de grandes sistemas distribuídos [25].
- *HBase*^{TM}: Um sistema gerenciador de banco escalável e distribuído, para o armazenamento de *big data* [13].
- Hive^{TM}: Um software de *data warehouse* para consultas e gerenciamento de grandes conjuntos de dados armazenados em sistemas de arquivos distribuídos [15].

  Mahout^{TM}: Uma biblioteca escalável para o aprendizado de máquinas e *data mining* [16].
- *Pig*^{TM}: Uma plataforma para análise de grandes bases de dados, que consiste de uma linguagem de alto nível para expressar programas visando a análise de dados [21].
- *Spark*^{TM}: Um mecanismos rápido e geral para o processamento de dados em larga escala [23].
- *ZooKeeper*^{TM}*:* Um serviço centralizado para manter informações de configurações, que proporciona sincronização distribuída e serviços de grupo [24].

Conforme *White*[79], o *framework Hadoop* é mais conhecido por seus projetos centrais, o *Hadoop MapReduce* e o seu sistema de arquivos distribuído, o HDFS (*Hadoop Distributed Filesystem*), que serão abordados nas próximas seções.

3.2 HDFS (*Hadoop Distributed File System*)

Para um melhor entendimento do funcionamento do Hadoop é necessário conhecer seu sistema de arquivos distribuídos. Segundo *Borthakur*[4], o HDFS é um sistema de arquivos distribuído, projetado para ser executado em servidores comuns. Para *White*[79], o HDFS é um sistema de arquivos distribuído destinado ao armazenamento de grandes arquivos (chegando a *terabytes*) com fluxos de acesso a dados padronizados, executado em *clusters* de servidores comuns.

De forma geral, os principais autores pesquisados [4, 71, 79] descrevem o HDFS como um sistema de arquivos tolerante a falhas, destinado a grandes arquivos e que suporta fluxo de dados. Além disso, o HDFS é similar aos sistemas de arquivos comuns, no tocante à organização hierárquica de arquivos, mas é mais adequado para aplicações que realizam poucas gravações e muitas leituras e que necessitem de portabilidade entre plataformas de hardware e software heterogêneos.

No HDFS, os arquivos são divididos em blocos e replicados em vários nós, e mesmo sendo executado em hardware comum e sujeito a falhas de máquinas, o HDFS disponibiliza replicação, detecção de falhas e recuperação de blocos de dados, automaticamente, mantendo a integridade do sistema de arquivos [4].

Apesar de suas qualidades, é importante destacar que o HDFS possui certas limitações. Por exemplo, uma vez que os blocos de dados dos arquivos estão segregados na rede, para o HDFS ter um bom desempenho é necessário que seus nós estejam conectados por uma rede com elevada largura de banda, por consequência, ele não é destinado a aplicações que necessitam de baixa latência no acesso a dados, como o processamento *online* de transações (OLTP). Outros fatores limitantes estão relacionados à arquitetura *mestre/escravo* que ele adota, pois o mestre contém os metadados de todo o sistema de arquivos em memória, ou seja, a quantidade de memória determina o número de arquivos que um *cluster Hadoop* pode ter, assim, não é recomendado manter vários arquivos pequenos no HDFS.

Conforme citado, o HDFS utiliza o padrão de arquitetura *mestre/escravo*, onde o *mestre* é denominado de *NameNode* e o *escravo* de *DataNode*. A Figura 3.1 apresenta uma visão geral da arquitetura do HDFS e seus principais componentes, que serão discutidos nas próximas seções.

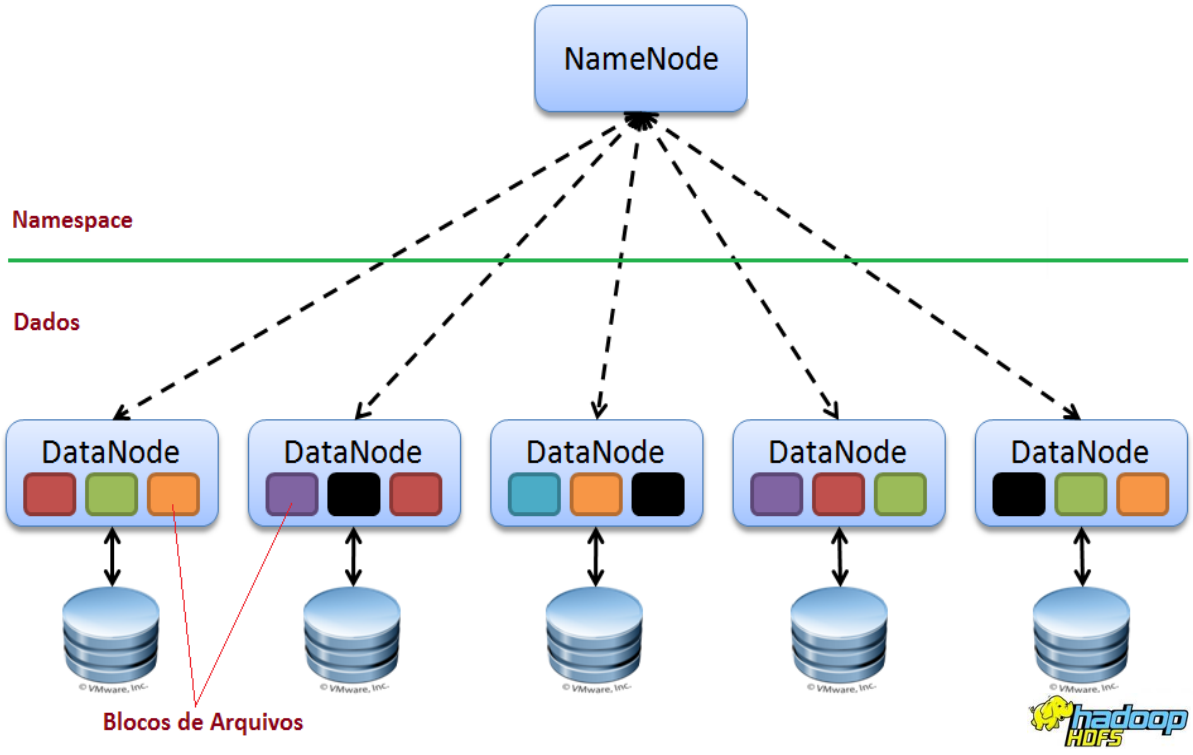

Figura 3.1: Arquitetura do HDFS, adaptado de Yoyoclouds [85].

3.2.1 *NameNode*

O *NameNode* é um serviço centralizador no *cluster Hadoop* que é responsável pelo gerenciamento de todo o sistema de arquivos distribuído do *Hadoop*. Além disso, o *NameNode* realiza ainda atividades como o balanceamento de carga dos *DataNodes*, o *garbage collection* e o atendimento às requisições dos clientes.

O *NameNode* gerencia a árvore do sistema de arquivos, os metadados para todos os arquivos e diretórios dessa árvore. Além disso, controla o mapeamento dos arquivos com seus respectivos blocos, a localização dos blocos que são armazenados nos *DataNodes*, dentre outros. O *NameNode* registra os *logs* das modificações que ocorrem no sistema de arquivos, para manter atualizada a representação dos metadados do sistema de arquivos mantidos em memória [79].

Segundo *White*[79], o *NameNode* armazena localmente as informações do sistema de arquivos e conhece os *DataNodes* onde os blocos de um determinado arquivo estão armazenados, mas essas localizações ficam na memória RAM e são reconstruídas sempre que o HDFS é iniciado.

O *NameNode* disponibiliza operações comuns de um sistema de arquivos, tais como listar, abrir, fechar, renomear e remover. No entanto, todas as operações realizadas diretamente com os arquivos são realizadas pelos *DataNodes*, a pedido do *NameNode*.

Conforme *White*[79], sem o *NameNode* o sistema de arquivos não pode ser utilizado, desta forma, se a máquina com o *NameNode* falhar todos os arquivos do HDFS podem ser perdidos. Por isso é importante configurar o *cluster Hadoop* resiliente a falhas no *NameNode*.

O HDFS possui mecanismos para evitar esse ponto de falha único, por meio da gravação de *backup* dos arquivos que contém os metadados de arquivos do sistema, ou por meio do uso do *HDFS Federation*, que particiona o *namespace* do HDFS por meio de múltiplos *NameNodes* [79] que gerenciam o mesmo conjunto de *DataNodes*, conforme ilustrado na Figura 3.2.

3.2.2 *DataNode*

De acordo com *Borthakur*[4], os *DataNodes* armazenam localmente os blocos dos arquivos, ou seja, são os servidores de armazenamento. Cada bloco é salvo como um arquivo separado no sistema de arquivos local do *DataNode*. Ele aceita requisições de leitura e/ou escrita de clientes locais ou remotos, a pedido do *NameNode*. Além disso, eles executam a criação, a exclusão e a replicação de blocos como parte das operações do HDFS.

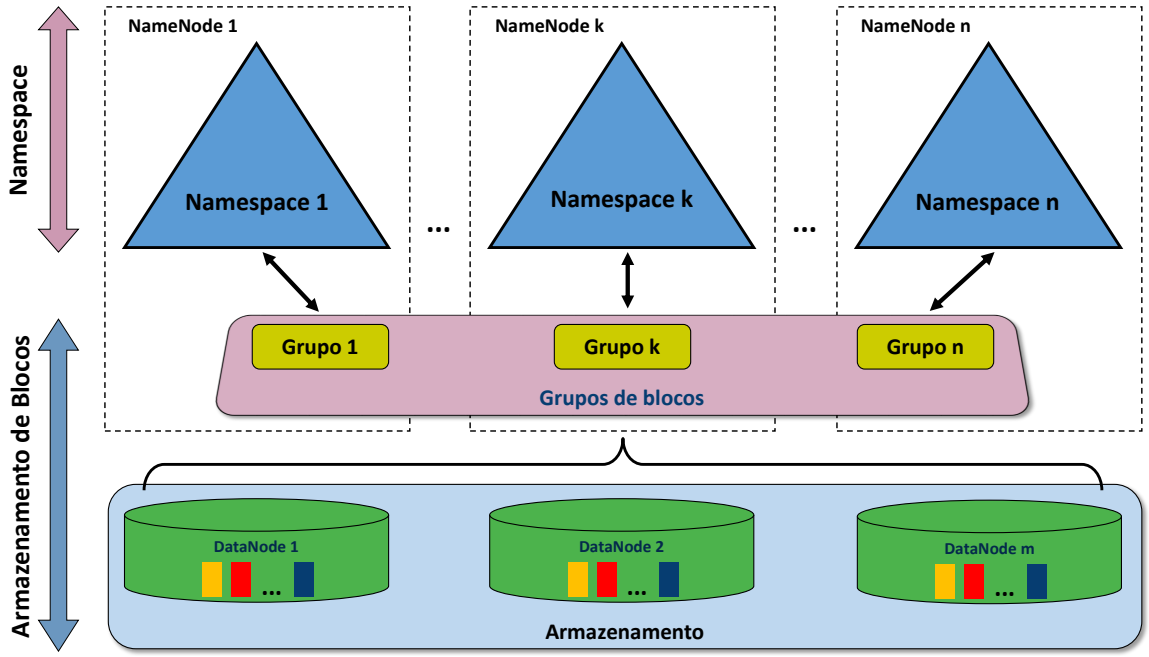

Figura 3.2: Múltiplos *NameNodes/Namespaces*, adaptado de *Apache* [14].

Para manter o sistema de arquivos atualizado, o *DataNode* relata, periodicamente, ao *NameNode* informações sobre todos os blocos de dados armazenados. Além disso, conforme citado no parágrafo anterior, os *DataNodes* necessitam de comunicação com outras instâncias para a replicação de dados. Desta forma, para que isso seja possível, eles periodicamente informam ao *NameNode* que estão ativos, por meio de mensagens. Quando o *NameNode* deixa de receber essas mensagens, que indicam que o nó ainda está ativo, de algum *DataNode*, ele marca o nó como inativo no *cluster* e solicita a replicação dos blocos de dados que estavam nesse *DataNode*, a partir de outros nós que também possuem os respectivos blocos.

3.2.3 Blocos de Arquivos

Conforme *Borthakur*[4], o HDFS foi projetado para suportar arquivos muito grandes, por isso, os aplicativos recomendados para o HDFS são aqueles que necessitam trabalhar com grandes conjunto de dados, que geralmente fazem uma gravação e muitas leituras desses dados, necessitando que essas leituras sejam realizadas em fluxo contínuo.

Para *White*[79], os discos rígidos possuem blocos, que representam a quantidade mínima de dados que podem ser lidos ou escritos. Nos sistemas de arquivos não distribuídos esses blocos, normalmente, possuem 512 *bytes*. O HDFS também utiliza o conceito de blocos, mas eles são muito maiores, 64 MB por padrão, visando mininar o custo de pesquisas dos dados para aproximadamente 1% do tempo de transferência de dados em redes de computadores.

Os dados no HDFS são divididos em pedaços menores, conhecidos como *chunks*, mas que podem ser traduzidos como blocos, e replicados no *cluster Hadoop*. Essa característica permite o acesso aos dados com altas taxas de transferência [4] e fornece condições ideais para aplicativos *Hadoop MapReduce*, apresentado na Seção 3.3. Além disso, permite que o processamento seja executado em subconjuntos menores dos dados e nos locais que esses blocos estão armazenados [89].

O tamanho atual de um bloco utilizado no HDFS é configurável, ou seja, durante a gravação de arquivos no HDFS, estes são divididos em blocos independentes, no tamanho configurado, e cada bloco é alocado em diferentes *DataNodes*, caso seja possível [4]. Todavia, ao contrário de um sistema de arquivos de disco único, um arquivo no HDFS que é menor que um único bloco, não ocupa o valor de um bloco cheio no sistema de armazenamento [79].

Para *White*[79], a abstração de blocos para um sistema de arquivos distribuídos possibilita gravar um arquivo maior que qualquer disco na rede. Além disso, ele simplifica o subsistema de armazenamento e funciona muito bem com o serviço de replicação para fornecer tolerância a falhas e disponibilidade.

3.2.4 *Namespace*

De forma simplificada, o *Namespace* funciona como um mapeamento para a localização dos dados no HDFS. Ele é formado por estrutura de árvore de diretórios do sistema de arquivos local, que documenta vários aspectos do HDFS, como os locais de blocos de arquivos, o fator de replicação dos arquivos, o balanceamento de carga, os direitos de acesso dos clientes às informações dos arquivos. Conforme *White*[79], o *NameNode* mantém o *Namespace* do sistema de arquivos, além disso, registra todas as mudanças que ocorrem no *Namespace*, ou em suas propriedades.

Para Borthakur (2008)[4], o *Namespace* é desacoplado dos dados, ou seja, reside apenas no *NameNode*, e armazena todos os metadados do sistema de arquivos, conforme demonstrado na Figura 3.3. Além disso, o *Namenode* mantém a imagem de todo o *Namespace* em memória. Um *backup* do *Namespace* é armazenado periodicamente no disco local do *NameNode*. Quando o *NameNode* for reinicializado, ele recolhe o *Namespace* do sistema de arquivos a partir da cópia.

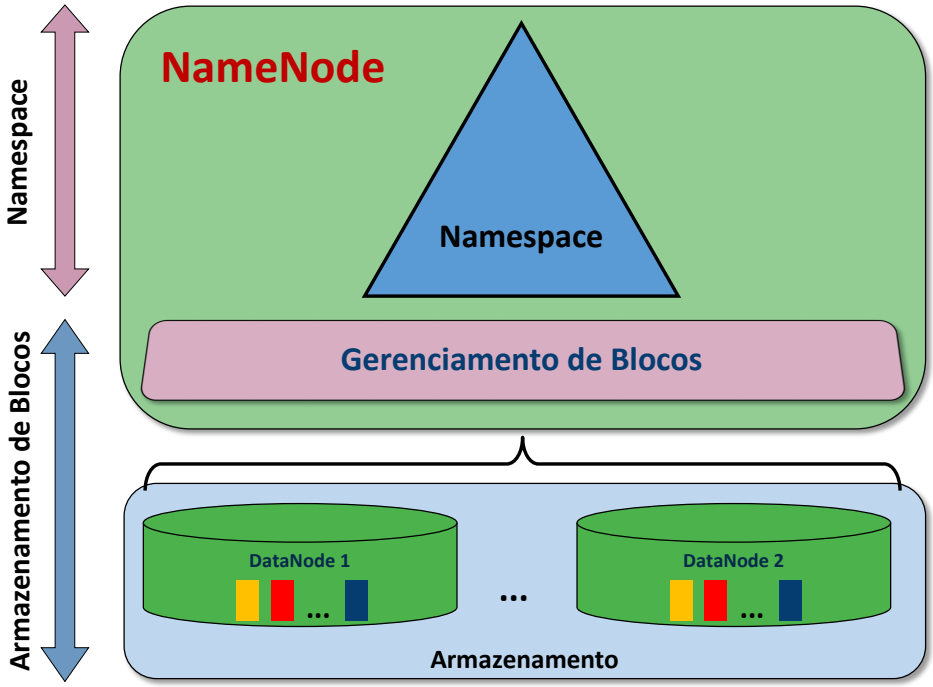

Figura 3.3: *Namespace* do HDFS, adaptado de *Srinivas* [75].

3.3 *Hadoop MapReduce*

O *MapReduce* é um paradigma computacional para processamento de dados distribuído em centenas de computadores, que foi introduzido por *Dean e Guemawat* [10] e popularizado por meio de grandes empresas de tecnologia como o Google[38], a IBM[54], a Intel[6], a Oracle[7], o FaceBook[37], o Yahoo[42] e a Apache Foundation[22], além de outras [59].

É possível encontrar diversas definições para o *MapReduce* na literatura especializada, mas apesar das diferentes abordagens, fica claro que ele é uma solução eficaz para o processamento de grandes bases de dados, conforme explanado nos próximos parágrafos.

Para *Zikopoulos et al.* [89], o *MapReduce* é um paradigma de programação que permite a escalabilidade através de centenas ou milhares de servidores em um *cluster*. Conforme *Lublinsky et al.* [53], o *MapReduce* é a solução para os problemas computacionais das grandes bases de dados, baseado nos princípios de divisão e conquista, onde os dados de entrada são divididos em pedaços e processados em paralelo.

Para *Miner e Shook* [59] o *MapReduce* é robusto, mas não fornece uma solução geral para o fenômeno conhecido como "*big data*", que refere-se as grandes massas de dados que não podem ser processadas ou analisadas por meio dos processos ou ferramentas tradicionais [89]. Segundo *Zikopoulos et al.* [89], *big data* é definido não apenas em função do volume, mas também da variedade e da velocidade necessárias para o processamento das grandes massas de dados, conforme ilustrado na Figura 3.4.

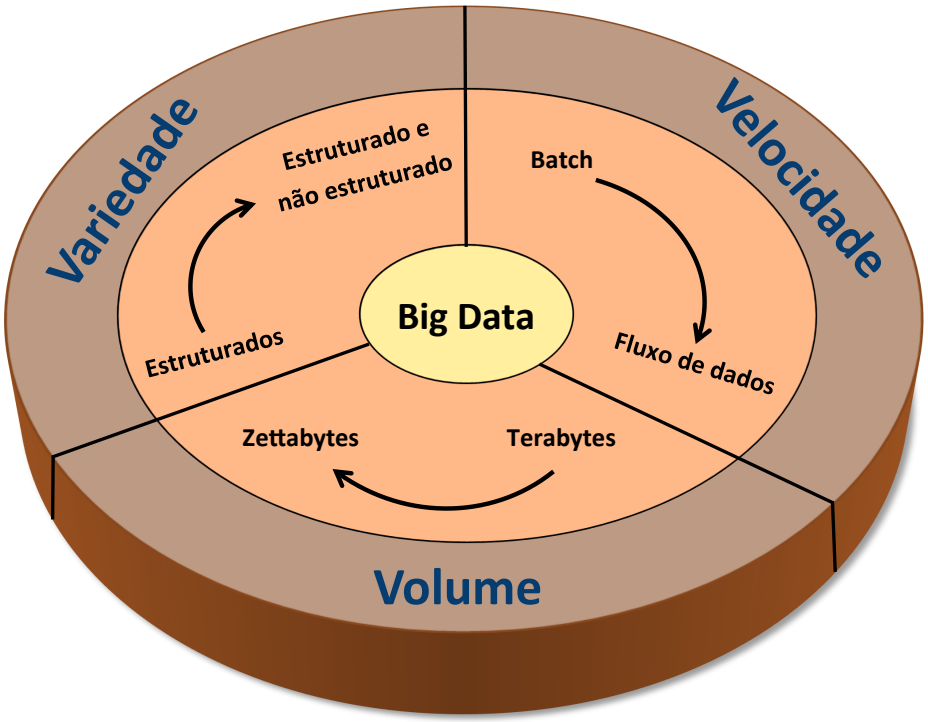

Figura 3.4: Caracterização de *big data*, adaptado de *Zikopoulos et al.* [89].

Conforme *Zikopoulos et al.* [89], a grande vantagem da estrutura de aplicação do *Hadoop MapReduce* é que o desenvolvedor não precisa lidar com os conceitos do *NameNode* e onde os dados estão armazenados, pois o *framework Hadoop* consultará o *NameNode* para saber quais os servidores possuem os blocos a serem utilizados no processamento e, em seguida, enviará o aplicativo *MapReduce* para ser executado localmente nesses nós.

Para *White*[79], no contexto do *Hadoop MapReduce* tem-se o *job*, que é a unidade de trabalho que o cliente quer realizar, contendo os dados de entrada, o programa *MapReduce* e as informações de configurações. Além disso, o *Hadoop* executa o *job* dividindo-o em tarefas, as quais são conhecidas como *MapTask* e *ReduceTask*.

O *Hadoop MapReduce*, de forma semelhante ao HDFS, também utiliza a arquitetura *mestre/escravo*, desta forma, possui dois processos principais, o *JobTracker* (*mestre*) e o *TaskTracker* (*escravo*).

3.3.1 *JobTracker*

Para *Sammer* [66], o *JobTracker* é o processo *mestre*, responsável por aceitar submissões de *jobs* a partir de clientes, agendamento de tarefas para executar nos *TaskTrackers*, e disponibilizar funções administrativas, tais como verificar os nós ativos, monitorar o progresso dos *jobs* no *cluster* e reiniciar tarefas que falharam.

Segundo *White*[79], o *JobTracker* coordena a execução de todos os *jobs* no sistema, através do agendamento de tarefas nos *TaskTrackers*. Simplificadamente, o *JobTracker* é o responsável pelo gerenciamento da execução dos programas *MapReduce*, procurando agendar a execução no *TaskTracker* que já possui os dados a serem processados.

Segundo *Sammer* [66], há um *JobTracker* por *cluster Hadoop* e ele necessita de um hardware confiável, pois caso ele falhe todos os *jobs* em execução também falharão.

3.3.2 *TaskTracker*

Conforme *Perera e Gunarathne*[61], os *TaskTrackers* são os *escravos*, que executam as *tasks* de um *job*. Para *Sammer*[66], os *TaskTrackers* aceitam as atribuições de tarefas dos *JobTrackers*, instanciam o código do usuário, executam as *tasks* localmente e, periodicamente, enviam relatórios de progresso das *tasks* ao *JobTracker*.

Segundo *White* [79], os *TaskTrackers* executam as tarefas e enviam relatórios de progresso ao *JobTracker*, que mantém um registro sobre a evolução geral de cada *job*. De forma geral, os *TaskTrackers* são responsáveis pela execução dos *jobs Hadoop MapReduce*.

Para *Apache* [17], geralmente os *TaskTrackers* são executados nos mesmos nós que os *DataNodes*, permitindo que o processamento ocorra onde os dados já estejam presentes, melhorando o desempenho como um todo.

Vale ressaltar que, conforme *Lublinsky et al.* [53], quaisquer dados armazenados no HDFS ou fora do *Hadoop* (por exemplo, em um banco de dados) pode ser usado como entrada para um *job Hadoop MapReduce*, além disso, o resultado do processamento também pode ser armazenado no HDFS ou fora do *Hadoop*. No entanto, no escopo desta pesquisa, considera-se apenas o HDFS como origem e destino dos dados.

3.4 Fluxo de Execução do *Job Hadoop MapReduce*

Segundo *Markey* [56], os *jobs Hadoop MapReduce* executam um conjunto de funções de programação simples nos valores de entrada e, finalmente, ordenam e apresentam os resultados. Portanto, o *MapReduce* trabalha dividindo o processamento em duas fases distintas, *map* e *reduce*.

Para *White*[79], cada fase (*map/reduce*) possui sua respectiva função (*map/ reduce*) e um par chave/valor como entrada e como saída. Conforme *Zikopoulos et al.* [89], o *Hadoop MapReduce* possui duas tarefas separadas e distintas que um *job* executa: a primeira é o *MapTask*, que recebe um conjunto de dados e converte-o em um outro conjunto de dados, onde os elementos individuais são discriminados em tuplas (pares chave/valor). A segunda tarefa é o *ReduceTask* que recebe as saídas dos *MapTask* como entrada e combina as tuplas de dados em um conjunto menor de tuplas.

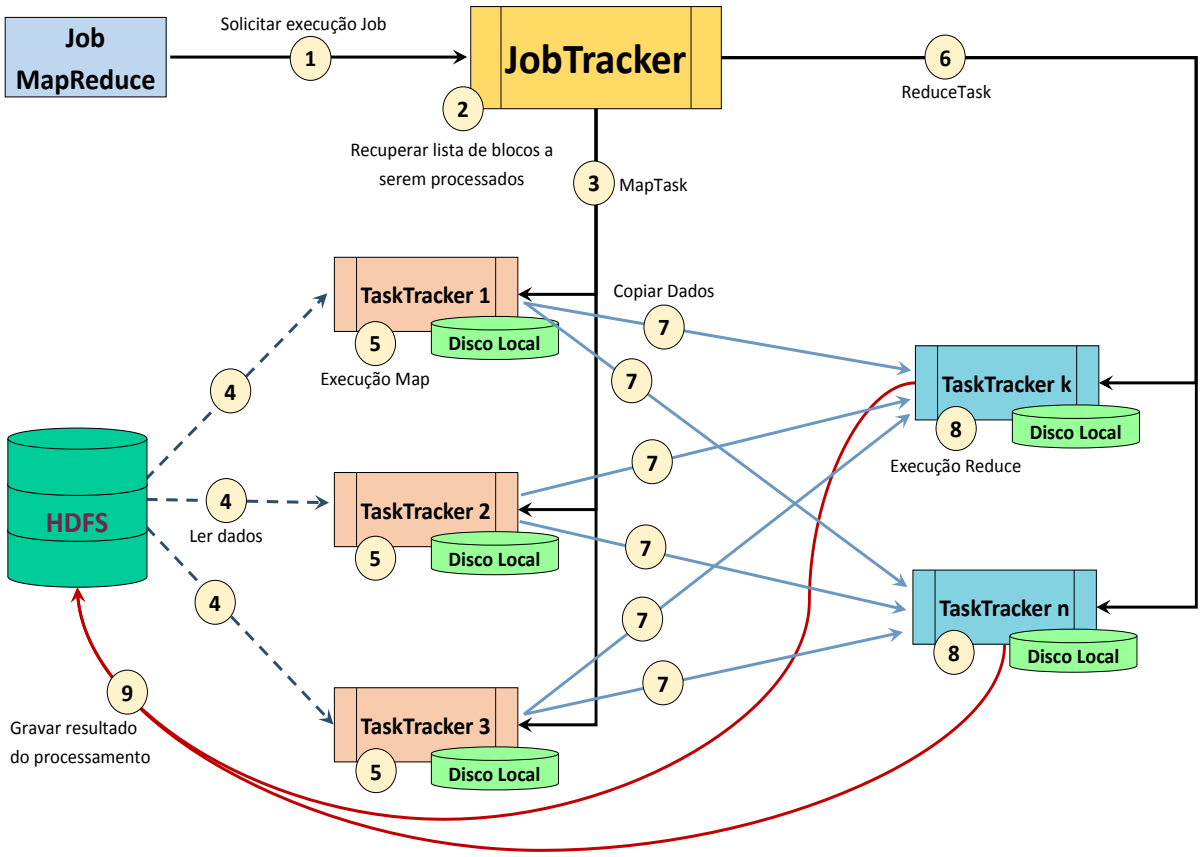

Figura 3.5: Fluxo de Execução de um *Job MapReduce*, baseado em *White*[79].

O fluxo de execução de um *job Hadoop MapReduce* possui muitos detalhes, mas considerando uma abstração de alto nível, pode-se sugerir que a execução de um *job Hadoop MapReduce* ocorra conforme ilustrado na Figura 3.5. Abaixo são descritos os fluxos apresentados na referida figura:

1. Submissão do *job Hadoop MapReduce* ao *JobTracker*;
2. O *JobTracker* recupera a lista dos blocos do arquivo de entrada, a partir do *NameNode*;
3. O *JobTracker* procura alocar os *MapTask* nos *TaskTracker* que possuem os blocos do arquivo de entrada;
4. Os *TaskTracker* recuperam o(s) respectivo(s) bloco(s) no HDFS. Essa entrada de dados é conhecida como *input split*;
5. Os *TaskTrackers* realizam o processamento dos *MapTask* localmente, além disso, gravam os resultados intermediários no disco local e informam o *JobTracker* da conclusão do respectivo processamento;

6. O *JobTracker* aloca as *ReduceTask* em *TaskTracker* disponíveis;
7. Os *TaskTracker* copiam os resultados do processamento dos *MapTask* dos respectivos *TaskTracker* que realizaram esses processamentos;
8. Os *TaskTracker* realizam o processamento dos *ReduceTask* localmente;
9. Os *TaskTracker* gravam os resultados do processamento no HDFS e informam ao *JobTracker* a conclusão do referido processamento.

De forma geral, a execução de um *job Hadoop MapReduce* ocorre conforme ilustrado na Figura 3.5, mas internamente cada fase (*map/reduce*) executa vários passos para realizar seus respectivos processamentos. Nas próximas seções serão discutidas as execuções das tarefas *Map* e *Reduce*.

3.4.1 *MapTask*

Para *Zikopoulos et al.* [89], os *jobs Hadoop MapReduce* que são executados nativamente no *Hadoop* são escritos em *Java*, os quais são distribuídos pelo *JobTracker* para todos os *TaskTrackers* do *cluster Hadoop* para que as *MapTasks* e *ReduceTasks* sejam executadas. Entretanto, o *Hadoop MapReduce* pode executar *jobs MapReduce* escritos em várias linguagens de programação. A Figura 3.6 ilustra, em alto nível, a arquitetura de execução dos *MapTasks* e, em seguida, são apresentados os principais componentes do *pipeline* dessa execução.

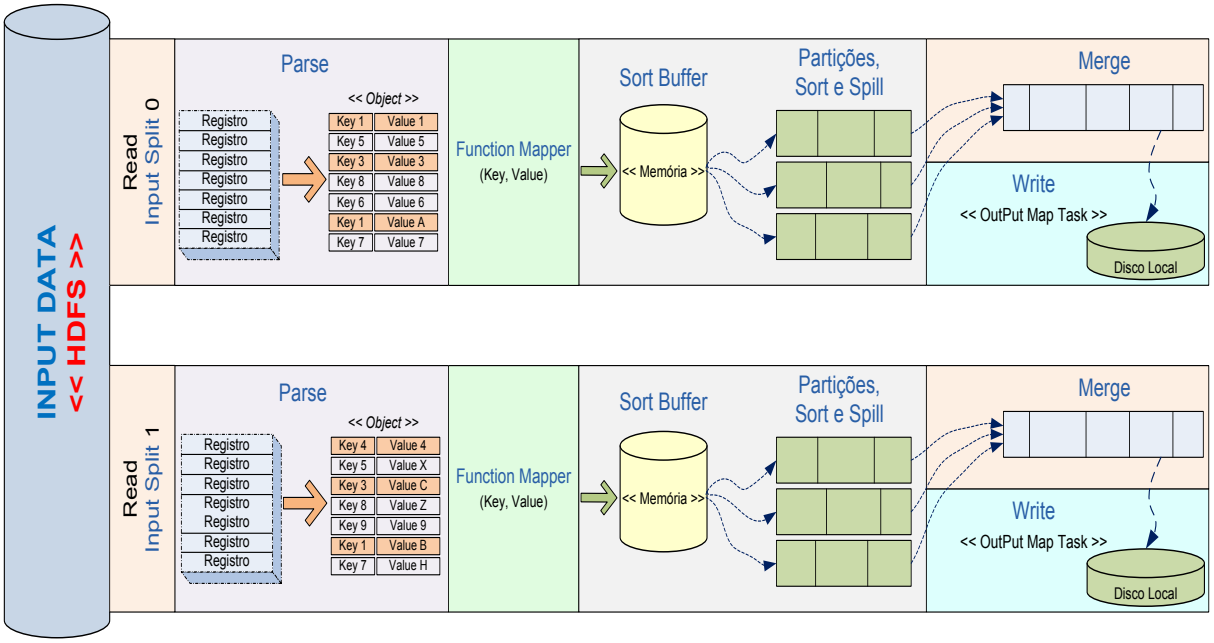

Figura 3.6: *Pipeline* de execução do *MapTask*, baseado em *Markey* [56].

Input Data

Representa os dados a serem processados por um *job Hadoop MapReduce*, podendo ser formado por um ou mais arquivo. No contexto desta pesquisa, considera-se que esses dados estejam localizados no HDFS do *cluster Hadoop* e que possuam algumas dezenas de *gigabytes* ou mais.

Input Split

Representa os dados de entrada a serem processados por cada *MapTask*. Segundo *Lublinsky et al.* [53], o *Hadoop MapReduce* divide a entrada a ser processada em blocos, chamados de *input split*, e para cada *split* é criado um *MapTask*, ou seja, cada *split* será processado em paralelo, caso exista no *cluster Hadoop* disponibilidade para a criação de vários *MapTask*.

Uma vez que os *input splits* são calculados, o *Hadoop MapReduce* define o número de *MapTasks* que devem ser executados e quais as respectivas localizações desejáveis. Além disso, também define o número de *TaskReduces* que serão utilizadas.

Na maioria dos casos, o tamanho do *input split* é semelhante ao tamanho de um bloco de arquivo do HDFS, desta forma, o *JobTracker* vai, preferencialmente, executar os *MapTasks* onde os *input split* estão localmente armazenados para que o processamento seja local, sem a necessidade de copiar os dados de outro *DataNode* por meio da rede.

Parse

Os *input split* são formados por vários registros, que o *framework MapReduce* vai ler e converter para objetos no formato *key/value*, que serão encaminhados para processamento pela função *Mapper*.

Function Mapper

Representa a implementação da classe *Mapper*, do *job Hadoop MapReduce*, desenvolvido pelo usuário. Essa função recebe como entrada um registro do *input split*, convertido em objeto no formato *key/value*. Essa função será executada consecutivamente para cada registro existente no *input split*. De forma geral, a *Function Mapper* recebe um objeto (*key/value*) como entrada e produz outro objeto (*key/value*) como saída.

Sort Buffer, Partições, Sort e Spill

De acordo com *Lublinsky et al.* [53], os *MapTask* definem um intervalo de chaves dos objetos (*key/value*) de saída para cada *TaskReduce*, esses subconjuntos são conhecidos como partições e serão as entradas de dados para as *TaskReduce*.

Para *White*[79], cada *MapTask* possui uma memória circular, de tamanho pré-determinado, chamada de *Sort Buffer* onde são escritos os objetos de saída do processamento da *Function Mapper*. Sempre que os dados armazenados atingem um certo limiar eles são salvos em um arquivo no disco local, processo conhecido como *spill*. Entretando, antes do *Hadoop MapReduce* gravar os objetos no disco local eles serão divididos em partições, de acordo com as respectivas chaves desses objetos, correspondendo às *TaskReduce* para os quais os intervalos de chaves serão encaminhados.

O processo *spill* será executado sempre que o *sort buffer* atingir o seu limiar e poderá gerar vários arquivos. Além disso, esse processo também ocorre quando a *Function Mapper* conclui o processamento de todos os registros, ainda que os dados armazenados não tenham atingido o limiar do *sort buffer*.

Merge

Após a conclusão do processamento de todos os registros do *input split* pela *Function Mapper* o *Hadoop MapReduce*, vai unificar todos os arquivos gerados pelos processos *spill* em um único arquivo de saída, que será salvo no disco local, ordenando-o e particionando-o. Este processo é conhecido como *merge*. Após o *merge* o *MapTask* é finalizado. Além disso, o *JobTracker* é notificado por meio de uma mensagem, encaminhada pelo *JobTracker* do respectivo *MapTask*.

3.4.2 *ReduceTask*

Os arquivos de saída de cada *MapTask* são gravados no disco local das máquinas onde foram executados, e as *ReduceTask* precisam copiar suas respectivas partições de cada *MapTask* do *cluster Hadoop*. Para *White*[79], uma vez que os *MapTask* podem ser concluídos em tempos diferentes, as *ReduceTasks* iniciam as cópias de suas partições assim que cada *MapTask* é concluído. A Figura 3.7 ilustra, em alto nível, a arquitetura de execução dos *ReduceTasks*, apresentando os principais componentes do *pipeline* dessa execução, os quais são descritos a seguir.

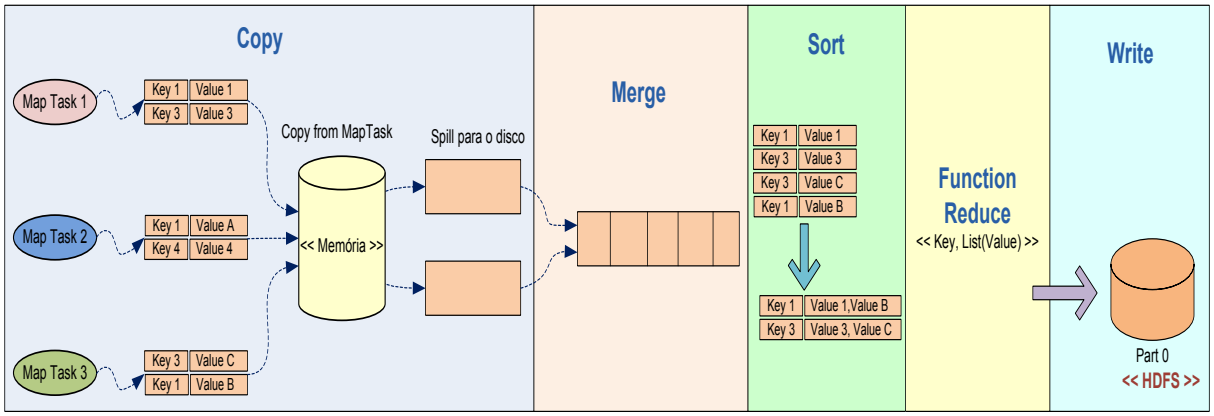

Figura 3.7: *Pipeline* de Execução do *ReduceTask*, baseado em *Markey* [56].

Copy

Conforme as *MapTask* vão sendo concluídas, os *ReduceTasks* copiam suas respectivas partições de dados, através da rede. Conforme *White*[79], as saídas das *MapTasks* são copiadas para uma área de memória reservada para a *ReduceTask*. Sempre que essa memória atingir um certo limiar, os dados serão gravados em um arquivo no disco local (*spill*). Vale destacar que todos os registros de uma determinada *chave* serão copiados apenas para um determinado *ReduceTask*, pois todas essas chaves foram previamente gravadas nas respectivas partições desse *ReduceTask*, independente do *MapTask* de origem.

Merge

Após as saídas de todos os *MapTasks* serem copiadas para o *ReduceTask* o *Hadoop MapReduce* vai unificar todos os arquivos gerados durante a cópia em um único arquivo no disco local.

Sort

Conforme *White*[79], o *Hadoop MapReduce* vai unificando os arquivos e ordenando os registros em rodadas por um número pré-determinado de arquivos, até que todos os arquivos gerados no *spill* sejam unificados. O *sort* gera um registro para cada chave, contendo todos os seus respectivos valores.

Function Reducer

Representa a implementação da classe *Reducer*, do *job Hadoop MapReduce*, desenvolvido pelo usuário. Essa função recebe como entrada cada registro do arquivo gerado durante o *merge/sort*, convertido em objeto no formato *key/List<value>*.

Write

Para *White*[79], a saída da *Function Reducer* é escrita diretamente para o sistema de arquivos de saída. Uma vez que geralmente os *DataNodes* e os *TaskTracker* são executados no mesmo *host*, o primeiro bloco de arquivos da saída do *ReduceTask* será gravado localmente e as demais réplicas seguem o *pipeline* normal do HDFS.

Após a conclusão do *write*, o *ReduceTask* é finalizado. Após todos os *ReduceTasks* serem concluídos o *Hadoop MapReduce* finaliza a execução do *job*.

3.4.3 Considerações Finais

Neste capítulo foram apresentadas as principais características do *framework Hadoop*, seus projetos centrais, o HDFS e o *MapReduce*. Além disso, foi apresentado um estudo sobre o *pipeline* de execução de um *job Hadoop MapReduce*, em níveis de detalhes necessários a este trabalho.

Dessa forma, percebe-se que executar o *Hadoop* em uma plataforma de sistemas distribuídos, como o *cluster* de computadores, é uma solução adequada para o armazenamento e o processamento de grandes bases de dados. Todavia, utilizar um *cluster Hadoop* apresenta alguns desafios, como a definição inicial do tamanho do *cluster* necessário para realizar determinado processamento em um tempo conhecido. Assim sendo, o objeto deste trabalho é definir um modelo que seja capaz de estimar o *cluster* adequado para *framework Hadoop* executar um *job MapReduce* para processar determinada carga de trabalho em um dado intervalo de tempo.

Capítulo 4

Trabalhos Relacionados

Este capítulo apresenta alguns trabalhos correlacionados com a pesquisa realizada nesta dissertação.

Existem diversas pesquisas abordando as mais variadas características do *Cluster Hadoop* que são importantes para este trabalho, como a sua utilização em ambientes totalmente virtualizados, a otimização do seu desempenho, a avaliação do *cluster Hadoop* através de técnicas de *benchmark* e pesquisas que apresentam modelos de desempenho.

Desta forma, para apresentar uma breve análise dos trabalhos relacionados, este capítulo foi dividido em quatro seções: *Hadoop* em Ambiente Virtualizado, Otimização do Desempenho do *Cluster Hadoop*, *Benchmarking* do *Cluster Hadoop* e Modelos de Performance do *Cluster Hadoop*.

4.1 *Cluster Hadoop* em Ambiente Virtualizado

A virtualização está em pleno desenvolvimento, cada vez mais presente nas grandes e médias organizações. No escopo desta pesquisa a avaliação do *Cluster Hadoop* ocorre em ambiente virtualizado, pois montar um *cluster* com dezenas, centenas ou até milhares de máquinas físicas não é prático e nem econômico.

Em princípio, empiricamente, pode-se supor que a sobrecarga da camada de virtualização, como os processos que controlam a comunicação da rede, I/O, memória e alocação de espaço em disco, afetam demasiadamente o desempenho do sistema porque são executados como processos [46]. Todavia, foi observado que o *Cluster Hadoop* melhorou seu desempenho computacional em um determinado ambiente totalmente virtualizado, além dos ganhos com a escalabilidade e a tolerância a falhas [46]. Esses resultados são ratificados em [39, 41], que demonstraram que o *Cluster Hadoop*, em ambientes totalmente virtualizados, quando os *hosts* físicos possuem até quatro máquinas virtuais, podem apresentar um desempenho superior que um ambiente semelhante formado por máquinas físicas.

Segundo a *VMware* [40], executar o *Hadoop* em um ambiente virtualizado pode ter vários benefícios, como a redução do tempo de implementação, a utilização sob demanda, a alta disponibilidade e a tolerância a falhas, a consolidação de hardware, a melhoria da eficiência do *datacenter* e a segurança. Entretanto, para a utilização de um ambiente virtual é fundamental que seja escolhido um *hypervisor* que apresente alto desempenho com o *cluster Hadoop*. Nesse sentido, *Yang et al.* [84] realizaram testes utilizando os ambientes de virtualização baseados nos *hypervisors Xen*[1] e KVM[2] (*Kernel-based Virtual Machine*), concluindo que o *Xen* possui melhor desempenho e estabilidade para a virtualização do *Hadoop*.

Em uma pesquisa mais recente, *Li et al.* [48] comparando os *hypervisors Xen*, KVM e uma outra solução comercial não especificada, concluíram que eles apresentam diferentes características de desempenho, de acordo com a variação das cargas de trabalho. Portanto, a escolha do *hypervisor* deveria ocorrer em função das cargas de trabalho. Além disso, o ideal é que as VM (*Virtual Machines*) sejam distribuídas no máximo de máquinas físicas possíveis, como uma medida geral para melhorar o desempenho [48].

Para *Li et al.* [48], e *Kontagora e Gonzalez-Velez* [46], em determinadas situações é possível melhorar o desempenho do *cluster Hadoop*, em ambientes virtuais, criando mais VMs no mesmo *host*, visando aumentar o paralelismo e uma melhor utilização dos recursos. Contudo, aumentar muito o paralelismo também pode prejudicar o desempenho [48], desta forma, é necessário encontrar um ponto de equilíbrio, visando não saturar os recursos do hospedeiro.

Segundo a *VMware Inc.* [39], os resultados utilizando o *cluster Hadoop* virtualizado sob o *hypervisor* VMWare[3] demonstraram que até quatro VMs por *host* físico, sob certas condições, apresentaram os melhores resultados de utilização de CPU e desempenho de escrita em disco.

Conforme *Ishii et al.* [44], a utilização do *cluster Hadoop* em máquinas virtuais degrada o seu desempenho em função da sobrecarga causada pela camada de virtualização. Além disso, concluem que o I/O de disco do servidor físico representa um gargalo para o processamento dos *jobs Hadoop MapReduce* maior que o uso intensivo de CPU. Desta forma, considerando ambientes virtualizados, para melhorar o desempenho de *jobs MapReduce*, com intenso I/O de disco, a melhor prática seria aumentar o número de máquinas

[1] *Xen* é um *hypervisor open-source*, que permite a execução de várias instâncias de um sistema operacional ou de diferentes sistemas operacionais, em paralelo em uma única máquina (http://wiki.xenproject.org/wiki/Xen_Overview).

[2] KVM (Kernel-based Virtual Machine) é uma solução de virtualização para o sistema operacional *Linux*, baseado na utilização de módulos do *kernel* que fornece a infraestrutura de virtualização de acordo com o modelo do processador do *host* (http://www.linux-kvm.org/page/Main_Page).

[3] *VMWare* é um *hypervisor* que virtualiza servidores, possibilitando consolidar as aplicações com menos hardware (http://www.vmware.com/products/vsphere-hypervisor/).

virtuais e não o número de CPUs virtuais das VMs, assim como, diminuir o número de *jobs Hadoop MapReduce* executados simultaneamente.

Apesar dos resultados apresentados por *Ishii et al.* [44], é importante salientar que os gargalos de I/O de disco podem ser amenizados por meio de soluções de sistemas de armazenamento inteligentes e redes de armazenamento, combinados com soluções de virtualização de armazenamento [73].

As pesquisas que realizaram comparações da performance do *cluster Hadoop* em ambientes virtuais concluíram que o *Hypervisor Xen* possui a melhor performance e, naturalmente, seria uma boa escolha para um *cluster Hadoop* em ambiente virtualizado.

Quanto a performance do *cluster Hadoop* em ambientes virtuais, foi observado que, de forma geral, a camada de virtualização degrada a performance, mas que em determinadas situações o desempenho do *cluster Hadoop*, em ambiente virtualizado, pode ser superior ao mesmo em um ambiente físico, pois melhora a eficiência na utilização dos recursos dos servidores, principalmente, se for considerado o poder de processamento (CPU, RAM, HD) dos servidores atuais.

As pesquisas também apontaram que o I/O de disco pode ser o principal limitador da performance do *Cluster Hadoop* em ambientes virtuais. Porém, em plataformas de nuvem os usuários terão pouca ou nenhuma gerência sobre a quantidade de VMs por *host* físico, mas pode-se reduzir a quantidade de *tasks* executadas simultaneamente em cada VM, visando reduzir a concorrência pelo uso do disco virtual. Além de proporcionar desempenho e tolerância a falhas, o modelo de computação em nuvem é uma tendência na área de computação [83] e não pode ser ignorado.

Diante dos estudos analisados, fica transparente que o *cluster Hadoop* é viável em plataformas virtualizadas, e por consequência, em plataformas de nuvens. Desta forma, também foi confirmado a viabilidade do trabalho proposto.

4.2 Otimização do Desempenho do *Cluster Hadoop*

O *Hadoop* pode rodar em ambientes com milhares de máquinas, portanto, esforços para implementar melhorias de performance, em nível de hardware ou em nível de software, podem representar ganhos consideráveis no processamento final. Nesta ótica, o primeiro nível de otimizações seria utilizar hardwares robustos, com CPU rápida composta de vários núcleos, memória RAM abundante, vários discos rígidos de alta performance e uma rede de alta velocidade [27, 43], o que em princípio gera um certo conflito com um dos objetivos do *Hadoop* que seria a execução em hardware comum [4]. Além disso, pode-se levar a subutilização dos recursos computacionais e, consequentemente, a gastos desnecessários.

Na literatura são encontradas diversas pesquisas com diferentes enfoques, abordando a problemática da otimização do *cluster Hadoop*. Em sua pesquisa, *Wlodarczyk et al.* [80] realizaram análises das dependências de parâmetros como tamanho da entrada de dados, número de nós, o número de *ReduceTasks* e a sobrecarga da cópia dos dados para o *HDFS*, concluindo que a complexidade do processamento, a quantidade de *ReduceTasks*, e o desempenho da rede são importantes fatores. Além disso, concluiu que a performance dos discos rígidos são fatores limitantes em *MapTasks* que produzem grandes saídas de dados. Segundo *Zhuoyao et al.* [87], a escolha do número de *ReduceTask* não é trivial e depende do tamanho do *cluster Hadoop*, dos dados a serem processados e da quantidade de recursos disponíveis para o processamento do *job*.

Conforme *Rizvandi et al.* [64], o desempenho de processamento do *cluster Hadoop* melhora sensivelmente com a adição de mais nós, resultados que foram ratificados por *Maurya e Mhajan* [58]. Todavia, considerando a implementação padrão do *Hadoop*, onde o HDFS possui um único *NameNode*, a adição de nós ao *cluster Hadoop* é limitada, principalmente, pela quantidade de memória RAM disponível no *NameNode* [70], pois todo o *namespace* e endereço dos blocos de arquivos do sistema de arquivos distribuído ficam armazenados na memória RAM [71]. Outros limitadores naturais para o número de nós são os recursos disponíveis em cada organização e a sobrecarga do uso da rede de computadores gerada pela troca de mensagens que ocorre entre os nós escravos e o mestre [70].

Em sua pesquisa, Premchaiswadi e Romsaiyud [62] propõe a utilização de uma extensão no *Cluster Hadoop* que pode automaticamente ajustar as configurações do *Hadoop MapReduce*, através da redistribuição dos dados, da realocação do número de *slots map/-reduce* e a adoção de um mecanismos de agendamento de roteamento híbrido para a fase *shuffle*, a qual é o processo de cópia dos dados gerados no processamento das *Map Task* e gravadas localmente para os *hosts* que executarão as *ReduceTasks*, através da rede. Todavia, apesar dos resultados expressivos obtidos, a utilização de extensões não mantidas pela Apache gera um complicador, pois essas extensões podem não ser compatíveis e/ou portadas para novas versões do *Hadoop*.

Para *Heger* [27], o desempenho do *cluster Hadoop* pode ser melhorado com a adição de mais discos rígidos nos *DataNodes*. Além disso, considera que ajustar o desempenho do *cluster Hadoop* é um processo bastante demorado, exigindo uma análise aprofundada dos códigos do *job Hadoop MapReduce*, dos recursos físicos e lógicos utilizados pela carga de trabalho da aplicação.

Segundo *Joshi* [45], é possível melhorar o desempenho do *Hadoop* através de ajustes nos parâmetros de configuração, como a quantidade de *MapTask* e *ReduceTask* por servidor, o tamanho dos blocos de arquivos, a quantidade de memória alocada para essas *tasks*, o aumento da área de memória reservada para ordenação de dados e a utilização de compactação de dados na saída dos *MapTasks*. Além disso, propõe a utilização de vários discos de dados, ajustes nos sistemas operacionais dos servidores.

Muitas pesquisas focam nos ajustes de hardware e/ou do sistema operacional para melhorar a performance do *cluster Hadoop*. Uma vez que ajustes de hardware não fazem parte do escopo deste trabalho, será necessário implementar ajustes de software para evitar gargalos na rede, otimizar a performance de acesso aos dados e a execução do processamento. A otimização por software será alcançada por meio de ajustes sucessivos no ambiente de testes. Neste sentido, pode-se considerar como um ponto de partida as otimizações citadas pela INTEL [43], com avaliações e refinamentos baseados nos ajustes dos *jobs Hadoop MapReduce* e fluxo de trabalho do *Hadoop MapReduce*, ambos descritos por *White* [79].

Considerando que os testes serão realizados no EMR (*Amazon Elastic MapReduce*), também é importante salientar que os resultados observados podem ser diretamente influenciados pela heterogeneidade dos hospedeiros físicos, em função do agendador do *cluster Hadoop* distribuir os *jobs* entre os *TaskTracker* considerando algumas suposições implícitas [86], bem como por erros de arquitetura nas aplicações *MapReduce*, *overhead* da portabilidade do *Java*, suposições de portabilidade [69] e controles de localidade [81]. Outro fator relevante é a cópia de dados entre os nós, quando há impossibilidade de processamento nos *TaskTracker* que possuem os dados necessários [81].

Outro importante ponto a ser explorado é a utilização do *Hadoop MapReduce* de forma correta e eficaz para o processamento a ser realizado. Isso implica em conhecer em detalhes o funcionamento do *framework* do *MapReduce* e a utilização de *design patterns* adequados para o problema. Nesse aspecto, os *design patterns* citados por Miner and Shook (2013)[59], fornecem o arcabouço para a elaboração de *jobs Hadoop MapReduce* simples e funcionais.

Eventualmente, podem ocorrer falhas nos nós do *cluster Hadoop* e afetar o desempenho durante a execução dos *jobs*. Além disso, os ajustes mais finos no *framework Hadoop* são específicos, de acordo com o tipo de processamento a ser realizado, os hardwares e suas configurações utilizadas, o sistema de virtualização utilizado, o sistema operacional, o sistema de arquivos e outras variáveis. Portanto, o que se busca neste trabalho é a utilização de uma camada leve de otimização que seja genérica o suficiente para atender a maioria dos casos em que o *cluster Hadoop* seja a solução adotada, sem aumentar a complexidade do projeto e os custos relacionados.

4.3 *Benchmarking* do *Hadoop*

A utilização de técnicas de *benchmark* disponibilizam métodos eficazes para a realização de medições, avaliações e comparações de processamentos diversos, inclusive dos processamentos realizados pelo *Hadoop*, por isso, existem diversas pesquisas e publicações abordando este assunto.

Segundo *Sangroya et al.* [68], existem várias soluções de *benchmarking* de performance para pesquisas e padrões industriais. Desta forma, é necessário utilizar corretamente as técnicas de *benchmarking* no *Hadoop* visando atingir a melhor performance, através dos ajustes na configuração do *framework Hadoop*.

O *Hadoop* possui várias ferramentas de *benchmarks* que acompanham o produto, os quais podem ser facilmente utilizados. Cada ferramenta de *benchmark* realiza determinado tipo de teste, visando avaliar determinadas características do *cluster Hadoop*. Abaixo são apresentada as principais ferramentas de *benchmark* que estão presentes nas distribuições do *Hadoop* [79]:

- ***TestDFSIO*:** *benchmark* utilizado para testes da performance de I/O do sistema de arquivos distribuído do *Hadoop*;
- ***Sort*:** *benchmark* que realiza testes de ordenação de dados para aferir a performance de execução dos *jobs Hadoop MapReduces*;
- ***MRBench*:** *benchmark* que executa um job *Hadoop MapReduce* em uma quantidade pré-definida de vezes;
- ***NNBench*:** *benchmark* que realiza testes de carga de trabalho no *NameNode*, visando avaliar sua capacidade de processamento.

Além desses, também pode ser utilizado o pacote de *benchmark* GridMix [79], que permite a modelagem realística de cargas de trabalho para um *cluster Hadoop*, imitando uma variedade de padrões de acessos a dados conhecidos.

O *Mochi Benchmark*, apresentado por *Tan et al.* [77], apresenta um *benchmark* para o *Hadoop* que extrai e visualiza informações sobre os *jobs Hadoop MapReduce* por meio da análise e do correlacionamento dos *logs* de execução dos *jobs* no *cluster Hadoop*. O *Mochi Benchmark* fornece uma visão unificada fim-a-fim, chamada de JCDF (*Job-Centric Data Flow*) que é um grafo direcionado com vértices representando os estágios de processamento e itens de dados. Conforme *Tan et al.* [77], o *Mochi Benchmark* correlaciona a execução dos *TaskTrackers* e *DataNodes* no espaço, tempo e volume, visando identificar quando os dados foram lidos ou gravados no HDFS, com a finalidade de apresentar o caminho dos dados que estão sendo lidos no HDFS, processados na estrutura do *cluster Hadoop* e gravados no HDFS.

Para *Huang et al.* [35], o *HiBench Benchmark Suite* é um pacote que utiliza alguns dos algoritmos de *benchmark* padrão do *Hadoop* e agrega aplicações do mundo real aos testes, permitindo uma análise mais completa do *cluster* ao utilizar os resultados experimentais para avaliar e definir a estrutura do *Hadoop* em termos de tempo de execução, tarefas concluídas por minuto, largura de banda do HDFS e recursos do sistema (CPU, RAM e I/O de disco).

Segundo *Sangroya et al.* [67, 68], o *benchmark* MRBS (*MapReduce Benchmark Suite*) é um conjunto de ferramentas capaz de avaliar a confiabilidade e o desempenho de sistemas *MapReduce*, considerando várias métricas de medição como latência de clientes, *throughput*, custo, tamanho dos dados de leitura/escrita, taxas de transferência entre os *jobs* e as *task* do *MapReduce*. Além disso, permite diferenciar cargas de trabalhos, considera aplicações *batch* e interativas, dentre outras. Segundo seus autores, o MRBS expande os *benchmarks* tradicionais, permitindo análise multi-critérios e testes mais realísticos.

Para que seja possível concluir sobre os resultados de uma otimização do *cluster Hadoop* é primordial uma rápida e precisa análise dos resultados dos *jobs* processados através de uma ferramenta de *benchmark* confiável. Não foi encontrado nenhum óbice quanto a *benchmarks* em ambiente totalmente virtualizados, conforme pode ser observado em [46].

Após análises nas ferramentas de *benchmark* citadas, optou-se por utilizar o *HiBench - The Hadoop Benchmark Suite* [26], por possuir um conjunto de programas com várias cargas de trabalho típicas do *framework Hadoop*, que auxiliam a avaliação em termos de desempenho, *throughput* de rede, utilização de recursos do sistema e padrões de acesso a dados, alinhado à simplicidade na instalação, configuração e utilização, além de ser utilizado por grandes empresas como IBM [54] e Intel [6]. Suas principais aplicações para testes de estresse são: *Sort, WordCount, TeraSort, Nutch Indexing, PageRank, Bayesian Classification, K-means Clustering* e *Enhanced DFSIO*.

4.4 Modelos Estimadores de Custo de *Job MapReduce*

Ao longo do tempo alguns pesquisadores buscaram elaborar modelos que fossem capazes de predizer o custo de processamento de cargas de trabalho e gerar otimizações para os parâmetros de configuração do *framework Hadoop*, visando melhorar o desempenho de processamento dos *jobs Hadoop MapReduce*.

Herotodos et al. [34], apresentaram o Projeto *Starfish*, um dos modelos de performance mais detalhados, que foi desenvolvido ao longo de várias pesquisas [29–33, 49–51]. O *Starfish* é um sistema capaz de fazer ajustes automáticos dos parâmetros de configuração de um *cluster Hadoop* para um determinado *job*, que procura se ajustar às necessidades dos usuários e cargas de processamento do sistema para prover um bom desempenho.

Além disso, o *Startfish* é formado por uma combinação de técnicas de otimizações de custo baseada em consulta de banco de dados, um processador de consultas robusto e adaptativo, programas de análise estática e dinâmica, amostragem de dados dinâmica com geração de perfis de tempo de execução e aprendizado de máquina estatístico aplicado aos fluxos de dados dos sistemas.

O Projeto Starfish é um estimador baseado em modelos de custo, formado por vários componentes, dentre os quais há o *Elastisizer* [33], que propõe uma solução para a problemática do dimensionamento do *cluster*. Todavia, sua abordagem utilizando perfis de *jobs*, perfis de servidores, estimadores baseados em aprendizado de máquinas e simulações, visa responder como será o desempenho de um *job Hadoop MapReduce* se o número de nós de um *cluster Hadoop* for alterado. Além disso, existe a necessidade de realizar novos treinamentos para novos *jobs* e sua sistemática de coleta de informações gera sobrecarga de processamento e interfere no tempo de execução dos *jobs*.

O Projeto *Starfish* foca na otimização de desempenho dos *jobs Hadoop MapReduce* e gera estimativas ajustando as configurações do *framework Hadoop*. Todavia, uma vez que o *framework Hadoop* possui mais de 200 parâmetros de configuração, fica difícil controlar os ajustes utilizando poucos nós e pouca sobrecarga de processamento, sendo necessário grandes bases de dados e um *cluster* com centenas de nós para visualizar o impacto das alterações de todos os parâmetros, além de várias rodadas de testes de processamento.

Conforme *Song et al.* [74], um dos desafios na análise do *Hadoop* é predizer o desempenho de *jobs* individualmente. Para superar esse desafio, eles propuseram um *modelo* que busca predizer o tempo de execução de um *job Hadoop MapReduce*, utilizando uma amostra dos dados de entrada e analisando a complexidade dos *jobs MapReduce*. O modelo proposto é composto por um analisador de *jobs*, responsável por analisar os *jobs* executados e coletar informações relacionadas com os *jobs* e com o *cluster Hadoop* e um módulo estimador, baseado nas pesquisas de *Lin et al.* [52], responsável por estimar o desempenho do *job Hadoop MapReduce*, utilizando métodos de regressão linear.

Na pesquisa de *Song et al.* [74] há necessidade de modificar a aplicação *Hadoop MapReduce* e a sistemática de coleta de informações gera sobrecarga de processamento e interfere no tempo de execução dos *jobs*. Além disso, há a necessidade de treinar o modelo para novos *jobs*, pois apesar de ser possível utilizar dados de *jobs* semelhantes, a precisão do modelo fica comprometida.

O modelo proposto por *Zhang et al.* [88] e o modelo proposto por *Zhuoyao et al.* [87] são capazes de estimar o tempo de conclusão de um *job Hadoop MapReduce* para uma nova entrada de dados a ser processada. O modelo coleta informações e cria perfis a partir de *jobs* já executados. Além disso, os perfis são criados com base nas fases genéricas do processamento de um *job Hadoop MapReduce*, procurando estimar o custo de cada uma

dessas fases para predizer o tempo de execução do mesmo *job* para uma nova base de dados.

Lin et al. [52] apresentaram um modelo capaz de estimar a complexidade e o tempo de execução das *tasks* do *Hadoop MapReduce*, utilizando um método que divide o processamento a partir da perspectiva de utilização de recursos, e utiliza fórmulas detalhadas para estimar o tempo de um *MapTask* e de um *ReduceTask*, separadamente. Além disso, apresentou um conceito de complexidade computacional relativa para as aplicações *Hadoop MapReduce*, visando medir e estimar a complexidade das aplicações *MapReduce* desenvolvidas pelos usuários.

A maioria dos modelos estudados são específicos para cada *job*, sendo necessárias novas análises para novos *jobs*. Em todos os casos é necessário fazer vários testes com amostras da base de dados a ser processada, para coletar informações do processamento (análise *post mortem* dos *jobs Hadoop MapReduce*) para alimentar os respectivos modelos. A maioria dos modelos é capaz de predizer o tempo de processamento apenas para o mesmo *job*, úteis quando há mudanças na quantidade de nós ou no tamanho da entrada de dados a ser processada.

Para atingir os objetivos desta pesquisa se torna necessária a utilização de um modelo capaz de predizer o tempo médio para a conclusão de um *job Hadoop MapReduce*, de forma rápida, simples e precisa, porém sem a necessidade de realização de constantes e/ou longos treinamentos no modelo. Além disso, buscou-se um modelo que não gere distorções no tempo de processamento, pois há muitas estimativas envolvidas.

Diante deste cenário, optou-se por utilizar como base, o modelo proposto por *Lin et al.* [52], pois não gera sobrecarga no processamento dos *jobs*, é de fácil implementação e pode ser aplicado à novos *jobs*, com poucas restrições. Além disso, o fato de utilizar um método que divide o processamento do *job Hadoop MapReduce* a partir da perspectiva de utilização de recursos, facilita estudar e estimar o consumo de diferentes tipos de recursos para cada *job*.

4.4.1 Considerações Finais

Neste capítulo foram apresentados alguns trabalhos que estão correlacionados com a pesquisa desta dissertação, abordando estudos que tratam sobre a utilização do *Hadoop* em ambiente virtualizado, a otimização do seu desempenho, alguns *benchmarks* capazes de avaliá-lo e sobre modelos existentes que possam ajudar a construir um modelo estimador de *cluster Hadoop*, objeto do próximo capítulo.

Capítulo 5

Modelo Estimador de *Cluster Hadoop* - HCE_m

Nos capítulos anteriores, foram apresentados o *Hadoop*, plataforma utilizada neste trabalho, e estudos sobre a utilização do *cluster Hadoop* em ambiente virtualizado. Além disso, também foram apresentados a otimização do seu desempenho, as ferramentas de *benchmark* para o *cluster Hadoop* e os modelos estimadores do custo de processamento de *jobs Hadoop MapReduce*, estudos que formaram o arcabouço do objeto deste trabalho.

Neste capítulo é apresentado um modelo para estimar o tamanho de um *cluster* para o *framework Hadoop*, de acordo com a carga de trabalho de um determinado *job Hadoop MapReduce* e considerando um dado intervalo de tempo, intitulado de Modelo Estimador de *Cluster Hadoop* (HCE_m).

5.1 Considerações Iniciais

Atualmente, com o advento da computação em nuvem, qualquer usuário, mesmo os com pouca experiência em TI, pode provisionar um *cluster Hadoop* em poucos minutos, executar seus *jobs Hadoop MapReduce* e pagar apenas pelos recursos utilizados. Mas agora, apesar da flexibilidade da plataforma de nuvem, os usuários são regularmente confrontados com os complexos problemas de dimensionamento do *cluster Hadoop* [33], que envolvem definir a capacidade de processamento e de armazenamento necessários para a carga de trabalho para uma aplicação *MapReduce*.

No mundo real, processamentos de pequeno porte podem ser rapidamente absorvidos pelas infraestruturas de TI existentes, mas quando o alvo é o processamento de grandes bases de dados, de processamentos que ficam várias horas em execução, os recursos computacionais disponíveis para as organizações realizarem esses processamentos são limitados, bem como o tempo disponível para a utilização dessas infraestuturas, seja porque há outros processamentos a serem realizados, seja porque há limites para custos com TI.

Considerando a utilização de um serviço pago, como o Amazon EMR[3] (*Amazon Elastic MapReduce*) da plataforma AWS (*Amazon Web Services*), subalocação resulta em mais tempo de processamento, enquanto superalocação resulta na alocação de recursos não utilizados ou pouco utilizados. Ambos os casos significam mais custos para realizar o processamento.

Assim, quando a plataforma é uma nuvem privada, é natural que os recursos computacionais sejam limitados, tanto pela infraestrutura da nuvem, quanto pela utilização dos vários processamentos que são alocados ao longo do tempo. Desta forma, é natural que as nuvem privadas possuam restrições para alocação de recursos, bem como janelas de tempo para utilização desses recursos. Também é necessário considerar que em muitos casos os recursos para o *cluster Hadoop* serão alocados apenas para realizar o processamento, depois os recursos da nuvem serão disponibilizados para outros processamentos. Portanto, em uma nuvem privada, utilizada em ambiente de produção, o problema de dimensionamento de um *cluster Hadoop* torna-se ainda mais crítico, sendo latente a necessidade de definir a capacidade de processamento considerando um dado intervalo de tempo.

Para que o *framework Hadoop* apresente um bom desempenho não basta uma infraestrutura adequada, ainda é necessário ajustar seus parâmetros de configuração de acordo com a carga de trabalho e os recursos computacionais disponibilizados. Desta forma, pode-se observar a real necessidade de mecanismos que possibilitem definir a infraestutura de *cluster* para o *framework Hadoop* processar um *job MapReduce* para uma determina carga de trabalho, considerando um determinado intervalo de tempo. Além disso, é necessário que esses mecanismos sejam capazes de definir as configurações básicas dos *hosts* e os principais parâmetros de configuração do *Hadoop*.

A problemática de dimensionamento do *cluster Hadoop* e os trabalhos prévios que, direta ou indiretamente, buscaram soluções, foram motivadores para investigar um modelo capaz de estimar o tamanho de um *cluster* para o *framework Hadoop*, com recursos suficiente para realizar o processamento de determinado *job MapReduce*, para uma determinada carga de trabalho e considerando um dado intervalo de tempo. Estimando ainda os recursos básicos dos *hosts* do *cluster* e os principais parâmetros de configuração do *framework Hadoop*.

A idéia principal é evitar que a infraestrutura de *cluster* e do *framework Hadoop* sejam subestimados ou superestimados, evitando custos desnecessários com reprocessamento ou desperdícios causados por alocação de recursos não utilizados ou subutilizados.

Visando atingir esses objetivos, foi proposto o HCE$_m$ (Modelo Estimador de *Cluster Hadoop*), um modelo que busca estimar a infraestrutura de um *cluster* para o *framework Hadoop*, processar determinada carga de trabalho considerando um dado intervalo de tempo. Além disso, o HCE$_m$ também deve definir os recursos computacionais básicos (CPU, RAM, HD) dos *hosts* do *cluster* e ajustar os principais parâmetros de configuração do *framework Hadoop*, de acordo com a carga de trabalho e os recursos computacionais disponíveis estimados.

5.2 Escopo do HCE$_m$

O escopo do HCE$_m$ é o *Hadoop*, versão 1.x, executado em plataforma virtualizada, como uma nuvem interna ou uma IaaS, com todos os *hosts* semelhantes e no processamento *batch* de grandes arquivos.

As otimizações propostas pelo HCE$_m$ se limitam aos principais parâmetros de configuração do *framework Hadoop*, portanto também não fazem parte do escopo desta pesquisa ajustes nos *hypervisors* utilizados, na camada de hardware, nos sistemas operacionais e controles de localidade [81]. Além disso, o HCE$_m$ não realiza considerações sobre as cargas de processamento que estejam em execução nos *hosts* físicos da plataforma de virtualização.

O foco do HCE$_m$ são os processamentos *batch* de grandes arquivos sequenciais, ou outros processamentos em que os arquivos de entrada sejam do tipo *write-once-read-many*. Portanto, o processamento *online* de transações também não fazem parte do escopo deste trabalho.

O HCE$_m$ considera que a base de dados a ser processada está no HDFS do *cluster Hadoop*, portanto não propõe alternativas para os problemas da importação inicial dos dados [82] para o *Hadoop*. Além disso, as análises abrangem apenas os *jobs Hadoop MapReduce*, implementados na linguagem Java, apesar do *Hadoop* poder executar aplicações implementadas em outras linguagens.

5.3 HCE$_m$

Esta Seção apresenta o HCE$_m$, ilustrado na Figura 5.1, a partir de uma visão de processo, sendo organizado por dois subprocessos principais, um denominado de *Perfil do Cluster Hadoop* e o outro Estimador de *Cluster Hadoop*.

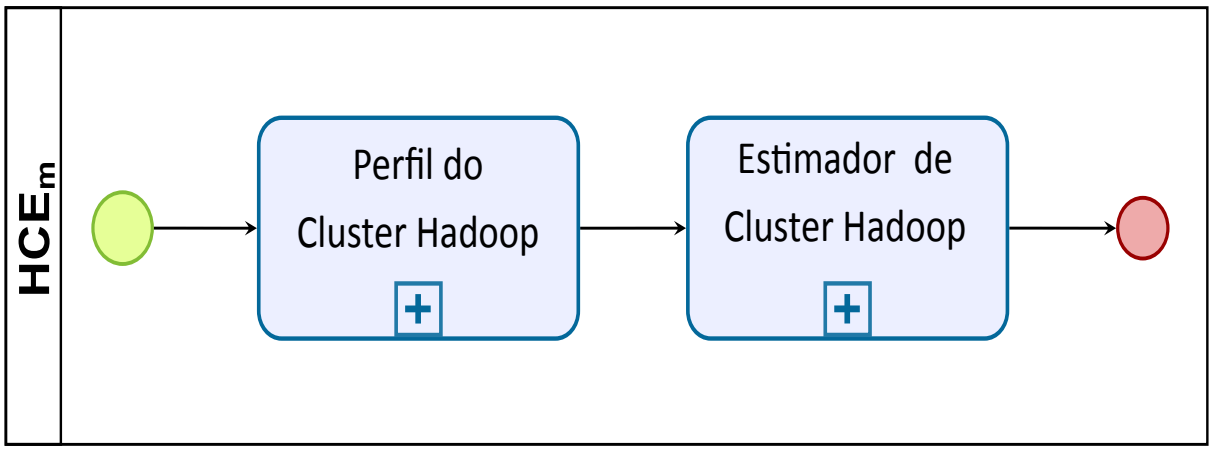

Figura 5.1: Visão Geral do HCE$_m$.

No subprocesso *Perfil do Cluster Hadoop* são mapeadas as principais características do *cluster* onde o *Hadoop* está sendo executado, sem se preocupar com as características dos *jobs MapReduce* para os quais se deseja estimar um *cluster*. No subprocesso Estimador de *Cluster Hadoop* são coletadas informações sobre o *job MapReduce*, objetivo do HCE$_m$, e estimado o tamanho de um *cluster* para o *framework Hadoop* executar um *job MapReduce* no processamento de determinada carga de trabalho, considerando um dado intervalo de tempo. Nas próximas seções esses subprocessos serão apresentados em mais detalhes.

5.3.1 *Perfil do Cluster Hadoop*

O HCE$_m$ considera que no *Hadoop MapReduce* a maioria dos processamentos ocorre localmente nos *TaskTrackers* e que a adição de mais nós ao *cluster* não degrada o desempenho individual dos *hosts* quanto à recursos como o I/O de disco local, utilização da memória RAM e a eficiência computacional da CPU. Assim sendo, a adição de mais *hosts* ao *cluster* pode reduzir do tempo total para processar uma carga de trabalho, pois permite aumentar a quantidade de *tasks* (*map/reduce*) executadas simultaneamente.

Pode-se supor que o desempenho dos recursos computacionais (CPU, HD, RAM) dos *hosts* é o mesmo para quaisquer cargas de trabalho. Desta forma, pode-se utilizar determinadas características dos *hosts* e utilizá-las para estimar o tempo de processamento de quaisquer cargas de trabalho.

Neste subprocesso do HCE$_m$ é gerado um perfil para o *cluster* utilizado para o *Hadoop*, de forma que ele possa ser utilizado para auxiliar a estimar o tempo de processamento de quaisquer *jobs MapReduce*. O grande diferencial desse subprocesso é que ele só precisa ser gerado uma vez para o *cluster*, pois ele mapeia características que não estão diretamente relacionadas com os *jobs MapReduce* a serem executados, assim sendo, ele pode ser gerado uma vez e aplicado a todos os *jobs* que forem executados neste mesmo *cluster*.

O perfil do *cluster Hadoop* é utilizado para estimar o tempo de execução e para otimizar as *tasks* (*map/reduce*), gerando informações que permitem estimar o tamanho do *cluster*. Desta forma, o perfil do *cluster Hadoop* foi definido com base em algumas das definições propostas por *Lin et al.* (2012)[52], apresentadas a seguir.

Job Padrão

Representa um *job Hadoop MapReduce* utilizado para testar a eficiência computacional da CPU e medir o RCC (*Relative Computacional Complexity*) da *Function Mapper* ou da *Function Reducer* [52].

RCC

Representa a relação do custo de executar uma *Function Mapper* (ou *Function Reducer*) e o custo de executar o *job padrão* no processamento da mesma carga de trabalho [52].

ParCef

Representa o custo de converter uma linha da base de entrada de dados em um objeto do tipo *key/value* [52], que no *Hadoop* também é definido como um registro. O *ParCef* é diretamente proporcional ao tamanho do registro e definido por: $ParCef = pa1 + pa2 * RLenght$, onde $pa1$ e $pa2$ são coeficientes do *cluster*, e *RLenght* é o tamanho de um registro em *bytes*.

SortCef

Representa o custo de CPU nas operações de ordenação internas realizadas durante a execução de um *job Hadoop MapReduce* [52]. Uma vez que a ordenação interna do *Hadoop MapReduce* é composta do *Quick Sort*, *Heap Sort* e *Merge Sort* é difícil expressá-la matematicamente com precisão, todavia ela pode ser observada por meio de testes, por exemplo, comparando os tempos de processamentos de uma base de dados não ordenada, com o processamento da mesma base de dados ordenada.

SerCef

Representa o custo de CPU na operação de escrita de um registro, objeto no formato *key/value*, dentro do *buffer* de memória e serialização do registro do *buffer* para o fluxo de saída padrão [52]. O *SerCef* também é diretamente proporcional ao tamanho do registro e definido por: $SerCef = se1 + se2 * RLenght$, onde $se1$ e $se2$ são coeficientes do *cluster* e *RLenght* é o tamanho de um registro em *bytes*.

O perfil do *cluster Hadoop*, ilustrado na Figura 5.2, pode ser definido como duas atividades distintas, uma que coleta os dados do *cluster* e outra que coleta os dados a partir da análise dos resultados de processamentos do *Job* Padrão, utilizando bases de dados geradas especificamente para permitir a observação de certos parâmetros do perfil do *cluster Hadoop*.

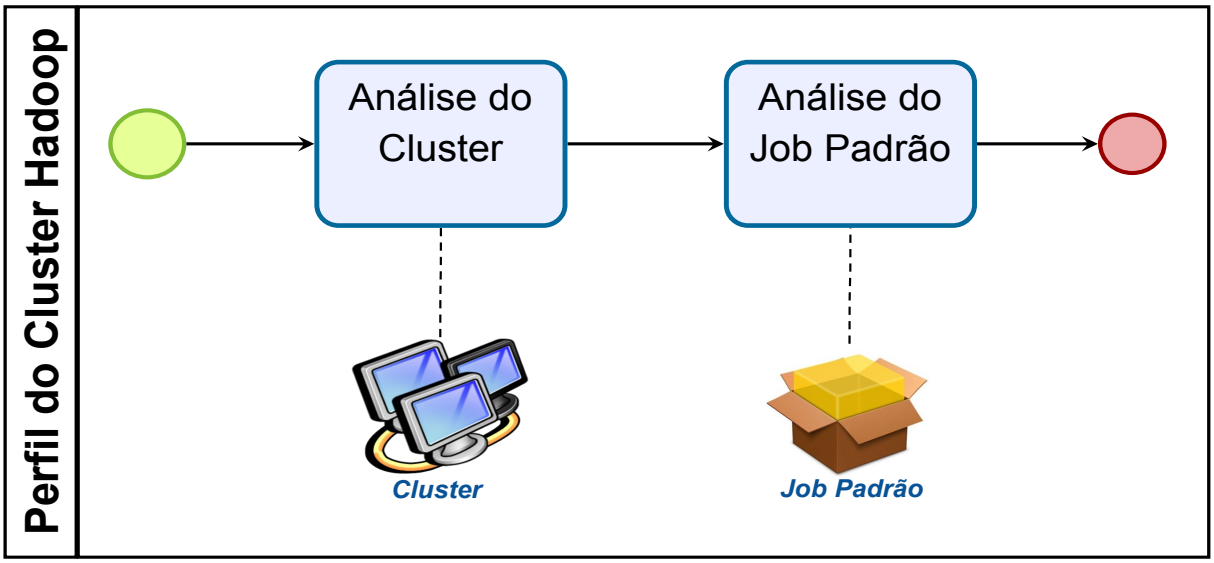

Figura 5.2: *Perfil do Cluster Hadoop.*

Tabela 5.1: Parâmetros do *Perfil do cluster Hadoop.*

Nr	Nome	Descrição
1	*BandWidth*	Largura de banda da rede
2	*SeqRead*	Desempenho de leitura do disco rígido
3	*SeqWrite*	Desempenho de escrita do disco rígido
4	*Access Time*	Média do tempo de acesso ao disco rígido
5	*Cef*	Média do custo de CPU do *job padrão* para processar um *byte* de entrada de dados
6	*SortCef*	Coeficiente de ordenação
7	*SerCef*	Coeficiente de serialização
8	*ParCef*	Coeficiente de conversão de registro em objeto *key/value*
9	*MapSysCost*	Custo do sistema para iniciar o *MapTask*
10	*ReduceSysCost*	Desempenho de leitura do disco rígido

As atividades Análise do *Cluster* e Análise do *Job* Padrão geram o perfil do *cluster Hadoop*, que é formado por dez parâmetros, definidos conforme apresentado na Tabela 5.1, e baseados na pesquisa de *Lin et al.* (2012)[52]. Alguns desses parâmetros podem ser obtidos antecipadamente, enquanto outros só poderão ser obtidos por meio de testes com o *Job* Padrão.

Os parâmetros de 1 a 4 podem ser coletados diretamente do *cluster*, por meio de comandos do sistema operacional *Linux*. Os demais parâmetros são mais complexos e calculá-los com precisão representa um grande desafio, pois eles estão relacionados ao custo de CPU para determinados processamentos realizados pelo *Hadoop MapReduce*, mas podem ser obtidos através de testes utilizando o *Job* Padrão no processamento de bases de dados elaboradas para essa finalidade.

5.3.2 *Estimador do Cluster Hadoop*

O segundo subprocesso do HCE_m é o Estimador do *Cluster Hadoop*, ilustrado na Figura 5.3, formado por cinco atividades, detalhadas a seguir.

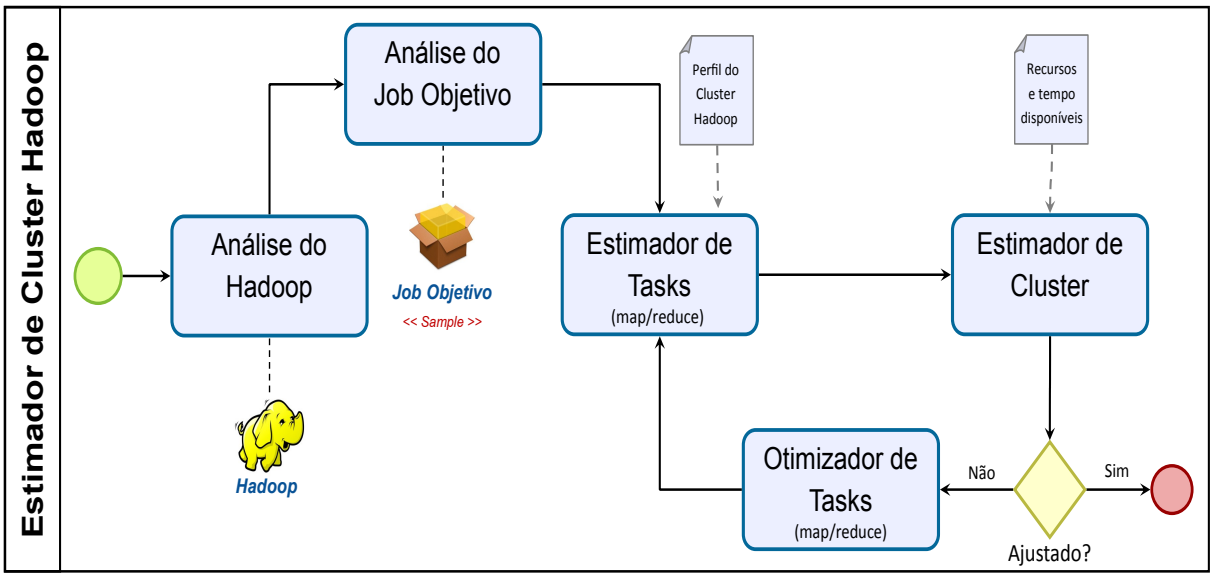

Figura 5.3: Estimador do *Cluster Hadoop*.

Atividade 1 - Análise do *Hadoop*

A atividade Análise do *Hadoop* possui o objetivo de coletar informações do *framework Hadoop*. São coletados treze parâmetros, conforme definidos na Tabela 5.2, dos arquivos de configurações do *framework Hadoop*. Esses parâmetros podem ser coletados antecipadamente e são utilizados como *input* para as atividades Estimador de *Tasks*, Otimizador de *Tasks* e Estimador de *Cluster*.

Tabela 5.2: Parâmetros de Análise do *Hadoop*.

Nr	**Nome**	**Descrição**
1	*Index Size*	Tamanho do índice para um registro no *sort buffer*
2	*SortBufferSize*	Quantidade de memória alocada para uso, enquanto ordenando arquivos nas *MapTask* (io.sort.mb)
3	*BufferPercent*	Percentual do *sort buffer* usado para dados (io.sort.spill.percent)
4	*IndexPercent*	Percentual do *sort buffer* usado para índices (io.sort.record.percent)
5	*MergeFator*	Número de fluxos para *merge* enquanto ordenando arquivos (io.sort.factor)
6	*ReduceJVM*	Memória alocada para a *ReduceTask* (mapred.child.java.opts)
7	*ShuMergeThresold*	Limiar de *segments* para *shuffle merge* (mapred.inmem.merge.threshold)
8	*ShuBufferPercent*	Percentual da memória a ser alocada para salvar os dados de saída dos *MapTaks*, na fase *shuffle* (mapred.job.shuffle.input.buffer.percent)
9	*ShuMergePercent*	Limite de uso da memória, definida em *ShuBufferPercent*, para iniciar o *merge* (mapred.job.shuffle.merge.percent)
10	*RedInBufferPercent*	Percentual de *ReduceJVM* permitido para salvar dados, das *MapTasks*, durante a fase reduce (mapred.job.reduce.input.buffer.percent)
11	*HDFSReplica*	Número de réplicas do sistema de arquivos distribuído do *Hadoop* (dfs.replication)
12	*HDFSBlockSize*	Tamanho de um bloco no sistema de arquivos distribuídos do *Hadoop* (dfs.block.size)
13	*ParallelCopies*	Quantidade de cópias paralelas executadas pelo *ReduceTask* (mapred.reduce.parallel.copies)

Atividade 2 - Análise do *Job Objetivo*

Nesta atividade é utilizado o *Job Objetivo*, que é o *job Hadoop MapReduce* para o qual se deseja estimar o tamanho do *cluster*. Nesta atividade são coletados onze parâmetros, conforme definidos na Tabela 5.3, que são gerados por meio do processamentos do *Job Objetivo* utilizando uma amostra da base de dados como entrada.

Nesta atividade é primordial gerar uma amostra de dados consistente e ao mesmo tempo significativa, pois uma amostra muito grande consome muito tempo, enquanto uma amostra muito pequena pode não ser significativa. Desta forma, é necessária a utilização de um bom algoritmo para gerar essa amostra. Neste trabalho foi utilizado o algoritmo apresentado por *Song et al.* [74], detalhado no Algoritmo 5.1.

Tabela 5.3: Parâmetros de Análise do *Job Objetivo*.

Nr	Nome	Descrição
1	*NumMaps*	Número de *MapTasks*
2	*NumReduces*	Número de *ReduceTasks*
3	*MapInput*	Tamanho dos dados de entrada dos *MapTasks*
4	*MapOutput*	Tamanho dos dados de saída dos *MapTasks*
5	*MIRNumber*	Número de registros (*records*) de entrada dos *MapTasks*
6	*MORNumber*	Número de registros (*records*) de saída dos *MapTasks*
7	*ReduceInput*	Tamanho dos dados de entrada dos *ReduceTasks*
8	*ReduceOutput*	Tamanho dos dados de saída dos *ReduceTasks*
9	*RIRNumber*	Número de registros (*records*) de entrada dos *ReduceTasks*
10	*MapComplex*	Complexidade do *Function Mapper*
11	*ReduceComplex*	Complexidade do *Function Reducer*

```
ENTRADA: - Arquivo de entrada: FILE; Quantidade de linhas para sample: k;
         - Maximo numero de iteracoes (depende do tamanho do arquivo): MAX.
SAIDA:   - Amosta com k linhas (cada linha possui a probabilidade k/n): sk.
-----------------------------------------------------------------------------
Sk = [1; 2; :::; k] //Take the top k lines in FILE;
if k > n then
   return Sk
end if
for new line i to EndOfFile do
   if MAX < 0 then
      break;
   end if
   MAX --;
   int r = random(0; i)
   if r < k then
      S[r]= new line i;
   end if
end for
return Sk;
```

Algoritmo 5.1: Algoritmo de *Sampling* [74].

Atividade 3 - Estimador de *Tasks* (*Map/Reduce*)

A terceira atividade é o Estimador de *Tasks* (*Map/Reduce*), que utiliza os dados coletados nas atividades anteriores (Tabelas 5.2 e 5.3) e mais o perfil do *cluster Hadoop* (Tabela 5.1) para estimar o tempo médio individual das *MapTask* e das *ReduceTask* em função de alterações nos parâmetros de configuração do *framework Hadoop* e na carga de trabalho.

Conforme abordado na Seção 4.4, há vários estimadores na literatura, o HCE_m foi baseado no trabalho de *Lin et al.* [52], que divide o processamento dos *jobs* a partir de uma perspectiva de utilização de recursos, o que facilita estudar o impacto das alterações dos parâmetros de configurações do *framework Hadoop* na utilização de cada recurso computacional abordado (CPU, RAM, HD).

Desta forma, para estimar o tempo de execução de um *MapTask* ou de um *ReduceTask* é necessário analisar o *pipeline* de execução de um *job MapReduce*, conforme estudos apresentado na Seção 3.4, e determinar os custos das fases internas de cada *task* (*map/reduce*), individualmente. A Seção 5.4 abordará em detalhes o estimador implementado neste trabalho.

Atividade 4 - Otimizador de *Tasks (Map/Reduce)*

Vários esforços, incluindo [27, 45, 80, 87] e *What-If Engine* [30] que é parte do Projeto *Starfish* [34], buscam otimizar o desempenho de *jobs Hadoop MapReduce*. Na abordagem deste trabalho buscou-se aplicar uma otimização simplificada, de fácil utilização, mas ao mesmo tempo ampla o suficiente para ser aplicada a maioria dos *jobs MapReduce*. As otimizações propostas foram aplicadas em nível de *tasks* (*map/reduce*).

Analisando os arquivos de configurações do *framework Hadoop*, pode-se observar que existem mais de 200 parâmetros para ajustes, portanto, não é uma tarefa trivial ajustar minuciosamente o *Hadoop*. Segundo *Song et al.* (2013)[74], na maioria dos casos, os usuários simplesmente optam por utilizar os valores padrões ou ajustar os valores dos parâmetros empiricamente.

Neste trabalho foram utilizados apenas alguns parâmetros do *framework Hadoop*, visando reduzir a complexidade da camada de otimização e ao mesmo tempo manter o alinhamento com o Estimador de *Tasks* (*Map/Reduce*) utilizado. Além disso, a utilização de poucos parâmetros facilita a observação dos resultados em um *cluster* com poucos nós, ambiente essencial para gerar testes e estimar o tamanho do *cluster*, permitindo fazer ajustes rápidos e simples.

O Otimizador de *Tasks* (*MapReduce*) utiliza um grupo de parâmetros para otimizar as *MapTaks*, discriminados na Tabela 5.4, e outro grupo para otimizar as *ReduceTasks*, discriminados na Tabela 5.5. Além desses parâmetros o otimizador ajusta outros parâmetros, baseados nas recomendações de *White* [79] e discriminados na Tabela 5.6, que não estão diretamente relacionados aos *MapTasks* ou *ReduceTask*, mas são necessários para sustentar a escalabilidade do *cluster Hadoop*.

Tabela 5.4: Parâmetros para Otimização do *MapTask*.

Nr	**Nome**	**Descrição**
1	*HeapSize*	Quantidade de memória RAM reservada para cada *task*.
2	*BlockSize*	Tamanho de um bloco de arquivo no HDFS.
3	*IoSortMb*	Tamanho do *buffer* de memória utilizado para ordenar a saída da *MapTask*.
4	*IoSortRecordPercent*	Percentual de *IoSortMb* reservada para os metadados dos registros em memória.
5	*IoSortSpillPercent*	Limiar do *buffer* de memória *IoSortMb*, para iniciar o processo de *spills* para disco.
6	*IoSortFactor*	Número de fluxos para *merge* de arquivos.

O objetivo principal desta atividade é reduzir as dispendiosas operações de I/O de disco, pois conforme *Ishii et al.* [44] o I/O de disco do servidor físico representa um gargalo para o processamento dos *jobs Hadoop MapReduce* maior que o uso intensivo de CPU. Além disso, é notório que mesmo com os diversos avanços nos sistemas de armazenamento, os *drives de disco* ainda são as mídias de armazenamento mais populares utilizadas nos computadores modernos para armazenar e acessar dados [73], porém esses dispositivos são limitados fisicamente por suas tecnologias e não possuem as mesmas taxas de desempenho que a CPU ou a RAM.

Desta forma, em relação aos *MapTask*, visa reduzir vários *spills* para o disco, enquanto nos *ReduceTask*, visa manter mais dados intermediários em memória durante a fase *copy*. De forma resumida, busca-se manter o máximo de informações na memória RAM dos *TaskTrackers*. Todavia, simplesmente aumentar a quantidade de memória RAM alocada para cada *task* (*map/reduce*) não necessariamente garante um melhor desempenho de processamento, pois é necessário ajustar os parâmetros correlacionados. Além disso, a quantidade de memória RAM do *TaskTracker* é limitada e deve ser alocada em quantidade suficiente para os *daemons tasktracker* e *datanode*, para o sistema operacional e para todas as *tasks* que forem executadas em paralelo no mesmo *host*.

Tabela 5.5: Parâmetros para otimização do *ReduceTask*.

Nr	Nome	Descrição
1	*HeapSize*	Quantidade de memória RAM reservada para cada *task*.
2	*ParallelCopies*	Número de cópias paralelas executadas pela *ReduceTask* durante a fase *shuffle*.
3	*IoSortFactor*	Número de fluxos para *merge* de arquivos.
4	*ShuffleInputBufferPercent*	O percentual de memória de *Heap Size* a ser alocada para armazenar saídas do *MapTask*, durante a fase *shuffle*.
5	*ShuffleMergePercent*	O limiar de uso do *ShuffleInputBufferPercent* para iniciar o processo de *merge* de arquivos temporários.
6	*InmemMergeThreshold*	O limiar do número de arquivos em memória para iniciar o processo de *merge*.
7	*ReduceInputBufferPercent*	A proporção do total de *heap size* para ser usada para reter *MapTasks outputs* em memória durante a *reduce function*.
8	*NumeroReduces*	A quantidade de *ReduceTasks* que serão lançadas pelo *job*.

Tabela 5.6: Outros Parâmetros a Serem Otimizados.

Nr	Nome	Descrição
1	*IoBufferSize*	Tamanho do *buffer* para operações de I/O.
2	*TrackerHttpThreads*	Quantidade de *threads* por *TaskTracker* para atender requisições de cópia das saídas das *MapTask*, solicitados pelos *ReduceTask*.
3	*HttpThreadsNameNode*	Quantidade de *threads* para tratar requisições de clientes, nos *NameNodes* e *DataNodes*.
4	*HttpThreadsTaskTracker*	Número de *threads* por *TaskTracker* para servir as saídas dos *MapTasks* para as *ReduceTasks*.
5	*HttpThreadsTaskTracker*	Número de *threads* por *TaskTracker* para servir as saídas dos *MapTasks* para as *ReduceTasks*.
6	*SlowStartCompletedMaps*	Fração do número de *MapTasks* no *job* que devem ser concluídos antes dos *ReduceTasks* iniciarem.

Os parâmetros apresentados nas Tabelas 5.4 e 5.5 influenciam diretamente as estimativas realizadas pela atividade Estimador de *Tasks* (*Map/Reduce*), o que permite aplicar valores mais razoáveis. Os parâmetros constantes da Tabela 5.6 são ajustados automaticamente por meio de uma base de regras, apresentada no Anexo A. Além disso, os ajustes são limitados pelos recursos computacionais (CPU, HD, RAM) disponíveis na plataforma de virtualização utilizada, que deve ser fornecido pelo usuário.

Nos *MapTasks*, os ajustes visam reduzirr o valor do parâmetro $\boldsymbol{r}$ do passo T_{m8}, que representa o número de arquivos para *merge*, para o valor mais próximo de um, ajustando principalmente os parâmetros *BlockSize* e *HeapSize*, conforme as estimativas do tamanho da saída do *MapTask*. O parâmetro $\boldsymbol{r}$ influencia diretamente os valores dos passos T_{m6}, T_{m7}, T_{m8}, T_{m9} e T_{m10}.

Nos *ReduceTasks*, os ajustes visam reduzir o valor do parâmetro $\boldsymbol{P}$ do passo T_{r5}, que representa o número de *rounds* de I/O, para o valor mais próximo de zero, ajustando principalmente os parâmetros *HeapSize* e *NumeroReduces*, conforme as estimativas do tamanho da entrada de dados para o *ReduceTask*. O parâmetro $\boldsymbol{P}$ influencia diretamente os valores dos passos T_{r5}, T_{r6}, enquanto a quantidade de *ReduceTasks* aumenta o paralelismo do processamento da maioria dos passos, mas implica em um maior número de *tasks* e de *hosts* para atender os requisitos de tempo.

A análise dos resultados de testes realizados com o *HiBench Benchmaks Suite* [26, 35], um *benchmark* que possui nove testes com cargas de trabalho típicas do *Hadoop*, em termos de tempo de execução, tarefas concluídas por minuto, largura de banda do HDFS e recursos do sistema (CPU, HD, RAM), permitiu confirmar a efetividade do Estimador de *Tasks* (*Map/Reduce*). Mas é importante frisar que o objetivo foi elaborar uma camada leve e simples de implementar, que proporcionasse uma visão mais realista para o tempo de execução das *tasks*, a ser utilizado para estimar o tamanho do *cluster*. Além disso, podem ser aplicadas camadas de otimização mais específicas para as aplicações MapReduce e suas respectivas cargas de trabalho visando atingir um melhor índice de desempenho.

Atividade 5 - Estimador de *Cluster Hadoop*

O Estimador de *Cluster Hadoop* utiliza um grupo de parâmetros como entrada para estimar o tamanho do *cluster*, conforme discriminados nas Tabelas 5.7 e 5.7. Alguns desses parâmetros foram previamente estimados nas atividades apresentadas anteriormente, outros deverão ser informados pelos usuários e estão relacionados com os recursos computacionais (CPU, HD, RAM) disponíveis na plataforma de virtualização. Esse conjunto de parâmetros proporcionam uma boa precisão para estimar o tamanho do *cluster*, pois capturam os detalhes internos dos processamentos das *tasks* (*map/reduce*).

Tabela 5.7: Parâmetros de Entrada para o Estimador de *Cluster Hadoop*, a serem informados pelos usuários.

Parâmero	**Descrição**
DataInput	Tamanho da base de dados a ser processada em *megabytes*.
TimeDisponivelJob	Tempo disponível para o processamento do *job MapReduce*, em segundos.
CPUDisponivel	Quantidade de CPU disponível na plataforma de virtualização, para o *cluster Hadoop*.
RAMDisponivel	Quantidade de memória RAM disponível na plataforma de virtualização, para o *cluster Hadoop*.
HDDisponivel	Quantidade de espaço de disco disponível na plataforma de virtualização, para utilização no *cluster Hadoop*.
FatorReplicacao(FR)	Fator de replicação para os blocos do HDFS. O valor padrão é três.
MapByCPU	Quantidade máxima de *MapTasks* por CPU do *TaskTracker*. O valor padrão é dois.
ReduceByCPU	Quantidade máxima de *ReduceTasks* por CPU do *TaskTracker*. O valor padrão é dois.

Tabela 5.8: Parâmetros de Entrada para o Estimador de *Cluster Hadoop*, estimados por atividades anteriores.

Parâmero	**Descrição**
MapTime	Tempo médio para executar um *MapTask* em segundos, estimado pela atividade Estimador de *Tasks*.
ReduceTime	Tempo médio para executar um *ReduceTask* em segundos. Valor estimado pela atividade Estimador de *Tasks*.
MapTaskMaximum	Máxima quantidade de *MapTask* que podem ser executadas simultaneamente em um mesmo *TaskTracker*.
ReduceTaskMaximum	Máxima quantidade de *ReduceTask* que podem ser executadas simultaneamente em um mesmo *TaskTracker*.
BlockSize	Tamanho do bloco de arquivos do HDFS em *megabytes*, definido pela atividade Otimizador de *Tasks*.
MapOutput	Tamanho da saída do *MapTask* em *megabytes*. Valor estimado pela atividade Estimador de *Tasks*.
ReduceOutput	Tamanho da saída do *ReduceTask* em *megabytes*. Valor estimado pela atividade Estimador de *Tasks*.
HeapSize	Quantidade de memória RAM, em *megabytes*, alocada para cada *task* (*map*/*reduce*). Valor calculado pela atividade Otimizador de *Tasks*.

Para calcular o tamanho do *cluster* são utilizados os parâmetros definidos nas Tabelas 5.7 e 5.8, onde alguns são informados pelos usuários e outros são resultados de processamentos das atividades anteriores, além das fórmulas e definições a seguir:

TempoDisponivelMap

Representa o tempo disponível para a execução das *MapTasks*, calculado em função do tempo disponível para o processamento do *job* e da estimativa de tempo do MapTask. É dado pela Equação 5.1.

$$TempoDisponivelMap = \left(\left\lceil \frac{\left(\frac{DataInput}{BlockSize} * MapTime \right)}{\left(\frac{DataInput}{BlockSize} * MapTime \right) + (QuantReduce * ReduceTime)} \right\rceil * TimeDisponivelJob \right) \tag{5.1}$$

TempoDisponivelReduce

Representa o tempo disponível para a execução das *ReduceTasks*, calculado em função do tempo disponível para o processamento do *job* e da estimativa de tempo do ReduceTask. É calculado por meio da Equação 5.2.

$$TempoDisponivelReduce = \left(\left\lceil \frac{(QuantReduce * ReduceTime)}{\left(\left(\frac{DataInput}{BlockSize} \right) * MapTime \right) + (QuantReduce * ReduceTime)} \right\rceil * TimeDisponivelJob \right) \tag{5.2}$$

MapSimultaneos

Representa a quantidade de *MapTasks* que necessitam ser executados em paralelo, para atingir o tempo calculado em *TimeDisponivelJob*. Essa quantidade é calculada por meio da Equação 5.3.

$$MapSimultaneos = ROUND \left\lceil \frac{\left(\frac{DataInput}{BlockSize} \right)}{\left(\frac{TempoDisponivelMap}{MapTime} \right)} \right\rceil \tag{5.3}$$

Onde ROUND é uma função que arredonda uma expressão numérica recebida para o número inteiro mais próximo.

ReduceSimultaneos

Representa a quantidade de *ReduceTasks* que necessitam ser executados em paralelo, para atingir o tempo calculado em *TimeDisponivelJob*. Essa quantidade é calculada por meio da Equação 5.4.

$$ReduceSimultaneos = ROUND \left[\frac{QuantReduce}{\left(\frac{TempoDisponivelReduce}{ReduceTime} \right)} \right] \tag{5.4}$$

NumSlaves

Representa a quantidade de *hosts* necessário para os escravos executarem o *job MapReduce* e processar o *DataInput*, na janela de tempo definida por *TimeDisponivelJob*. Esse valor é calculado pela Equação 5.5.

$$\begin{aligned} &NumSlaves \\ &= MAX \left(ROUND \left(\frac{MapSimultaneos}{MapTaskMaximum} \right); ROUND \left(\frac{ReduceSimultaneos}{ReduceTaskMaximum} \right) \right) \end{aligned} \tag{5.5}$$

RamSlaves

Representa a quantidade de memória RAM por escravo, necessária para o *job MapReduce* processar o *DataInput* na janela de tempo definida por *TimeDisponivelJob*. É denotado pela Equação 5.6.

$$\begin{aligned} &RamSlaves = (MapMaximum * HeapSize) + (ReduceMaximum \\ &* HeapSize) + 2500 \end{aligned} \tag{5.6}$$

Onde, o valor 2500 é a quantidade e memória para o *host* executar o sistema operacional e os *daemons TaskTracker* e *DataNode*.

CpuSlaves

Representa a quantidade de CPU por nó escravo, necessária para o *job MapReduce* processar o *DataInput*, na janela de tempo definida por *TimeDisponivelJob*. Esse quantidade é calculada por meio da Equação 5.7.

$$CpuSlaves = MAX \left(\left(\frac{MapMaximum}{MapByCPU} \right); \left(\frac{ReduceMaximum}{ReduceByCPU} \right) \right) + 1 \tag{5.7}$$

Onde, o valor 1 é relativo à CPU alocada para o sistema operacional e os *daemons TaskTracker* e *DataNode*.

DiscoLocal

Representa a quantidade de espaço, em *gigabytes*, para o disco destinado ao sistema operacional e arquivos temporários, relativos ao processamento do *job*. Neste parâmetro, considera-se que é utilizado um disco exclusivo para o sistema operacional. Além disso, o valor estimado é acrescido para evitar que o disco tenha mais de 70% de utilização e o tempo de resposta cresça exponencialmente [73]. Esse valor é estimado pela Equação 5.8.

$$DiscoLocal = \frac{\left(\left(\left(\left(MapOutput * \frac{QuantMap}{NumSlaves}\right) * 2\right) + 15360\right) * \frac{100}{70}\right)}{1024} \tag{5.8}$$

Onde, o valor 15360 representa o espaço necessário para o sistema operacional.

DiscoHDFS

Representa a quantidade de espaço, em *gigabytes*, para o disco destinado ao HDFS. Neste parâmetro, considera-se que o *TaskTracker* utiliza, pelo menos, um disco exclusivo para o sistema de arquivos distribuído. Além disso, o valor estimado é acrescido para evitar que o disco tenha mais de 70% de utilização e o tempo de resposta cresça exponencialmente [73]. Esse valor é calculado pela Equação 5.9.

$$DiscoHDFS = \frac{\left[\left(\left(\frac{FatorReplicacao * DataInput}{NumSlaves}\right) + \left(\frac{ReduceOutput * QuantReduce * FR}{NumSlaves}\right)\right) * \frac{100}{70}\right]}{1024} \tag{5.9}$$

RamMaster

Representa a quantidade de memória RAM para o *host* mestre gerenciar o *cluster Hadoop*. Essa quantidade é dada pela Equação 5.10.

$$RamMaster = ROUND\left(\left[\frac{\left(\frac{NumSlaves * DiscoHDFS}{BlockSize * FatorReplicacao}\right)}{1024}\right] + 1\right) \tag{5.10}$$

Para estimar o tamanho do *cluster*, isto é, a quantidade de *hosts*, bem como suas configurações básicas (CPU, HD, RAM), o Estimador de *Cluster Hadoop* deve ser integrado às atividades Otimizador de *Tasks (Map/Reduce)* e Estimador de *Tasks (Map/Reduce)*, desta forma, em função dos parâmetros *TimeDisponivelJob*, *CPUDisponivel*, *RAMDisponivel* e *HDDisponivel*, serão definidos os valores de dois parâmetros chaves, o *MapMaximum* e o *ReduceMaximum*, que representam a quantidade de *tasks* simultâneas que podem ser executadas por um *TaskTracker*, que são essenciais para definir o tamanho do *cluster*.

Em relação às configurações básicas do *host* mestre do *cluster Hadoop*, o Estimador de *Cluster Hadoop* se preocupa apenas com a quantidade de memória RAM, os demais recursos também são incluídos nos cálculos, mas utilizam valores baseados nas recomendações propostas por White (2012)[79]. Desta forma, são computados, três CPU, uma para o sistema operacional, uma para o *daemon JobTracker* e outra para o *daemon NameNode*; um HD de 15 GB para o sistema operacional; mais 2 GB de RAM, além da quantidade de RAM estimada, sendo um para o sistema operacional e outro para o *daemon JobTracker*.

Uma vez que todos os valores sejam calculados, o tamanho do *cluster* é o somatório da quantidade de nós escravos acrescida de um nó para o servidor, enquanto os recursos computacionais para o *cluster* são definidos em função de somatórios das quantidades de memória RAM, de CPU e de HD, conforme demonstrado, respectivamente, nas Equações 5.11, 5.12 e 5.13.

$$Quant.RAM = \sum_{i=1}^{NumSlaves} (RamSlaves) + (RamMaster + 2) \tag{5.11}$$

$$Quant.CPU = \sum_{i=1}^{NumSlaves} (CPUSlaves) + (3) \tag{5.12}$$

$$Quant.HD = \sum_{i=1}^{NumSlaves} (DiscoLocal + DiscoHDFS) + (15) \tag{5.13}$$

É importante salientar que o objetivo é que as atividades Estimador de *Tasks*, Otimizador de *Tasks* e Estimador de *Cluster Hadoop* trabalhem de forma integrada, ou seja, o ajuste em um dos parâmetros por uma dessas atividades vai imediatamente afetar os parâmetros e resultados das demais atividades. Desta forma, o HCE_m possibilita simular vários cenários, considerando os parâmetros utilizados no modelo.

5.4 Estimador de Tempo de *Tasks (Map/Reduce)*

Esta seção apresenta os detalhes do estimador de tempo de *tasks* (*map/reduce*) utilizado pelo HCE_m.

Foi utilizado o estimador proposto por *Lin et al.* (2010)[52], onde foram incluídas adaptações para considerar a execução de *tasks* em paralelo no mesmo *TaskTracker*. A seguir são apresentados os detalhes para estimar o tempo de execução de um *MapTask* e de um *ReduceTask*.

5.4.1 Estimador de Tempo do *MapTask*

Segundo *Lin et al.* (2012)[52], o processamento do *MapTask* ocorre por meio de dez passos, conforme ilustrado na Figura 5.4, onde pode ser observado que alguns passos ocorrem de forma sequencial, outros de forma paralela e há também passos sobrepostos, como a leitura do *input data*, pois os dados podem estar localizados localmente no *TaskTracker* ou em outro nó do *cluster*, ou seja, a leitura pode ser local ou via rede, mas não ambos. Além disso, foi constatado que determinados passos são mais dependentes de certos recursos computacionais, como CPU, HD e Rede. Esse modelo torna possível focar na otimização do *Hadoop* visando recursos que apresentam maior custo, como por exemplo o I/O de disco.

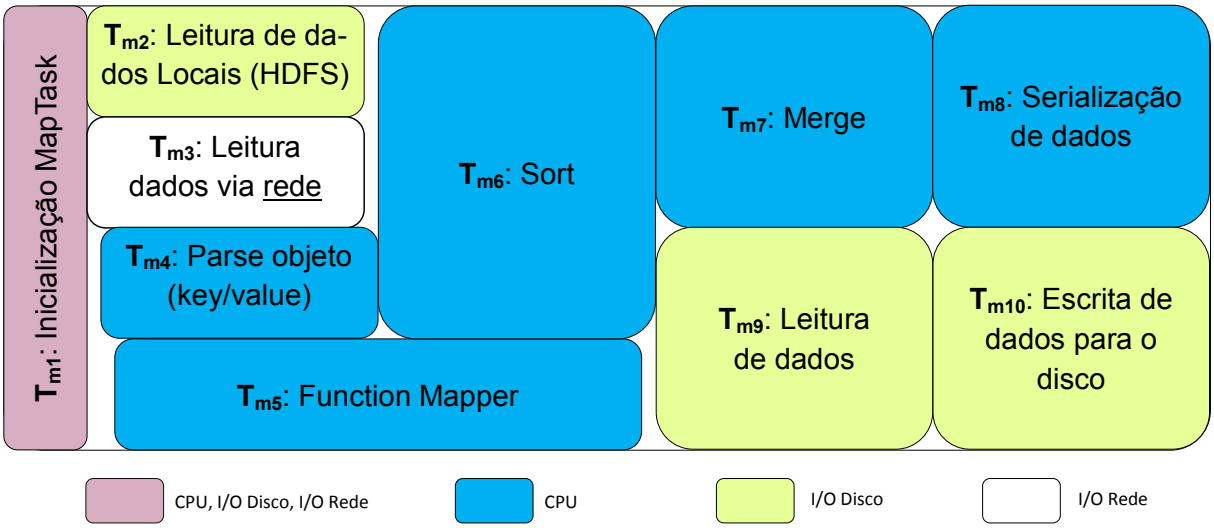

Figura 5.4: *Pipeline* de execução do *MapTask*, adaptada de *Song et al.* (2013)[74].

Para *Lin et al.* (2012)[52], o custo do *MapTask*, denotado por T_{Map}, é uma composição desses passos sequenciais, paralelos ou sobrepostos, definidos como um vetor, formado por uma lista ordenada finita com 10 objetos, indicando as possíveis variações dos valores dos parâmetros, conforme demonstrado na Equação 5.14.

$$T_{Map} = (T_{m1}, T_{m2}, T_{m3}, T_{m4}, T_{m5}, T_{m6}, T_{m7}, T_{m8}, T_{m9}, T_{m10}). \tag{5.14}$$

A seguir é apresentado um resumo de cada um desses passos e as equações utilizadas para definir os seus respectivos valores. É válido salientar que os parâmetros utilizados nas equações foram previamente definidos por meio das tabelas 5.1, 5.2 e 5.3. Além disso, algumas equações foram ajustadas na adaptação para este trabalho, visando considerar o compartilhamento de recursos computacionais por *tasks* executadas simultaneamente no mesmo *TaskTracker*.

T_{m1}: Inicialização *MapTask*

Representa o tempo de inicialização do *MapTask*, ou seja, o tempo necessário para inicializar processos, leitura de arquivos internos, dentre outros. É obtido pela Equação 5.15.

$$T_{m1} = MapSysCost \tag{5.15}$$

T_{m2}: Leitura de dados (HDFS)

Representa o custo de I/O de disco na leitura sequencial de toda a entrada de dados do *MapTask*. A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.16.

$$T_{m2} = \left\lceil \frac{MapInput}{\left(\frac{SeqRead}{TasksSimultaneas}\right)} \right\rceil \tag{5.16}$$

T_{m3}: Leitura de dados via rede

Representa o custo de I/O de rede na leitura do *input split*. Quando o *input split* está no mesmo *TaskTracker* esse custo é zero. A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.17

$$T_{m3} = \left\lceil \frac{MapInput}{\left(\frac{BandWidth}{TasksSimultneas}\right)} \right\rceil \tag{5.17}$$

T_{m4}: *Parse* objeto (*key/value*)

Representa o custo de ler uma linha do *input split*, fazer o *parse* e chamar a *Function Mapper* para processar o objeto (*key/value*). A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É obtido pela Equação 5.18.

$$T_{m4} = ParCef * TasksSimultneas * MIRNumber \tag{5.18}$$

T_{m5}: *Function Mapper*

Representa o custo total de execução da *Function Mapper* para todo o *input split*. A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.19.

$$T_{m5} = MapInput * CEF * MapComplex * TasksSimultneas \tag{5.19}$$

T_{m6}: *Sort*

Representa o custo de CPU na ordenação interna dos dados. É obtido pela Equação 5.20.

$$T_{m6} = SortCef * MORNumber * \left(\log\left(MORNumber\right) - \log\left(\frac{MapOutput}{SortBufferSize}\right)\right) \tag{5.20}$$

T_{m7}: *Merge*

Representa o custo interno para o *merge* dos arquivos temporários. A equação foi ajustada visando considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.21.

$$T_{m7} = SortCef * TasksSimultneas * MORNumber * \log\left(\frac{MapOutput}{SortBufferSize}\right) \tag{5.21}$$

T_{m8}: Serialização de dados

Representa o custo para escrever dados para o *sort buffer* e para serializar os dados para o fluxo de saída padrão. É denotado pela Equação 5.22.

$$T_{m8} = SerCef * MORNumber * \left(1 + \left[\log_{MergeFactor} r\right]\right) \tag{5.22}$$

Onde ***r*** representa o número de arquivos para *merge* na primeira rodada da serialização, sendo calculado pela Equação 5.23.

$$r = max\left(1, \left[\frac{MapOutput}{(SortBufferSize * IndexPercent)}\right], \left[\frac{(MORNumber * IndexSize)}{(SortBufferSize * IndexPercent)}\right]\right) \tag{5.23}$$

T_{m9}: Leitura de dados

Representa o custo de ler dados nos estágios *sort* e *spill*. A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É obtido pela Equação 5.24.

$$T_{m9} = MapOutput * \left(\frac{\log_k r}{\left(\frac{SeqRead}{TasksSimultaneas} \right)} \right) \tag{5.24}$$

T_{m10}: Escrita de dados para o disco

Representa o custo de escrever dados nos estágios *sort* e *spill*. A equação foi ajustada visando considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.25.

$$T_{m10} = MapOutput * \left(\frac{1 + \log_k r}{\left(\frac{SeqRead}{TasksSimultaneas} \right)} \right) \tag{5.25}$$

Uma vez que os passos ocorrem de forma sequenciais, paralelos e sobrepostos, o custo do MapTask não poderia ser estimado simplesmente somando os elementos de T_{Map}. Desta forma, *Lin et al.* (2012)[52] definiu uma equação para estimar o tempo de execução do *MapTask* baseada no *pipeline* de processamento, conforme ilustrado na Figura 5.4, denotada por $T_{MapTime}$ e definida por meio da Equação 5.26.

$$T_{MapTime} \equiv T_{m1} + max((T_{m2}, T_{m3}, (T_{m4} + T_{m5})) + T_{m6} + max(T_{m7}, T_{m9}) + max(T_{m8}, T_{m10}) \tag{5.26}$$

5.4.2 Estimador de Tempo do *ReduceTask*

Para *Lin et al.* (2012)[52], o processamento do *ReduceTask* também ocorre por meio de dez passos, conforme ilustrado na Figura 5.5, onde pode ser observado que alguns passos ocorrem de forma sequencial e outros de forma paralela. Além disso, foi constatado que determinados passos são mais dependentes de certos recursos computacionais, como CPU, HD e Rede. Isso torna o modelo mais interessante, pois permite ajustar o *framework Hadoop* de acordo com os recursos disponíveis nos *hosts* que formam o *cluster*.

Segundo *Lin et al.* (2012)[52], o custo do *ReduceTask*, denotado por T_{Reduce}, de forma semelhante ao *MapTask*, é uma composição dos passos sequenciais e paralelos, definidos como um vetor, formado por uma lista ordenada finita com 10 objetos, indicando as possíveis variações dos valores dos parâmetros, conforme demonstrado na Equação 5.27.

$$T_{Reduce} = (T_{r1}, T_{r2}, T_{r3}, T_{r4}, T_{r5}, T_{r6}, T_{r7}, T_{r8}, T_{r9}, T_{r10}) \tag{5.27}$$

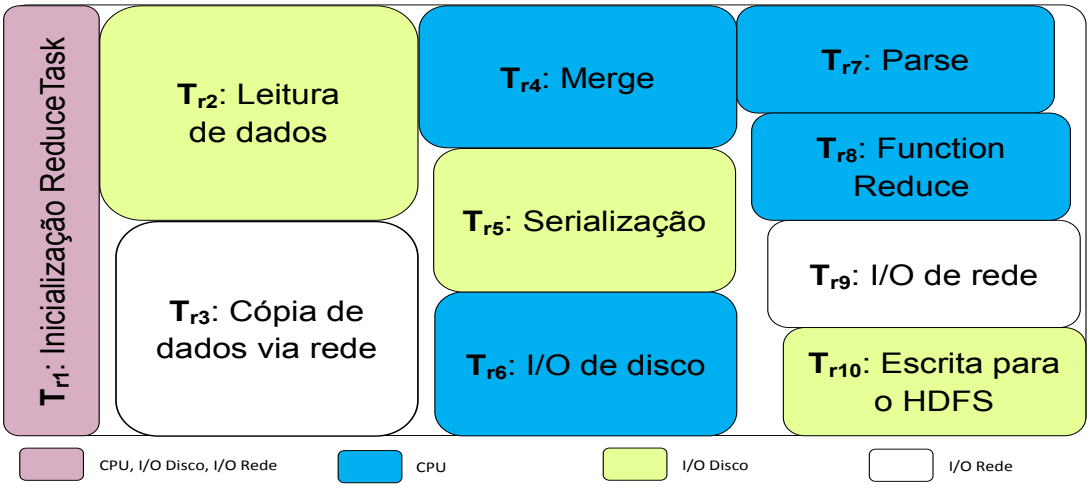

Figura 5.5: *Pipeline* de execução do *ReduceTask*, adaptada de *Song et al.* (2013)[74].

A seguir é apresentado um resumo de cada um desses passos e as equações utilizadas para definir seus valores. É válido salientar que os parâmetros utilizados nessas equações foram previamente definidos por meio da tabelas 5.1, 5.2 e 5.3. Além disso, algumas equações foram ajustadas na adaptação para este trabalho, visando considerar o compartilhamento de recursos computacionais por *tasks* executadas em paralelo no mesmo *TaskTracker*.

R_{m1}: Inicialização do *ReduceTask*

Representa o tempo de inicialização do *ReduceTask*, necessário para inicializar processos, leitura de arquivos internos, dentre outros. É calculado pela Equação 5.28.

$$T_{r1} = ReduceSysCost \tag{5.28}$$

T_{r2}: Leitura de dados

Representa o custo de I/O de disco na leitura de dados na fase *copy*. A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.28.

$$T_{r2} = ReduceInput * \left(\frac{(SegmentLength + \left(\frac{SeqRead}{TasksSimultaneas} \right) * AccessTime)}{SegmentLength * \left(\frac{SeqRead}{TasksSimultaneas} \right)} \right) \tag{5.29}$$

Onde *SegmentLength* é definido pela Equação 5.30.

$$SegmentLength = \left(\frac{MapOutput}{NumberOfReduce} \right) \tag{5.30}$$

T_{r3}: Cópia de dados via rede

Representa o custo de I/O de rede para copiar os dados, a partir dos *TaskTracker* que executaram os *MapTask*, na fase *copy*. A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.31.

$$T_{r3} = \left(\frac{ReduceInput}{\left(\frac{BandWidth}{TasksSimultaneas} \right)} \right) \tag{5.31}$$

T_{r4}: *Merge*

Representa o custo de CPU para o *merge* de arquivos na fase *copy*. A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.32.

$$T_{r4} = SortCef * RIRNumber * \log(NumberOfMap) * TasksSimultaneas \tag{5.32}$$

T_{r5}: Serialização

Representa o custo de serializar os dados na fase *copy*. É denotado pela Equação 5.33.

$$T_{r5} = SerCef * RIRNumber * P \tag{5.33}$$

Onde P indica o número de rodadas de I/O, podendo ser calculado por meio do Algoritmo 5.2.

```
import java.math.BigDecimal;

public class p {
   public static void main(String[] args) {
       double MORNumber, NumberOfReduce, ShuffleBuffer, ReduceJVM, ShuffleMergeThreshold;
       double ShuffleMergePercent, ReduceInBufferPercent, ShuffleBufferPercent;
       double ReduceInput, K, P, Q, M, MB, RB, SMT, RI, NumberOfMap, RIRLenght;
       BigDecimal F;
       P = 0;
       Q = 0;
       K = 100;
       F = 0;
       RIRLenght = 50;
       MORNumber = 722000;
       NumberOfReduce = 800;
       NumberOfMap = 8000;
       ReduceJVM = 419430400; //400MB
       ReduceInput = 161536500;
       M = MORNumber / NumberOfReduce;
       ShuffleBufferPercent = 0.70;
       ShuffleBuffer = (ReduceJVM * ShuffleBufferPercent);
       ShuffleMergePercent = 0.66;
       MB = (ShuffleMergePercent * ShuffleBuffer);
       ReduceInBufferPercent = 0.0;
       RB = (ReduceJVM * ReduceInBufferPercent);
       ShuffleMergeThreshold = 1000;
       SMT = ShuffleMergeThreshold;
       RI = ReduceInput;

       if (RI < MB & RI < RB & NumberOfMap < SMT) {
           P = 0;
       } else if ((RI / NumberOfMap) > (MB / SMT)) {
           if ((RI % MB) > RB) {
               F = BigDecimal.valueOf((RI / MB));
               Q = RI - (RI % MB) + RB;
           } else {
               F = BigDecimal.valueOf(RI / MB);
               Q = RI - (RI % MB);
           }
           P = (1 + ((Math.log(F.doubleValue())) / (Math.log(2 * K - 1))));
       } else {
           if ((RI % (SMT * M * RIRLenght)) > RB) {
               F = BigDecimal.valueOf(NumberOfMap / SMT);
               Q = RI + RB - (RI % (SMT * M * RIRLenght));
           } else {
               F = BigDecimal.valueOf(NumberOfMap / SMT);
               Q = RI - (RI % (SMT * M * RIRLenght));
           }
           P = (1 + ((Math.log(F.doubleValue())) / (Math.log(2 * K - 1))));
       }
       System.out.println("P = " + P + "  -  Q = " + (long) Q);
   }
}
```

Algoritmo 5.2: Código para encontrar P e Q, adaptado de *Lin et al.* (2012)[52].

T_{r6}: I/O de Disco

Representa o custo de I/O de disco na fase *copy*. A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.34.

$$T_{r6} = P * Q * \left[\frac{1}{\left(\frac{SeqRead}{TasksSimultaneas} \right)} + \frac{1}{\left(\frac{SeqWrite}{TasksSimultaneas} \right)} \right] \tag{5.34}$$

Onde Q indica a quantidade de dados sendo escrita em cada rodada, podendo ser calculado por meio do Algoritmo 5.2.

T_{r7}: *Parse*

Representa o custo de serializar dados para a *Function Reducer*. A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.35.

$$T_{r7} = ParCef * RIRNumber * TasksSimultaneas \tag{5.35}$$

T_{r8}: *Function Reducer*

Representa o custo de executar a *Function Reducer*. A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.36.

$$T_{r8} = ReduceInput * ReduceComplex * Cef * TasksSimultaneas \tag{5.36}$$

T_{r9}: I/O de rede

Representa o custo total de I/O de rede para transferir os dados, quando está gravando a saída do *ReduceTask* no HDFS. A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.37.

$$T_{r9} = HDFSReplica * \left[\frac{ReduceOutput}{\left(\frac{Bandwidth}{TasksSimultaneas} \right)} \right] \tag{5.37}$$

T_{r10}: Escrita para o HDFS

Representa o custo total de I/O de disco para escrever os dados de saída do *ReduceTask* no HDFS. A equação foi ajustada para considerar a execução simultânea de *tasks* no *TaskTracker*. É denotado pela Equação 5.38.

$$T_{r10} = HDFSReplica * \left[\frac{ReduceOutput}{\left(\frac{SeqWrite}{TasksSimultaneas} \right)} \right] \tag{5.38}$$

De forma semelhante ao *MapTask*, há processamentos que ocorrem de forma sequencial e outros em paralelo, logo o custo do *ReduceTask* não poderia ser estimado com a simples soma dos elementos de T_{Reduce}. Desta forma, *Lin et al.* (2012)[52] definiu uma equação para estimar o tempo de execução baseada no *pipeline* do processamento, conforme ilustrado na Figura 5.5, denotada por $T_{TReduceTime}$ e definida pela Equação 5.39.

$$T_{ReduceTime} \equiv T_{r1} + max \left\{ \left(\frac{T_{r2}}{min\left(ParallelCopies, NumberOfMap\right)} \right), T_{r3} \right\} + max\{(T_{r4} + T_{r5}), T_{r6}\} + max \left\{ \left(T_{r7} + T_{r8}\right), T_{r9}, \left(\frac{T_{r10}}{HDFSReplica} \right) \right\} \tag{5.39}$$

O trabalho de *Lin et al.* (2012)[52] apresentou meios para calcular o custo das *tasks* (*map/reduce*). Além disso, apresentou meios para calcular o custo de CPU, I/O de disco e I/O de rede. Todavia, o importante para este trabalho foram os estudos sobre a previsão de tempo das *tasks* (*map/reduce*), pois permitiram simular otimizações e estimar os novos tempos de conclusão dos processamento das *tasks (map/reduce)*. Possibilitando ao HCE_m simular as possíveis configurações de tamanhos do *cluster* para o *framework Hadoop*.

5.5 Considerações Finais

Neste capítulo foi apresentado o HCE_m, sua arquitetura interna e o detalhamento do funcionamento das atividades responsáveis por estimar o tamanho de um *cluster* para o *job Hadoop MapReduce* executar um *job* e processar uma determinada carga de trabalho em um intervalo de tempo aproximado.

No próximo capítulo será apresentado uma análise dos resultados de testes do HCE_m, realizados com uma plataforma real de virtualização, o serviço *Amazon Elastic MapReduce*[3], da plataforma *Amazon Web Services* [2].

Capítulo 6

Análise dos Resultados

Neste capítulo é apresentado um resumo da avaliação de desempenho do HCE_m. Os testes foram realizados por meio do serviço Amazon EMR [3] (*Amazon Elastic MapReduce*), da plataforma AWS [2] (Amazon Web Service).

6.1 Considerações Iniciais

Visando uma análise com dupla abordagem, foram realizadas duas baterias de testes, todas utilizando o *Amazon EMR*. Na primeira bateria de testes, foi criado um *cluster* no *Amazon EMR*, onde foram executados vários processamentos e comparados com as estimativas geradas pelo HCE_m. Na segunda bateria, os testes foram realizados de forma inversa, o *cluster* foi estimado pelo HCE_m, alocado no *Amazon EMR* e então realizado os processamentos.

Todos os testes realizados no Amazon EMR, utilizaram o perfil *m1.large*, que possui as configurações discriminadas na Tabela 6.1. Além disso, os *cluster* foram criados na estrutura de rede da AWS existente no Brasil.

Tabela 6.1: Configuração da Instância *m1.large* do AWS.

RAM	**CPU**	**HD**	**Bandwidth**	**Softwares Principais**
7.5 GB	4 (2 core x 2) 64-bit	840 GB (2 * 420 GB)	500 Mbps	Sistema operacional *Linux Red Hat* 4.4.6-3, *Java* 1.7 e *Hadoop* 1.0.3.

As bases de dados utilizadas nos testes foram originadas a partir de arquivos de *logs* do *Squid*[12], arquivos sequenciais de texto, compostos por linhas semelhantes ao ilustrado na Figura 6.1, porém com tamanhos diferentes para cada bateria de testes, 92 GB para a primeira bateria e 1 TB para a segunda bateria. Além disso, em ambas as baterias de testes, o HDFS estava configurado para a utilização de blocos de arquivos com 128 MB.

```
987548934.123 91 53.467.750.734 TCP_HIT/200 466 GET http://xxxxxxxxxx - 8 NONE - img.jpg
987548934.123 19 180.986.795.910 TCP_HIT/200 1146 GET http://xxxxxxx - 8 NONE - img.jpg
```

Figura 6.1: Exemplo de linhas do arquivo utilizado como entrada para os testes.

Durante a alocação de *cluster*, foi observado que o Amazon EMR implementa vários ajustes nos parâmetros de configuração do *framework Hadoop*. Desta forma, os parâmetros de configuração foram alterados para os valores padrões, recomendados pela Apache [14], pois os ajustes gerados pelo Amazon EMR poderiam gerar distorções nas análises do HCE_m, uma vez que alguns dos parâmetros ajustados pelo Amazon EMR não fazem parte do grupo de parâmetros utilizados no HCE_m.

No HCE_m, o *perfil do cluster Hadoop* (Seção 5.3.1) e as tarefas Análise do Hadoop e Análise do Job Objetivo, do subprocesso Estimador de *Cluster Hadoop* (Seção 5.3.2), foram processados uma única vez, pois foi utilizado o mesmo *Job Objetivo* e o mesmo perfil *m1.large* de *hosts* da AWS para as duas baterias de testes.

6.2 Gerando o Perfil do *Cluster Hadoop*

O primeiro passo para realizar as baterias de teste foi gerar o *perfil do cluster Hadoop*, conforme detalhado na Seção 5.3.1. Para isso, foi criado um *cluster* no *Amazon EMR*, denominado de *Cluster* Desenvolvimento, composto por quatro *hosts*, sendo um mestre e três escravos, conforme ilustrado na Figura 6.2.

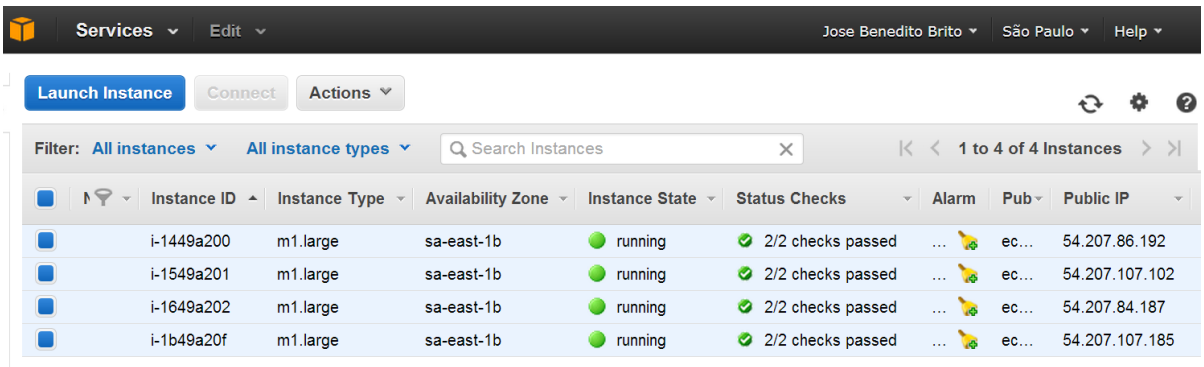

Figura 6.2: *Cluster* Desenvolvimento.

Nesta fase o objetivo foi gerar os valores para os parâmetros que formam o *perfil do cluster*, apresentados na Tabela 5.1, um dos pontos mais críticos do HCE_m, pois ele é a base de cálculos para as tarefas estimadoras, e necessita de várias interações para obter os valores para todos os parâmetros.

Foram implementados ajustes nas configurações do *framework Hadoop*, visando uma maior controle sobre o ambiente de testes. Desta forma, durante a geração do perfil do *cluster*, o *Hadoop* foi configurado para não executar *tasks* simultâneas, no mesmo *TaskTracker*, ainda que empiricamente fosse transparente que as configurações dos *hosts* suportariam uma maior carga de processamento, pois poderia haver distorções nos resultados observados, em função de que algumas *tasks* poderiam ser executadas isoladamente, e outras compartilhando os recursos do *TaskTracker*. Desta forma, foram alterados os valores dos parâmetros de configuração *mapred.tasktracker.map.tasks.maximum* e *mapred.tasktracker.reduce.tasks.maximum* para o valor 1.

Outra providência, para evitar desvios nos valores observados, foi processar diferentes bases de dados intercaladas, visando reduzir o uso de *cache* de disco nas leituras de dados. Além disso, não foram executados *jobs* em paralelo.

O *perfil do cluster Hadoop* é definido em função de testes realizados com o *Job Padrão* (Seção 5.3.1). Neste trabalho foi utilizado uma versão do *Word Count*, apresentado na Seção B.1 do Anexo B.

Alguns parâmetros do *perfil do cluster Hadoop* podem ser obtidos rapidamente, por meio de testes diretamente no sistema operacional ou consulta a arquivos. Desta forma, o parâmetro *BandWidth* foi obtido visualizando as configurações do perfil *m1.large* no AWS. Enquanto os parâmetros *SeqRead*, *SeqWrite* e *AccessTime* foram obtidos por meio de testes utilizando os comandos *time*[1] e *dd*[2] do sistema operacional *Linux*, que é utilizado nos *hosts* do AWS.

Os demais parâmetros necessitam da execução do *Job Padrão* no processamento de bases de dados criadas especificamente para permitir observar a evolução dos valores desses parâmetros. Nas próximas seções é apresentado um resumo de como esses parâmetros do *perfil do cluster Hadoop* foram gerados.

MapSysCost e ReduceSysCost

Para calcular esses parâmetros, o *Job Padrão* foi executado utilizando uma base de dados com tamanho zero. Assim, o tempo de execução das *tasks* é apenas o custo de inicializar o *MapTask* ou *ReduceTask*.

[1] Comando utilizado para medir o tempo de execução de programas.
[2] Comando utilizado para converter e copiar arquivos.

Cef

O *Cef* foi obtido em função da execução da *Map Function*. Para calcular o valor do *Cef*, a classe *Mapper* do *Job Padrão* foi modificada para não escrever a saída do processamento. Desta forma, o custo de execução das *MapTasks* seriam apenas o custo de inicialização da *task*, o custo de *parse* dos dados de entrada em objeto (*key/value*) e o custo de executar o *Map Function*. Assim, esse valor pode ser encontrado ao subtrair os valores de *MapSysCost* e o custo de *parse* dos dados de entrada.

ParCef

O *ParCef* é o custo de converter um registro em um objeto (*key/value*), que será encaminhado para a *Map Function* processar. Para obter esse parâmetro, o *Job Padrão* foi alterado para não realizar nenhum processamento, e, por consequência, não gravar nenhum valor na saída padrão. Desta forma, o *framework Hadoop* vai inicializar as *MapTask*, fazer o *parse* dos registros e finalizar. Assim sendo, o valor do *ParCef* poderá ser calculado ao subtrair o tempo do *MapSysCost*.

SerCef

O *SerCef* é o custo de CPU para serializar um registro. Para obter esse parâmetro foi necessário controlar os dados de saída da *MapTask*, para que houvesse apenas uma rodada de serialização, ou seja, todos os dados ficaram no *buffer* de memória até a conclusão da *Map Function*.

Assim sendo, foram criados arquivos com tamanhos de registros diferentes, mas que ao serem processados geravam saídas que podiam ser contidas no *buffer* de memória. Analisando o tempo total de serialização com os tamanhos de registros foi possível calcular o tempo médio de serialização de um registro.

SortCef

Para gerar o parâmetro *SortCef*, foi utilizado o *Job Sort* (Seção B.2 do Anexo B), que apenas escreve os dados de entrada na saída. Para esse *job*, foram criados dois conjuntos de arquivos. Um conjunto de quatro arquivos com conteúdo aleatório, registros de 8 bytes e contendo respectivamente 1024, 2048, 4096 e 8192 mil registros cada. O outro conjunto foi formado por arquivos com o mesmo conteúdo, mas devidamente ordenados.

O objetivo foi utilizar o *MapReduce* para ordenar os arquivos, utilizando seu mecanismo interno de ordenação. Desta forma, o valor de *SortCef* é estimado por meio das diferenças de custo de CPU observados no processamentos dos arquivos ordenados e os não ordenados.

Assim sendo, por meio de testes com o *Job Padrão*, aplicando os métodos expostos, foi gerado o perfil do *cluster Hadoop*, apresentado na Tabela 6.2.

Tabela 6.2: Perfil *Cluster* calculado.

Parâmetro	Valor
BandWidth (B/ms)	67109
SeqRead (B/ms)	55575
SeqWrite (B/ms)	54526
AccessTime (ms)	12
Cef	0,0000175083
ParCef (ms): pa1	0,0000872950
ParCef (ms): pa2	-0,0000004161
SerCef (ms): sa1	0,0427477209
SerCef (ms): sa2	-0,0000558799
SortCef (ms)	0,0003793633
MapSysCost (ms)	5000,00
ReduceSysCost (ms)	11000,00

6.3 Desenvolvimento dos Testes

Seguindo o fluxo de execução do HCE_m, os próximos passos seriam as tarefas *Análise do Hadoop* e *Análise do Job Objetivo*. A Seção B.3 do Anexo B, apresenta a codificação do *Job Objetivo* utilizado nas baterias de testes.

A análise do *Hadoop* foi realizada por meio da leitura e coleta dos parâmetros necessários, discriminados na Tabela 5.2, dos arquivos de configuração do *framework Hadoop*.

Para a análise do *Job Objetivo* foram geradas três amostras da base de dados de 92 GB, por meio da implementação do Algoritmo 5.1, com aproximadamente 1,2 GB. Além disso, foi criada uma variante do *Job Objetivo*, que não gravava a saída do processamento, definida como job'. Então, esses *jobs* foram executados algumas vezes, para cada amostra da base de dados, permitindo a coleta dos parâmetros constantes da Tabela 5.3, por meio da análise dos resultados dos processamentos desses *jobs*.

O objetivo dessas análises iniciais foi permitir ao HCE_m uma visão geral do *framework Hadoop*, por meio de seus respectivos parâmetros de configuração e do resultado de processamentos do *Job Objetivo* utilizando a amostra de dados. Nas próximas seções é apresentado um resumo dos resultados observados em cada bateria de testes.

6.3.1 Resultados Observados no *Cluster* Desenvolvimento

A primeira bateria de teste foi realizada no *Cluster Desenvolvimento*, que foi alocado com quatro *hosts*, sendo um mestre e três escravos. Além disso, foi utilizada uma base de dados com aproximadamente 92 GB, de texto sequencial.

Os valores dos parâmetros de configuração *mapred.tasktracker.map.tasks.maximum* e *mapred.tasktracker.reduce.tasks.maximum* foram alterados para 2, seus respectivos valores padrões, recomendados pela *Apache*.

Neste teste, o *Job Objetivo* foi executado no *Cluster Desenvolvimento* e obteve o resultado apresentado na Figura 6.3. Além disso, a Figura 6.4 apresenta um trecho do arquivo *job configuration*, arquivo que apresenta os valores dos parâmetros de configuração utilizados na execução de um *job MapReduce*, e os respectivos valores de alguns parâmetros que são utilizados no HCE_m.

```
Finished At: 26-Jun-2014 15:25:10 (2hrs, 31mins, 59sec)
------------------------------------------------------------------------------------
        Total   Total   Tasks
Kind    Tasks   Sucess  Failed  Start Time          Finish Time
Map       734     734       0   26-06-14 12:53:21  26-06-14 14:22:45 (1hrs, 29mins, 23sec)
Reduce     22      21       1   26-06-14 12:58:37  26-06-14 15:25:02 (2hrs, 26mins, 24sec)

   Counter                                  Map          Reduce           Total
   Launched reduce tasks                      0               0              23
   Launched map tasks                         0               0             734
   Data-local map tasks                       0               0             734
   Bytes Written                              0     715,109,692     715,109,692
   FILE_BYTES_READ               14,967,196,360  14,846,498,046  29,813,694,406
   HDFS_BYTES_READ               98,437,719,585               0  98,437,719,585
   FILE_BYTES_WRITTEN            29,711,807,886  14,847,012,291  44,558,820,177
   HDFS_BYTES_WRITTEN                         0     715,109,692     715,109,692
   Bytes Read                    98,437,643,249               0  98,437,643,249
   Map input records                658,174,446               0     658,174,446
   Reduce shuffle bytes                       0  14,846,590,278  14,846,590,278
   Map output bytes              13,530,148,902               0  13,530,148,902
   CPU time spent (ms)               26,999,050      11,606,010      38,605,060
   SPLIT_RAW_BYTES                       76,336               0          76,336
   Reduce input records                       0     658,174,446     658,174,446
   Reduce input groups                        0      36,565,247      36,565,247
   Reduce output records                      0      36,565,247      36,565,247
   Map output records               658,174,446               0     658,174,446
```

Figura 6.3: Resumo de Processamento do *Job Objetivo*, executado no *Cluster Desenvolvimento*.

Para viabilizar um estudo comparativo, o HCE_m foi carregado com os valores obtidos do *perfil do cluster Hadoop* e das tarefas *Análise do Hadoop* e *Análise do Job Objetivo*, utilizando a base com amostra dos dados. Em seguida, o *cluster* foi estimado para processar a base de dados de 92 GB, considerando que cada *TaskTracker* poderia executar no máximo dois *MapTask* e dois *ReduceTask*, que o *job* deveria utilizar 21 *ReduceTasks* e os recursos disponíveis do perfil *m1.large*, da AWS.

```
name value
io.sort.mb 100
io.sort.spill.percent 0.80
io.sort.record.percent 0.05
io.sort.factor 10
mapred.tasktracker.reduce.tasks.maximum 2
mapred.tasktracker.map.tasks.maximum 2
mapred.inmem.merge.threshold 1000
mapred.job.shuffle.merge.percent 0.66
mapred.job.shuffle.input.buffer.percent 0.70
mapred.job.reduce.input.buffer.percent 0.0
dfs.replication 3
dfs.block.size 134217728
mapred.reduce.parallel.copies 5
```

Figura 6.4: Trecho do *Job Configuration* do *Job Objetivo*, executado no *Cluster* Desenvolvimento.

Analisando os resultados do processamento do *Job Objetivo* e os valores estimados pelo HCE_m, resumidos na Figura 6.5, pode ser observado que a diferença de tempos entre o executado e o estimado foi de aproximadamente 12 minutos, que representa menos de 5% do tempo total. Além disso, considerando os recursos computacionais (CPU, RAM, HD) do perfil *m1.large*, fica explícito que o *Cluster Desenvolvimento* estaria superestimado para o *Job Objetivo* processar a base de 92 GB em 2:30 h. As estimativas geradas pelo HCE_m sugerem que os *hosts* necessitariam de apenas 50% do total de memória RAM e 20% da capacidade de armazenamento originais.

Item	Job Objetivo	HCEm
Quantidade de MapTasks	734	733
Tempo médio das MapTask	64 seg	45 seg
Tempo médio das ReduceTask	17 min	20 min
Tempo estimado para conclusão do job	02:31:59 h	2:44 h
Quantidade de hosts para os escravos	3	3
Quantidade de RAM dos escravos	7,5 GB	3,3 GB
Quantidade de CPU dos escravos	4	2
Tamanho do HD local	420 GB	33 GB
Tamanho do HD para o HDFS	420 GB	133 GB

Figura 6.5: Análise comparativa *Job Objetivo* x HCE_m.

6.3.2 Resultados Observados no *Cluster* Produção

Na segunda bateria de teste, foi utilizada uma abordagem diferente. O HCE_m estimou um *cluster*, onde o *Job Objetivo* foi executado, visando processar uma base de dados de 1 TB de dados, composto por 20 arquivos de 500 GB cada, em um dado intervalo de tempo, que foi definido em 200 minutos. Além disso, os recursos computacionais (CPU, RAM, HD) dos *hosts* foram limitadas às configurações do perfil *m1.large* da AWS.

Uma vez que seriam utilizados *hosts* do mesmo perfil *m1.large*, não foi necessário gerar um novo perfil de *cluster Hadoop*. Além disso, foram utilizados os dados das análises dos *Hadoop* e do *Job Objetivo* obtidos durante a primeira bateria de testes, pois o *job MapReduce* seria o mesmo.

Por meio do HCE_m, foi estimado a quantidade de memória RAM para cada *task* (*map/reduce*) e a quantidade de *tasks* (*map/reduce*) por *TaskTracker*, considerando o tempo proposto e os recursos computacionais do perfil *m1.large*, do AWS. Além disso, o HCE_m também foi utilizado para calcular a quantidade de *ReduceTasks*, em função do tempo médio de execução, estimado em aproximadamente 7 minutos.

Assim sendo, o HCE_m estimou o tamanho do *cluster* necessário para uma base de dados de 1 TB, gerando o seguinte resultado:

1. Tempo total proposto: 200 minutos;
2. Tamanho da *HeapSize/Task*: 400 MB;
3. Tempo estimado para o processamento dos *MapTasks*: 2:10 h;
4. Tempo estimado para o processamento dos *ReduceTasks*: 1:10 h;
5. Quantidade de *MapTasks*: 8000;
6. Quantidade de *ReduceTasks*: 800 (definido);
7. Tempo médio das *MapTask*: 84 segundos;
8. Tempo médio das *ReduceTask*: 453 segundos;
9. Quantidade de *hosts* para os escravos: 15;
10. Quantidade de RAM dos escravos: 6,3 GB;
11. Quantidade de CPU dos escravos: 3 GB;
12. Tamanho do HD local: 47 GB;
13. Tamanho do HD para o HDFS: 287 GB.

Em seguida, foi alocado um *cluster* na Amazon EMR, denominado de *Cluster Produção*, utilizando o perfil *m1.large*, composto por 16 *hosts*, conforme ilustrado na Figura 6.6, sendo um mestre e quinze escravos.

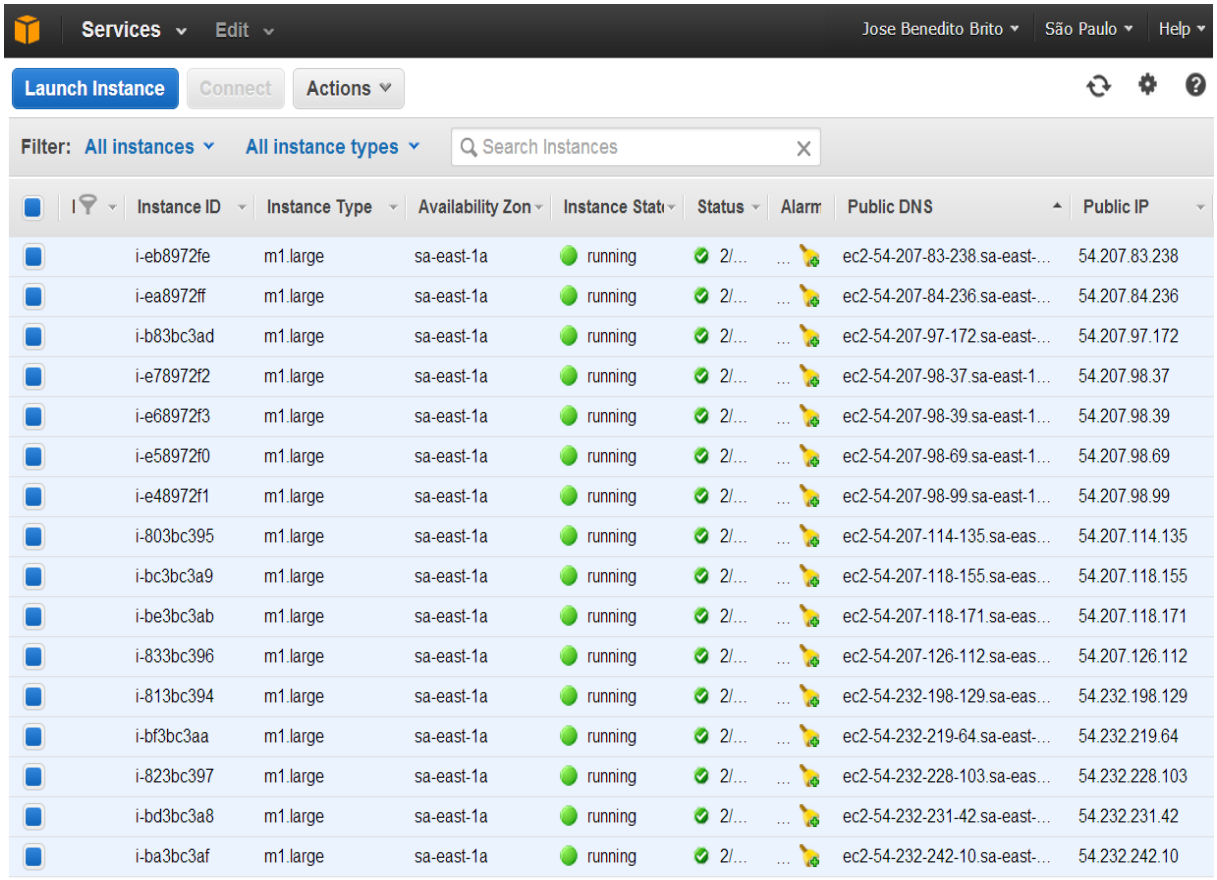

Figura 6.6: *Cluster Producao*.

Alguns parâmetros de configuração do *framework Hadoop* foram ajustadas de acordo as recomendações geradas pelo HCE_m, conforme ilustrado na Figura 6.7. Desta forma, o *Job Objetivo* executado no *Cluster Produção* obteve o resultado apresentado na Figura 6.8.

Analisando os resultados do processamento do *Job Objetivo* e os valores estimados pelo HCE_m, resumidos na Figura 6.9, é possível observar que o *cluster* estimado para o *framework Hadoop*, bem como os ajustes nos parâmetros de configuração, foram adequados para o processamento da carga de trabalho do *Job Objetivo*.

Considerando o tempo de processamento, percebe-se que a estimativa do HCE_m atingiu 86% do tempo efetivamente gasto no processamento. Todavia, essa diferença real tende a ser menor, pois 18 MapTasks e 58 ReduceTasks falharam.

```
name value

io.sort.factor 100
tasktracker.http.threads 40
mapred.job.shuffle.merge.percent 0.66
mapred.reduce.slowstart.completed.maps 0.80
dfs.block.size 134217728
mapred.map.tasks 8240
mapred.inmem.merge.threshold 0
mapred.reduce.tasks 800
io.sort.spill.percent 0.80
mapred.job.shuffle.input.buffer.percent 0.70
io.sort.record.percent 0,09
mapred.tasktracker.reduce.tasks.maximum 12
dfs.replication 3
mapred.job.reduce.input.buffer.percent 0.0
mapred.reduce.parallel.copies 5
io.sort.mb 280
io.file.buffer.size 131072
 mapreduce.input.num.files 20
 mapred.tasktracker.map.tasks.maximum 12
```

Figura 6.7: Trecho do *Job Configuration* do *Job Objetivo*, executado no *Cluster* Produção.

```
Finished At: 28-Jun-2014 00:54:52 (3hrs, 51mins, 33sec)
Analyse This Job
-------------------------------------------------------------------------------------------
          Total  Total   Tasks
 Kind     Tasks  Sucess  Failed  Start Time         Finish Time
 Map       8258   8240      18   27-06-14 21:03:29  27-06-14 23:45:24 (2hrs,41mins,54sec)
 Reduce     858    800      58   27-06-14 23:08:39  28-06-14 00:54:44 (1hrs,46mins,4sec)

 Counter                                    Map            Reduce            Total
 Launched reduce tasks                        0                 0              859
 Rack-local map tasks                         0                 0            3,363
 Launched map tasks                           0                 0            8,258
 Data-local map tasks                         0                 0            4,895
 Bytes Written                                0       300,860,875      300,860,875
 FILE_BYTES_READ                  1,712,934,535,320  136,513,383,336  1,849,447,918,656
 HDFS_BYTES_READ                  1,105,491,885,432                 0  1,105,491,885,432
 FILE_BYTES_WRITTEN                 273,364,828,151  136,533,173,296    409,898,001,447
 HDFS_BYTES_WRITTEN                           0       300,860,875      300,860,875
 Bytes Read                       1,105,491,056,900                 0  1,105,491,056,900
 Map input records                    5,937,764,100                 0    5,937,764,100
 Reduce shuffle bytes                         0  136,552,928,280    136,552,928,280
 Map output bytes                   124,638,906,680                 0    124,638,906,680
 CPU time spent (ms)                    236,570,130       121,266,280      357,836,410
 SPLIT_RAW_BYTES                            828,532                 0          828,532
 Reduce input records                         0     5,937,234,800      5,937,234,800
 Reduce output records                        0        14,462,285       14,462,285
 Map output records                   5,937,234,800                 0    5,937,234,800
```

Figura 6.8: Resumo de Processamento do *Job Objetivo*, executado no *Cluster Produção*.

Nos tempos individuais das *tasks*, foi observado um grande desvio na estimativa dos tempos das *ReduceTasks*, todavia essa divergência é natural, pois essas *tasks* iniciam seus processamentos após determinado percentual de *MapTasks* serem concluídos. Desta forma, as *ReduceTasks* ficam aguardando pela conclusão de alguns *MapTasks* para concluírem seus respectivos processamentos. Além disso, é importante salientar que as estimativas foram geradas a partir da análise de uma amostra, de outra base de dados, que em termos gerais equivale a apenas 0,07% da base processada.

Item	Job Objetivo	HCEm
Quantidade de MapTasks	8240	8000
Tempo médio das MapTasks	64 seg	84 seg
Tempo médio das ReduceTasks	1020 seg	453 seg
Tempo estimado para conclusão do job	03:51:33 h	03:20:00 h
Quantidade de hosts para os escravos	15	15
Quantidade de RAM dos escravos	7,5 GB	6,3 GB
Quantidade de CPU dos escravos	4	4
Tamanho do HD local	420 GB	47 GB
Tamanho do HD para o HDFS	420 GB	287 GB

Figura 6.9: Análise comparativa *Job Objetivo* x HCE$_m$.

6.4 Considerações Finais

Neste capítulo foi apresentado uma análise de testes do HCE$_m$ utilizando o serviço Amazon EMR [3], da plataforma *Amazon Web Service* [2], um ambiente real, utilizado por várias organizações. É válido salientar que não é possível ter uma visão ou controle das cargas de trabalho a que estão submetidos os *hosts* físicos dessa plataforma.

Outrossim, durante os laboratórios foi observado, em determinados horários, gargalos Amazon EMR, que em alguns casos invalidaram os testes em execução. A maioria do congestionamento ocorreu no período vespertino, entre 14 e 17 horas. Assim sendo, o desempenho de processamento é sensível á carga de trabalho dos *hosts* físicos dessa plataforma de nuvem. Ou seja, pode ocorrer desvios que eventualmente não ocorreriam em um ambiente controlado, todavia, esse foi o objetivo utilizar um ambiente real, com todas as suas intempéries, visando encontrar situações e dificuldades que acontecem no dia-a-dia dos ambientes de TI das organizações.

As análises dos resultados dos processamentos dos testes realizados comprovaram a efetividade do HCE$_m$, demonstrando que ele é capaz de estimar um *cluster* adequado para o *Hadoop* executar um *job* e processar determinada carga de trabalho em um dado intervalo de tempo.

Além disso, uma vez que as tarefas do HCE$_m$ trabalham de forma integrada, a partir do momento que todos os parâmetros de análise estão disponíveis, é possível realizar simulações para diversos cenários, baseadas em alterações nos parâmetros pertencente ao modelo, como o tamanho de bloco de arquivos, a quantidade de *tasks* por CPU, a quantidade de *tasks* por *TaskTracker* e alterações nos parâmetros de configuração do *framework Hadoop*. Além disso, ele permite identificar e evitar gargalos de I/O de disco nos processos de *merge* e *shuffle*.

Capítulo 7

Conclusões

Este trabalho foi inicialmente motivado por um debate sobre redução de custos de TI, relacionados ao processamento *batch* de grandes arquivos, realizados em plataformas centralizadas ou ambiente de *mainframes*. Onde vislumbrou-se que um *cluster* com o *framework Hadoop* possui um desempenho eficiente para o processamento de grandes bases de dados. Além disso, o *Hadoop* é escalável, confiável e sem custo de software para implementação.

Com o advento da computação em nuvem, um *cluster* com o *framework Hadoop* pode ser alocado em minutos, todavia, garantir que o *Hadoop* tenha um bom desempenho para realizar seu processamento apresenta vários desafios, como as necessidades de ajustes das configurações do *Hadoop* às cargas de trabalho, alocar um *cluster* apenas com os recursos necessários para realizar determinados processamentos e definir os recursos necessários para realizar um processamento em um intervalo de tempo conhecido.

Este trabalho analisou em detalhes a arquitetura do *Hadoop MapReduce*, sua utilização em ambientes virtualizados, metodologias para otimizar o seu desempenho, mecanismos de *benchmark* para medir o seu desempenho e modelos capazes de estimar o custo de processamento de cargas de trabalho.

Diante dos estudos realizados foi proposto um modelo que visa auxiliar na problemática da definição inicial do tamanho de um *cluster*, para utilização do *framework Hadoop* para o processamento de grandes bases de dados em um dado intervalo de tempo. Dessa forma, ele forneceria uma estratégia alternativa ao processamento *batch* de grandes arquivos, realizados em plataformas centralizadas.

O modelo proposto, chamado HCE$_m$, foi avaliado utilizando o serviço Amazon EMR da plataforma AWS, utilizando diferentes bases de dados. Em uma bateria de testes, buscou-se realizar um processamento na plataforma de nuvem, e utilizar o HCE$_m$ para analisar o *cluster*, quanto a recursos computacionais (CPU, RAM, HD), utilizado no processamento. Foi observado que o HCE$_m$ pode facilmente identificar problemas relacionados a superalocação de recursos computacionais.

Na outra bateria de testes, o HCE$_m$ foi utilizado para estimar o tamanho de um *cluster* para o *Hadoop* executar um *job* e processar uma determinada carga de trabalho em um dado intervalo de tempo. O *cluster* estimado foi alocado e realizado o processamento previsto, onde foi possível observar que o HCE$_m$ foi eficaz ao estimar um *cluster* adequado para o processamento da carga de trabalho, maximizando os recursos computacionais alocados. Além disso, o *cluster* estimado pelo HCE$_m$, realizou o processamento da carga de dados com 86% de precisão do tempo proposto.

Desta forma, o HCE$_m$ mostrou-se eficaz para auxiliar na problemática de definição do tamanho inicial de um *cluster* para o *Hadoop* realizar o processamento de uma determinada carga de trabalho em um dado tempo. Além disso, mostrou-se promissor para identificar problemas relacionados a superalocação de recursos computacionais em um *cluster Hadoop*.

Assim sendo, este trabalho apresentou um modelo capaz de estimar o tamanho de um *cluster*, ajustado de acordo com a carga de trabalho e o intervalo de tempo para o *framework Hadoop*. O modelo mostrou-se eficiente em suas estimativas, mas são necessários mais esforços para refinar o modelo e definir uma metodologia mais simples para gerar o *perfil do cluster*.

7.1 Contribuições

Este trabalho contribui para o domínio dos sistemas distribuídos de várias formas. Primeiro explorar a arquitetura do *Hadoop MapReduce* analisando o seu *pipeline* de execução, buscando mecanismos para abstrair essa complexa infraestrutura e facilmente prever o seu tempo de execução. Segundo explorar as possibilidades para prover uma camada de otimização leve, simples e funcional, visando reduzir os gargalos de I/O de discos no processamento das *tasks*. Terceiro, apresenta um modelo capaz de estimar o tamanho de um *cluster* adequado para a execução de *jobs MapReduce* para o processamento de determinada carga de trabalho em um dado intervalo de tempo. Finalmente, ele realiza a avaliação do modelo proposto.

Assim sendo, as principais contribuições específicas são as seguintes:

Exploração da arquitetura do *Hadoop MapReduce*

Este trabalho primeiro analisa e explica os vários processos existentes no *pipeline* de execução de um *job Hadoop MapReduce*. Ele brevemente discute várias questões relacionadas com as fases de execução, o uso de CPU, de memória RAM, de disco e como o *MapReduce* interage com o sistema de arquivos distribuídos do *Hadoop*.

Otimização do *Hadoop MapReduce*

Este trabalho discute meios para implementar uma camada de otimização leve, simples e aplicável à maioria dos *jobs Hadoop MapReduce*. Ele foca na redução dos gargalos de I/O de disco, procurando otimizar o uso de memória RAM para evitar que dados intermediários sejam gravados nos discos de dados, através da utilização de uma base de regras, permitindo que essa facilidade seja facilmente portada e utilizada.

Modelo Estimador de *Cluster Hadoop*

Este trabalho apresenta um modelo capaz de estimar o tamanho do *cluster* de acordo com a carga de trabalho e um dado intervalo de tempo. Ele estima a quantidade de *hosts* e suas configurações básicas de CPU, RAM e HD, permitindo que usuários possam rapidamente ter uma noção da infraestrutura de *cluster* que deve ser alocada para atender as suas respectivas necessidades.

Avaliação do Modelo Estimador de *Cluster Hadoop*

Apresenta uma avaliação do modelo proposto, utilizando o serviço Amazon EMR a renomada plataforma AWS. Permitindo um estudo do *Hadoop* em um ambiente real de produção, cenário que permite a observação de novas situações à medida que podem ocorrer diversos fatores não previstos em um ambiente simulado, influenciando diretamente as análises e ajustes necessários ao *Hadoop*. Além disso, a utilização de um ambiente real permite obter os melhores resultados finais, uma vez que os testes são realizados sem a necessidade de estimar outros valores.

7.2 Impactos na Empresa

O HCE$_m$ poderá ser utilizado como uma importante ferramenta de apoio a decisão, atuando principalmente em duas frentes:

- Evitar despesas e retrabalhos associados a subalocação e superalocação de recursos da nuvem privada, quando da alocação de um *cluster* para o *framework Hadoop* para realizar determinado processamento;
- Redução de gastos de TI, ao permitir identificar determinados processamentos *batch*, processados no ambiente de *mainframes*, com um elevado custo financeiro, que poderão ser migrados para um *cluster* com o *framework Hadoop*, na nuvem privada da empresa, uma plataforma existente e, portanto, sem novos custos para utilização, considerando os recursos computacionais disponíveis e os tempos necessários para o processamento, respectivamente.

7.3 Trabalhos Futuros

O HCE$_m$ mostrou-se eficaz em suas estimativas, mas o seu campo de atuação ainda está bastante limitado, pois a maioria das aplicações reais do *Hadoop MapReduce* utilizam a compactação de dados na saída das *MapTasks*. Desta forma, torna-se necessário ampliar o HCE$_m$ para incluir a compactação de dados em suas estimativas.

Neste trabalho, foram realizados esforços para considerar o impacto do compartilhamento de disco e de rede, em virtude da execução de *tasks* simultâneas no mesmo *TaskTracker*, mas ainda faltam estudos para definir ajustes semelhantes para as equações relacionadas com o uso intensivo de CPU, considerando a execução de *tasks* simultâneas por CPU e por *TaskTracker*.

Finalmente, implementar o modelo como um *framework*, capaz de realizar todos os passos iniciais de forma automatizada, onde uma rotina executada gera as bases de testes, executa os *jobs* e coleta todas as informações, abstraindo toda a complexidade da geração do *perfil do cluster*. Onde os usuários poderão de forma simples e transparente, simular e estimar diferentes cenários, de acordo com as suas respectivas necessidades. Além disso, esse *framework* deve realizar todas as alterações nos arquivos de configuração do *framework Hadoop*.

Referências

[1] Amazon.com. Amazon. http://www.amazon.com/, Jun 2014. Acesso em 14 Jun 2014. 14

[2] Amazon.com. Amazon web services (aws) - serviços de computação em nuvem. http://aws.amazon.com/pt/, Jun 2014. Acesso em 14 Abr 2014. 64, 65, 75

[3] Amazon.com. Aws - amazon elastic mapreduce (emr). http://aws.amazon.com/pt/elasticmapreduce/, Jun 2014. Acesso em 14 Jun 2014. 39, 64, 65, 75

[4] Dhruba Borthakur. Hdfs architecture guide. http://hadoop.apache.org/docs/r1.2.1/hdfs_design.pdf, 2008. Acesso em 25 Jul 2013. 15, 16, 17, 18, 19, 31

[5] Liang T. Chen and Deepankar Bairagi. Developing parallel programs - a discussion of popular models. http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation/ oss-parallel-programs-170709.pdf, 2010. Acesso em 10 Jun 2014. 9

[6] Intel Corporation. Intel. http://www.intel.com/, Jun 2014. Acesso em 14 Jun 2014. 14, 20, 35

[7] Oracle Corporation. Oracle. http://www.oracle.com/, Jun 2014. Acesso em 14 Jun 2014. 14, 20

[8] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordom Blair. *DISTRIBUTED SYSTEMS Concepts and Design*. Addison-Wesley, fifth edition, 2012. xi, 7, 8, 9, 10

[9] Mário Dantas. *COMPUTAÇÃO DISTRIBUÍDA DE ALTO DESEMPENHO - REDES, CLUSTERS E GRIDS COMPUTACIONAIS*. Axcel Books do Brasil Editora, 2005. 2, 4, 7, 9

[10] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association. 20

[11] M. Ebbers, J. Kettner, W. O'Brien, and B. Ogden. *Introduction to the New Mainframe - z/OS Basics*, volume 1. IBM Redbooks, third (march 2011) edition, Mar 2011. 1, 8

[12] Squid Software Foundation. Squid: Optimising web delivery. http://www.squid-cache.org/, Jun 2014. Acesso em 14 Jun 2014. 65

[13] The Apache Software Foundation. Apache hbase. http://hbase.apache.org/, Abr 2013. Acesso em 05 Jul 2013. 15

[14] The Apache Software Foundation. Hadoop. http://hadoop.apache.org/, Abr 2013. Acesso em 05 Jul 2013. xi, 2, 14, 18, 66

[15] The Apache Software Foundation. Hiive. http://hive.apache.org/, Abr 2013. Acesso em 05 Jul 2013. 15

[16] The Apache Software Foundation. Mahout. http://mahout.apache.org/, Abr 2013. Acesso em 05 Jul 2013. 15

[17] The Apache Software Foundation. Mapreduce tutorial. http://hadoop.apache.org/docs/stable/mapred_tutorial.pdf, Abr 2013. Acesso em 18 Jul 2013. 22

[18] The Apache Software Foundation. Apache abro! http://avro.apache.org/, Jun 2014. Acesso em 05 Jun 2014. 14

[19] The Apache Software Foundation. Apache ambari. http://ambari.apache.org/, Jun 2014. Acesso em 05 Jun 2014. 14

[20] The Apache Software Foundation. Apache cassandra. http://cassandra.apache.org/, Jun 2014. Acesso em 05 Jun 2014. 14

[21] The Apache Software Foundation. Apache pig! http://pig.apache.org/, Jun 2014. Acesso em 05 Jun 2014. 15

[22] The Apache Software Foundation. The apache software foundation. http://apache.org/, Jun 2014. Acesso em 14 Jun 2014. 2, 14, 20

[23] The Apache Software Foundation. Apache spark - lightning-fast cluster computing. http://spark.apache.org/, Jun 2014. Acesso em 05 Jun 2014. 15

[24] The Apache Software Foundation. Apache zookeeper. http://zookeeper.apache.org/, Jun 2014. Acesso em 05 Jun 2014. 15

[25] The Apache Software Foundation. Chukwa. http://hbase.apache.org/, Abr 2014. Acesso em 14 Abr 2013. 15

[26] Inc GitHub. Hibench. https://github.com/intel-hadoop/HiBench, Fev 2014. Acesso em 18 Mar 2014. 35, 50

[27] D. Heger. Hadoop performance tuning - a pragmatic & iterative approach. In *MeasureIT*, http://www.cmg.org/publications/measureit/2013-2/mit97/, Mar 2013. Acesso em 14 Set 2013. 31, 32, 47

[28] Herodotos Herodotou. *Automatic Tuning of Data-Intensive Analytical Workloads*. Doutorado, Duke University, 2012. 1

[29] Herodotos Herodotou and Shivnath Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. In *Proceedings of the VLDB Endowment*, volume 4, pages 1111–1122, 2011. 35

[30] Herodotos Herodotou and Shivnath Babu. A what-if engine for cost-based mapreduce optimization. *IEEE Data Eng. Bull.*, 36(1):5–14, 2013. 4, 35, 47

[31] Herodotos Herodotou, Nedyalko Borisov, and Shivnath Babu. Query optimization techniques for partitioned tables. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 49–60, New York, NY, USA, 2011. ACM. 35

[32] Herodotos Herodotou, Fei Dong, and Shivnath Babu. Mapreduce programming and cost-based optimization? crossing this chasm with starfish. In *Proceedings of the VLDB Endowment*, volume 4, pages 1446–1449, 2011. 2, 3, 4, 35

[33] Herodotos Herodotou, Fei Dong, and Shivnath Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 18:1–18:14, New York, NY, USA, 2011. ACM. 35, 36, 38

[34] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. Starfish: A self-tuning system for big data analytics. In *CIDR*, pages 261–272, 2011. 35, 47

[35] Shengheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 41–51, 2010. 35, 50

[36] IBM. z/os basic skills information center. http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp, 2008. Acesso em 15 Jul 2013. 1

[37] Facebook Inc. Facebook. http://www.facebook.com/, Jun 2014. Acesso em 14 Jun 2014. 14, 20

[38] Google Inc. Google. http://www.google.com/, Jun 2014. Acesso em 14 Jun 2014. 14, 20

[39] VMware Inc. A benchmarking case study of virtualized hadoop performance on vmware vsphere 5. http://www.vmware.com/files/pdf/techpaper/VMW-Hadoop-Performance-vSphere5.pdf, Out 2012. Acesso em 03 Ago 2013. 29, 30

[40] VMware Inc. Virtualizing apache hadoop. http://www.vmware.com/files/pdf/Benefits-of-Virtualizing-Hadoop.pdf, Jun 2012. Acesso em 03 Ago 2013. 30

[41] VMware Inc. Virtualized hadoop performance with vmware vsphere 5.1. http://www.vmware.com/files/pdf/techpaper/hadoop-vsphere51-32hosts.pdf, Abr 2013. Acesso em 05 Ago 2013. 29

[42] Yahoo! Inc. Yahoo! http://www.yahoo.com/, Jun 2014. Acesso em 14 Jun 2014. 14, 20

[43] INTEL. Optimizing hadoop* deployments. http://www.intel.com/content/www/us/en/cloud-computing/cloud-computing-optimizing-hadoop-deployments-paper.html, Out 2010. Acesso em 28 Jul 2013. 31, 33

[44] Masakuni Ishii, Jungkyu Han, and Hiroyuki Makino. Design and performance evaluation for hadoop clusters on virtualized environment. *Information Networking (ICOIN), 2013 International Conference on*, pages 244–249, Jan 2013. 30, 31, 48

[45] Shrinivas B. Joshi. Apache hadoop performance-tuning methodologies and best practices. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pages 241–242, New York, NY, USA, 2012. ACM. 33, 47

[46] Maryam Kontagora and Horacio Gonzalez-Velez. Benchmarking a mapreduce environment on a full virtualisation platform. *Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on*, pages 433–438, Fev 2010. 29, 30, 35

[47] Hermann Kopetz. *Real-Time Systems Design Principles for Distributed Embedded Applications*. Springer, second edition, 2011. 6

[48] Jack Li, Qingyang Wang, Deepal Jayasinghe, Junhee Park, Tao Zhu, and Calton Pu. Performance overhead among three hypervisors: An experimental study using hadoop benchmarks. *2013 IEEE International Congress on Big Data. BigData Congress 2013*, pages 8–16, 2013. 30

[49] Harold Lim, Yuzhang Han, and Shivnath Babu. How to fit when no one size fits. In *CIDR*, 2013. 4, 35

[50] Harold Lim, Herodotos Herodotou, and Shivnath Babu. Stubby: A transformation-based optimizer for mapreduce workflows. *PVLDB*, 5(11):1196–1207, 2012. 4, 35

[51] Harold C. Lim, Shivnath Babu, and Jeffrey S. Chase. Automated control for elastic storage. In *Proceedings of the 7th International Conference on Autonomic Computing*, ICAC '10, pages 1–10, New York, NY, USA, 2010. ACM. 35

[52] Xuelian Lin, Zide Meng, Chuan Xu, and Meng Wang. A practical performance model for hadoop mapreduce. In *Proceedings of the 2012 IEEE International Conference on Cluster Computing Workshops*, CLUSTERW '12, pages 231–239, Washington, DC, USA, 2012. IEEE Computer Society. xiii, 4, 36, 37, 42, 44, 47, 55, 56, 59, 62, 64

[53] Boris Lublinsky, Kevin T. Smith, and Alexey Yakubovich. *Professional Hadoop Solutions*. Jonh Wiley & Sons, Inc., 2013. 20, 22, 25

[54] International Business Machines. Ibm. http://www.ibm.com/, Jun 2014. Acesso em 14 Jun 2014. 2, 14, 20, 35

[55] Dan C. Marinescu. *CLOUD COMPUTING - Theory and Practice*. Morgan Kaufmann, first edition, 2013. 11

[56] Steven C. Markey. Deploy an openstack private cloud to a hadoop mapreduce environment. http://www.ibm.com/developerworks/cloud/library/cl-openstack-deployhadoop/, Out 2012. Acesso em 15 Ago 2013. xi, 22, 24, 27

[57] Tim Mather, Subra Kumaraswamy, and Shahed Latif. *Cloud Security and Privacy*. O'Reilly Media, Inc., 2009. 2, 10

[58] Mahesh Maurya and Sunita Mahajan. Performance analysis of mapreduce programs on hadoop cluster. *Information and Communication Technologies (WICT), 2012 World Congress on*, pages 505 – 510, Nov 2012. 32

[59] Donald Miner and Adam Shook. *MapReduce Design Patterns*. O'Reilly Media, Inc., first edition, 2013. 20, 33

[60] Oracle. Java. http://www.java.com/pt_BR/download/faq/whatis_java.xml, Abr 2014. Acesso em 02 Abr 2014. 14

[61] Srinath Perera and Thilina Gunarathne. *Hadoop MapReduce Cookbook*. Packt Publishing, 2013. 22

[62] Wichian Premchaiswadi and Walisa Romsaiyud. Optimizing and tuning mapreduce jobs to improve the large-scale data analysis process. *Int. J. Intell. Syst.*, 28(2):185–200, February 2013. 32

[63] George Reese. *Cloud Application Architectures - Building Applications and Infrastructure in the Cloud*. O'Reilly Media, Inc., first edition, 2009. 11

[64] Nikzad Babaii Rizvandi, Albert Y. Zomaya, Ali Javadzadeh Boloori, Javid. Taheri, and University of Sydney. *Preliminary Results: Modeling Relation Between Total Execution Time of Mapreduce Applications and Number of Mappers/Reducers*. School of Information Technologies, University of Sydney Sydney, 2011. 32

[65] Navin Sabharwal and Prashant Wali. *Cloud Capacity Management*. Apress, first edition, 2013. xi, 10, 11

[66] Eric Sammer. *Hadoop Operations*. O'Reilly Media, Inc., 2012. 21, 22

[67] Amit Sangroya, Damian Serrano, and Sara Bouchenak. Benchmarking dependability of mapreduce systems. *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 21–30, Out 2012. 35

[68] Amit Sangroya, Damian Serrano, and Sara Bouchenak. Mrbs: towards dependability benchmarking for hadoop mapreduce. *Proceeding Euro-Par'12 Proceedings of the 18th international conference on Parallel processing workshops*, pages 3–12, 2012. 34, 35

[69] Jeffrey Shafer, Scott Rixner, and Alan L. Cox. The hadoop distributed filesystem: Balancing portability and performance. *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*, pages 122–133, Mar 2010. 33

[70] Konstantin V. Shvachko. Hdfs scalability: the limits to growth. *LOGIN*, 35(2):6 – 16, Abr 2010. 32

[71] Konstantin V. Shvachko, Hairong Shva, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. *MSST '10 Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010. 15, 32

[72] Abraham Silbertschatz and Peter Baer Calvin. *OPERATING SYSTEM CONCEPTS*. Addison Wesley Longman, Inc., fifth edition, 1998. 7

[73] G. Somasundaram, Alok Shrivastava, and EMC. *Armazenamento e Gerenciamento de Informações - Como armazenar, gerenciar e proteger informações digitais*. Bookman, 2011. 1, 31, 48, 54

[74] Ge Song, Zide Meng, Fabrice Huet, Frederic Magoules, Lei Yu, and Xuelian Lin. A hadoop mapreduce performance prediction method. *2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 820–825, Nov 2013. xi, xiii, 4, 36, 46, 47, 56, 60

[75] Suresh Srinivas. An introduction to hdfs federation. http://hortonworks.com/blog/an-introduction-to-hdfs-federation/, Ago 2011. Acesso em 20 Set 2013. xi, 20

[76] K. Stanoevska and T. Wosniak. *Grid and Cloud Computing*. Springer, 2009. 7, 8, 9

[77] Jiaqi Tan, Xinghao Pan, Soila Kavulya, Rajeev Gandhi, and Priya Narashimhan. Mochi: Visual log-analysis based tools for debugging hadoop. *HotCloud'09 Proceedings of the 2009 conference on Hot topics in cloud computing*, 18, 2009. 34

[78] Andrew S. Tanenbaum and Maarten Van Steen. *DISTRIBUTED SYSTEMS - Principles and Paradigms*. PEARSON - Prentice Hall, second edition, 2007. 7, 8, 9, 14

[79] Tom White. *Hadoop The Definitive Guide*. O'Reilly Media, Inc., third edition, 2012. xi, 4, 15, 17, 18, 19, 21, 22, 23, 26, 27, 28, 33, 34, 48, 55

[80] T.W. Wlodarczyk, Yi Han, and Chunming Rong. Performance analysis of hadoop for query processing. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, pages 507–513, March 2011. 32, 47

[81] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, and Yun Tian. Improving mapreduce performance through data placement in heterocluster hadoop cluster. *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–9, Abr 2010. 33, 40

[82] Weijia Xu, Wei Luo, and N. Woodward. Analysis and optimization of data import with hadoop. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1058–1066, May 2012. 40

[83] Gaizhen Yang. The application of mapreduce in the cloud computing. In *Intelligence Information Processing and Trusted Computing (IPTC), 2011 2nd International Symposium on*, pages 154–156, Oct 2011. 31

[84] Yand Yang, Xiang Long, Xiaoqiang Dou, and Chengjian Wen. Impacts of virtualization technologies on hadoop. *Intelligent System Design and Engineering Applications (ISDEA), 2013 Third International Conference on*, pages 846–849, Jan 2013. 30

[85] YOYOCLOUDS. Hdfs architecture. http://yoyoclouds.wordpress.com/2011/12/15/hdfsarchitecture/, Dez 2011. Acesso em 18 Set 2013. xi, 16

[86] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. *OSDI'08 Proceedings of the 8th USENIX conference on Operating systems design and implementation*, pages 29–42, 2008. 33

[87] Zhuoyao Zhang, L. Cherkasova, and Boon Thau Loo. Getting more for less in optimized mapreduce workflows. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 93–100, May 2013. 32, 36, 47

[88] Zhuoyao Zhang, Ludmila Cherkasova, and Boon Thau Loo. Benchmarking approach for designing a mapreduce performance model. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 253–258, New York, NY, USA, 2013. ACM. 4, 36

[89] Paul C. Zikopoulos, Chris Eaton, Dirk deRoos, Thomas Deutsch, and George Lapis. *Undestanding Big Data - Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill, 2012. xi, 1, 19, 20, 21, 22, 24

Anexo A

Base de Regras

Tabela A.1: Base de Regras da Tarefa Otimizador de Tasks (Map/Reduce).

Parâmetro	**Regra**	**Comentário**
dfs.namenode.handler.count	50%	% do número de nós do *cluster*.
dfs.datanode.handler.count	25%	% do número de nós do *cluster*.
mapred.job.tracker.handler.count	5%	% do número de nós do *cluster*, o valor mínimo é 10.
tasktracker.http.threads	40/500	Valor é duplicado a cada 500 nós.
dfs.namenode.handler.count	50%	% do número de nós do *cluster*.
dfs.datanode.handler.count	25%	% do número de nós do *cluster*.
mapred.job.tracker.handler.count	5%	% do número de nós do *cluster*, valor mínimo é 10.
mapred.reduce.parallel.copies	Raiz(nr nós)	Raiz quadrada do número de nós. O valor mínimo é 5.
io.sort.mb	70%	% de *Heap Size*. 30% do valor é reservardo para a sobrecarga da *linguagem Java*.
io.sort.record.percent	16/(16+recordSize)	% de io.sort.mb para metadados dos registros.
io.sort.spill.percent	80%	% Limiar do uso de de IoSortMB para iniciar o processo de *spill*.
io.sort.factor	100	Valor sugerido por *White*(2012).
mapred.inmem.merge.threshold	-1	O valor negativo significa que não há limiar, por isso o textitspill é governado apenas por mapred.job.shuffle.merge.percent.
mapred.job.shuffle.input.buffer.percent	70%	% de *Heap Size*. 30% do valor é reservardo para a sobrecarga da *linguagem Java*.
mapred.job.shuffle.merge.percent	70%	Valor sugerido por *White* (2012).
mapred.job.reduce.input.buffer.percent	10%	Valor sugerido por *White*(2012).
io.file.buffer.size	131072	Valor sugerido por *White* (2012).

Anexo B

Jobs MapReduce Utilizados no HCE_m

B.1 Job Padrão

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package com.hadoop.mapreduce;
7
8
9  import org.apache.hadoop.conf.Configuration;
10 import org.apache.hadoop.fs.FileSystem;
11 import org.apache.hadoop.fs.Path;
12 import org.apache.hadoop.io.IntWritable;
13 import org.apache.hadoop.io.Text;
14 import org.apache.hadoop.mapreduce.Job;
15 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
16 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
17 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
18 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
19
20 public class WordCount {
21
22     public static void main(String[] args) throws Exception {
23
24         Path inputPath = new Path(args[0]);
25         Path outputDir = new Path(args[1]);
26
27         // Create configuration
28         Configuration conf = new Configuration(true);
29         // Delete output if exists
30         FileSystem fs = FileSystem.get(conf);
31         if (fs.exists(outputDir)) {
32             fs.delete(outputDir, true);
33         }
34
35         // Create job
36         Job job = new Job(conf, "SP-Com Processamento");
```

```
37          job.setJarByClass(WordCount.class);
38
39          // Setup MapReduce
40          job.setMapperClass(WordCountMapper.class);
41          job.setReducerClass(WordCountReducer.class);
42          job.setNumReduceTasks(3);
43
44          // Specify key / value
45          job.setOutputKeyClass(Text.class);
46          job.setOutputValueClass(IntWritable.class);
47
48          // Input
49          FileInputFormat.addInputPath(job, inputPath);
50          job.setInputFormatClass(TextInputFormat.class);
51
52          // Output
53          FileOutputFormat.setOutputPath(job, outputDir);
54          job.setOutputFormatClass(TextOutputFormat.class);
55
56          // Execute job
57          int code = job.waitForCompletion(true) ? 0 : 1;
58          System.exit(code);
59
60      }
61
62  }
```

Algoritmo B.1: Job Padrão.

B.1.1 Classe *Mapper* do *Job Padrão*

```
1   /*
2    * To change this license header, choose License Headers in Project Properties.
3    * To change this template file, choose Tools | Templates
4    * and open the template in the editor.
5    */
6
7   package com.hadoop.mapreduce;
8
9   import java.io.IOException;
10
11  import org.apache.hadoop.io.IntWritable;
12  import org.apache.hadoop.io.Text;
13  import org.apache.hadoop.mapreduce.Mapper;
14
15  public class WordCountMapper extends
16          Mapper<Object, Text, Text, IntWritable> {
17
18      private final IntWritable ONE = new IntWritable(1);
19      private Text word = new Text();
20
21      public void map(Object key, Text value, Context context)
22              throws IOException, InterruptedException {
23
```

```
24          String[] csv = value.toString().split(",");
25          for (String str : csv) {
26              word.set(str);
27              context.write(word, ONE);
28          }
29      }
30  }
```

Algoritmo B.2: Classe Mapper do Job Padrão.

B.1.2 Classe *Reducer* do *Job Padrão*

```
1   /*
2    * To change this license header, choose License Headers in Project Properties.
3    * To change this template file, choose Tools | Templates
4    * and open the template in the editor.
5    */
6
7   package com.hadoop.mapreduce;
8
9   import java.io.IOException;
10
11  import org.apache.hadoop.io.IntWritable;
12  import org.apache.hadoop.io.Text;
13  import org.apache.hadoop.mapreduce.Reducer;
14
15  public class WordCountReducer extends
16          Reducer<Text, IntWritable, Text, IntWritable> {
17
18      public void reduce(Text text, Iterable<IntWritable> values, Context context)
19              throws IOException, InterruptedException {
20          int sum = 0;
21          for (IntWritable value : values) {
22              sum += value.get();
23          }
24          context.write(text, new IntWritable(sum));
25      }
26  }
```

Algoritmo B.3: Classe Reducer do Job Padrão.

B.2 Job Sort

```
1  package com.unb.hadoop;
2
3  import org.apache.hadoop.conf.Configuration;
4  import org.apache.hadoop.fs.FileSystem;
5  import org.apache.hadoop.fs.Path;
6  import org.apache.hadoop.io.IntWritable;
7  import org.apache.hadoop.io.Text;
8  import org.apache.hadoop.mapreduce.Job;
9  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
10 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
11 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
12 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
13
14 /**
15  *
16  * @author Jobe
17  */
18 public class Sort {
19
20     public static void main(String[] args) throws Exception {
21
22         if (args.length != 3) {
23             System.err.println("Uso: IpJob <Label> <input path> <output path>");
24         }
25
26         String label = args[0];
27         Path inputPath = new Path(args[1]);
28         Path outputDir = new Path(args[2]);
29
30         // Create configuration
31         Configuration conf = new Configuration(true);
32         // Delete output if exists
33         FileSystem fs = FileSystem.get(conf);
34         if (fs.exists(outputDir)) {
35             fs.delete(outputDir, true);
36         }
37
38         // Create job
39         Job job = new Job(conf, "Sort-Proc - " + label);
40         job.setJarByClass(Sort.class);
41 //         job.setNumReduceTasks(3);
42
43         // Setup MapReduce
44         job.setMapperClass(MapSort.class);
45         job.setReducerClass(RedSort.class);
46         //job.setNumReduceTasks(1);
47
48         // Specify key / value
49         job.setOutputKeyClass(Text.class);
50         job.setOutputValueClass(IntWritable.class);
51
52         // Input
53         FileInputFormat.addInputPath(job, inputPath);
54         job.setInputFormatClass(TextInputFormat.class);
```

```
55
56          // Output
57          FileOutputFormat.setOutputPath(job, outputDir);
58          job.setOutputFormatClass(TextOutputFormat.class);
59
60          // Execute job
61          int code = job.waitForCompletion(true) ? 0 : 1;
62          System.exit(code);
63
64      }
65
66  }
```

Algoritmo B.4: Job Sort.

B.2.1 Classe *Mapper* do *Job Sort*

```
1   package com.unb.hadoop;
2
3   import java.io.IOException;
4   import org.apache.hadoop.io.Text;
5   import org.apache.hadoop.mapreduce.Mapper;
6   import org.apache.hadoop.io.IntWritable;
7
8   /**
9    *
10   * @author Jobe
11   */
12  public class MapSort extends Mapper<Object, Text, Text, IntWritable> {
13
14      @Override
15      public void map(Object key, Text value, Context context) throws IOException,
            InterruptedException {
16          context.write(value, new IntWritable(0));
17      }
18  }
```

Algoritmo B.5: Classe Mapper do Job Sort.

B.2.2 Classe *Reducer* do *Job Sort*

```
1   package com.unb.hadoop;
2
3   import java.io.IOException;
4   import org.apache.hadoop.io.IntWritable;
5   import org.apache.hadoop.io.Text;
6   import org.apache.hadoop.mapreduce.Reducer;
7
8   /**
9    *
10   * @author Jobe
11   */
```

```
12 public class RedSort extends Reducer<Text, IntWritable, Text, IntWritable> {
13
14     protected void reduce(Text key, IntWritable values, Context context) throws
           IOException, InterruptedException {
15         context.write(key, new IntWritable(0));
16     }
17 }
```

Algoritmo B.6: Classe Reducer do Job Sort.

B.3 Job Objetivo

```
1  package com.unb.hadoop;
2
3  import org.apache.hadoop.conf.Configuration;
4  import org.apache.hadoop.fs.FileSystem;
5  import org.apache.hadoop.fs.Path;
6  import org.apache.hadoop.io.IntWritable;
7  import org.apache.hadoop.io.Text;
8  import org.apache.hadoop.mapreduce.Job;
9  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
10 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
11 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
12 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
13
14 /**
15  *
16  * @author Jobe
17  */
18 public class IpJob {
19
20     public static void main(String[] args) throws Exception {
21
22         if (args.length != 2) {
23             System.err.println("Uso: IpJob <input path> <output path>");
24         }
25
26         Path inputPath = new Path(args[0]);
27         Path outputDir = new Path(args[1]);
28
29         // Create configuration
30         Configuration conf = new Configuration(true);
31         // Delete output if exists
32         FileSystem fs = FileSystem.get(conf);
33         if (fs.exists(outputDir)) {
34             fs.delete(outputDir, true);
35         }
36
37         // Create job
38         Job job = new Job(conf, "IpCount-Process");
39         job.setJarByClass(IpJob.class);
40
41         // Setup MapReduce
42         job.setMapperClass(MapperIP.class);
```

```
43          job.setReducerClass(ReducerIP.class);
44          job.setNumReduceTasks(800);
45
46          // Specify key / value
47          job.setOutputKeyClass(Text.class);
48          job.setOutputValueClass(IntWritable.class);
49
50          // Input
51          FileInputFormat.addInputPath(job, inputPath);
52          job.setInputFormatClass(TextInputFormat.class);
53
54          // Output
55          FileOutputFormat.setOutputPath(job, outputDir);
56          job.setOutputFormatClass(TextOutputFormat.class);
57
58          // Execute job
59          int code = job.waitForCompletion(true) ? 0 : 1;
60          System.exit(code);
61
62      }
63  }
```

Algoritmo B.7: Job Objetivo.

B.3.1 Classe *Mapper* do *Job Objetivo*

```
1   package com.unb.hadoop;
2
3   import com.unb.hadoop.util.Utils;
4   import java.io.IOException;
5   import org.apache.hadoop.io.IntWritable;
6   import org.apache.hadoop.io.Text;
7   import org.apache.hadoop.mapreduce.Mapper;
8
9   /**
10   *
11   * @author Jobe
12   */
13  public class MapperIP extends Mapper<Object, Text, Text, IntWritable> {
14
15      private Text textIp = new Text();
16
17      @Override
18      public void map(Object key, Text value, Context context) throws IOException,
            InterruptedException {
19          String aux = Utils.parseLineGetIp(value.toString());
20          if (aux != null) {
21              textIp.set(aux.concat(";"));
22              context.write(textIp, new IntWritable(1));
23          }
24      }
25  }
```

Algoritmo B.8: Classe Mapper do Job Objetivo.

B.3.2 Classe *Reducer* do *Job Objetivo*

```
1  package com.unb.hadoop;
2
3  import java.io.IOException;
4  import org.apache.hadoop.io.IntWritable;
5  import org.apache.hadoop.io.LongWritable;
6  import org.apache.hadoop.io.Text;
7  import org.apache.hadoop.mapreduce.Reducer;
8
9  /**
10  *
11  * @author Jobe
12  */
13 public class ReducerIP extends Reducer<Text, IntWritable, Text, LongWritable> {
14
15     private LongWritable result = new LongWritable();
16
17     @Override
18     protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws
           IOException, InterruptedException {
19         int count = 0;
20
21         for (IntWritable value : values) {
22             count++;
23         }
24         result.set(count);
25         context.write(key, result);
26     }
27 }
28
29 \subsection{Classe \textit{Reducer} do \textit{Job Objetivo}}
30
31 \begin{lstlisting}[caption=Classe Reducer do Job Objetivo.]
```

Algoritmo B.9: Classe Reducer do Job Objetivo.

B.3.3 Classe *Utils* do *Job Objetivo*

```
1  package com.unb.hadoop.util;
2
3  import com.unb.hadoop.Ip;
4  import com.unb.hadoop.Log;
5
6  /**
7  *
8  * @author Jobe
9  */
10 public class Utils {
11
12     public static final String DELIMITADOR = " ";
13     //Linha:
14     //987548934.123 19 73.310.233.314 TCP_HIT/200 4771 GET http://europe.cnn.com EUROPE/
           potd/2001/04/17/tz.pullitzer.ap.jpg - 8 NOME/- image.jpg
```

```
15
16      public static Log parseLine(String line) {
17          try {
18              String split[] = line.split(DELIMITADOR);
19              Log log = new Log(split[0], split[1], split[2], split[3], split[4], split[5],
                    split[6], split[7], split[8], split[9], split[10]);
20              return log;
21          } catch (Exception e) {
22              throw new RuntimeException(e.getMessage());
23          }
24      }
25
26      public static String parseLineGetIp(String line) {
27          try {
28              String split[] = line.split(DELIMITADOR);
29              String logIp = split[2].toString();
30              return logIp;
31          } catch (Exception e) {
32              return null;
33  //              throw new RuntimeException(e.getMessage());
34          }
35      }
36
37      public static Ip parseLineIp(String line) {
38          try {
39              Ip ip = new Ip();
40              if (line.length() > 370) {
41                  ip = new Ip(line.substring(350, 370).trim());
42              }
43              return ip;
44          } catch (Exception e) {
45              throw new RuntimeException(e.getMessage());
46          }
47      }
48
49  }
```

Algoritmo B.10: Classe Reducer do Job Objetivo.