UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANÁLISE DE ESCALABILIDADE DE APLICAÇÕES HADOOP/MAPREDUCE POR MEIO DE SIMULAÇÃO

FABIANO DA GUIA ROCHA

ORIENTADOR: PROF. DR. HERMES SENGER

São Carlos - SP
Fevereiro/2013

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANÁLISE DE ESCALABILIDADE DE APLICAÇÕES HADOOP/MAPREDUCE POR MEIO DE SIMULAÇÃO

FABIANO DA GUIA ROCHA

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

São Carlos - SP
Fevereiro/2013

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

R672ae

Rocha, Fabiano da Guia.
Análise de escalabilidade de aplicações Hadoop/Mapreduce por meio de simulação / Fabiano da Guia Rocha. -- São Carlos : UFSCar, 2013.
78 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2013.

1. Computação. 2. Sistemas multiprocessados. 3. Escalabilidade. 4. Modelo mapreduce. I. Título.

CDD: 004 (20ª)

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

"Análise de Escalabilidade de Aplicações Hadoop/Mapreduce por meio de Simulação"

Fabiano da Guia Rocha

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Hermes Senger
(Orientador - DC/UFSCar)

Prof. Dr. Hélio Crestana Guardia
(DC/UFSCar)

Profa. Dra. Líria Matsumoto Sato
(POLI/USP)

São Carlos
Fevereiro/2013

Dedico este trabalho à minha família.

Sem o incentivo, carinho e compreensão de meus pais, irmãs e sobrinho nada disso seria possível.

AGRADECIMENTO

Agradeço primeiramente a Deus, pela saúde, fé e perseverança que tem me dado.

À minha família Pedro, Odália, Karoline, Kamille e Kauã pelo companheirismo, confiança e apoio nas horas de tribulação. Em especial aos meus pais Pedro e Odália, a quem honro o esforço com o qual me proporcionaram condições de galgar êxito na sociedade letrada.

Ao meu cunhado Jordão Carvalho pelos momentos de descontração e incentivo.

Aos colegas Diego Desani, Carlos Spinetti, Samuel Silva, Pedro Bignatto, Douglas Santana, Murilo Nogueira e Augusto Cesar por compartilhar experiências e, também pela convivência diária no DC-UFSCar.

Aos meus amigos Thiago Rodrigues, Jacqueline Tenório, Daniel Schafer, Juciara Nepomuceno, Barbara Castanheira e Vinicius Ferreira pela excepcional convivência em São Carlos, e em especial as companheiras de todas as horas Eliane Mahl e Mariane Nan, por compartilhar o verdadeiro sentido da palavra Amigo e pelo incentivo a busca de novos conhecimentos.

Aos meus avós Lázaro, Ernesta, Margarida e a Tia Eva pelas orações e carinho incondicional.

À Tia Maria, pelo incentivo à minha profissão em fazer educação, sabendo dos desafios do educador no contexto atual.

Ao professor e orientador Hermes Senger, pela sabedoria e dedicação, que muito contribuiu com minha formação acadêmica e profissional.

“Eu na trilha incerta dos meus passos, mergulhado no destino imenso dos meus sonhos vou percorrendo as estradas do mundo.” (Pe. Fábio de Melo)

RESUMO

Durante os últimos anos, houve um significativo crescimento na quantidade de dados processados diariamente por companhias, universidades e outras instituições. Mapreduce é um modelo de programação e um *framework* para a execução de aplicações que manipulam grandes volumes de dados em máquinas compostas por milhares de processadores ou núcleos. Atualmente, o Hadoop é a implementação como *software* livre de Mapreduce mais largamente adotada. Embora existam relatos na literatura sobre o uso de aplicações Mapreduce em plataformas com cerca de quatro mil núcleos processando dados da ordem de dezenas de petabytes, o estudo dos limites de escalabilidade não foi esgotado e muito ainda resta a ser estudado. Um dos principais desafios no estudo de escalabilidade de aplicações Mapreduce é o grande número de parâmetros de configuração da aplicação e do ambiente Hadoop. Na literatura há relatos que mencionam mais de 190 parâmetros de configuração, sendo que 25 podem afetar de maneira significativa o desempenho da aplicação. Este trabalho contém um estudo sobre a escalabilidade de aplicações Mapreduce executadas na plataforma Hadoop. Devido ao número limitado de processadores disponíveis, adotou-se uma abordagem que combina experimentação e simulação. A experimentação foi realizada em um *cluster* local de 32 nós (com 64 processadores), e para a simulação empregou-se o simulador MRSG (*MapReduce Over SimGrid)*. Como principais resultados, foram identificados os parâmetros de maior impacto no desempenho e na escalabilidade das aplicações. Esse resultado foi obtido por meio de simulação. Além disso, apresentou-se um método para a calibração do simulador MRSG, em função de uma aplicação representativa escolhida como *benchmark*. Com o simulador calibrado, avaliou-se a escalabilidade de uma aplicação bem otimizada. O simulador calibrado permitiu obter uma predição sobre a escalabilidade da aplicação para uma plataforma com até 10 mil nós.

Palavras-chave: sistemas multiprocessados, escalabilidade, Mapreduce.

ABSTRACT

During the last years we have witnessed a significant growing in the amount of data processed in a daily basis by companies, universities, and other institutions. Many use cases report processing of data volumes of petabytes in thousands of cores by a single application. MapReduce is a programming model, and a framework for the execution of applications which manipulate large data volumes in machines composed of thousands of processors/cores. Currently, Hadoop is the most widely adopted free implementation of MapReduce. Although there are reports in the literature about the use of MapReduce applications on platforms with more than one hundred cores, the scalability is not stressed and much remain to be studied in this field. One of the main challenges in the scalability study of MapReduce applications is the large number of configuration parameters of Hadoop. There are reports in the literature that mention more than 190 configuration parameters, 25 of which are known to impact the application performance in a significant way. In this work we study the scalability of MapReduce applications running on Hadoop. Due to the limited number of processors/cores available, we adopted a combined approach involving both experimentation and simulation. The experimentation has been carried out in a local cluster of 32 nodes, and for the simulation we have used MRSG (*MapReduce Over SimGrid*). In a first set of experiments, we identify the most impacting parameters in the performance and scalability of the applications. Then, we present a method for calibrating the simulator. With the calibrated simulator, we evaluated the scalability of one well-optimized application on larger clusters, with up to 10 thousands of nodes.

Keywords: multiprocessor systems, scalability, MapReduce.

LISTA DE FIGURAS

Figura 2.1: Modelo de fluxo de dados Mapreduce, adaptada de White (2009). 20

Figura 2.2: Modelo de execução Mapreduce, adaptada de DEAN e GHEMAWAT (2004). 25

Figura 2.3: Representação do fluxo de dados do Hadoop, adaptada de HADOOP (2008). 30

Figura 2.4: Arquitetura do *Hadoop Distributed File System*, adaptada de HADOOP (2008). 31

Figura 2.5: Representação do *JobTracker* e do *TaskTracker*, adaptada de O´MALLEY (2011). 32

Figura 2.6: Arquitetura geral do SimGrid versão 3.8.1, adaptada de SIMGRID (2012). 35

Figura 2.7: Exemplo de arquivo deployment.xml para quatro nós, adaptada de SimGrid (2012). 37

Figura 2.8: Exemplo de arquivo *platform.xml,* adaptada de SimGrid (2012). 38

Figura 2.9: Exemplo do arquivo *mrsg.conf*. 40

Figura 2.10: Exemplo de saída do MRSG. 40

Figura 3.1: Percentual de impacto dos fatores para o primeiro experimento. 51

Figura 3.2: Percentual de impacto dos fatores para o segundo experimento. 52

Figura 4.1: Gráfico de *Makespan* da aplicação de índices invertidos........59

Figura 4.2: Gráfico de *Speedup* da Fase *Map*, Fase *Reduce* e Total. 60

Figura 4.3: Comparação entre o Makespan Real e o Makespan Simulado. 64

Figura 4.4: Gráfico de M*akespan* com 26 tarefas *reduce* e 100 mil tarefas map. 66

Figura 4.5: Gráfico de Speedup com 26 tarefas *reduce* e 100 mil tarefas *map*........67

Figura 4.6: Gráfico de Eficiência com 26 tarefas *reduce* e 100 mil tarefas *map*........67

Figura 4.7: Gráfico de *Speedup* com 676 tarefas *reduce* e 100 mil tarefas *map*........70

Figura 4.8: *Makespan, Speedup* e Eficiência com 676 tarefas *reduce* e *switch infiniband*........71

LISTA DE TABELAS

Tabela 2.1: Correspondência entre o nível do fator e a resposta. 44

Tabela 3.1: Fatores e níveis do primeiro experimento 2^k fatorial.............................. 50

Tabela 3.2: Percentual de impacto dos fatores sobre a eficiência do primeiro experimento... 51

Tabela 3.3: Fatores e níveis do segundo experimento 52

Tabela 3.4: Percentual de impacto de cada fator sobre a eficiência do segundo experimento... 53

Tabela 4.1: Tempo de Execução.. 58

Tabela 4.2: *Speedup* da Fase *Map*, Fase *Reduce* e Total. 59

Tabela 4.3: Simulação da Fase *Map*. 62

Tabela 4.4: Simulação da Fase *Reduce.* 62

Tabela 4.5: Comparação entre o Makespan Real e o Makespan da Simulação. 63

Tabela 4.6: Comparação entre o Makespan Real e o Makespan da Simulação. 63

Tabela 4.7: Resultado Simulado das Fases map, *reduce* e do *makespan.*............... 65

Tabela 4.8: *Makespan*, *Speedup* e Eficiência com 26 tarefas *reduce* e 100 mil tarefas *map*. 66

Tabela 4.9: *Makespan*, *Speedup* e Eficiência com 676 tarefas *reduce* e 100 mil tarefas map. 69

Tabela 4.10: *Makespan*, Speedup e Eficiência com 676 tarefas *reduce* e switch infiniband 71

LISTA DE SIGLAS

AMOK - *Advanced Metacomputing Overlay Kit*

API - *Application Programming Interface*

CPU - *Central Processing Unit*

DFS -*Distributed File System*

DFO *- Derivative Free Optimization*

EDR - *Enhanced Data Rate*

FLOPS - *FLoating-point Operations Per Second*

GRAS - *Grid Reality And Simulation*

HDFS - Hadoop Distributed File System

JVM - *Java Virtual Machine*

MPI - *Message Passing Interface*

MRSG - *MapReduce Over SimGrid*

MSG - *Meta-SimGrid*

RPC - *Remote Procedure Call*

SMPI - *Simulated MPI*

SST – *Sum of Squares Total*

SURF - *SimUlator aRtiFact*

XBT - *eXtended Bundle of Tools*

XML - *Extensible Markup Language*

WARC - *Web ARChive*

SUMÁRIO

CAPÍTULO 1 - INTRODUÇÃO....................13

1.1 Contexto....................13

1.2 Objetivos....................15

1.3 Metodologia de Desenvolvimento do Trabalho....................16

1.4 Organização do Trabalho....................17

CAPÍTULO 2 - FUNDAMENTOS TECNOLÓGICOS....................18

2.1 Aplicações Mapreduce....................18

2.1.1 O Modelo de Programação Mapreduce....................19

2.1.2 Fase Map....................21

2.1.3 Operação Shuffle....................22

2.1.4 Fase Reduce....................22

2.1.5 Execução do Mapreduce....................24

2.2 O Framework Hadoop para a Execução de Aplicações Mapreduce....................26

2.2.1 Framework Hadoop....................27

2.2.2 Subprojetos do Hadoop....................27

2.2.3 Hadoop Mapreduce....................29

2.2.4 O Hadoop *Distributed File System* (HDFS)....................30

2.2.5 Componentes do framework Hadoop Mapreduce....................32

2.3 Ferramentas de Simulação....................33

2.3.1 O Simulador SimGrid....................35

2.3.2 Simuladores de Aplicações Mapreduce....................38

2.3.2.1 O Simulador *MapReduce Over SimGrid* (MRSG)....................39

2.4 Medidas de Escalabilidade....................41

2.5 Projeto de Experimentos....................42

2.5.1 Projeto 2^k Fatorial....................44

CAPÍTULO 3 - ANÁLISE DOS FATORES DE MAIOR IMPACTO....................47

3.1 Ambiente Experimental....................47

3.2 Projeto de Experimentos 2^k Fatorial....................48

3.3 Fatores e seus níveis....................49

3.3.1 Primeiro Experimento 50
3.3.2 Segundo Experimento 52
3.4 Considerações do Capítulo 53
CAPÍTULO 4 - ANÁLISE DE ESCALABILIDADE DE UMA APLICAÇÃO MAPREDUCE 54
4.1 Ambiente De Execução Real 54
4.2 Aplicação utilizada nos Testes 55
4.3 Tempo de Execução da Aplicação (*makespan*) 58
4.4 Calibração do Simulador MRSG 61
4.5 Testes de Escalabilidade por meio de Simulação 65
4.5.1 Aumentando o paralelismo da Fase *Reduce* com 676 tarefas 69
4.5.2 Teste com o Uso de *Switch Infiniband* e 676 tarefas *reduce* 70
4.6 Considerações do Capítulo 72
CAPÍTULO 5 - CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS 73
REFERÊNCIAS 77

Capítulo 1

INTRODUÇÃO

1.1 Contexto

O desenvolvimento e o uso de aplicações paralelas e distribuídas para o processamento de grandes volumes de dados têm crescido consideravelmente. Com o objetivo de maximizar a eficiência e a competitividade no mercado, grandes organizações têm investido em pesquisas voltadas à análise de dados. No meio científico, a disponibilização de grandes bases de dados geradas a partir de experimentos científicos tem fortalecido uma nova forma de fazer ciência, apoiada fortemente pela análise desses dados.

A crescente popularidade da Internet, associada a fatores como a disponibilidade de computadores com grande capacidade de processamento e redes de alta velocidade tem impulsionado o crescimento da quantidade de dados gerados diariamente nas mais diversas atividades. Esse fato tem mudado a maneira de se processar grandes volumes de informações, utilizando plataformas computacionais e modelos de programação que apresentem baixo tempo de resposta de processamento.

Mapreduce é um modelo de programação desenvolvido para o processamento de grandes volumes de dados em *clusters* com milhares de processadores (DEAN; GHEMAWAT, 2004). Esse modelo tem sido amplamente utilizado por diversas empresas como Yahoo!, Google, Amazon, Facebook, entre

outras. Nesse modelo, a execução das tarefas ocorre em três fases, denominadas *map* e *reduce*. A fase *map* manipula os dados de entrada na forma de lista de pares (chave, valor) e produz dados intermediários, que são processados pela fase *shuffle*, recombinados e, posteriormente, consumidos e processados pela fase *reduce* (WHITE, 2009).

O *framework* Hadoop (DEAN; GHEMAWAT, 2004), que opera no modelo Mapreduce, é uma das mais difundidas tecnologias para a implementação e a execução de aplicações paralelas voltadas ao processamento de grandes bases de dados. Sua arquitetura alvo são as plataformas compostas tipicamente por milhares de nós, incluindo *clusters* locais, máquinas localizadas em *datacenter* ou em nuvens computacionais (*clouds)*.

Atualmente, Hadoop é a implementação distribuída como *software* livre, mais conhecida e difundida de Mapreduce. O projeto Hadoop teve origem na empresa Yahoo! em atividades voltadas ao processamento de grandes bases de dados e no sistema de busca. Atualmente, Hadoop é um projeto da *Apache Software Foundation* e conta com o apoio de diversos colaboradores. Desenvolvido para executar aplicações em clusters com milhares de nós utilizando o suporte de um sistema de arquivos distribuídos, Hadoop implementa mecanismos de tolerância a falhas, balanceamento de cargas e distribuição de dados (WHITE, 2009; DEAN; GHEMAWAT, 2004).

Existem relatos de aplicações Hadoop que manipulam mais de 20 Petabytes de dados por dia, executando em mais de quatro mil máquinas e 40 mil tarefas concorrentemente (O'MALLEY, 2011). Tal grau de paralelismo tem despertado grande interesse da comunidade de pesquisa que está começando a utilizar Hadoop/Mapreduce na implementação de aplicações em diversas áreas, como bioinformática, processamento de linguagem natural, aprendizagem de máquina e análise de imagens, entre outras.

A popularização do uso de Mapreduce/Hadoop despertou o interesse em avaliar os limites de escalabilidade de tais aplicações, os gargalos gerados e em como obter uma melhor configuração da aplicação, que proporcione maior escalabilidade. Apesar dos recentes estudos sobre Mapreduce/Hadoop encontrados na literatura científica, observa-se que há carência de estudos mais aprofundados e sistemáticos sobre os principais gargalos e sobre os limites de escalabilidade.

A ferramenta Hadoop possui mais de 190 parâmetros configuráveis, e destes, foram observadas que aproximadamente 25 causam maior impacto nas aplicações (BABU, 2010). A realização de estudos envolvendo esse grande número de parâmetros possibilita configurar a aplicação de maneira a obter o melhor desempenho possível. Esse trabalho apresenta um estudo sobre o impacto de cada fator (parâmetro) ou a combinação de parâmetros na escalabilidade das aplicações. Para tal, foi utilizado o projeto de experimentos 2^k fatorial.

Para a análise de escalabilidade, seria conveniente realizar experimentos em plataformas de grande porte. Entretanto, a disponibilidade de tais plataformas é uma das grandes limitações para um estudo dos limites de escalabilidade. Nesse sentido, a simulação contribui com esse cenário, possibilitando modelar diferentes plataformas (inclusive plataformas com até milhares de nós) e aplicações.

Uma boa ferramenta de simulação, devidamente calibrada e ajustada, permite aos desenvolvedores prever com certo grau de acurácia e o comportamento de aplicações, bem como seus gargalos e limites de escalabilidade.

1.2 Objetivos

Este trabalho tem por objetivo principal avaliar a escalabilidade de aplicações Mapreduce. Para avaliar esses limites, é necessário identificar os parâmetros de configuração que têm maior impacto sobre o desempenho e a escalabilidade.

O objetivo é identificar, ainda que de maneira aproximada, qual o comportamento esperado em termos de escalabilidade para uma dada aplicação, em plataformas compostas por milhares de nós. Uma vez identificados os gargalos que limitam a escalabilidade, é possível trabalhar a fim de reduzi-los, aumentando, assim, a escalabilidade das aplicações.

Por fim, com o estudo da escalabilidade dos parâmetros de maior impacto e dos gargalos, pode-se fornecer um conjunto de boas práticas a serem adotadas por desenvolvedores e usuários.

1.3 Metodologia de Desenvolvimento do Trabalho

A metodologia utilizada neste trabalho procurou contornar a limitação em termos de recursos computacionais disponíveis e, para isso, faz-se uso da combinação de experimentação real e de simulação. Apesar de haver infraestruturas disponíveis, tais como o Grid5000 (GRID5000, 2013), usuários que tenham interesse em avaliar a escalabilidade de suas aplicações podem sofrer das mesmas limitações de acesso a tais infraestruturas.

Nesse sentido, a metodologia utilizada pode ser útil para prever a escalabilidade de aplicações sem depender da disponibilidade de uma plataforma real. Por meio de simulação, pode-se avaliar cenários com diferentes parâmetros, abrangendo uma grande variedade que não seria possível via experimentação real.

Para a experimentação em ambiente real, utilizou-se uma aplicação de índices reversos, bastante eficiente e otimizada, conhecida por *Per-Posting list* (MCCREADIE; MACDONALD; OUNIS, 2011). Esse método de construção de índices reversos está presente no sistema de recuperação de informações denominado Terrier (TERRIER, 2012). Tal aplicação foi executada em um ambiente real composto de 32 nós (*cluster* DC-UFSCar), que possibilitou a coleta de dados necessários ao ajuste do simulador.

Esse ajuste foi necessário para que fornecesse resultados tão próximos quanto possível de uma aplicação real para a plataforma de 32 nós disponível. A esse ajuste, dá-se designação de calibração. Uma vez calibrado, o simulador permite extrapolar o número de nós da plataforma e avaliar a escalabilidade da aplicação-alvo para plataformas de grande porte, da ordem de milhares de nós. Com a simulação das diferentes plataformas, é possível identificar, ainda que de maneira aproximada, qual seria o comportamento esperado em termos de escalabilidade da aplicação nessas plataformas.

1.4 Organização do Trabalho

Inicialmente, o Capítulo 2 apresenta uma revisão bibliográfica dos fundamentos tecnológicos envolvidos. No Capítulo 3, apresenta-se uma análise sobre os fatores de maior impacto no desempenho das aplicações Mapreduce, por meio da análise do projeto 2^k fatorial. No capítulo 4, apresenta-se uma análise de escalabilidade de aplicações Mapreduce, além de uma estratégia de calibração do simulador MRSG. Finalmente, as considerações finais e trabalhos futuros encontram-se no Capítulo 5, seguido pelas referências bibliográficas.

Capítulo 2

FUNDAMENTOS TECNOLÓGICOS

Nesta seção são apresentados os principais fundamentos tecnológicos e o ferramental utilizado neste trabalho. Inicialmente, descrevemos o modelo de programação paralela Mapreduce e o framework Hadoop, que é voltado ao processamento de grandes bases de dados em clusters com milhares de máquinas. Além disso, são descritas as ferramentas de simulação utilizadas para realização do estudo de escalabilidade. Por fim, as métricas de escalabilidade e o projeto de experimentos utilizados neste trabalho são apresentados.

2.1 Aplicações Mapreduce

Com o objetivo de simplificar a implementação e a execução de tarefas de processamento paralelo na empresa Google Inc., em especial a construção de índices reversos para o seu mecanismo de busca, Dean e Ghemawat (2004) propuseram um modelo de programação e um framework conhecido por Mapreduce. Esse modelo destina-se à implementação de aplicações voltadas ao processamento e à geração de grandes bases de dados em plataformas compostas por milhares de processadores.

Os proponentes do modelo argumentam que, apesar da complexidade da maioria das aplicações paralelas, diversas delas podem ser implementadas de forma mais simples e com pouco esforço. O modelo Mapreduce consegue abstrair a complexidade de paralelismo e, por meio de uma interface simples, provê mecanismos de tolerância a falhas, a balanceamento de cargas e a distribuição de dados (WHITE, 2009; DEAN; GHEMAWAT, 2008). Assim, os programadores não

precisam tratar tais questões diretamente e podem manter seu foco nas aplicações. Essa vantagem tem favorecido a popularização de aplicações Mapreduce que tratam do processamento *offline* de grandes volumes de dados, tais como o processamento de documentos e de *logs* de requisições da web.

Dean e Ghemawat (2008) observaram que as bases de dados das aplicações analisadas eram particionadas e distribuídas entre milhares de máquinas a fim de executar as aplicações em menos tempo. Nesse contexto, o framework Mapreduce foi desenvolvido para facilitar o desenvolvimento de aplicações que possuam as mesmas características de manipulação de dados. Nesse framework, em uma primeira fase de processamento, diversas tarefas executam um mesmo conjunto de operações (fase *map*) sobre diferentes blocos de dados, em paralelo. Na fase seguinte (*reduce*), operações de agregação ou de redução são aplicadas aos resultados da fase anterior. O modelo é baseado em primitivas *map* e *fold* de linguagens funcionais, tais como Lisp.

Dean e Ghemawat relatam que em 2008 o modelo Mapreduce era utilizado pela empresa Google para processar mais de vinte petabytes de dados por dia em atividades de processamento de textos e de grafos em larga escala, de aprendizado de máquina e de estatística de tradução automática. Na literatura há relatos que, entre 2004 e 2008, a Google desenvolveu mais de dez mil programas Mapreduce e diariamente processava cerca de cem mil *jobs* em diversos *clusters*. Esse modelo de programação tem sido largamente explorado por grandes empresas, tais como Yahoo!, Google, Amazon, Facebook, entre outras (DEAN; GHEMAWAT, 2008).

Vance (2009) cita que a utilização de Mapreduce tornou possível a melhoria de produtos como o Google Maps e o Google *Earth*, sendo utilizado para manipular e montar imagens compostas partindo de grandes quantidades de imagens provenientes de satélites. No mecanismo de busca Google, Mapreduce tornou possível criar perfis de busca pela identificação de relacionamentos entre diferentes grupos de sites, imagens e documentos.

2.1.1 O Modelo de Programação Mapreduce

No modelo de programação Mapreduce, as tarefas de processamento são distribuídas entre os nós, implementando uma arquitetura *master-slave*, na qual existe um único mestre gerenciando um determinado número de escravos (*workers*).

O nó mestre escalona as tarefas aos escravos, determinando qual nó irá realizar uma tarefa *map* ou uma tarefa *reduce*. Sempre que um nó escravo completar a execução de uma tarefa, ele sinaliza ao mestre, para que esse possa escalonar uma nova tarefa ao escravo que está disponível (DEAN; GHEMAWAT, 2008).

Mapreduce é utilizado em conjunto com um sistema de arquivos distribuídos, especialmente projetado para dar suporte a aplicações distribuídas que manipulam grandes volumes de dados (e.g. HDFS, Google GFS). White (2009) descreve que um sistema de arquivos distribuídos (*Distributed File System* - DFS) deve fornecer transparência de localidade e de redundância a fim de melhorar a disponibilidade dos dados e de funcionar como uma plataforma de armazenamento central.

No sistema que implementa o modelo Mapreduce, cada nó recebe blocos de dados *(chunks)* como entrada, os quais estão armazenados no sistema de arquivos do *cluster*. Então, cada nó processa em diferentes máquinas. O processamento das tarefas é abstraído em três fases distintas: a fase *map* e a fase *reduce,* acessíveis ao programador, e a fase intermediária chamada *shuffle* que é criada pelo sistema em tempo de execução (WHITE, 2009).

O modelo de programação Mapreduce, ilustrado pela Figura 2.1, apresenta um exemplo que conta o número de ocorrências de palavras em textos. Neste exemplo, têm-se dois registros armazenados no sistema de arquivos, no qual cada registro é um bloco de dados (*chunk)* enviado para processamento.

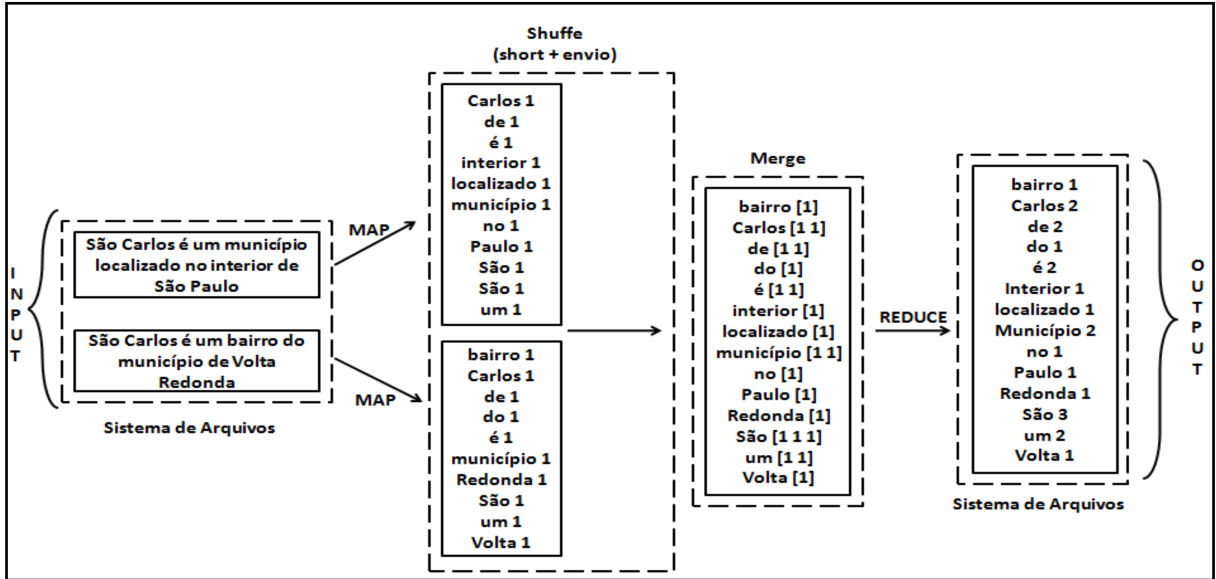

Figura 2.1: Modelo de fluxo de dados Mapreduce, adaptada de White (2009).

2.1.2 Fase Map

O código executado na fase *map* deve ser definido pelo usuário na forma de uma função *map.* Essa função recebe um conjunto de dados de entrada, gera dados intermediários e os armazena na máquina local. A Figura 2.1 ilustra o modelo de fluxo de dados Mapreduce, no qual, na fase *map,* faz-se a contagem de ocorrências de chaves mapeadas no documento *b* e as associam ao valor "1", produzindo dados intermediários na forma <chave, valor>. O Algoritmo 2.1 representa a função *map* do exemplo *wordcount.*

Algoritmo 2.1: Função *Map*

1: classe **Mapper**
2: método **Map (doc_id *a*, document *b*)**
3: para todos **term *t* Є document *d*** faça
4: **EMIT (term *t*, 1)**
5: fim for

No pseudocódigo anterior, o método *map* recebe como entrada o conteúdo de um documento *b*, extrai os *tokens* e, para cada *token* encontrado, emite um par <chave, 1>. O exemplo ilustrado na Figura 2.1, o processamento da fase *map* produz o seguinte conjunto intermediário de dados:

Bloco 1:

<São, 1>
<Carlos, 1>
<é, 1>
<um, 1>
<município, 1>
<localizado, 1>
<no, 1>
<interior, 1>
<de, 1>
<São, 1>
<Paulo, 1>

Bloco 2:

<São, 1>
<Carlos, 1>
<é, 1>
<um, 1>
<bairro, 1>
<do, 1>
<município, 1>
<de, 1>
<Volta, 1>
<Redonda, 1>

No exemplo acima, na saída gerada pela fase *map,* pode-se observar que há a ocorrência do *token* “São” em mais de um par <São, 1> em ambos os blocos de dados.

2.1.3 Operação Shuffle

A operação *shuffle* é realizada automaticamente entre a fase *map* e a fase *reduce*, executando operações distintas: *sort* e *merge*. Durante o *shuffle*, o nó que executou a tarefa *map* ordena as chaves (operação *sort)*, serializa os dados e os envia ao nó responsável pela execução do *reduce* correspondente. Após as operações *sort* e *merge*, o ambiente de execução envia os pares de mesma chave ao mesmo nó que irá realizar a tarefa *reduce*.

Supondo que exista uma tarefa *reduce* e considerando o conjunto de dados intermediários ilustrados na seção 2.1.2, o seguinte conjunto de dados seria enviado aos nós onde são realizadas as tarefas *reduce*:

<bairro, [1]>
<Carlos, [1,1]>
<de, [1,1]>
<do, [1]>
<é, [1,1]>
<interior, [1]>
<localizado, [1]>
<município, [1,1]>
<no, [1]>
<Paulo, [1]>
<Redonda, [1]>
<São, [1,1,1]>
<um, [1,1]>
<Volta, [1]>

2.1.4 Fase Reduce

A fase *reduce* deve também ser implementada pelo usuário por meio da função *reduce,* que recebe como entrada os dados intermediários gerados pela função *map* e tratados pela operação *shuffle.* Esses dados são compostos por um

conjunto de chaves intermediárias e valores a elas associados no formato <chave, [valor, valor, ...]>. Esse agrupamento de mesmas chaves em uma lista evita repetições desnecessárias de chaves na saída do *map* e reduz a quantidade de dados a ser transmitida na rede.

Após o processamento da fase *reduce*, novas saídas são produzidas. Nessa fase, a função *reduce* efetua a combinação de dois ou mais valores em um único, através da unificação de chaves iguais e cria-se um possível conjunto menor de valores. Assim, na Figura 2.1, a função *reduce* soma todas as ocorrências para uma <chave> específica. A seguir, o Algoritmo 2.2 ilustra o comportamento da função *reduce* do exemplo *wordcount*.

Algoritmo 2.2: Função *Reduce*

1: classe ***Reducer***
2: método ***Reduce* (term *t*, counts $[c_1, c_2, \ldots, c_n]$)**
3: **sum ← 0**
4: para todo ***c* ∈ counts** $[c_1, c_2, \ldots, c_n]$ faça
5: **sum ← sum + *c***
6: fim para
7: EMIT (term *t*, 1)

Supondo que os dados intermediários ilustrados na seção 2.1.3 sejam os dados de entrada da fase *reduce*, pode-se observar que cada chave está associada a uma lista de contadores. A fase *reduce* emite a seguinte saída:

<bairro, [1]>

<Carlos, [2]>

<de, [2]>

<do, [1]>

<é, [2]>

<interior, [1]>

<localizado, [1]>

<município, [2]>

<no, [1]>

<Paulo, [1]>

<Redonda, [1]>

<São, [3]>

<um, [2]>

<Volta, [1]>

O fluxo completo de dados do Mapreduce é composto basicamente de cinco etapas: *input*, *map*, *shuffle*, *reduce* e *output* (WHITE, 2009).

2.1.5 Execução do Mapreduce

A execução de uma aplicação Mapreduce, ilustrada na Figura 2.2 a seguir, pode ser compreendida pelos itens I a VII descritos a seguir (DEAN; GHEMAWAT, 2004):

I. A base de dados armazenada no DFS é composta por diversos registros que são divididos em N blocos de dados *(chunks)* que, por padrão variam de 16 MB a 64 MB, com tamanho configurável pelo usuário. Inicialmente, os *chunks* são distribuídos e replicados pelos nós da arquitetura e, em seguida, replica-se o código do usuário em todas as máquinas ativas;

II. O modelo Mapreduce implementa a arquitetura mestre/escravo *(master/slave).* Existem tarefas *map* e tarefas *reduce,* ora implementadas pelo usuário, a serem atribuídas e cabe ao nó mestre determinar qual tarefa um nó escravo deve executar;

III. Um nó escravo recebe uma tarefa *map* a ser executada tendo como entrada um *chunk* de dados previamente recebido. Uma vez iniciado o processamento, os pares <chave,valor> são analisados e processados de acordo com a função *map* definida pelo usuário. Os conjuntos de pares <chave,valor> intermediários são armazenados na memória temporária, para posterior envio à próxima fase;

IV. Periodicamente, um nó armazena temporariamente os pares em um *buffer* no disco local. A saída produzida é escrita no disco local, no qual a função *map* foi executada (e não no DFS) por questões de desempenho. A localização desse *buffer* é transmitida ao mestre, que se encarrega de repassar a localização desse armazenamento temporário aos escravos responsáveis pelas tarefas *reduce*, que pode ou não ser executada no mesmo nó;

V. Um nó escravo, ao ser notificado pelo mestre sobre a localização do *buffer* que contém os pares intermediários, usa chamadas remotas para ler tais dados no disco local do escravo que realizou a tarefa *map*.

Quando o escravo que executa a tarefa *reduce* termina de ler os dados intermediários, ele os classifica. Caso a quantidade de dados intermediários seja superior à capacidade da memória, utiliza-se um armazenamento externo;

VI. O nó escravo que executa a função *reduce* analisa a classificação intermediária dos pares e procura, para cada <chave>, o conjunto de valores intermediários a serem reduzidos. Ao final, o arquivo de saída é gerado;

VII. Quando todas as tarefas *map* e *reduce* são finalizadas, o nó mestre continua executando o programa do usuário. Após a sua conclusão, o resultado da execução do Mapreduce é disponibilizado no arquivo de saída.

O sistema de execução de programas Mapreduce é responsável por gerenciar a execução de todas as tarefas *map, shuffle* e *reduce* e por efetuar a manipulação de dados de forma automatizada.

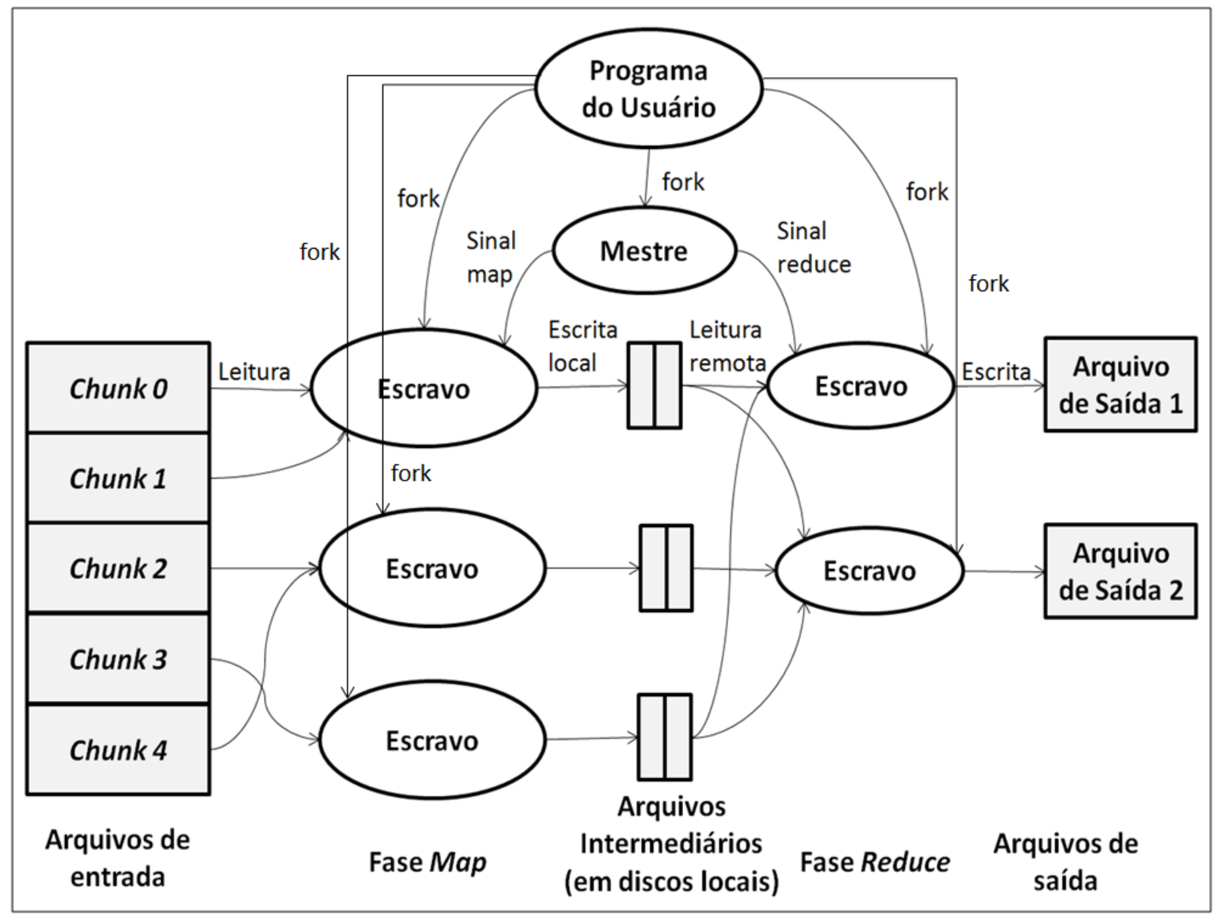

Figura 2.2: Modelo de execução Mapreduce, adaptada de DEAN e GHEMAWAT (2004).

Barroso e Hoelzle (2009) descrevem algumas funcionalidades desse sistema, tais como: a paralelização e a distribuição da computação entre as máquinas disponíveis no *cluster,* mecanismos de tratamento e de tolerância a falhas nas máquinas, o escalonamento das tarefas de computação e de comunicação intermáquinas e o gerenciamento dos recursos de rede e de discos.

No sistema de execução, o nó mestre é responsável por detectar falhas e constantemente supervisionar os nós escravos usando técnicas como *pinging e heartbeat. Heartbeat* é uma técnica utilizada para testar a conectividade, na qual os nós escravos enviam ao mestre, por padrão, um sinal a cada 3 segundos. Esse intervalo de envio de sinal pode ser alterado em plataformas de grande porte para evitar o congestionamento do tráfego de rede e a consequente degradação do desempenho da aplicação.

O monitoramento do sistema permite ao nó mestre detectar o mau desempenho de um determinado escravo. Ao detectar uma falha devido ao não recebimento de resposta por um determinado tempo, o nó mestre reescalona as tarefas em uma máquina escravo disponível. Mapreduce executa réplicas das tarefas em outras máquinas com o objetivo de acelerar a computação e garantir a confiabilidade dos dados.

2.2 O Framework Hadoop para a Execução de Aplicações Mapreduce

Existem várias ferramentas que implementam o modelo de programação Mapreduce, cada qual criada para atender a necessidades específicas nas mais diversas linguagens de programação. As principais ferramentas são: Google Mapreduce (C++), Hadoop Mapreduce (Java), Greenplum (Python), Phoenix (C), Skynet (Ruby), MapSharp (C#), entre outras. A seguir, serão descritos o framework Hadoop e seus componentes.

2.2.1 Framework Hadoop

O projeto Hadoop foi criado por Doug Cutting e implementado pela Apache (*APACHE*, 2011). Hadoop é a implementação mais conhecida e utilizada do modelo Mapreduce. O projeto foi desenvolvido tendo como foco o processamento em c*lusters* com milhares de máquinas e sustentado sobre duas bases: Mapreduce, que provê as interfaces de programação, e HDFS (*Hadoop Distributed File System*), que fornece um sistema de arquivos distribuídos. HDFS se mostra um grande aliado ao projeto Hadoop, pois permite explorar a localidade de arquivos na arquitetura, escalonando tarefas aos nós que possuem os arquivos de entrada para execução (WHITE, 2009).

Em ambientes distribuídos, Hadoop é capaz de gerenciar a arquitetura distribuída, realizando, de forma automática, atividades como o escalonamento de tarefas, a interação com o sistema de arquivos distribuídos e a troca de mensagens entre nós, permitindo que o usuário se preocupe apenas com a lógica da aplicação.

Um exemplo de aplicação do framework Hadoop é o ambiente utilizado pela rede social Facebook que, segundo Borthakur (2010), processa aproximadamente 21 petabytes de dados em um *cluster* com 2.000 máquinas, sendo que 1.200 delas possuem 8 núcleos e 800 possuem 16. Cada máquina executa aproximadamente 15 tarefas *map*/*reduce* e conta com 12 terabytes de disco e 32 de memória RAM.

2.2.2 Subprojetos do Hadoop

Há outros subprojetos que proporcionam serviços complementares, ou até mesmo incrementam as estruturas básicas do *framework* Hadoop. Abaixo, são descritos alguns dos principais subprojetos existentes no Hadoop (APACHE, 2011):

- *Hadoop Common:* é composto por um conjunto de utilidades que oferece suporte aos subprojetos Hadoop. Esse subprojeto inclui bibliotecas para sistemas de arquivos, RPC (*Remote Procedure Call*) e serialização de objetos;
- *Apache Avro:* é basicamente um sistema de serialização de dados composto por várias estruturas que permite a utilização de arquivos para o armazenamento persistente de dados, RPC e apresenta um formato binário de dados compacto e rápido;

- *Hadoop* Mapreduce*:* é um modelo de programação destinado ao desenvolvimento de aplicações capazes de processar grandes bases de dados;
- *Hadoop Distributed File System (HDFS):* é o sistema de arquivos distribuídos utilizado por aplicações feitas com o Hadoop e é responsável por distribuir e criar réplicas dos blocos de dados nos nós da arquitetura;
- *Apache Pig:* é uma plataforma de alto nível destinada à análise de grandes bases de dados de aplicações Mapreduce, como as desenvolvidas no projeto Hadoop;
- *Apache HBase:* considerada a base de dados do Hadoop, é um sistema distribuído de armazenamento de dados baseado no modelo *BigTable,* proposto pela empresa Google. Esse subprojeto é utilizado na leitura e na escrita em tempo real e em acessos aleatórios ao banco de dados. Orientado por colunas, o HBase fornece estruturas para a manipulação de grandes tabelas que podem ser compostas por bilhões de linhas com até milhões de colunas;
- *Apache ZooKeeper:* trata-se de um serviço centralizado que oferece suporte a operações do HBase e pode ser integrado ao Mapreduce a fim de tolerar falhas nos nós escravos. É composto por um conjunto de ferramentas que têm por objetivo a construção de aplicações distribuídas que lidem com falhas parciais de componentes.;
- *Hive:* esse subprojeto é uma infraestrutura de data *warehouse,* composta por ferramentas que objetivam a indexação de dados, consultas *ad-hoc,* a criação de relatórios e a análise de grandes quantidades de dados armazenados em arquivos Hadoop;
- *Chukwa:* É um sistema distribuído para coletar dados para monitoramento e para analisar *logs* dinamicamente.

Observa-se que a análise e o processamento de extensas bases de dados utilizando o framework Hadoop e seus subprojetos têm sido largamente empregados tanto na área científica quanto em grandes empresas que utilizam e contribuem com o projeto, como Facebook, Yahoo!, Last.fm, Google, Amazon, IBM, Adobe, Cloudera e entre outras.

Hadoop foi projetado para realizar tarefas que exigem processamento paralelo e pode ser configurado para ser utilizado em uma única máquina ou para ser executado em um conjunto de máquinas. De maneira transparente ao programador, Hadoop efetua a distribuição da base de dados e o balanceamento de carga e realiza o processamento paralelo de grandes quantidades de dados com mecanismos de tolerância a falhas.

O núcleo do Hadoop tem sido desenvolvido em Java, portanto, faz-se necessário que os componentes computacionais possuam uma Máquina Virtual Java (JVM). Hadoop foi desenvolvido para ser utilizado tanto em ambientes GNU/Linux quanto no sistema operacional Windows.

O´Malley (2011) relata que a empresa Yahoo! investiu esforços significativos em pesquisas no Hadoop em 2001, com o objetivo de obter maior grau de confiabilidade e de disponibilidade na execução de suas aplicações. Como meta, a empresa deseja executar as aplicações em *clusters* com até 6.000 máquinas, 16 núcleos cada, 48 Gigabytes de RAM e discos de 24 Terabytes. Isso tornará possível a execução de 100.000 tarefas e 10.000 *jobs* simultâneos. Tais limites correspondem às características de gerenciamento, de comunicação e de processamento da versão 0.20 do Hadoop.

2.2.3 Hadoop Mapreduce

O projeto Hadoop Mapreduce é uma implementação do modelo de programação Mapreduce, desenvolvido no modelo de software livre e em linguagem Java, que tem por objetivo produzir aplicações distribuídas com alto grau de disponibilidade, confiabilidade, escalabilidade e tolerância a falhas.

No Hadoop Mapreduce, os dados de entrada são inicialmente divididos em N partes (*chunks*), sendo N o número de tarefas *map*. Esses *chunks* são distribuídos e replicados aos nós da arquitetura e representam os dados de entrada para as tarefas *map*. O nó mestre aloca as tarefas *map* que efetuam o processamento dos dados de entrada e o resultado intermediário é enviado às tarefas *reduce*.

A função *reduce* efetua a computação dos dados intermediários recebidos e grava os dados de saída no HDFS. No ambiente de execução, uma tarefa *reduce* somente se inicia após o término de todas as tarefas *map*. O esquema ilustrado na Figura 2.3 demonstra o fluxo de dados no Hadoop Mapreduce.

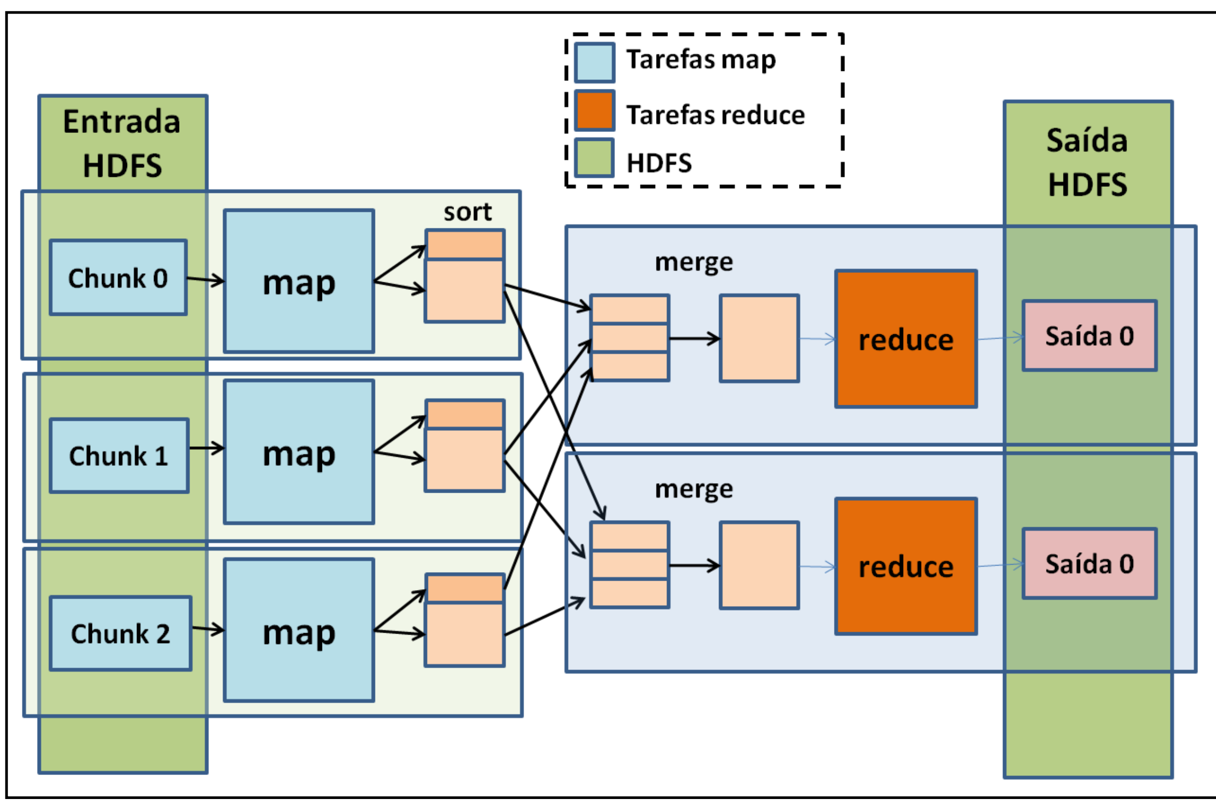

Figura 2.3: Representação do fluxo de dados do Hadoop, adaptada de HADOOP (2008).

No framework Hadoop Mapreduce, os dados de entrada, a aplicação Mapreduce e os arquivos de configuração constituem um *job*. Primeiramente, Hadoop executa o *job* subdividindo-o em diversas tarefas (*tasks*), que podem ser tarefas *map* ou tarefas *reduce.* No exemplo ilustrado na Figura 2.3, observa-se a incidência de três tarefas *map* e duas tarefas *reduce*.

White (2009) menciona que a operação *shuffle,* realizada nas *tasks map* e nas *tasks reduce*, é uma das mais importantes para o bom desempenho das aplicações Mapreduce no Hadoop. Além da transferência de dados de saída das funções *map* para a entrada das funções *reduce*, a operação *shuffle (*com a execução das operações *sort* e *merge)* procura garantir que toda entrada de dados da fase *reduce* esteja ordenada pelos pares de chaves.

2.2.4 O Hadoop *Distributed File System* (HDFS)

Hadoop Distributed File System (*HDFS*) é o principal sistema de armazenamento utilizado pelas aplicações Hadoop (HADOOP, 2008). A Figura 2.4 ilustra a arquitetura do sistema de arquivos HDFS.

O sistema de arquivos HDFS opera segundo uma arquitetura mestre-escravo e é composto por um processo mestre denominado *NameNode*, o qual tem a função de um servidor de metadados e de banco de dados. Esse processo mestre é responsável por gerenciar e controlar o acesso ao sistema de arquivos e manter metadados que indicam a localização física dos dados armazenados e replicados nos nós escravos. Ele executa operações como *open, close, rename* de arquivos e diretórios no sistema de arquivos. O componente *SecondaryNameNode* opera como assistente do *NameNode*, oferecendo tarefas de verificação e de manutenção periódica.

O ambiente de sistemas de arquivos distribuídos pode conter vários *DataNode e* apenas um *NameNode*. Cada nó escravo possui um processo *DataNode* que armazena um conjunto de blocos de dados do DFS. Esse processo escravo é responsável pelo armazenamento e pela recuperação dos blocos de dados no sistema de arquivo local. Ele se comunica regularmente com o processo *NameNode* e executa operações como a exclusão de bloco local ou a cópia de blocos de dados de outros *DataNodes*. Cada *DataNode* executa operações de leitura e de escrita de dados.

Antes de iniciar a execução de um *job,* a base de dados a ser manipulada é copiada para o sistema de arquivos HDFS e subdivida em blocos de dados, conhecidos por *chunks*. Tais blocos são distribuídos e replicados aos *DataNodes* que gerenciam o armazenamento local. Por padrão, o tamanho de cada bloco é de 64 Megabytes, sendo que esse valor é configurável.

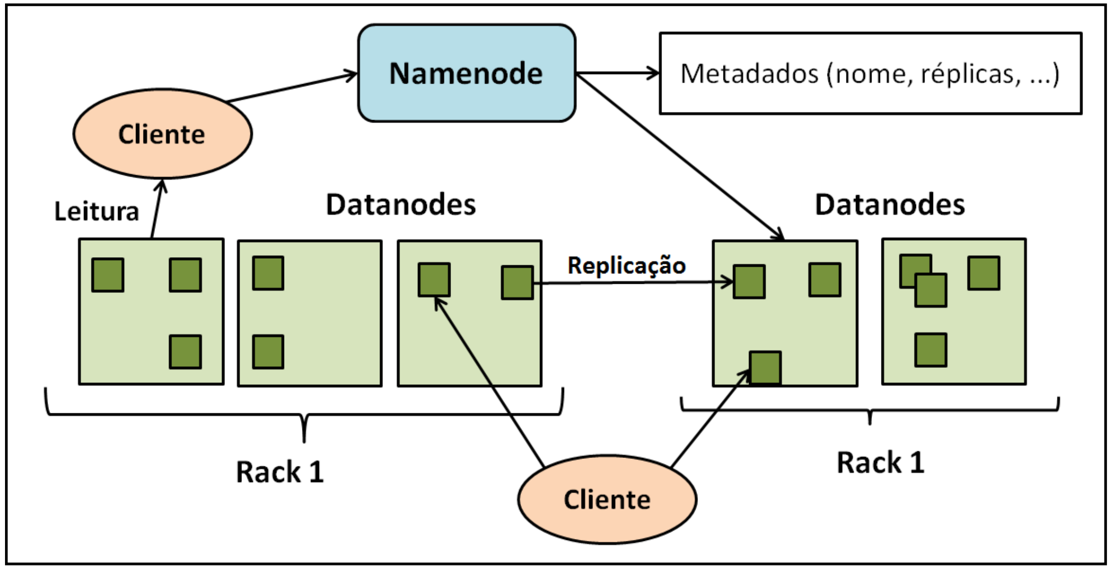

Figura 2.4: Arquitetura do *Hadoop Distributed File System*, adaptada de HADOOP (2008).

2.2.5 Componentes do framework Hadoop Mapreduce

O ambiente de execução Hadoop utiliza o modelo mestre-escravo para a execução das tarefas. Na versão 1.0.4 do framework Hadoop, a execução de aplicações Mapreduce é gerenciada por dois processos: *JobTracker* e *TaskTracker*. Cada conglomerado de nós de processamento possui um único processo *JobTracker* executando no nó mestre e que funciona como um agendador de tarefas e é o responsável pela manipulação das submissões e pelo monitoramento dos vários nós de processamento que possuem um processo *TaskTracker.*

O processo *TaskTracker* provê suporte à execução das tarefas *map* e *reduce*, ou seja, é o executor do código JAVA das tarefas. Trata-se de um processo que está presente em cada nó escravo e que aceita as tarefas (*map, reduce* e operação *shuffle*) submetidas pelo *JobTracker* e as executa. Todo *TaskTracker* é configurado com um conjunto de *slots* que indica o número de tarefas concorrentes que esse pode executar.

A Figura 2.5 ilustra a relação mestre-escravo entre os processos *JobTracker* e *TaskTracker.* O *JobTracker* distribui tarefas aos *DataNodes* e, o *TaskTracker* as executa e envia constantemente as informações relativas ao status de cada tarefa ao *JobTracker*. Esse, por sua vez, é responsável pela validação do trabalho solicitado e, em caso de falha na execução em algum *DataNode*, determina sua re-execução.

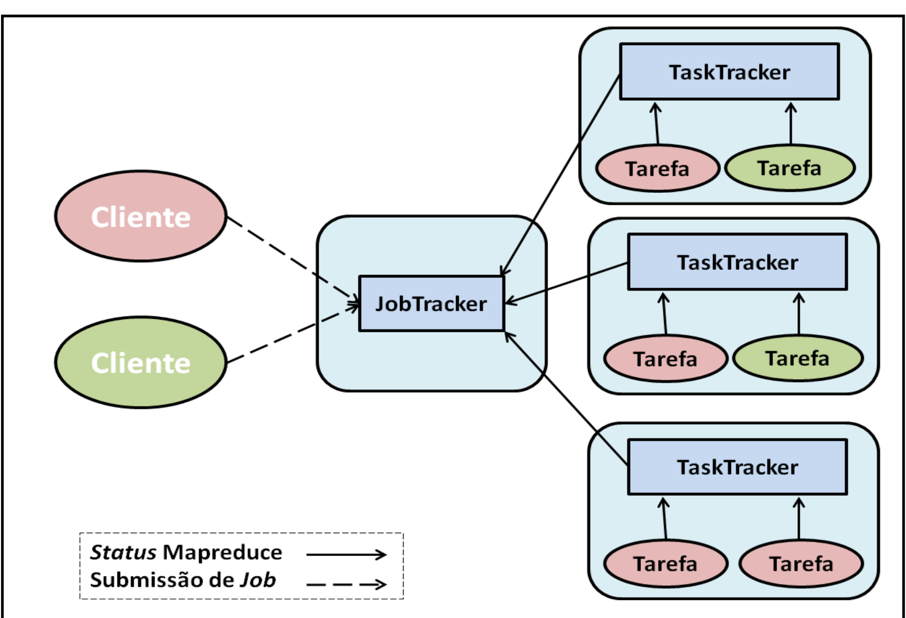

Figura 2.5: Representação do *JobTracker* e do *TaskTracker*, adaptada de O´MALLEY (2011).

No ambiente de execução, as tarefas são lançadas de maneira que cada uma delas esteja associada a um *chunk* de dados. No Hadoop, a computação é movida para os nós onde residem os dados, sempre que possível. Com a replicação dos blocos de dados, o escalonador do Hadoop possui um maior número de opções de execução, pois qualquer nó que possua uma réplica dos dados é candidato à execução. Caso todos os nós que possuam a réplica de dados necessária à execução da tarefa estejam ocupados, essa poderá ser escalonada em um nó diferente, contanto que haja a transferência de dados para esse nó. Por padrão, Hadoop tenta escalonar a execução da tarefa no mesmo *rack*, com o objetivo de minimizar os efeitos da comunicação.

Nas primeiras versões do Hadoop, o fato de haver apenas um processo *JobTracker* determinava um ponto de falha que poderia ocasionar a perda de todo o processamento. Na sua versão atual (Hadoop 1.0.4), esse problema foi resolvido. Periodicamente, Hadoop salva o status de cada *job* em execução; assim, o framework pode instanciar um novo processo *JobTracker* e continuar a execução em caso de falha.

2.3 Ferramentas de Simulação

Casanova (2008) relata que certas análises em ambientes de c*luster* de computadores com centenas ou milhares de nós podem se tornar uma tarefa inaplicável em plataformas reais. Dentre os fatores que impossibilitam a realização dos experimentos e os tornam suscetíveis a falhas não previstas destacam-se a indisponibilidade tanto de máquinas quanto de rede, o número insuficiente de máquinas na plataforma e as condições de utilização que podem flutuar devido ao compartilhamento dos recursos.

Como a realização de experimentos em ambiente real pode ser uma tarefa impraticável, a simulação se mostra uma solução alternativa para a essa realização. Por meio de ferramentas e de técnicas específicas, simulação permite estudar o comportamento e as reações de determinados sistemas através de diferentes modelos.

Hammoud (2010) afirma que simulação permite traçar comportamentos reais da aplicação, minimizar riscos e custos, emular diferentes cenários, determinando, assim, os efeitos de diferentes configurações, bem como a visualização de todo o processo. Com simulação, é possível realizar experimentos com diferentes configurações que não se dispõem na prática, avaliar situações teóricas e observar o comportamento de um sistema em diversos cenários. Um simulador pode vir a sanar problemas restritivos de tempo e torna possível um grande número de experimentos em diversos cenários. Um simulador de Mapreduce pode ser útil para estimar o tempo e o custo da execução de aplicações para a submissão de *jobs*.

Casanova (2008) defende o uso de simulação de experimentos pois, dentre as suas principais características, essa permite configurar diversos experimentos, repeti-los em ambientes controlados e, em geral, em menor tempo que um experimento real. Comparando tais características com aplicações em ambientes reais, o número de configurações a serem exploradas pode ser limitado devido a características como o tempo total de execução, custo, ausência de equipamentos na quantidade desejada e entre outros. Assim, uma das principais vantagens de um simulador é a possibilidade que o usuário tem de controlar as condições em que os diferentes tipos e modelos de simulação ocorrem.

Diversos aspectos podem ser analisados quanto ao desempenho de sistemas, tais como: tempo de resposta, robustez, *throughput*, escalabilidade e tolerância a falhas. Uma questão importante mencionada por Casanova (2008) é como comparar várias soluções em cenários relevantes e efetuar a análise dos sistemas levando em consideração a combinação de diferentes aspectos. Nesse sentido, simulação se mostra uma saída viável para a realização de experimentos e para a análise de diferentes cenários.

Com objetivo de viabilizar e validar esse projeto de pesquisa optou-se pelo uso de um software que permita simular o framework Hadoop Mapreduce. A ideia é que uma ferramenta de simulação permita avaliar diferentes configurações antes de uma implementação real, possibilitando a escolha de modelos mais adequados à solução do problema avaliado.

2.3.1 O Simulador SimGrid

Henri Casanova, tendo por base o simulador criado por Arnaud Legrand, propôs uma versão mais genérica de um simulador composto por uma API (*Application Programming Interface)* simples, o SimGrid versão 1.0, para a simulação de eventos discretos e com foco em aplicações científicas paralelas em plataformas distribuídas. Nessa primeira versão, forneceu-se um conjunto de abstrações e funcionalidades que podem ser utilizadas de maneira mais simples para a simulação de aplicações específicas (CASANOVA, 2001).

O simulador SimGrid versão 2.0 contou com os avanços propostos por Arnaud Legrand, que incorporou ao simulador uma nova camada em alto nível, composta por duas interfaces: o SimGrid e o MetaSimGrid (MSG). O MSG foi um simulador construído a partir da utilizaçãodo SimGrid e que tornou possível simulações mais realistas orientadas à aplicação. Essa versão incorporou o conceito de *threads* e a possibilidade de simular tarefas de comunicação assíncrona (CASANOVA, 2001).

O simulador SimGrid versão 3.0 é mantido em conjunto coma Universidade do Havaí em Manoa*,* com o *LIG Laboratory,* em Grenoble – França, e com a Universidade de Nancy, também na França. O simulador é composto por alguns ambientes de programação desenvolvidos sobre um único núcleo de simulação. Basicamente, o simulador pode ser dividido em três camadas: programação, simulação do *kernel* e camada base (SIMGRID, 2012). A Figura 2.6 ilustra a arquitetura geral do SimGrid versão 3.8.1.

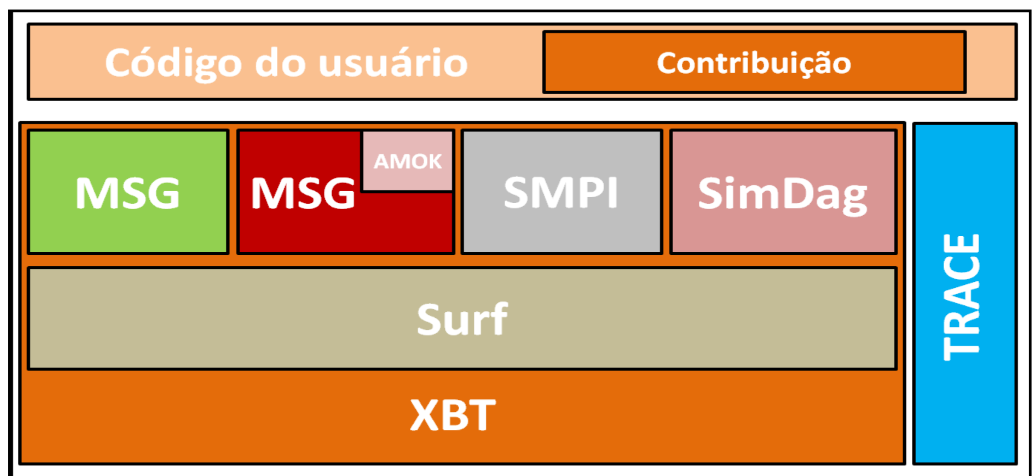

Figura 2.6: Arquitetura geral do SimGrid versão 3.8.1, adaptada de SIMGRID (2012).

A camada de programação do simulador é composta por vários ambientes de programação sobre um núcleo de simulação. Cada ambiente possui características específicas e um paradigma de programação diferente, tais como programação estruturada, orientada a objetos e programação concorrente. As principais APIs da camada de programação são:

- *MSG* (*Meta-SimGrid*)*:* ambiente simples de programação utilizado para simulações no nível de aplicação, o qual possui duas APIs: Java (API jMSG) e C. Nesse ambiente, a implementação da aplicação é desnecessária;
- *GRAS* (*Grid Reality and Simulation*)*:* esse framework facilita o desenvolvimento de aplicações reais orientadas a eventos. Com o GRAS é possível desenvolver e testar a aplicação no simulador e portá-la para plataformas reais. Essa API trabalha em conjunto com o toolkit chamado AMOK (*Advanced Metacomputing Overlat Kit*), que implementa em alto nível diversos serviços necessários às aplicações distribuídas. Esse toolkit fornece ferramentas úteis ao GRAS, tais como o teste de largura de banda entre dois nós e o gerenciamento remoto de servidores;
- *SMPI* (*Simulated MPI*)*:* interface que utiliza técnicas de emulação para simulações do comportamento de códigos MPI (*Message Passing Interface*) em ambientes multiprocessados;
- *SIMDAG* (*Simulation Direct Acyclic Graph*)*:* ambiente dedicado à simulação de aplicações paralelas, por meio do modelo de DAGs (*Direct Acyclic Graphs*). Com ele, é possível especificar relações de dependência entre tarefas de um programa paralelo.

A camada de simulação do *kernel* é o núcleo das funcionalidades do simulador e é responsável pela simulação da plataforma virtual. Essa, por sua vez, é composta pelo módulo SURF (*SimUlation aRtiFact*) e é a base para a camada de nível superior (Ambiente de programação).

A camada base é composta pelo módulo XBT (*eXtended Bundle of Tools*). O XBT é uma biblioteca portável que fornece suporte a registros (*logs*) e algumas estruturas de dados, tais como: Fifo (fila genérica), Dict (dicionário genérico), *Heap*

(*heap* genérico), *Set* (conjunto de dados genéricos) e Swag (tipo de dado baseado em listas encadeadas).

SimGrid é um projeto de código aberto (*opensource)*. O simulador opera em modo texto e está disponível para ambientes Linux, Windows e MacOS. Ele é implementado na linguagem de programação C e utiliza arquivos XML (*Extensible Markup Language*) como entrada. Nesses arquivos são definidas as características da simulação, tais como a topologia de rede e as características e responsabilidades dos nós.

No simulador, as aplicações são modeladas pela manipulação de tarefas e de recursos. Na configuração, as tarefas são divididas em “tarefas de computação”, que utilizam os recursos de processamento (CPU), ou “tarefas de transmissão”, que utilizam o canal de comunicação. Fica sob responsabilidade do programador assegurar a correspondência adequada entre tarefas de computação/processadores e tarefas de transferência de dados/canais de comunicação.

O ambiente de simulação é definido por um arquivo fonte que contém a codificação do que se deseja simular. O SimGrid faz uso de dois arquivos de entrada: o *deployment.xml* e o *platform.xml*. O arquivo (*deployment.xml*) contém as especificações dos processos, a quantidade de tarefas, de computação embutida em cada tarefa e de comunicação de rede necessária para executar cada tarefa.

A Figura 2.7 ilustra o arquivo XML utilizado para definir os parâmetros do ambiente da simulação com quatro nós. Nesse exemplo, o “*Host 0*” possui a função de nó mestre e os “*Host 1*”, “*Host 2*”, “*Host 3*” e “*Host 4*” são definidos como nós escravos.

```
1: <?xml version='1.0'?>
2: <!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
3: <platform version="3">
4:       <process host="Host 0" function="master"/>
5:       <process host="Host 1" function="worker"/>
6:       <process host="Host 2" function="worker"/>
7:       <process host="Host 3" function="worker"/>
8:       <process host="Host 4" function="worker"/>
9: </platform>
```

Figura 2.7: Exemplo de arquivo deployment.xml para quatro nós, adaptada de SimGrid (2012).

A plataforma computacional é definida por um arquivo fonte (*platform.xml*), no qual descreve-se a topologia de rede e suas características, como latência e largura

de banda, o poder computacional do nó e a quantidade de núcleos. Os recursos computacionais são modelados por duas características de desempenho: o poder computacional (unidades de trabalho executadas por unidade de tempo) dado em FLOPS (operações em ponto flutuante por segundo) e os links de comunicação, caracterizados pela largura de banda (bytes por segundo) e pela latência.

A Figura 2.8 ilustra o arquivo XML utilizado para definir os parâmetros da simulação de uma plataforma com quatro nós. No exemplo, especificam-se os *hosts*, o poder computacional, os canais de comunicação (largura de banda e latência) e as rotas de comunicação entre os *hosts*.

```
1: <?xml version='1.0'?>
2: <!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
3: <platform version="3">
4:   <AS id="AS0" routing="Full">
5:    <host id="Host 0" power="6393254000.0" core="2" />
6:    [...]
7:    <host id="Host 4" power="6393254000.0" core="2" />
8:    <link id="l1" bandwidth="392000000.0" latency="1e-4" />
9:    [...]
10:   <link id="l4" bandwidth="392000000.0" latency="1e-4" />
11:   <route src="Host 0" dst="Host 1">
12:       <link_ctn id="l1"/>
13:   </route>
14:   [...]
15:   <route src="Host 1" dst="Host 2">
16:               <link_ctn id="l1"/>
17:               <link_ctn id="l2"/>
18:      </route>
19:   [...]
20: </AS>
21: </platform>
```

Figura 2.8: Exemplo de arquivo *platform.xml,* adaptada de SimGrid (2012).

2.3.2 Simuladores de Aplicações Mapreduce

Dentre os simuladores de aplicações Mapreduce existentes, o simulador de eventos discretos Mumak (TANG, 2009) permite simular um ambiente homogêneo com milhares de nós e reproduzir o comportamento de uma aplicação com boa acurácia. Porém, ele não simula operações de sincronização de entrada/saída, omite a fase *shuffle,* calculando-a por estimativa, bem como não simula a comunicação pelo envio de *heartbeat.*

O simulador MRSim foi desenvolvido com o objetivo de estimar o custo de execução de aplicações Mapreduce em *grids* remotos, como o EC2 e o SunGrid. Ele é composto por um simulador de eventos discretos (SimJava) e um simulador de rede *GridSim, que efetua o split e a replicação* de dados local (HAMMOUD, 2010).

O MRPerf é um simulador proposto por Wang (2009) e foi projetado com o objetivo de planejar a implantação de ambientes de larga escala. Ele limita-se a um único dispositivo de armazenamento por nó, não simula características como execução especulativa e nem a replicação dos blocos de entrada de dados.

2.3.2.1 O Simulador *MapReduce Over SimGrid* (MRSG)

Com o objetivo de comprovar e viabilizar o desenvolvimento de experimentos de ambientes homogêneos e heterogêneos em larga escala, um grupo de pesquisa com foco em Mapreduce da Universidade Federal do Rio Grande do Sul desenvolveu o simulador MRSG (*MapReduce Over SimGrid*) (KOLBERG; ANJOS, 2011). Este foi desenvolvido sobre o SimGrid para simular o ambiente e o comportamento do Mapreduce em diferentes plataformas (ANJOS, 2012). Assim, o gerenciamento da execução de aplicações Mapreduce e o escalonamento de tarefas são realizados pelo MRSG e a computação dos nós e a simulação do ambiente pelo SimGrid.

Assim como no SimGrid, as definições da arquitetura, tais como o número e as características dos processadores, enlaces de rede e topologia são descritas em um arquivo de plataforma no formato XML. As configurações do *job* são especificadas em um arquivo com extensão *.conf*, no qual se define: o número de tarefas *reduce*, o tamanho dos *chunks*, o tamanho da entrada de dados, o número de réplicas, o percentual de saída de dados produzido pela fase *map*, o custo de cada tarefa *map* e *reduce* e o número de *slots* disponíveis para as tarefas *map* e *reduce*. A Figura 2.9 ilustra o arquivo *.conf* utilizado no simulador MRSG.

O simulador MRSG necessita de uma descrição simplificada dos dados de entrada, como o tamanho e o custo das tarefas. Assim, de maneira determinística pode-se simular a execução das tarefas, determinando quais dados serão mais relevantes para o experimento e o simulador deverá produzi-los.

```
1: reduces 26
2: chunk_size 178
3: input_chunks 192
4: dfs_replicas 3
5: map_slots 2
6: reduce_slots 2
```

Figura 2.9: Exemplo do arquivo *mrsg.conf.*

Uma vez especificados os dados de entrada (*platform.xml*, *deploy.xml* e *mrsg.conf*), deve-se especificar na API do MRSG os seguintes parâmetros:

- *My_map_output_function*: indica a quantidade de bytes que uma tarefa *map* irá emitir para uma tarefa *reduce*;
- My_*task*_cost_function: usuário indica o custo de uma tarefa *map* e uma tarefa *reduce*, medida em FLOPs.

A execução do MRSG emite diversas informações de saída, tais como ilustra a Figura 2.10.

```
[Host 0:master:(1) 0.000000] [msg_test/INFO] JOB CONFIGURATION:
[Host 0:master:(1) 0.000000] [msg_test/INFO] slots: 2 map, 2 reduce
[Host 0:master:(1) 0.000000] [msg_test/INFO] chunk replicas: 3
[Host 0:master:(1) 0.000000] [msg_test/INFO] chunk size: 178 MB
[Host 0:master:(1) 0.000000] [msg_test/INFO] input chunks: 192
[Host 0:master:(1) 0.000000] [msg_test/INFO] input size: 34176 MB
[Host 0:master:(1) 0.000000] [msg_test/INFO] maps: 192
[Host 0:master:(1) 0.000000] [msg_test/INFO] reduces: 26
[Host 0:master:(1) 0.000000] [msg_test/INFO] workers: 28
[Host 0:master:(1) 0.000000] [msg_test/INFO] grid power: 1.79011e+11 flops
[Host 0:master:(1) 0.000000] [msg_test/INFO] average power: 6.39325e+09 flops/s
[Host 0:master:(1) 0.000000] [msg_test/INFO] heartbeat interval: 3s
[Host 0:master:(1) 0.000000] [msg_test/INFO] JOB BEGIN
[Host 0:master:(1) 0.001301] [msg_test/INFO] map 1 assigned to Host 1
[Host 0:master:(1) 0.003903] [msg_test/INFO] map 2 assigned to Host 2
[...]
[Host 0:master:(1) 204.091070] [msg_test/INFO] reduce 0 assigned to Host 1
[Host 0:master:(1) 204.093672] [msg_test/INFO] map 58 assigned to Host 2
[Host 0:master:(1) 204.094973] [msg_test/INFO] reduce 1 assigned to Host 2
[...]
[Host 0:master:(1) 612.356465] [msg_test/INFO] map 190 assigned to Host 24 (non-
local)
[Host 0:master:(1) 817.925829] [msg_test/INFO] MAP PHASE DONE
[Host 0:master:(1) 1399.239751] [msg_test/INFO] REDUCE PHASE DONE
[Host 0:master:(1) 1399.239751] [msg_test/INFO] JOB STATISTICS:
[Host 0:master:(1) 1399.239751] [msg_test/INFO] local maps: 191
[Host 0:master:(1) 1399.239751] [msg_test/INFO] non-local maps: 1
[Host 0:master:(1) 1399.239751] [msg_test/INFO] speculative maps (local): 0
[Host 0:master:(1) 1399.239751] [msg_test/INFO] speculative maps (remote): 0
[Host 0:master:(1) 1399.239751] [msg_test/INFO] total non-local maps: 1
[Host 0:master:(1) 1399.239751] [msg_test/INFO] total speculative maps: 0
[Host 0:master:(1) 1399.239751] [msg_test/INFO] normal reduces: 26
[Host 0:master:(1) 1399.239751] [msg_test/INFO] speculative reduces: 0
[Host 0:master:(1) 1399.239751] [msg_test/INFO] JOB END
```

Figura 2.10: Exemplo de saída do MRSG.

2.4 Medidas de Escalabilidade

A crescente demanda por processamento de grandes bases de dados requer sistemas que ofereçam altos níveis de desempenho e escalabilidade. À medida que a carga de trabalho aumenta, é desejável que o sistema mantenha desempenho proporcional.

Pode-se definir *speedup* como o aumento de velocidade observado na execução de uma determinada tarefa com vários processadores. O *speedup* pode ser medido pela razão entre o tempo de execução de um processo de maneira sequencial utilizando-se o melhor algoritmo e o tempo da execução desse processo em *p* processadores. Essa relação é expressa como:

$$S(p) = \frac{Ts}{Tp} \quad (1)$$

A expressão *S(p)* indica o aumento de velocidade com o uso de *p* processadores. Nessa expressão, T_s refere-se ao tempo de execução do melhor algoritmo sequencial no ambiente monoprocessado. O valor de T_p refere-se ao tempo de execução em um ambiente multiprocessado.

O ganho ideal de *speedup* com a inserção de *p* processadores deveria ser proporcional a *p*. Contudo, diversos fatores influenciam nessa relação, tais como a sobrecarga de comunicação entre os processadores e a rede, a parte não paralelizável do código, entre outros. O *speedup* máximo de um sistema relaciona o desempenho obtido com a fração de melhoria do sistema. Há casos em que se observa um aumento da velocidade em um valor maior do que *p.* A esse efeito denomina-se *speedup* superlinear.

A medida de eficiência considera a relação de *speedup* com o número de processadores e o aumento de desempenho proporcionado com a paralelização, por sua vez, indica a proporção de ganho com o uso dos *p* processadores por meio da expressão 2.

$$E = \frac{Ts}{Tp * P} \quad (2)$$

Nessa expressão, T_s refere-se ao tempo de execução com apenas um processador e T_p usando p processadores. A eficiência máxima, no caso ideal, seria o valor 1.0, equivalente a 100%.

2.5 Projeto de Experimentos

A fim de medir o desempenho de uma aplicação, uma abordagem consiste em analisar o efeito de cada fator isoladamente em relação aos demais. Um projeto de experimentos tem por objetivo obter o máximo de informação com o mínimo possível de experimentos, levando em consideração o trabalho a ser executado e o custo. A análise prévia dos experimentos visa a ajudar a separar os diversos fatores que podem afetar o desempenho, a mensurar erros e a delimitar parâmetros que não podem ser controlados (JAIN, 1991).

No projeto e na análise de experimentos, alguns termos são frequentemente usados, tais como:

- Variável de Resposta: é a resposta do experimento que, em geral, equivale à medida de desempenho do sistema. Nos experimentos realizados, considera-se como a variável de resposta o *timeout* (tempo total de execução) de cada experimento;
- Fator: cada variável que afeta a variável de resposta. Nos experimentos de simulação de aplicações Mapreduce, considera-se cada parâmetro utilizado na configuração do experimento como um fator, por exemplo, o número de tarefas *map* e *reduce*, o tamanho do *chunk* de dados, o número de replicações, entre outros;
- Nível: refere-se aos valores que cada fator pode assumir. Cada nível constitui uma alternativa para determinado fator, por exemplo, o tamanho do *chunk* de dados com dois níveis, 16 e 128 Megabytes;
- Fatores primários: os fatores cujos efeitos necessitam ser quantificados;
- Fatores secundários: fatores que impactam no desempenho, contudo não são quantificados;
- Replicação: refere-se à repetição de todos ou de alguns experimentos;

- Projeto: um projeto de experimentos consiste em especificar o número de experimentos, as combinações dos fatores e seus níveis e o número de replicações para cada experimento;
- Unidade experimental: toda entidade utilizada para o experimento;
- Interação: considera-se que dois fatores A e B interagem entre si caso o efeito de um dependa do nível do outro.

Há vários tipos de projeto de experimentos. Jain (1991) descreve os três tipos mais utilizados: projeto simples, projeto fatorial completo e o projeto fatorial fracionado. O projeto simples consiste na realização de experimentos tendo por base uma configuração típica na qual se varia um fator por vez, baseado no seu impacto no desempenho. Considerando k fatores com o *i-ésimo* fator tendo n_i níveis, o projeto simples requer n experimentos, onde

$$n = \prod_{i=1}^{k} \mathrm{n_i} \quad . \tag{3}$$

Esse tipo de projeto, na prática, não é estatisticamente eficiente e pode direcionar a análise a conclusões erradas.

O projeto fatorial completo faz uso da combinação de todas as configurações possíveis entre todos os fatores e seus níveis. Consequentemente, torna possível identificar o efeito gerado por todos os fatores, incluindo os efeitos secundários ocasionados pela combinação dos mesmos. O desempenho com k fatores e n_i níveis, requer n experimentos, como demonstrado pelo produto a seguir.

$$\mathrm{n} = 1 + \sum_{i=1}^{k} (\mathrm{n_i} - 1) \tag{4}$$

A vantagem do projeto fatorial consiste em examinar todas as possíveis combinações de fatores, encontrando o efeito de todos os fatores, incluindo os secundários e suas interações. Contudo, esse tipo de projeto apresenta um custo (tempo e recursos) elevado para ser executado, o que o torna inviável por não possibilitar a repetição dos experimentos.

O projeto fatorial fracionário é uma alternativa que possibilita a redução do número de experimentos, o que, consequentemente, reduz também o número de fatores e suas combinações. Assim, com a análise prévia das interações, o projeto

fatorial fracionário reduz o número de experimentos para n^k, sendo que n é a quantidade de níveis e k é a de fatores.

Jain (1991) relata que há três maneiras de reduzir o número de experimentos: reduzir o número de níveis de cada fator, reduzir o número de fatores e usar o projeto fatorial fracionado. Em muitos casos, representar cada fator com apenas dois níveis já é o suficiente para determinar o seu efeito no projeto.

2.5.1 Projeto 2^k Fatorial

O projeto 2^k fatorial determina o efeito de cada um dos fatores e seus níveis em relação à variável de resposta, ou seja, em relação à saída analisada. O 2^k fatorial analisa cada fator em dois níveis, ou seja, um valor mínimo e um máximo para cada fator. A determinação da influência de cada um deles é descrita pelo modelo de regressão linear a seguir:

$$y = x_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B \tag{5}$$

A variável y é a variável de resposta do experimento, que nesse projeto refere-se à eficiência computacional. Nesse modelo de regressão linear, cada x_i (i refere-se a cada fator no experimento, na equação entende-se por fatores A e B) pode assumir valores 1 e -1, o que representa os dois níveis de cada fator. Com a combinação dos fatores e seus níveis, assumindo um projeto de experimentos com 2^2 (k=2, fator A e fator B), pode-se observar os seguintes cenários:

$$\begin{aligned} y_1 &= q_0 - q_A - q_B + q_{AB} \\ y_2 &= q_0 + q_A - q_B - q_{AB} \\ y_3 &= q_0 - q_A + q_B - q_{AB} \\ y_4 &= q_0 + q_A + q_B + q_{AB} \end{aligned} \tag{6}$$

A correspondência entre o nível do fator e a resposta é apresentada na Tabela 2.1.

Tabela 2.1: Correspondência entre o nível do fator e a resposta.

Experimento	A	B	Y
1	-1	-1	y_1
2	1	-1	y_2
3	-1	1	y_3
4	1	1	y_4

Resolvendo as quatro equações para q_i´s em relação à variável de resposta (eficiência) para cada cenário, tem-se:

$$q_0 = \frac{1}{4}(y_1 + y_2 + y_3 + y_4)$$
$$q_A = \frac{1}{4}(-y_1 + y_2 - y_3 + y_4)$$
$$q_B = \frac{1}{4}(-y_1 - y_2 + y_3 + y_4)$$
$$q_{AB} = \frac{1}{4}(y_1 - y_2 - y_3 + y_4) \quad (7)$$

A importância de um fator é medida pela proporção da variação total da variável de resposta em relação a um fator. Por exemplo, considerando que um fator explica 90% e outro 10% da variação total da variável de resposta, o segundo fator pode ser considerado de menor importância na prática.

A variação total da variável de resposta *y* pode ser calculada pela soma dos quadrados do total (SST – *Sum of Squares Total*), descrita na equação 8,sendo que, *k* é a quantidade de fatores, y_i é a variável de resposta para o experimento *i* e $\bar{y}$ é a média da variação da variável de resposta *y* de todos os experimentos.

$$Variação\ total\ de\ y = SST = \sum_{i=1}^{2^k}(y_i - \bar{y})^2 \quad (8)$$

O SST pode ser descrito como o somatório da proporção da variação total explicada por cada fator e suas interações, conforme ilustrado na equação 9:

$$SST = 2^2\mathrm{q}_{\mathrm{A}}^2 + 2^2\mathrm{q}_{\mathrm{B}}^2 + 2^2\mathrm{q}_{\mathrm{AB}}^2 \quad (9)$$

Assim, para k = 2, o SST total é o somatório da variação explicado pelo fator A ($SSA = 2^2\mathrm{q}_{\mathrm{A}}^2$), somado à variação explicada pelo fator B ($SSB = 2^2\mathrm{q}_{\mathrm{B}}^2$) e somada à variação explicada pela interação AB ($SSAB = 2^2\mathrm{q}_{\mathrm{AB}}^2$) conforme descrito na equação 10.

$$SST = \mathrm{SSA} + \mathrm{SSB} + \mathrm{SSAB} \quad (10)$$

A fim de expressar o percentual da importância dos fatores, calcula-se a fração explicada por cada um deles em relação à variação total, como representado na equação 11.

$$Fração\ de\ variação\ explicada\ por\ A = \frac{SSA}{SST} \tag{11}$$

Na análise de escalabilidade dos experimentos, optou-se pela abordagem do projeto 2^k fatorial, no qual, dado um experimento, é possível observar o percentual do efeito dos k fatores em dois níveis possíveis. Assim, é possível definir quais fatores apresentam maior impacto no desempenho.

Capítulo 3

ANÁLISE DOS FATORES DE MAIOR IMPACTO

Neste capítulo, reporta-se a análise 2^k fatorial realizada para avaliar a influência de um parâmetro ou da combinação de um conjunto de parâmetros de uma aplicação Mapreduce. Para a realização dos testes, foram criadas diferentes configurações de experimentos que permitem estimar qual é o efeito dos k fatores e os seus impactos no desempenho de uma aplicação.

3.1 Ambiente Experimental

O ambiente Hadoop possui aproximadamente 190 parâmetros de configuração, dos quais observou-se que aproximadamente 25 interferem mais significativamente no desempenho de um *job* (BABU, 2010).

Nesse contexto, a fim de avaliar os limites de escalabilidade, faz-se necessário identificar os parâmetros de maior impacto, bem como encontrar a configuração ideal do conjunto de parâmetros que permitirá obter a máxima escalabilidade de uma aplicação. Portanto, foram criadas diferentes configurações de experimentos e cada uma delas era composta pela combinação entre parâmetros e seus valores de referência. A análise dessa combinação tornou possível avaliar o impacto dos parâmetros (ou sua combinação) na execução.

O ambiente de simulação era composto por um conjunto de programas que criam os dados de entrada, executam as simulações e extraem as informações

desejadas. Neste trabalho, o ambiente utilizado foi o simulador SimGrid, que simula plataformas como *clusters*, *grids* e sistemas P2P (CASANOVA, 2008).

No Simgrid, um arquivo *XML* permite descrever o poder computacional dos nós, a largura de banda dos *links* e a topologia da rede do sistema. Para simular as aplicações, optou-se pelo MRSG (*MapReduce Over SimGrid*), que simula aplicações Mapreduce sob o SimGrid (KOLBERG; ANJOS, 2011). MRSG lê um arquivo texto (*mrsg.conf*) que descreve os parâmetros da aplicação, tais como: o número de tarefas *map*, o tamanho do *chunk* de dados, o custo das tarefas *map* e *reduce,* entre outros.

3.2 Projeto de Experimentos 2^k Fatorial

Na análise da influência dos fatores na escalabilidade dos experimentos, optou-se pela abordagem do projeto 2^k fatorial, com o objetivo de observar o impacto de cada um dos fatores no desempenho.

Dentre os 190 parâmetros de configuração do ambiente Hadoop, optou-se pela experimentação de 9 parâmetros, sendo estes os mais utilizados no ambiente de simulação para modelar a aplicação Mapreduce experimental. Para o melhor entendimento, conceituam-se os parâmetros utilizados nos experimentos:

- Número de nós: quantidade de nós que compõem o *cluster* simulado;
- Número de *reduces*: quantidade de tarefas *reduce* que serão executadas na simulação;
- *Chunk Size*: tamanho do bloco de dados em que a base deve ser particionada. Ex. base de dados de 1 Gigabyte particionada em 16 blocos de 64 Megabytes cada;
- DFS Replicas: quantidade de réplicas de cada bloco de dados na plataforma;
- *Map_Output*: percentual de saída de dados produzidos pela fase *map*, considerando o valor da base de entrada. Por exemplo, para a base de dados de entrada de 1 Gigabyte, um *mapout* de 10% iria produzir 102,4 Megabytes de dados na fase *map*;

- *Map_Cost* e *Reduce_Cost*: custo de processamento por *byte* em cada fase. Esses dois parâmetros são utilizados para estimar a quantidade de trabalho a ser executado em cada tarefa *map* e *reduce*. Dessa maneira, para modelar corretamente a aplicação no simulador, faz-se necessário o ajuste desses custos de processamento.

Além dos sete parâmetros descritos, o simulador MRSG considera o número de *slots map* e *slots reduce*. Nessa análise, considera-se o número de *slots* igual ao número de *core* de cada nó da plataforma experimental, ou seja, o valor fixo de dois *slots map* e dois *slots reduce*.

Após a escolha dos parâmetros iniciais, desenvolveu-se um *shell script*, responsável por criar as *k* combinações de cada fator e nível e, para cada combinação possível, criar o arquivo de configuração que é utilizado como entrada do simulador MRSG. A execução das simulações foi automatizada por outro *shell script* que executa e extrai o *log* e as informações necessárias para a análise.

Os cálculos da análise 2^k fatorial foram automatizados por meio da criação de uma planilha também utilizada para desenhar os gráficos que evidenciam os fatores de maior impacto.

3.3 Fatores e seus níveis

O projeto de experimentos 2^k fatorial considera o problema de analisar o impacto de 7 diferentes fatores em relação à variável de saída da aplicação ora simulada. Essa combinação de fatores e níveis foi utilizada como configuração da simulação no MRSG. Tais experimentos, considerando-se 7 fatores com 2 níveis cada (2^7experimentos), permitiram analisar 128 combinações diferentes.

Além das 128 combinações, foram geradas 32 combinações adicionais destinadas a representar o tempo de execução sequencial (considerando 1 nó) utilizado para o cálculo da eficiência.

A combinação de cada fator e de seus níveis foi gerada e salva em um arquivo, que juntamente com os arquivos de plataforma e de ambiente compõem os dados de entrada do simulador. Para todos os experimentos, a base de dados foi fixada em 20480 Megabytes (20 Gigabytes). Vale notar que o número de tarefas

map lançadas para a execução é calculado pela divisão da base de dados pelo tamanho do *chunk*. Por exemplo, considerando a base de dados acima e o tamanho de *chunk* de 128 Megabytes, um total de 160 tarefas *map* (160 *chunk*s) serão configuradas. Deve-se observar que o número de tarefas *map* torna-se fixo em função da base de dados de entrada e do tamanho do *chunk* de dados.

Após a execução da simulação, foram extraídos os seus *logs* com os dados necessários para a análise, em especial, o valor da eficiência. Esse valor foi calculado por meio do tempo da execução da configuração paralela e do tempo da execução da mesma combinação em um único nó, de forma sequencial. O valor de eficiência foi a variável de saída utilizada na análise do impacto dos fatores no experimento.

3.3.1 Primeiro Experimento

Os fatores e os níveis analisados nesse primeiro experimento estão descritos na Tabela 3.1. A escolha desses níveis foi realizada com o objetivo de analisar, inicialmente, valores extremos de configuração e, nos experimentos seguintes, refinar esses valores.

Tabela 3.1: Fatores e níveis do primeiro experimento 2^k fatorial

Número de Nós	Número de *Reduces*	*Chunk Size*	Dfs réplicas	*Map Output*	*Map Cost*	*Reduce_Cost*
10	20	16	1	1	1	1
1000	2000	128	10	1000	10000	10000

A representação gráfica (Figura 3.1) resultante do processamento das 128 configurações (1 a 127 no gráfico) possíveis entre fatores e níveis da tabela anterior descreve o impacto de cada combinação. Para proporcionar uma visualização do impacto de cada uma das 128 combinações, o gráfico da Figura 3.1 está ilustrando os valores de 0,0 a 1%. Vale observar que, na grande maioria das combinações, o impacto foi inferior a 0,70%, o que se considera negligenciável.

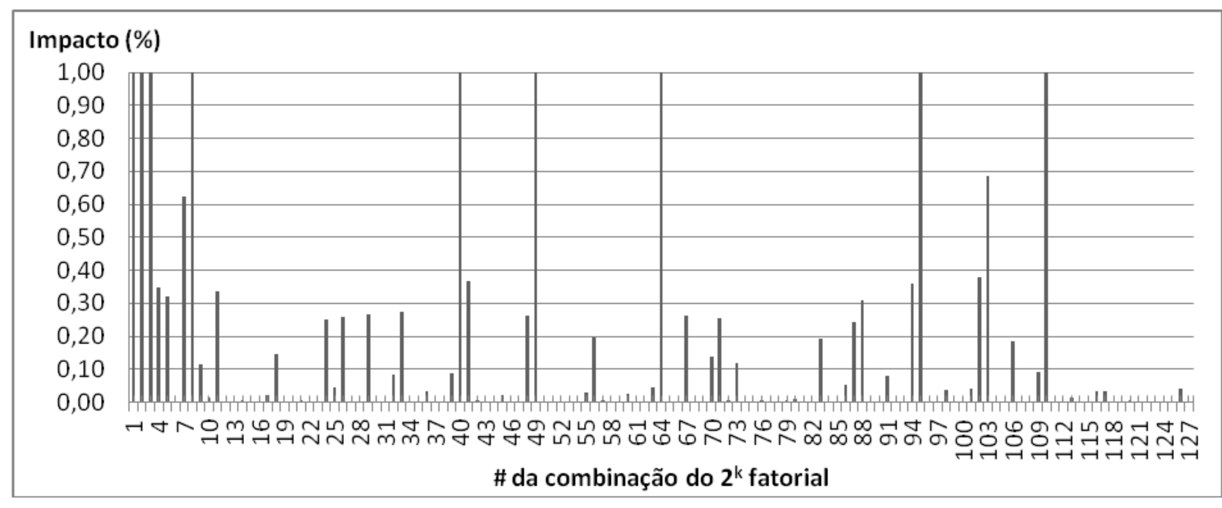

Figura 3.1: Percentual de impacto dos fatores para o primeiro experimento.

Na Tabela 3.2, encontra-se o resultado da análise evidenciando os fatores e as combinações de fatores de maior impacto. Na tabela evidenciamos apenas os fatores (ou combinações de fatores) que resultaram em maior impacto.

Tabela 3.2: Percentual de impacto dos fatores sobre a eficiência do primeiro experimento.

Número do Experimento	Fator/Combinação de fatores	Impacto (%)
02	Número de Nós	25,3
01	Número de *Reduce*s	22,0
64	Número de *Reduce*s + Map Output	12,4
40	Número de *Reduce*s + *Reduce*_Cost	10,8
110	*Reduce_Cost* + *Map Output*	8,0
49	Número de *Reduce*s + *Reduce_Cost* + *Map Output*	4,6
03	*Reduce Cost*	4,2
95	Número de Nós + *Map Output*	3,4

Observa-se que, para a configuração analisada, o número de nós é o parâmetro de maior impacto evidenciado. Essa resultante já é esperada como fator impactante pois o número de tarefas a ser executada é maior que o número de nós disponível, fato este que resulta em tarefas em espera a ser executada.

O número de tarefas *reduce* também é apresentado como de grande impacto. Este fato deve-se ao baixo paralelismo das tarefas reduce e, combinado com o grande volume de dados produzido pela fase map *(map output)* refletem no desempenho da aplicação, como observado na Tabela 3.2 nos experimentos 01 e 64.

Nos demais experimentos relacionados na tabela, observa-se que a combinação do número de tarefas *reduce*, o *map output* e o custo de processamento de tarefas *reduce* tiveram menor impacto.

3.3.2 Segundo Experimento

No segundo lote de experimentos, para representar outro cenário de aplicação Mapreduce, optou-se pela redução dos valores dos níveis dos fatores, conforme descrito na Tabela 3.3. Com uma variação menor, buscou-se avaliar o impacto dos fatores em situações menos adversas.

Tabela 3.3: Fatores e níveis do segundo experimento

Número de Nós	Número de *Reduces*	*Chunk Size*	Dfs réplicas	*Map Output*	*Map Cost*	*Reduce_ Cost*
10	20	16	1	50	10	10
1000	2000	128	10	200	100	100

A simulação dessas 128 configurações possíveis produziu o resultado gráfico ilustrado pela Figura 3.2. Observa-se no gráfico que, com a redução do percentual de saída de dados da fase *map*, o parâmetro *chunk size* é evidenciado como fator de alto impacto. Considerando que o *chunk size* refere-se à quantidade de dados que cada tarefa *map* deve processar, é natural observar que o número de nós seja um fator impactante, já que uma menor quantidade de nós reflete uma menor quantidade de tarefas *map* sendo executas concorrentemente.

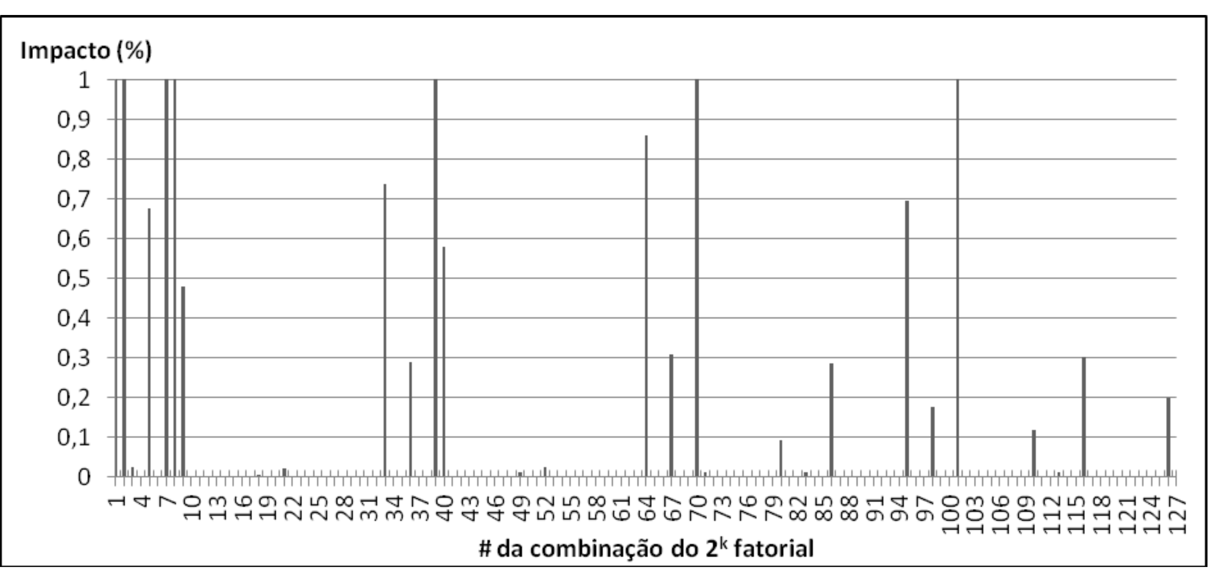

Figura 3.2: Percentual de impacto dos fatores para o segundo experimento.

Na Tabela 3.4 encontra-se o resultado da análise, evidenciando os fatores e as combinações de fatores de maior impacto.

Tabela 3.4: Percentual de impacto de cada fator sobre a eficiência do segundo experimento.

Número do Experimento	Fator/Combinação de fatores	Impacto (%)
02	Número de Nós	58,3
01	Número de *Reduce*s	16,5
08	Número de *Reduce*s + Número de Nós	11,4
07	*Chunk Size*	3,0
39	Número de *Reduce*s + Número de Nós + *Chunk Size*	2,2
101	Número de Nós + *Chunk Size*	1,4

3.4 Considerações do Capítulo

Plataformas reais, tal como o Hadoop, possuem grande número de fatores que podem ser configuráveis e muitos deles causam impacto no desempenho das aplicações. A análise 2^k fatorial permite identificar os fatores mais importantes na aplicação considerando uma variável de saída, que no caso do experimento realizado, era a eficiência.

O uso da eficiência como variável de saída significa não só medir o melhor tempo de resposta, mas determinar a escalabilidade, ou o benefício proporcional com o aumento do número de nós.

Nesta análise, os fatores mais impactantes foram o número de nós, o número de *reduces*, o tamanho do *chunk size*, a saída da fase *map* e o custo das tarefas *reduce*. Tais fatores serão utilizados nos experimentos de análise de escalabilidade descritos no próximo capítulo.

Capítulo 4

ANÁLISE DE ESCALABILIDADE DE UMA APLICAÇÃO MAPREDUCE

Para a realização da análise de escalabilidade de aplicações Mapreduce, uma estratégia interessante seria a realização de experimentos em plataformas de grande porte. Contudo, devido a limitações no acesso a tais plataformas, na realização deste trabalho, optou-se por combinar execução real com simulação. A execução real permitiu a extração de informações necessárias à modelagem da aplicação e a consequente calibração (ajuste) do simulador. A calibração foi necessária para que houvesse correspondência entre o ambiente simulado e o ambiente real. Uma vez calibrado e ajustado, foi possível estimar o custo de processamento das tarefas map e reduce para plataformas de maior porte, configurar o simulador para a realização de experimentos em larga escala e, posteriormente, analisar os limites de escalabilidade do modelo Mapreduce.

4.1 Ambiente De Execução Real

A metodologia adotada neste trabalho consiste, inicialmente, na execução de aplicações reais em um *cluster* do Departamento de Computação (DC-UFSCar). O ambiente de experimentação utilizado é composto por um *cluster* de 32 nós conectados por um *switch* Gigabit Ethernet. Cada nó de processamento possui 2 processadores AMD Opteron 246, 8 Gigabytes de memória RAM, 4 discos Maxtor Maxline de 250 Gigabytes cada e sistema operacional Linux Centos. O desempenho de cada nó foi de 6.393254 Gflops, medido através do *benchmark* Linpack.

Já a velocidade de transmissão da interface de rede entre nós no mesmo *switch* foi medida através do software *Iperf* que indicou largura de banda de 392 Mbps e latência de $1e^{-4}$. No *cluster* está instalado o Hadoop versão 1.0.4, com 4 *slots* para a execução concorrente de tarefas *map* e *reduce*. O valor *default* para o tamanho dos *chunks* de dados foi de 64 Megabytes, com fator de replicação igual a 3 e o intervalo entre os *heartbeats* gerados por cada nó *worker* foi de 3 segundos.

4.2 Aplicação utilizada nos Testes

Uma coleção de documentos é composta por um conjunto de registros ou palavras. A manipulação desse conjunto de dados pode ser realizada por meio de diferentes estratégias de recuperação textual e, em grandes coleções de dados, a operação de recuperação ou consulta é uma tarefa computacionalmente cara. Uma estratégia largamente empregada na recuperação de informação em grandes bases de dados é o índice invertido, composto por uma estrutura de busca que contém todos os termos existentes na base e, para cada termo, possui uma lista invertida que armazena os identificadores dos registros que contêm o termo, ou seja, o documento no qual o termo pode ser encontrado.

Em McCreadie, Macdonald e Ounis (2011) os autores apresentam o estudo da escalabilidade e da eficiência de 4 estratégias de indexação Mapreduce, sendo elas: a indexação *Per-Term*, a *Per-Document*, a *Per-Token* e a *Per-Posting list*. Em seu estudo, os autores concluem que a estratégia *Per-Posting list* é a mais escalável.

O processo de indexação da estratégia *Per-Posting list* é dividido em múltiplas tarefas *map*. Tais tarefas efetuam o processamento em um subconjunto de dados e comprimem os dados processados em listas para cada termo.

Ao final da execução de cada tarefa *map* é emitido um conjunto de pares <termo, lista> para todos os termos encontrados. Esse conjunto parcial de índices é ordenado por termos e as listas de cada termo são agrupadas para serem processadas pelas tarefas *reduce.* O pseudocódigo a seguir descreve a implementação das funções *map* (Algoritmo 4.1) e *reduce* (Algoritmo 4.2) para essa estratégia (MCCREADIE; MACDONALD; OUNIS, 2011).

Algoritmo 4.1: Função map

1: **Entrada**
  Chave: Identificação do Documento, Nome
  Valor: Conteúdo do Documento, DocCont
2: **Saída**
  Chave: Termo
  Valor: Posting List
3: **para cada** Termo **em** DocCont **faça**
4:   Stem (Termo)
5:   deleteStopword(Termo)
6:   **se** (Termo não encontrado na memória) **então**
    Adicione o documento atual do termo na Posting List em memória para este termo
7: **fim para**
8: Adicione o Documento no Índice de Documentos
9: **se** (últimoMap() ou MemóriaCheia()) **então**
  Para cada Termo na Posting List em memória, emita(Termo, Posting List)
10: **se** (últimoMap()) **então**
11:   escreva as informações sobre os documentos que este map processou (arquivos “side-effect”)

Algoritmo 4.2: Função reduce

1: **Entrada**
  Chave: um Termo
  Valor: Lista de *Posting List*, ParcialPostingLists
2: **Saída**
  Chave: Termo
  Valor: *Posting-List*
3: Lista Posting-List = new PostingList()
4: **Sort** ParcialPostingList por map e **flush**
5: **para cada** PostList **em** ParcialPostingLists **faça**
6: **para cada** doc-ID **em** PostList **faça**
7:   Revisar doc-ID
8:   **Merge** PostList **em** Posting-List
11: emit (Posting-List)

A ferramenta de recuperação de informações Terrier (TERRIER, 2012), que implementa a indexação *Per-Posting list,* é um sistema desenvolvido em Java voltado ao processamento de documentos em larga escala. Para a composição de índices invertidos, o Terrier analisa um *corpus* de texto de forma sequencial ou paralela, gerando um *job,* submetido para execução no *cluster* com Hadoop. Por fim, Terrier produz como resultado índices invertidos contendo dados estatísticos sobre a incidência de cada termo encontrado nos documentos da coleção.

Os testes de indexação distribuída através do Terrier (versão 3.5) podem ser realizados em diferentes bases de dados, como a WT2G, a .GOV, a .GOV2 e a ClueWeb, desde que essa última seja composta por um dos formatos previstos (WARC - *Web ARChive*, por exemplo).

Sendo assim, buscou-se um conjunto de dados com grande representatividade, ou seja, que dentre as suas características apresentasse grande volume de dados bem conhecidos e largamente utilizados e que fosse semelhante ao que as aplicações reais utilizam. Outras características levadas em consideração para a escolha da base foram o fato dos dados já terem sido amplamente testados e que essa fosse de domínio público, fato esse que facilita a reproducibilidade dos experimentos.

As páginas web que compõem o Clueweb são armazenadas no formato de arquivo WARC e compactadas no formato gzip. O formato WARC foi padronizado pela ISO 28500 e permite a concatenação de múltiplos objetos de dados em um arquivo. Esse formato é largamente utilizado na construção de aplicações para coleta, gerenciamento, acesso e troca de conteúdo.

A base de dados escolhida foi a coleção Clueweb09, criada pelo *Language Technologies Institute* da Universidade Carnegie Mellon com o objetivo de oferecer suporte às pesquisas relacionadas às tecnologias de recuperação de informações da linguagem natural. Essa coleção consiste em cerca de 1 bilhão de páginas web, coletadas em 10 idiomas no período entre janeiro e fevereiro de 2009. Esse conjunto de dados já foi largamente estudado e otimizado, sendo utilizado em diversas trilhas na TREC 2009 (MCCREADIE; MACDONALD; OUNIS, 2011).

A base de dados Clueweb09_English_1 é um subconjunto do *corpus* ClueWeb09 composto por mais de 50 milhões de documentos. Na execução dos testes utilizamos dois fragmentos do Clueweb09_English_1, o enwp00.00.warc.gz e o enwp00.01.warc.gz.

O primeiro arquivo possui 186.524.558 bytes (aproximadamente 178 Megabytes), que, descompactado contém 1.048.603.018 bytes (aproximadamente 1 Gigabyte). O segundo arquivo possui 186.485.308 *bytes* (aproximadamente 178 Megabytes) que, descompactado possui 1.048.603.523 *bytes* (aproximadamente 1 Gigabyte).

Na realização dos testes optou-se pelo uso da base de entrada compactada (formato tar.gz). Essa escolha teve por objetivo minimizar o tempo de E/S (entrada e

saída) que, em geral, melhora desempenho da aplicação. A base de dados foi gerada replicando-se os dois arquivos até gerar 192 arquivos (consequentemente 192 tarefas *map*), totalizando aproximadamente 34 Gigabytes no HDFS. A base produzida contém:

- Número de Documentos: 4.169.664;
- Número de *Tokens*: 4.287.926.208;
- Número de Termos: 565.557;
- Total de *bytes* lidos do HDFS: 35.810.457.876 *bytes* (aprox. 34 GB).

Por se tratar de uma base de dados voltada à recuperação de informações da língua natural, os experimentos foram conduzidos com número fixo de 26 tarefas *reduce* em que cada tarefa *reduce* trata de uma letra do alfabeto. Em todos os experimentos foi fixado o número de 192 tarefas *map*, correspondente aos 192 arquivos de entrada. O grau de replicação de dados utilizado no HDFS foi 3 e tamanho de *chunk* de 64 Megabytes.

4.3 Tempo de Execução da Aplicação (*makespan*)

Cada experimento foi replicado duas vezes para plataformas com 32, 28, 24, 20, 16, 12, 8, 4 e 1 nó. Na Tabela 4.1 são descritos os tempos médios (em segundos) de execução das fases *map*, *reduce* e o *makespan* (tempo total de execução da aplicação).

Tabela 4.1: Tempo de Execução.

Número de nós	Fase *Map* (s)	Fase *Reduce* (s)	*Makespan* (s)
01	17.305	18.173	19.088
04	4.582	4.739	5.118
08	2.398	2.644	2.850
12	1.626	1.881	2.088
16	1.170	1.540	1.744
20	953	1.327	1.535
24	790	1.173	1.374
28	766	1.013	1.218
32	650	995	1.192

A Figura 4.1 ilustra o *makespan* da execução real da plataforma de 1 a 32 nós.

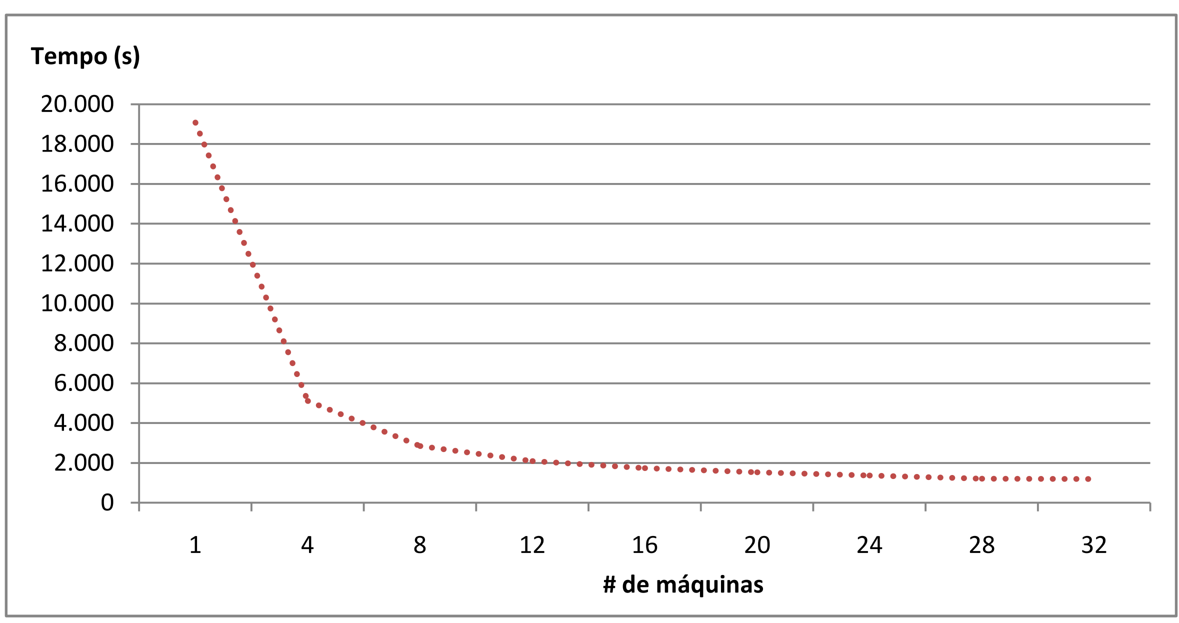

Figura 4.1: Gráfico de *Makespan* da aplicação de índices invertidos.

A seguir, calculamos o *speedup* para os tempos de execução das fases *map*, *reduce* e *makespan*. Os resultados estão representados na Tabela 4.2, e ilustrados de forma gráfica na Figura 4.2.

Tabela 4.2: *Speedup* da Fase *Map*, Fase *Reduce* e Total.

Número de nós	*Speedup* Fase *Map*	*Speedup* Fase *Reduce*	*Speedup* Total
04	3,78	3,84	3,73
08	7,22	6,87	6,70
12	10,64	9,66	9,14
16	14,78	11,80	10,95
20	18,16	13,70	12,44
24	21,91	15,50	13,90
28	22,58	17,94	15,67
32	26,62	18,26	16,02

Analisando o gráfico de *speedup* (Figura 4.2), observa-se que no caso do *map* há um comportamento sublinear, ou seja, linear a menos de uma constante. O *speedup* sublinear encontrado pode ser justificado pelos atrasos gerados pela sobrecarga do sistema na criação, comunicação, sincronização e finalização das tarefas e espera por operações de entrada e saída em disco.

Vale destacar o comportamento do *speedup* da plataforma com 28 nós, ilustrado na Figura 4.2. Considerando que, na plataforma com 28 nós são executadas 56 tarefas de maneira concorrente (duas em cada nó), e que no total devem ser executadas 192 tarefas *map*, restam 24 tarefas a serem executando na

última rodada. Portanto, tem-se na última rodada uma ociosidade de 16 nós (57%). Esta ociosidade causa o "cotovelo" que pode ser observado na Figura 4.2.

O *speedup* da fase *reduce* também apresenta um comportamento parecido. Nas execuções acima de 16 nós o *speedup* começa a cair, pois a aplicação possui apenas 26 tarefas *reduce*. Convém destacar que, apesar de haver apenas 26 tarefas *reduce* (o que causa ociosidade além de 13 nós), o *speedup* continua aumentando.

Este comportamento é justificado pelas otimizações realizadas na versão utilizada do Hadoop. A fase *reduce* tem início logo após o término das primeiras tarefas *map*. Em geral, isso reduz o tempo total de execução do *job*, mas aumenta a duração da fase *reduce* que deve aguardar pela execução de todas as tarefas da fase *map*. Uma evidência dessa otimização é verificada pela soma da duração das duas fases que é maior que o *makespan*.

Salienta-se que os resultados ora obtidos neste trabalho descrevem um comportamento um pouco diferente dos experimentos realizados em McCreadie; Macdonald e Ounir (2011), devido às diferentes versões do Hadoop utilizadas.

Com a execução real do Terrier na plataforma Hadoop e a base de dados de entrada ClueWeb foi possível extrair os tempos de execução bem como os demais parâmetros necessários para a etapa de calibração do simulador.

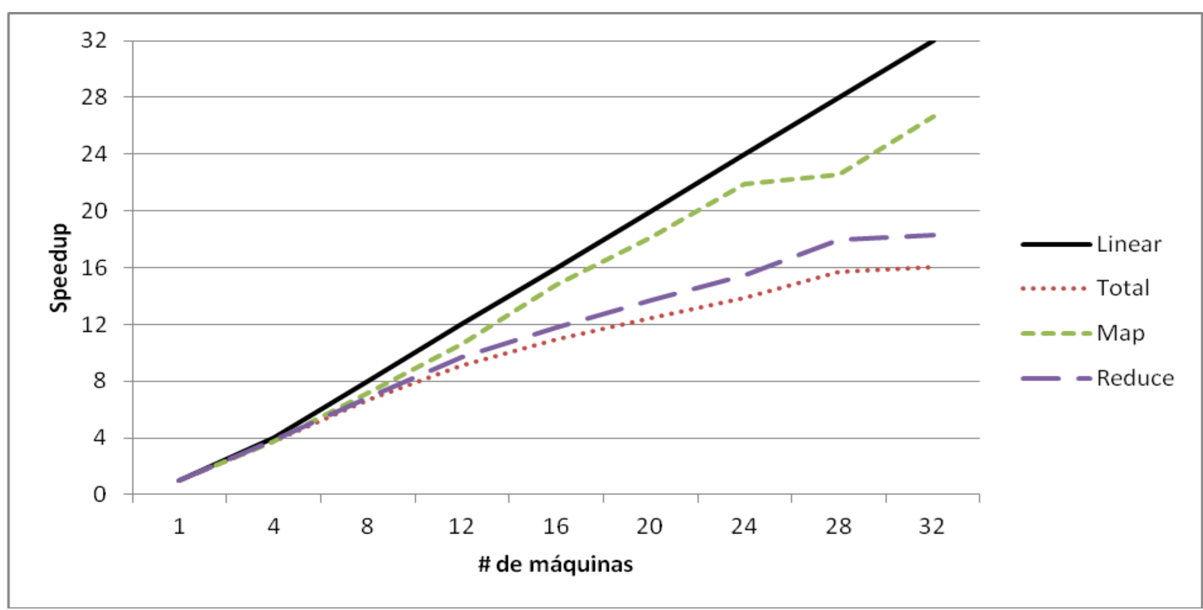

Figura 4.2: Gráfico de *Speedup* da Fase *Map*, Fase *Reduce* e Total.

4.4 Calibração do Simulador MRSG

Após a execução e coleta dos *logs* da experimentação real, procedemos com a calibração do simulador MRSG. Esta etapa teve por objetivo ajustar o simulador para que simule, da forma mais próxima possível, o comportamento da aplicação quando executando em uma plataforma real como a que foi descrita.

MRSG requer que sejam especificados diversos parâmetros referentes à plataforma e à aplicação a serem simuladas. A plataforma é especificada no *platform.xml* e os parâmetros da aplicação são configurados no arquivo *mrsg.conf*. Um dos parâmetros são os custos (*map_cost*, *reduce_cost*) que representam o processamento das tarefas, ou seja, esse custo depende do tipo de operação realizada pelas funções *map* e *reduce*.

Para representar adequadamente o comportamento de um sistema real, é necessário que diversos parâmetros do simulador sejam configurados e calibrados, ou seja, deve ser realizado o ajuste correto dos custos e demais parâmetros de aplicação permitindo que o simulador reproduza o comportamento da aplicação o mais próximo do real possível. Uma vez totalmente calibrado, pode-se realizar simulações e verificar a acurácia do simulador validando a simulação com êxito.

Espera-se que com o simulador calibrado e devidamente validado, ele possa ser utilizado na análise de diferentes cenários sem que seja necessária a implementação do sistema real.

O processo de calibração consiste em procurar valores de custo das tarefas *map* e *reduce* (*map_cost* e *reduce_cost*) do simulador, de tal modo que a diferença entre o tempo total simulado e o tempo total real sejam minimizadas. O custo por byte está relacionado à quantidade de trabalho que a tarefa deverá executar para cada byte recebido como entrada, possibilitando ao simulador estimar o tempo de processamento da tarefa.

Diversas execuções foram realizadas, de forma empírica, para encontrar o custo aproximado que resulte em simulações mais precisas. A cada execução manual da plataforma simulada foram realizados ajustes (aumentar ou diminuir o valor de custo) no parâmetro *map_cost* e *reduce_cost* de maneira que, o tempo de simulação corresponda o mais próximo possível do tempo real. Dessa maneira, foi

possível calibrar o simulador MRSG com os custos descritos na Tabela 4.3 para a fase *map* e na Tabela 4.4 para a fase *reduce*.

Tabela 4.3: Simulação da Fase *Map*.

Número de nós	Fase *Map* (Real) em segundos	*Map_Cost* (Sim)	Fase *Map* (Sim) em segundos	Erro (%)
01	17.305	1,10 E+12	16.719	3,50
04	4.582	1,17 E+12	4.402	4,09
08	2.398	1,17 E+12	2.237	7,19
12	1.626	1,29 E+12	1.634	0,49
16	1.170	1,23 E+12	1.175	0,42
20	953	1,22 E+12	962	0,94
24	790	1,25 E+12	793	0,37

Em relação a fase *map,* observa-se que nas execuções iniciais com 1, 4 e 8 nós o erro percentual observado foi maior, permanecendo na faixa de 3 a 7%. O erro percentual médio da fase *map* simulada ficou em torno de 2,43% em relação aos testes reais.

Analisando a fase *reduce* simulada, nota-se um comportamento semelhante à fase *map*, no qual o erro percentual foi maior nas execuções iniciais. Conforme ilustrado na Tabela 4.4, o erro percentual médio foi de aproximadamente 2,06% em relação à execução real.

Tabela 4.4: Simulação da Fase *Reduce*.

Número de nós	Fase *Reduce* (Real) em segundos	*Reduce_Cost* (Sim)	Fase *Reduce* (Sim) em segundos	Erro (%)
01	18.173	9,00 E+11	17.504	3,82
04	4.739	8,60 E+11	4.578	3,51
08	2.644	1,77 E+12	2.499	5,80
12	1.881	1,25 E+12	1.888	0,37
16	1.540	3,50 E+12	1.541	0,06
20	1.327	3,50 E+12	1.336	0,67
24	1.173	3,55 E+12	1.175	0,17

Com o objetivo de obter uma maior acurácia entre o makespan simulado e o real, buscamos ajustar os custos de *map* e *reduce* para execuções paralelas utilizando o valor do custo médio obtido da experimentação real (1 a 24 nós - Tabela 4.3 e Tabela 4.4) descontando-se os *outliers* (valor mínimo e o máximo) e o custo para um nó (execução sequencial).

Uma vez calculado o custo médio de *map* e *reduce,* repetimos os testes no MRSG de 01 a 24 nós conforme ilustrado na Tabela 4.5Tabela 4.6. Analisando o makespan dos testes, observa-se que o percentual de erro calculado entre o

makespan real e o *makespan* da simulação foi inferior a 1% com o erro percentual médio de aproximadamente 0,41%.

Tabela 4.5: Comparação entre o Makespan Real e o Makespan da Simulação.

Número de nós	*Makespan* (Real) em Segundos	*Makespan* (Sim) em Segundos	Erro (%)
01	19.088	19.248	0,83
04	5.118	5.127	0,17
08	2.850	2.871	0,73
12	2.088	2.095	0,33
16	1.744	1.739	0,28
20	1.535	1.528	0,45
24	1.374	1.373	0,07

Para validar o modelo de calibração ora executado, modelamos no MRSG um cenário com 28 e 32 utilizando o custo médio de *map* e *reduce* ora calculado. O *makespan* obtido da simulação foi comparado com o *makespan* extraído da execução real de 28 e 32 nós executada no cluster DC-UFSCar e calculado o percentual de erro que pode ser observado na Tabela 4.6.

Tabela 4.6: Comparação entre o Makespan Real e o Makespan da Simulação.

Número de nós	*Makespan* (Real) em Segundos	*Makespan* (Sim) em Segundos	Erro (%)
28	1.218	1.207	0,91
32	1.192	1.188	0,33

Observa-se na Tabela 4.6 que, com o ajuste dos custos das tarefas *map* e tarefas *reduce*, ambas as configurações simuladas apresentaram o erro inferior a 1% em relação aos testes reais.

O gráfico da Figura 4.3 ilustra os dados representados na Tabela 4.5 e Tabela 4.6 comparando o *makespan* real com o obtido na simulação para plataformas de 1 a 32 nós. Pode-se observar que o comportamento de ambos os experimentos se assemelham e que, como o erro foi relativamente baixo, utilizaremos o valor médio como custo para na modelagem de plataformas de maior porte.

Devido a limitações físicas de recursos, foi possível executar os testes na plataforma real com apenas 32 nós. Com tal margem de erro relativamente baixa, pode-se considerar que o simulador apresenta boa acurácia, podendo representar com certo grau de confiabilidade o ambiente real desta aplicação específica em uma plataforma pequena

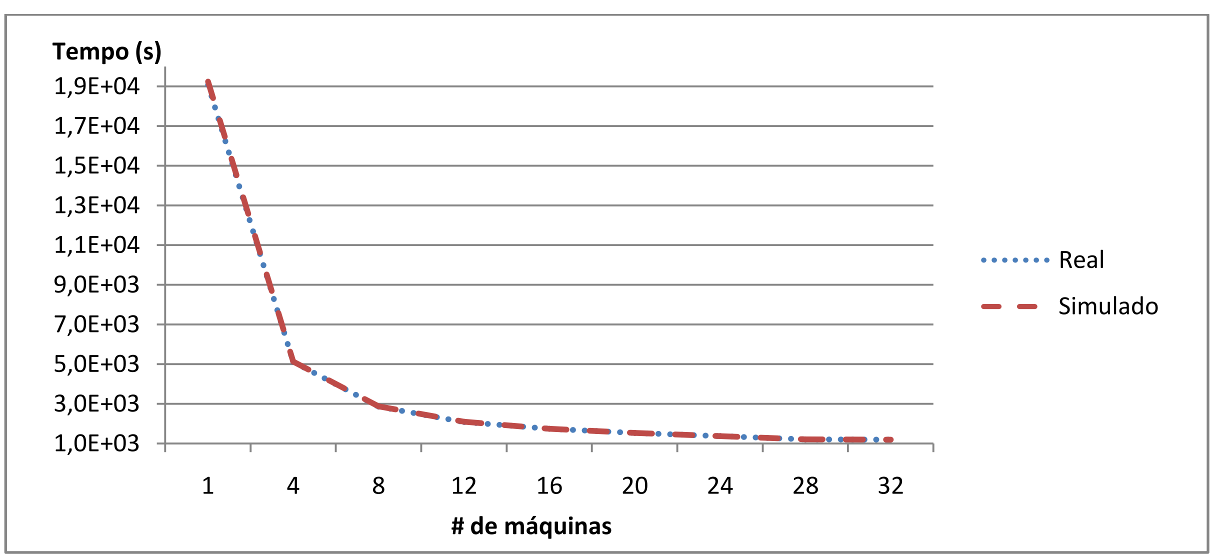

Figura 4.3: Comparação entre o Makespan Real e o Makespan Simulado.

Convém ressaltar que, para todos os experimentos, na simulação da plataforma sequencial (um nó) utilizou-se o valor de custo obtido nos experimentos de execução real.

O processo de calibração realizado pode ser resumido em:

1. Experimentação real no cluster de 01 a 32 nós;
2. Coleta do tempo de execução da fase *map*, *reduce* e *makespan*;
3. Modelagem da aplicação no simulador de 01 a 24 nós;
4. Experimentação e ajuste empírico do custo de *map* e *reduce*;
5. Coleta do tempo de execução da fase *map*, *reduce* e *makespan*;
6. Análise de erro;
7. Calculo da média dos custos de *map* e *reduce*;
8. *Reexecução* da simulação com o valor médio de custo;
9. Coleta do tempo de execução da fase *map*, *reduce* e *makespan*;
10. Análise de erro;
11. Execução da simulação com 28 e 32 nós;
12. Coleta do tempo de execução da fase *map*, *reduce* e *makespan*
13. Anállise de erro e validação do processo de calibração

4.5 Testes de Escalabilidade por meio de Simulação

Com o simulador calibrado, foi possível realizar diversos experimentos com plataformas compostas por milhares de nós. Tais experimentos tiveram por objetivo avaliar a escalabilidade da aplicação escolhida.

Na realização dos testes de escalabilidade, modelamos plataformas com 1, 2, 5, 10, 20, 50, 100, 200, 500, 1.000, 2.000, 5.000 e 10.000 nós. No primeiro cenário, considerou-se a seguinte configuração de aplicação:

- *Map output* gerado por cada tarefa *map*: 13343376 bytes;
- *Map* cost: 1.22867E+12 (número de operações a serem executadas, por *byte* de entrada da função *map*);
- *Reduce Cost*: 2.68667E+12 (número de operações a serem executadas, por *byte* de entrada da função *reduce*);
- Número de tarefas *reduce*: 26;
- Chunk Size: 178 MB;
- *Input Chunks* (tarefas *map*): 100.000;
- Dfs réplicas: 3;
- *Map Slots*: 2;
- *Reduce Slots*: 2.

Para cada experimento, foram extraídos os tempos de duração da fase *map*, da fase *reduce* e o *makespan* (Tabela 4.7).

Tabela 4.7: Resultado Simulado das Fases map, *reduce* e do *makespan*.

Número de nós	Fase *Map*	Fase *Reduce*	*Makespan*
01	8.707.546	8.158.951	9.029.708
02	4.879.228	4.554.412	5.042.337
05	1.951.695	1.819.932	2.015.104
10	975.849	905.803	1.003.390
20	487.926	439.568	488.363
50	195.173	176.107	195.626
100	97.589	88.309	98.070
200	48.797	44.448	49.330
500	19.519	28.829	30.785
1.000	10.015	28.642	29.652
2.000	5.029	28.551	29.155
5.000	2.087	28.545	28.816
10.000	1.181	28.556	28.822

O gráfico da Figura 4.4 ilustra o *makespan* obtido na simulação para plataformas de 1 a 10.000 nós. Como se pode observar, a partir de 500 nós, o tempo de execução praticamente se mantém no mesmo patamar. Isso ocorre devido ao baixo grau de paralelismo da fase *reduce*, que possui apenas 26 tarefas.

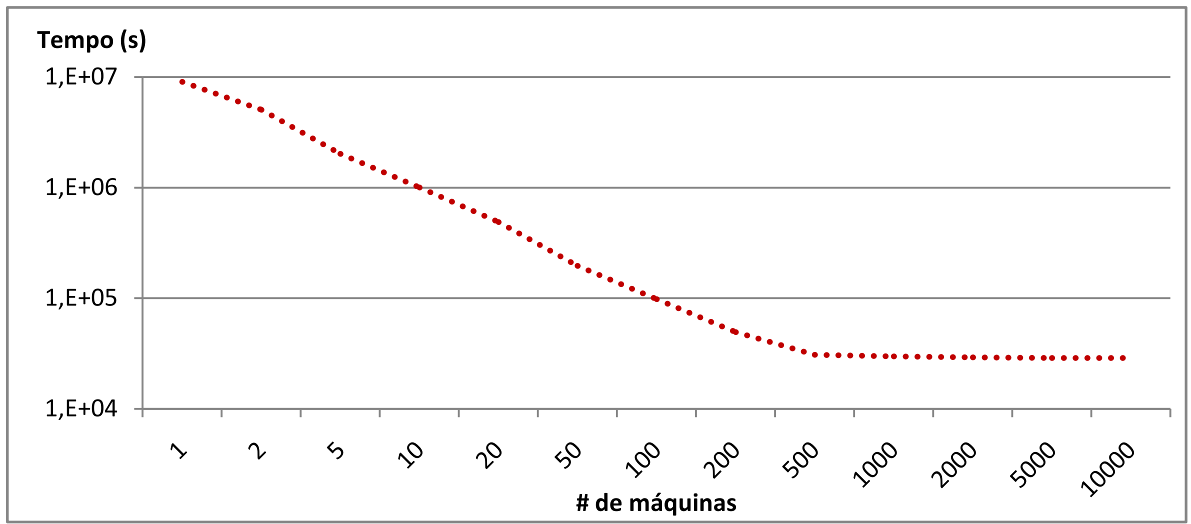

Figura 4.4: Gráfico de M*akespan* com 26 tarefas *reduce* e 100 mil tarefas map.

A Tabela 4.8 contém os valores de *makespan*, o *speedup* e a eficiência do experimento. A curva de *speedup* é ilustrada no gráfico da Figura 4.5 em escala logarítmica em base 10.

Tabela 4.8: *Makespan*, *Speedup* e Eficiência com 26 tarefas *reduce* e 100 mil tarefas *map*.

Número de nós	*Makespan (s)*	*Speedup* Total	Eficiência
01	9.029.708	1,000	1,000
02	5.042.337	1,791	0,895
05	2.015.104	4,481	0,896
10	1.003.390	8,999	0,900
20	488.363	18,490	0,924
50	195.626	46,158	0,923
100	98.070	92,074	0,921
200	49.330	183,047	0,915
500	30.785	293,315	0,587
1.000	29.652	304,523	0,305
2.000	29.155	309,714	0,155
5.000	28.816	313,357	0,063
10.000	28.822	313,292	0,031

Como se pôde confirmar, a aplicação não escala conforme o esperado a partir de 500 nós. A escalabilidade aumenta sublinearmente até 500 nós, quando, então se estabiliza.

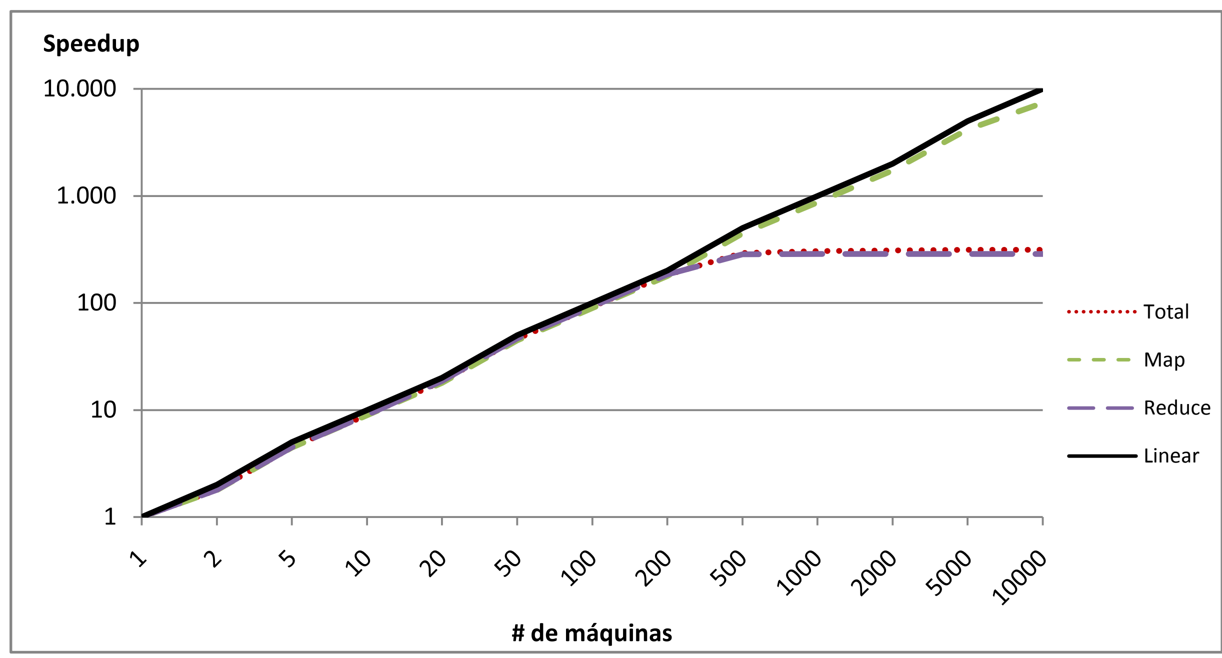

Figura 4.5: Gráfico de Speedup com 26 tarefas *reduce* e 100 mil tarefas *map*.

A eficiência da aplicação para a plataforma e as configurações ora simuladas mantiveram constantes até 200 nós e, após esse valor, observa-se uma queda na eficiência, conforme ilustra a Figura 4.6.

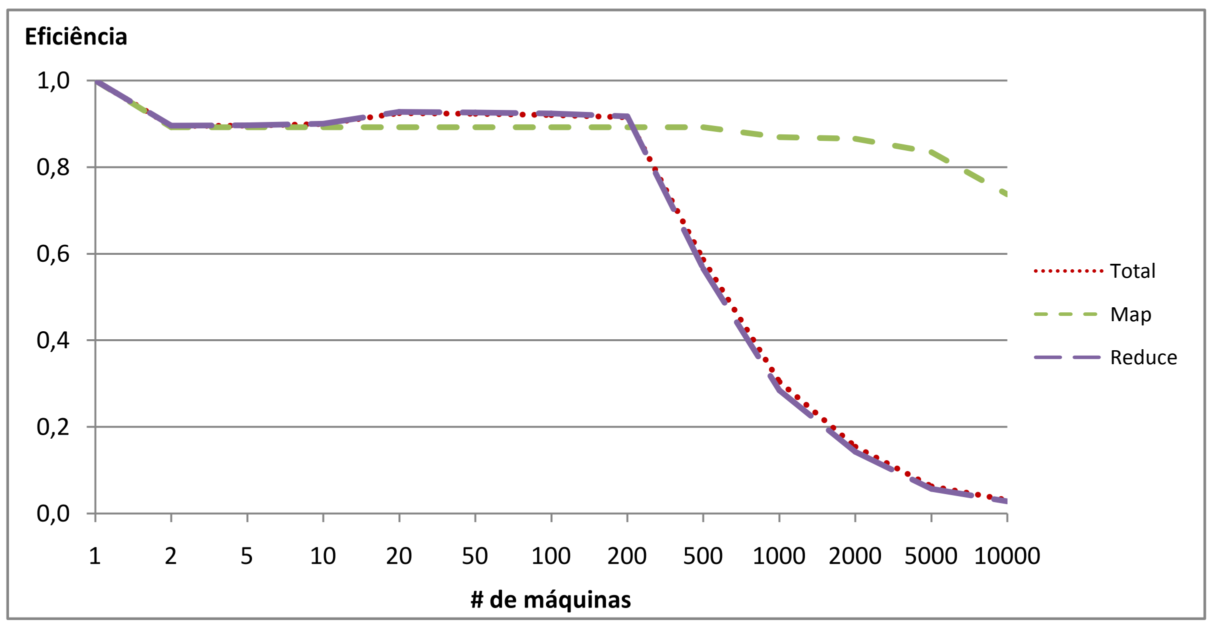

Figura 4.6: Gráfico de Eficiência com 26 tarefas *reduce* e 100 mil tarefas *map.*

Com esses experimentos, é possível observar que a fase *map* escala quase linearmente, desde que a quantidade de tarefas *map* seja suficientemente grande para manter os nós ocupados. Alguns fatores podem contribuir para que a fase *map* escale sublinearmente (ao invés de linearmente perfeito), tais como a geração de uma grande quantidade de tarefas não locais e o consequente aumento do tráfego de rede. Tais tarefas têm impacto direto na aplicação, pois exigem a transmissão dos *chunks* antes da execução.

A implementação da aplicação faz com que a fase *reduce* não escale além de 26 núcleos (13 nós) - devido as implementação de 26 tarefas reduce em que cada tarefa é responsável por uma letra do alfabeto, porém, a representação gráfica do *speedup* ilustrada na Figura 4.5 demonstra que a fase *reduce* escala até 500 nós. Isso ocorre devido a uma otimização realizada no Hadoop, que antecipa o início da fase *reduce*, logo que as primeiras tarefas *map* são finalizadas.

Na análise dos experimentos, observa-se que a fase reduce se mostrou um dos principais limitantes da escalabilidade da aplicação (*makespan*). Isso ocorre, pois a fase *reduce* atrasa a duração do *job*.

Outro fator que pode ter influenciado o comportamento observado da fase *reduce* é o gargalo ocasionado devido ao aumento dos tempos de *heartbeat.* O simulador MRSG considera, por padrão, o intervalo de *heartbeat* como 3 segundos. Caso esse valor fosse fixo, seria possível a ocorrência de gargalo no nó mestre e/ou congestionamento da rede, fato que pode impossibilitar o controle adequado da execução da aplicação.

Com objetivo de tentar minimizar o tráfego de rede para plataformas de grande porte, o MRSG otimiza o intervalo de *heartbeat* pela divisão do número de nós por 100. Caso esse valor seja inferior a 3, usa-se o valor padrão, caso contrário, otimiza-se esse intervalo. Nos experimentos realizados, o *heartbeat* foi de 3 segundos para plataformas de 1 a 200 nós, de 5 segundos para 500 nós, de 20 segundos para 2.000 nós, de 50 segundos para 5.000 nós e de 100 segundos para 10.000 nós.

Se considerarmos, por exemplo, que uma tarefa *map* tenha duração aproximada de 170 segundos, em uma plataforma de 5.000 nós com intervalo de *heartbeat* de 50 segundos, haverá uma ociosidade de 30 segundos até que ocorra o envio de uma nova tarefa a ser executada. Observa-se que a nó JobTracker

somente terá ciência que a tarefa terminou no instante de 200 segundos (*heartbeat* a cada 50 segundos) e consequentemente o nó já estaria ocioso.

4.5.1 Aumentando o paralelismo da Fase *Reduce* com 676 tarefas

Uma possível estratégia para aumentar a escalabilidade da aplicação é modificar o seu código a fim de produzir um número maior de tarefas *reduce*, pois, assim seria possível aumentar o seu grau de paralelismo. Por exemplo, os dois primeiros caracteres de cada termo poderiam ser verificados ao invés de apenas um (que resulta em apenas 26 tarefas *reduce*, uma para cada letra do alfabeto). Nesse caso, seria possível ter 26^2 tarefas *reduce,* de modo a utilizar mais processadores durante o processamento da fase.

No próximo cenário, são considerados os mesmos parâmetros de plataforma e de configuração utilizadas na seção 4.5, exceto o número de tarefas *reduce*, que foi fixado em 676 tarefas. Com o aumento do número de tarefas *reduce*, observa-se que o *speedup* da fase escala sublinearmente até 500 nós, conforme descrito na Tabela 4.9.

Tabela 4.9: *Makespan*, *Speedup* e Eficiência com 676 tarefas *reduce* e 100 mil tarefas map.

Número de nós	*Makespan (s)*	*Speedup* Total	Eficiência
01	17.750.840	1,000	1,000
02	9.434.726	1,881	0,941
05	4.346.400	4,084	0,817
10	2.738.452	6,482	0,648
20	1.562.024	11,364	0,568
50	705.375	25,165	0,503
100	342.457	51,834	0,518
200	138.850	127,842	0,639
500	58.271	304,626	0,609
1.000	38.480	461,300	0,461
2.000	32.287	549,783	0,275
5.000	28.530	622,182	0,124
10.000	28.633	619,943	0,062

Essa alteração na configuração possibilitou alguma melhora no desempenho, porém, isso simplesmente deslocou a curva para cima, elevando para uma quantidade entre 500 e 1000 nós, conforme ilustra a Figura 4.7.

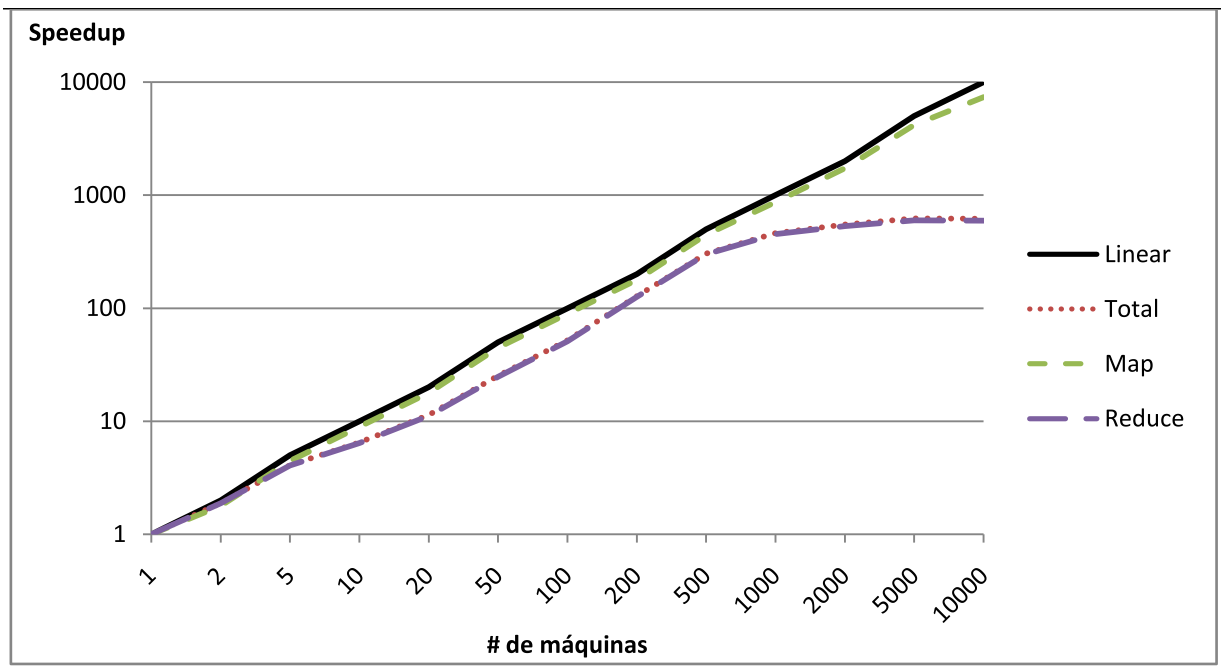

Figura 4.7: Gráfico de *Speedup* com 676 tarefas *reduce* e 100 mil tarefas *map*.

4.5.2 Teste com o Uso de *Switch Infiniband* e 676 tarefas *reduce*

Como alternativa para melhoria de *speedup,* buscamos investigar o comportamento da aplicação considerando o aumento da largura de banda da rede que interconecta os nós de modo a atender à demanda do tráfego de dados gerado principalmente na fase intermediária. O ambiente experimental utilizado se baseia em *gigabit ethernet*, contudo em clusters de melhor qualidade faz-se uso de outras tecnologias, tais como *infiniband* por exemplo.

Assim, criamos um cenário semelhante ao nosso, porém com uma rede rápida *infiniband* EDR (E*enhanced Data Rate*) com taxa de largura de banda de 300Gb/s (37,5 GBps) e latência de 100 *nanosegundos*. Considerando o experimento com 10 mil nós e 676 tarefas *reduce,* o *speedup* (vide Tabela 4.10) obtido foi de 5.739, conforme ilustra a Figura 4.8.

Observa-se que com o aumento do grau de paralelismo da fase *reduce* houve um significativo ganho na escalabilidade da aplicação. A representação gráfica do speedup ilustra o comportamento sublinear da fase *map* e *reduce*. Na plataforma de 10 mil, o *makespan* resultante foi 18 vezes menor que o observado no experimento da seção 4.5.1.

Tabela 4.10: *Makespan*, Speedup e Eficiência com 676 tarefas *reduce* e switch infiniband

Número de nós	*Makespan (s)*	*Speedup* Total	Eficiência
01	8.774.972	1,000	1,000
02	4.971.199	1,77	0,883
05	2.002.754	4,38	0,876
10	1.002.435	8,75	0,875
20	503.331	17,43	0,872
50	201.995	43,44	0,869
100	101.016	86,87	0,869
200	50.629	173,32	0,867
500	19.933	440,22	0,880
1.000	10.433	841,08	0,841
2.000	5.443	1.612,16	0,806
5.000	2.475	3.545,443	0,709
10.000	1.529	5.739,027	0,574

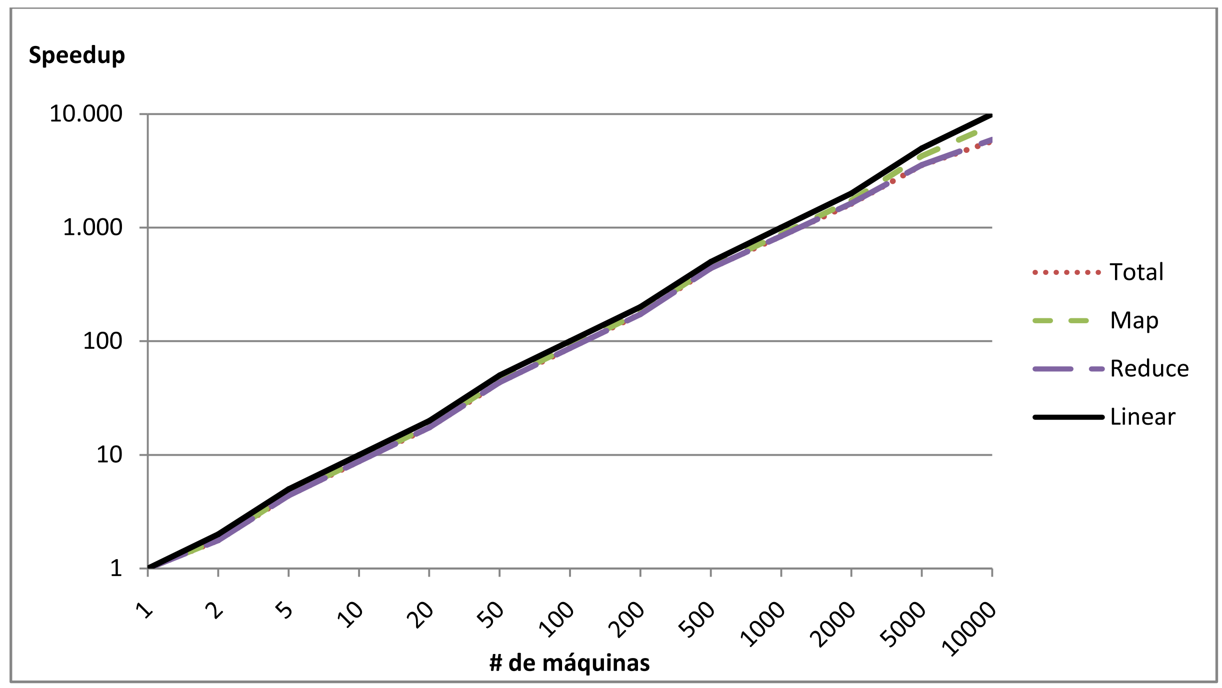

Figura 4.8: *Makespan, Speedup* e Eficiência com 676 tarefas *reduce* e *switch infiniband*

Apesar do ganho de speedup observado, vale destacar que na sua versão atual o MRSG possui limitações com relação à simulação do tráfego de rede durante fase intermediária. Na simulação, o MRSG acresce tempo durante as fases de *map* e de *reduce* como forma de compensar a limitação. Por conta disso, pode haver imprecisões nos resultados obtidos não refletindo de maneira precisa o comportamento para *Ethernet* e *infiniband*.

De todo modo, isto não invalida o método proposto, que é uma contribuição para usuários que queiram avaliar a escalabilidade de suas aplicações. Além disso, melhorias poderão ser efetuadas no MRSG no futuro, melhorando a sua acurácia e corrigindo possíveis imprecisões caso elas realmente existam.

4.6 Considerações do Capítulo

Neste trabalho, executou-se uma aplicação típica de indexação distribuída, denominada Terrier, na plataforma Hadoop, a qual teve por base de entrada a coleção ClueWeb.

A plataforma de teste foi composta por 1, 4, 8, 12, 16, 20, 24, 28 e 32 nós e os tempos de execução real foram utilizados para calibrar o simulador MRSG. A calibração foi uma etapa de grande importância neste trabalho e se fez necessária para que o simulador representasse de forma relativamente precisa o comportamento da aplicação utilizada.

Uma vez calibrado, o simulador permitiu avaliar a escalabilidade dessa aplicação em plataformas de ate 10 mil nós. Com os experimentos, pôde-se observar que a fase *map* escala sublinearmente e que a fase *reduce* se mostra o principal gargalo. Entretanto, com algum investimento é possível melhorar e aumentar o paralelismo durante essa fase.

Duas alternativas foram investigadas para a melhoria do *speedup* da aplicação. Uma delas é o aumento do grau de paralelismo da fase *reduce e a o*utra é o uso de *switches infiniband*, que poderia proporcionar uma significativa redução do gargalo de tráfego de rede e consequente melhoria do *speedup.*

Capítulo 5

CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Mapreduce é um modelo de programação que tem por objetivo o desenvolvimento de aplicações que manipulam grandes volumes de dados executando em plataformas compostas por milhares de processadores. No modelo, a execução das tarefas é feita segundo uma arquitetura mestre/escravo, em que o nó mestre é responsável por distribuir tarefas aos escravos e gerenciar sua execução.

Hadoop é, atualmente, a implementação do modelo Mapreduce mais popular, sendo amplamente utilizada por diversas companhias e no meio científico para o processamento de grandes bases de dados em plataformas compostas por milhares de nós. Hadoop dispõe de mecanismos de tolerância a falhas, balanceamento de cargas e distribuição de dados. Hadoop possui uma grande quantidade de parâmetros configuráveis, porém, aproximadamente 25 causam impacto significativo nas aplicações conforme relatado por BABU (2010).

Para o estudo de escalabilidade de aplicações Hadoop/Mapreduce se faz necessária a realização de experimentos em larga escala, com elevados números de tarefas e a variação de parâmetros da aplicação e da plataforma. Por meio do projeto de experimentos 2^k fatorial foi possível identificar os parâmetros ou conjunto de parâmetros que causam impacto no desempenho das aplicações.

Um dos grandes desafios na realização do estudo de escalabilidade é a indisponibilidade de uma infraestrutura física de grande porte que permita a análise de diferentes cenários. Para contornar tal limitação, neste trabalho, optou-se por realizar a combinação de experimentação real e simulada.

A experimentação real foi realizada no cluster local de 32 nós do DC-UFSCar. A aplicação Mapreduce escolhida para análise trata do processamento de índices invertidos e recuperação de informação conhecida por Terrier, sendo que esta foi instalada sobre a plataforma Hadoop. Esta aplicação é bem conhecida, representativa, otimizada e já foi bastante discutida na literatura, estando acessível em um sistema disponibilizado como software livre.

Com a execução da experimentação real foi possível analisar o seu comportamento e extrair as informações necessárias para a etapa de simulação. A simulação vem a contornar a limitação de estrutura física, possibilitando modelar diferentes aplicações e plataformas com até milhares de nós.

Para obter uma boa correspondência entre as avaliações reais e simuladas, foi necessário calibrar o simulador MRSG. A calibração foi realizada considerando a plataforma de 1 a 32 nós e, para cada configuração, fornecemos os custos das tarefas *map* e *reduce,* que permitiram reproduzir o *makespan* da aplicação com erro inferior a 1%.

Nos testes de escalabilidade analisamos o comportamento simulado da aplicação para plataformas de até 10 mil nós. Os testes mostram que a fase *map* escala sublinearmente. Contudo, observamos que um dos principais gargalos da aplicação é a fase *reduce.* Isto se deve ao baixo paralelismo do *reduce* (apenas 26 tarefas), bem como as otimizações realizadas no Hadoop, pois na versão atual a fase *reduce* inicia logo após o término das primeiras tarefas *map*.

Uma vez identificados tais gargalos, foram realizados novos experimentos, como a experimentação com 676 tarefas *reduce,* resultando no aumento do grau de paralelismo da fase *reduce*. Este resultado foi satisfatório, pois mostrou que é possível obter maior escalabilidade. Outro provável problema observado é a possível ociosidade causada devido ao aumento do intervalo do *heartbeat* para plataformas de grande porte. Neste cenário, para tentar evitar sobrecarga na rede ocorre o aumento do intervalo entre os *heartbeats*, contudo tal aumento pode acarretar ociosidade no nó entre a finalização de uma tarefa e o recebimento de uma nova.

Ressalta-se que, por se tratar de um conjunto de experimentos realizados via simulação faz-se necessário observar que não há garantias de que o comportamento real com centenas ou milhares de nós seja de fato semelhante ao que foi simulado, sendo estes limitados ao grau de acurácia do MRSG. Entretanto, o método adotado permite criar uma maior expectativa de que os resultados de simulação retratem de maneira mais próxima possível o comportamento esperado para casos reais.

Outro aspecto que deve ser destacado é a limitação dos resultados obtidos, que são válidos somente para a aplicação. Todavia, convém destacar que a aplicação utilizada é bastante representativa de uma classe de aplicações de indexação Mapreduce, de grande relevância e já foi bastante estudada e otimizada, o que proporciona uma expectativa de que aplicações otimizadas podem alcançar os mesmos níveis de escalabilidade.

Deste modo, nesse trabalho, concluímos que o modelo Mapreduce é escalável neste modelo de aplicação e os limites de escalabilidade estão diretamente relacionados a configuração dos parâmetros da aplicação e da plataforma. A análise prévia de tais configurações possibilitou identificar os parâmetros de maior impacto e os gargalos da aplicação.

Assim, pela técnica do projeto de experimento 2^k fatorial, é possível verificar os fatores de maior impacto na escalabilidade. Uma vez identificado tais parâmetros, por meio da combinação de experimentação real e simulação, é possível realizar os ajustes necessários minimizando os gargalos e assim obter bons níveis de escalabilidade para plataformas de grande porte. Ressalta-se que, com a adoção da metodologia utilizada neste estudo desenvolvedores de aplicações Mapreduce/Hadoop podem ter noção prévia do comportamento da aplicação e identificar os fatores que necessitam de ajustes para obtenção de bons limites de escalabilidade.

Como possíveis trabalhos futuros, pode-se verificar a aplicabilidade e acurácia do método utilizado para a calibração com outras aplicações Mapreduce. Alguns parâmetros de configuração (tais como tamanho dos *chunks*, número de réplicas, número de tarefas *reduce*, entre outros) podem ser otimizados com o objetivo de obter a máxima escalabilidade.

Essa otimização pode ser realizada com auxílio de métodos conhecidos como *Derivative Free Optimization* (DFO) (RIOS; SAHINIDI, 2013). A otimização pode

indicar aos desenvolvedores quais os parâmetros que devem ser otimizados. Uma vez calibrado e otimizado, o ambiente de simulação pode ser utilizado para gerar experimentos com quantidades significativas de nós para a extração de curvas de escalabilidade do melhor caso.

Por fim, podem-se propor melhorias no MRSG sobre como simular o tráfego em rede na fase intermediária.

REFERÊNCIAS

ANJOS, J. C. S. **Adequação da Computação Intensiva em Dados para Ambientes Desktop Grid com uso de MapReduce**. Instituto de Informática, UFRGS, 2012.

APACHE. **The Apache Software Foundation**. Hadoop website. Disponível em: <http://hadoop.apache.org>. Acesso em: janeiro/2011.

BABU, S. **Towards Automatic Optimization of MapReduce Programs.** Duke University. Durham, North Carolina, USA, 2010.

BARROSO, A.; HOELZLE, U. **The Datacenter as a Computer:** An Introduction to the Design of Warehouse-Scale Machines, 2009.

BORTHAKUR, D. **Facebook has the world's largest Hadoop *Cluster*!**. Disponível em: <http://hadoopblog.blogspot.com/2010/05/facebook-has-worldslargest-hadoop.html>. Acesso: janeiro/2011.

CASANOVA, H. **Simgrid**: A Toolkit for the Simulation of Application Scheduling. 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, USA. 2001.

CASANOVA, H.; LEGRAND, A.; QUINSON, M. **SimGrid**: a Generic Framework for Large-Scale Distributed Experiments". 10th IEEE International Conference on Computer Modeling and Simulation, 2008.

DEAN, J.; GHEMAWAT, S. **Mapreduce**: Simplifed Data Processing on Large *Clusters*. Communications of The ACM, Vol. 51, No. 1, 2008.

DEAN, J.; GHEMAWAT, S. **Mapreduce**: Simplifed Data Processing on Large *Clusters*. 6th Symposium on Operating Systems Design and Implementation, Berkeley, 2004.

GRID5000. **Grid´5000**. Disponível em: <http://www.grid5000.fr >. Acesso: fevereiro/2013.

HADOOP. **Powered by Hadoop**. Disponível em: <http://wiki.apache.org/hadoop/PoweredBy>. Acesso: janeiro/2011.

HADOOP. **The Hadoop Distributed File System**: Architecture and Design". 2008. Disponível em: <http://hadoop.apache.org/common/docs/r0.17.0/hdfs_design.html> A Janeiro/2011.

HAMMOUD, S.; LI, M.; LIU, Y.; ALHAM, N. K.; LIU, Z. **MRSim**: A discrete event based MapReduce simulator, 2010.

JAIN, R. **Art of Computer Systems Performance Analysis Techniques for Experimental Design Measurements Simulation and Modeling**, 1991.

KOLBERG, W.; ANJOS, J. C. S. **MRSG**: a mapreduce simulator for desktop grids. Instituto de Informatica, UFRGS, 2011.

MCCREADIE, R.; MACDONALD, C.; OUNIS, I. **MapReduce indexing strategies**: Studying scalability and efficiency. Information Processing and Management, Elsevier, 2011.

O'MALLEY, O. **Next Generation of Apache Hadoop MapReduce**. Disponível em <http://developer.yahoo.com/blogs/hadoop/posts/2011/02/mapreduce-nextgen/ Acesso: abril/2011.

RIOS, L. M.; SAHINIDI, N. V. **Derivative-free optimization: A review of algorithms and comparison of software implementations**, Journal of Global Optimization, Volume 56, Issue 3 , pp 1247-1293.

SIMGRID. **The SimGrid Project**. SimGrid website. Disponível em: <http://simgrid.gforge.inria.fr/>. Acesso: Janeiro/2011.

TANG, H. **Mumak**: Map-Reduce Simulator., 2009. Disponível em:

<https://issues.apache.org/jira/browse/MAPREDUCE-728>. Acesso em julho/2012.

TERRIER. **Terrier IR Platform**. Terrier website. Disponível em:

http://www.terrier.org/ Acesso: Outubro/2012.

VANCE, A. **Hadoop**: a Free Software Program, Finds Uses Beyond Search. The New York Times. Março, 2009. Disponível em: <http://www.nytimes.com/2009/03/17/technology/business-computing/17cloud.html# Acesso: fevereiro/2011.

WANG, G. BUTT, A. R.; PANDEY, P.; GUPTA, K. **Using realistic simulation for performance analysis of mapreduce setups**., New York, 2009.

WHITE, T. **Hadoop:** the Definitive Guide. O'Reily, 2009.