UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

EXPLORAÇÃO E BENCHMARKING DE UMA IMPLEMENTAÇÃO DE MAP REDUCE: O CASO DO HADOOP NA PLATAFORMA GRID'5000

TRABALHO DE GRADUAÇÃO

Vinicius Vielmo Cogo

Santa Maria, RS, Brasil

2010

EXPLORAÇÃO E BENCHMARKING DE UMA IMPLEMENTAÇÃO DE MAP REDUCE: O CASO DO HADOOP NA PLATAFORMA GRID'5000

por

Vinicius Vielmo Cogo

Trabalho de Graduação apresentado ao Curso de Ciência da Computação
da Universidade Federal de Santa Maria (UFSM, RS), como requisito
parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Profª. Andrea Schwertner Charão

Co-orientador: Prof. Marcelo Pasin

Trabalho de Graduação N. 297

Santa Maria, RS, Brasil

2010

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

EXPLORAÇÃO E BENCHMARKING DE UMA IMPLEMENTAÇÃO DE MAP REDUCE: O CASO DO HADOOP NA PLATAFORMA GRID'5000

elaborado por
Vinicius Vielmo Cogo

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Profª. Andrea Schwertner Charão
(Presidente/Orientador)

Prof. Benhur de Oliveira Stein

Profª. Patrícia Pitthan de Araújo Barcelos

Santa Maria, 18 de Janeiro de 2010.

AGRADECIMENTOS

Inicialmente, agradeço a todas as entidades divinas, espirituais e físicas, instanciadas neste texto ou não, que me acompanham desde sempre. Estes que, em conjunto, também me apoiaram durante os quatro anos de graduação, até então os mais intensos da minha vida.

É impossível fazer um agradecimento sem pensar na família e seus "agregados". Gostaria de agradecer em especial aos meus pais, Sandra e Marco e seus companheiros, Luis e Juliana, por sempre me mostrarem a realidade de cada decisão a ser tomada, bem como me apoiarem na grande maioria delas, assim como aos meus irmãos Vitor, Vitória e Mateus. Gostaria de agradecer aos meus avós, Abel, Irani, Luis e Zélia, pela educação dada aos meus pais e, consequentemente a mim, assim como agradecer aos meus padrinhos (Denise e Paulo), tios e primos.

Gostaria de agradecer à Leticia, minha grande companheira, por estar sempre ao meu lado em todos tipos de situações, sejam elas difíceis, fáceis, alegres ou tristes, bem como por ter me aguentado durante todos os finais de semestre do curso. Espero ainda termos muitos momentos juntos. Eles não precisam ser todos especiais, basta estarmos juntos e felizes. Também gostaria de agradecer a Ana Helena por todo apoio que nos deu desde o início, bem como pela sua serenidade em todos os momentos.

Quero prestar meu agradecimento a todos os professores, que muito mais do que ensinarem conteúdos, ensinam sobre a vida e como aprender com ela. Em especial à Andrea Charão pela orientação pessoal e profissional durante o tempo de PET, LSC e TG. Seus ensinamentos foram fundamentais para chegar ao fim desta etapa e serão utilizados por mim em todas as que virão. Também um especial agradecimento ao Marcelo Pasin, pelo apoio e acolhimento demonstrados durante a reta final do curso e o desejo de que essa parceria continue firme por muito tempo.

Agradeço aos meus amigos, desde o Ensino Médio (Harnye, Etiane, Hugo e Patrick), assim como os amigos e colegas de graduação, em especial ao Vitor Conrado, Gustavo Rissetti, Bruno Appel e Fernando Rivas. Agradeço também à todos "tugas"e "zukas"que conheci durante esta reta final do curso, por toda sua amizade e companheirismo que apenas está "a começar". Agradeço também a todos responsáveis e colegas do PET, LaSIGE, LSC, Decadium e InterativaSul pelas oportunidades oferecidas de crescimento profissional e pessoal, bem como por todos os ensinamentos que me foram dados.

A todos vocês, fica aqui o meu mais sincero obrigado, e a certeza de que podem contar comigo no que for preciso.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

EXPLORAÇÃO E BENCHMARKING DE UMA IMPLEMENTAÇÃO DE MAP REDUCE: O CASO DO HADOOP NA PLATAFORMA GRID'5000

Autor: Vinicius Vielmo Cogo
Orientador: Profª. Andrea Schwertner Charão
Co-orientador: Prof. Marcelo Pasin
Local e data da defesa: Santa Maria, 18 de Janeiro de 2010.

A partir da demanda pela análise de grandes quantidades de informações, em espaços de tempos viáveis, o modelo de programação MapReduce ganhou espaço científico e comercial, devido ao aproveitamento do paralelismo para dividir a carga de dados e obter ganho de desempenho. Uma das principais implementações deste modelo de programação é a Hadoop MapReduce, que foi selecionada como objeto de estudo deste trabalho. Neste trabalho, tem-se como objetivos gerais a criação de um ambiente de testes para Hadoop na plataforma Grid'5000, além da exploração e seleção de ferramentas de *benchmarking* para este *framework*. São apresentados também neste trabalho, uma revisão teórica sobre os conceitos já citados, o processo de desenvolvimento, bem como a apresentação e avaliação dos resultados obtidos.

Palavras-chave: MapReduce, Hadoop, Benchmarking, Grid'5000, Ambiente de Testes.

ABSTRACT

Trabalho de Graduação
Undergraduate Program in Computer Science
Universidade Federal de Santa Maria

EXPLORATING AND BENCHMARKING A MAPREDUCE IMPLEMENTATION: THE CASE OF HADOOP ON GRID'5000
Author: Vinicius Vielmo Cogo
Advisor: Prof[a]. Andrea Schwertner Charão
Coadvisor: Prof. Marcelo Pasin

Due to the demand for analysis of large amounts of information, in feasible time, the MapReduce programming model has obtained scientific and commercial visibility, as a result of the use of parallelism to share the data load to obtain performance gains. One of the main features of this programming model is Hadoop MapReduce, which was selected as the object of this study. This work has as general objectives the creation of a testing environment for the Hadoop on Grid'5000 platform, as well as the exploration and selection of benchmarking tools for this framework. This work also presents a theoretical review of the concepts already mentioned, the development process and the presentation and evaluation of results obtained in this work.

Keywords: MapReduce, Hadoop, Benchmarking, Grid'5000, Test Environment.

ACKNOWLEDGMENT

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

HDFS	Hadoop Distributed File System
JVM	Java Virtual Machine
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
MTBF	Mean Time Between Failures
NFS	Network File System
RPC	Remote Procedure Call
SSH	Secure Shell

SUMÁRIO

1 INTRODUÇÃO

Em 1965, Gordon E. Moore observou que o número de transistores em circuitos integrados aproximadamente dobraria a cada dois anos (MOORE, 1965) (Intel Corporation, 2005). Sua previsão passou a ser chamada de "Lei de Moore", esta que se encontra diretamente ligada a diversas medidas tecnológicas, como por exemplo a velocidade dos processadores e a capacidade de memória. Mais do que uma Lei natural, tornou-se uma meta mercadológica a ser alcançada por empresas que produzem processadores e outros componentes formados por transistores.

Com tais avanços na velocidade de processamento de informações, seria natural considerar nas mesmas proporções, o aumento da velocidade de leitura e escrita de tais dados. Porém não foi o que aconteceu, atualmente, os sistemas computacionais possuem grande capacidade de processamento sobre pequenas quantidades de informações. A análise de grandes quantidades de informações, em espaços de tempos viáveis, tornou-se uma das novas demandas na área de Tecnologia de Informação.

Demanda essa que proporcionou o surgimento da tendência pela distribuição dos dados em sistemas de larga escala, a fim de obter ganho de desempenho nas tarefas de análises. Surge então, neste contexto, o modelo de programação MapReduce, proposto pela empresa Google em 2004 (DEAN; GHEMAWAT, 2004), o qual consiste no aproveitamento do paralelismo para dividir a carga de dados, ao invés de paralelizar as etapas de processamento.

Após esta proposta inicial, surgiram diversas iniciativas de implementações deste modelo de programação em diferentes linguagens de programação. A implementação abordada neste trabalho é o Hadoop MapReduce (Apache Software Foundation, 2007a), e foi desenvolvida na linguagem de programação Java.

A escolha pelo Hadoop MapReduce deve-se ao fato deste trabalho estar inserido no

contexto do projeto FTH-Grid (*Fault-Tolerant Hierarchical Grid Scheduling*) (LIP6 and LaSIGE, 2009), que é uma cooperação entre o LIP6/França (*Laboratoire d'Informatique de Paris 6*) e o LaSIGE/Portugal (Large-Scale Informatics Systems Laboratory) e tem como principal objetivo, o desenvolvimento de arquiteturas e mecanismos para *grids* computacionais tolerantes a falhas e a intrusões.

Os objetivos gerais deste trabalho são: criar um ambiente de testes para Hadoop na plataforma Grid'5000 e explorar soluções de *benchmarking* para Hadoop. Um objetivo específico deste último é selecionar uma ou mais ferramentas de *benchmarking* que sejam capazes de comparar duas versões do Hadoop. Este objetivo está relacionado à previsão da criação de uma versão modificada do Hadoop, com mecanismos de replicação de tarefas, visando a tolerância a falhas bizantinas no *framework*.

As principais contribuições científicas deste trabalho são:

- Proposta e concretização de um ambiente público de testes para Hadoop na plataforma Grid'5000, onde outros projetos que utilizem este *framework* nesta plataforma, possam utilizar o ambiente de testes.

- Exploração e documentação de soluções de *benchmarking* para Hadoop, através de uma revisão de trabalhos e ferramentas existentes. Sendo este o primeiro trabalho a agrupar, em uma única análise, soluções de *benchmarking* baseadas em *workload*, em componentes específicos e em análise de logs.

- Seleção e apresentação dos *benchmarks* como solução para as necessidades do projeto FTH-Grid.

Tendo em vista os objetivos e atividades do presente trabalho, este documento possui seis capítulos e uma seção de apêndice, dos quais esta primeira introdução consiste o capítulo 1. No capítulo 2, é apresentada uma revisão literária e teórica, destinada a fundamentação e embasamento para os capítulos consequentes. Por sua vez, o capítulo 3 é aquele que possui a descrição da primeira das atividades gerais deste trabalho, o ambiente de testes, bem como seus requisitos, a preparação realizada e a sua concretização na plataforma Grid'5000. Referente a outra atividade, no capítulo 4, consta a apresentação dos requisitos aos *benchmarks*, as soluções existentes encontradas e as ferramentas selecionadas para o projeto FTH-Grid. Dando sequência, no capítulo 5, são apresentados os

resultados e avaliações das atividades anteriormente citadas, assim como no capítulo 6, encontram-se as conclusões gerais referentes a este trabalho.

2 FUNDAMENTAÇÃO

Neste capítulo é apresentada uma compilação crítica e retrospectiva de publicações sobre os temas a serem vistos neste documento, a fim de ilustrar seus estados atuais, bem como estabelecer um referencial teórico para dar suporte ao desenvolvimento deste trabalho. Os temas que serão abordados neste capítulo são: MapReduce, Hadoop, *benchmarking* e a plataforma computacional Grid'5000.

2.1 MapReduce

MapReduce é um modelo de programação, apresentado pela empresa Google (DEAN; GHEMAWAT, 2004), que tem como principais objetivos, proporcionar a geração, processamento e análise de grandes quantidades de dados, distribuídos em um sistema computacional de larga escala.

A metodologia para atingir tal objetivo consiste em aproveitar-se do paralelismo para dividir a carga de dados, ao invés de dividir as etapas de processamento. Em outras palavras, cada componente é responsável por processar completamente um pequeno grupo de dados, ao invés de processar todos os dados em uma determinada etapa da computação.

O formato das informações utilizadas no MapReduce são pares do tipo: *(chave, valor)*, onde a *chave* é o elemento identificador do par e o *valor* é o seu conteúdo. O processo de computação utiliza conjuntos de pares como este, tanto na entrada, quanto na saída. Neste modelo de programação o programador deve preocupar-se, basicamente, com a composição de duas funções:

- *map()*: A função *map* recebe como entrada um conjunto de pares, analisa cada um destes e gera um novo conjunto intermediário de pares. Os pares, deste conjunto de saída da função *map*, serão agrupados de acordo com suas chaves e serão enviados como conjunto de entrada para a função *reduce*.

- *reduce()*: A função *reduce* recebe como entrada o conjunto de pares intermediários gerado pela função *map*, normalmente organizados de acordo com suas chaves, realiza uma determinada computação sobre os valores destes pares e gera um novo conjunto de pares, que será considerado a saída da aplicação MapReduce.

Neste mesmo artigo (DEAN; GHEMAWAT, 2004), é apresentada uma concretização do Google para este modelo de programação, em forma de um *framework* homônimo, escrito na linguagem de programação C++. Este é executado internamente ao Google em um *cluster* de larga escala, formado por máquinas comerciais, onde uma típica computação de MapReduce é capaz de processar muitos TeraBytes de dados em milhares de máquinas.

A abstração utilizada para as funções *map* e *reduce*, neste modelo, são inspiradas nas primitivas *map* e *reduce* presentes em Lisp e em diversas outras linguagens do paradigma de programação funcional. Porém, as correspondências entre as funções em MapReduce e em Lisp não são triviais (LäMMEL, 2007). A função *map* do modelo de programação MapReduce pode ser considerada apenas um argumento da função *map* do Lisp e a função *reduce* do MapReduce pode ser uma aplicação do combinador *reduce* do Lisp, ou ainda também pode ser um argumento da primitiva *map* desta linguagem.

O fluxo completo de dados no modelo de programação MapReduce consiste basicamente de cinco etapas: *input*, *map*, *shuffle*, *reduce* e *output* (WHITE, 2009). Na figura 2.1, é apresentado um exemplo de fluxo real baseado na entrada de arquivos de texto com informações climáticas, no formato NCDC (*National Climatic Data Center*), e que gera como saída a maior temperatura global registrada em cada ano. Ainda nesta imagem, é apresentada uma solução semelhante através da utilização de *pipelines* do Unix, ainda que existam diferenças na forma da paralelização.

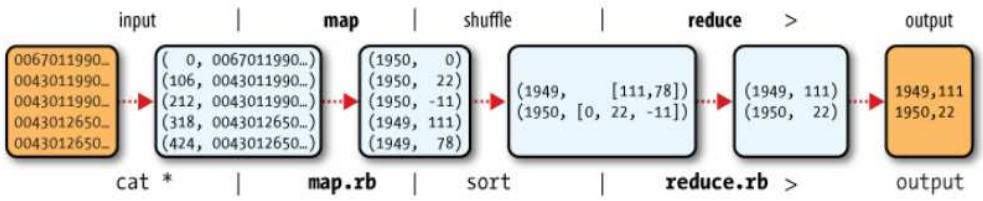

Figura 2.1: Fluxo completo dos dados em uma aplicação de MapReduce (WHITE, 2009)

Existe uma grande quantidade de implementações de MapReduce atualmente, em diversas linguagens de programação. As principais implementações conhecidas são Google MapReduce (C++) (DEAN; GHEMAWAT, 2004), Hadoop MapReduce (Java) (Apache

Software Foundation, 2007a), Greenplum (Python) (Greenplum, 2009), GridGain (Java) (GridGain, 2005), Phoenix (C) (Phoenix, 2007; RANGER et al., 2007), Skynet (Ruby) (Skynet, 2007), MapSharp (C#) (MapSharp, 2009), Disco (Erlang) (Disco, 2008), Holumbus (Haskell) (Holumbus, 2008; SCHMIDT, 2009), BashReduce (Bash Script) (BashReduce, 2009) entre diversas outras implementações criadas para atender demandas internas às empresas.

2.2 Hadoop

O Hadoop (Apache Software Foundation, 2007b) é uma coleção de sub-projetos, com foco em infra-estrutura para computação distribuída. Este projeto ganhou notoriedade pelo seu *framework* para MapReduce e pelo seu sistema distribuído de arquivos. Além destes, existem outros sub-projetos que proporcionam serviços complementares, ou ainda que incrementam as estruturas comuns, gerando novas abstrações. A seguir encontra-se uma breve explicação sobre cada sub-projeto existente no Hadoop:

- ***Common* (Apache Software Foundation, 2007c)**

  O Hadoop Common é um conjunto de utilidades que fornece suporte aos outros sub-projetos do Hadoop. Neste estão incluídas bibliotecas para sistemas de arquivos, RPC (*Remote Procedure Call*) e serialização de objetos.

- ***Avro* (Apache Software Foundation, 2008a)**

  O Avro é basicamente um sistema de serialização de dados, que proporciona a utilização de arquivos para armazenamento persistente de dados, RPC (*Remote Procedure Call*), bem como um formato binário de dados compacto e rápido.

- ***MapReduce* (Apache Software Foundation, 2007a)**

  O Hadoop MapReduce é um modelo de programação e um *framework*, destinados ao desenvolvimento de aplicações capazes de rapidamente processar enormes quantidades de dados em sistemas computacionais de grande escala, formados por equipamentos comerciais. Este *framework* será analisado com maior profundidade no decorrer do texto, visto que é o principal objeto de estudo deste trabalho.

- ***HDFS* (Apache Software Foundation, 2007d)**

O Hadoop Distributed File System (HDFS) é o sistema distribuído de armazenamento de arquivos utilizado pelas aplicações feitas com o Hadoop. O HDFS é responsável por criar múltiplas réplicas de blocos de dados e distribuí-los nos componentes do *cluster*.

- ***Pig* (Apache Software Foundation, 2007e)**

  O Pig é uma plataforma que permite a análise de grandes quantidades de dados e que consiste de uma linguagem de alto-nível para expressar aplicações que analisam tais dados. A plataforma Pig e a linguagem utilizada nesta, chamada de Pig Latin (OLSTON et al., 2008), foram contribuições ao *software* livre, concedidas pelo Yahoo!.

- ***HBase* (Apache Software Foundation, 2009a)**

  O HBase é considerado o banco de dados do Hadoop, este é um sistema distribuído de armazenamento de dados, orientado por colunas, de código aberto e baseado no modelo proposto pelo Google através do BigTable (CHANG et al., 2006). Este sub-projeto é bastante utilizado para acessos aleatórios aos dados disponíveis no sistema e quando são necessárias enormes quantidades de elementos nas tabelas de dados. Este sistema foi concebido pela empresa Powerset (atualmente pertencente a Microsoft), com o intuito de ser utilizado para buscas através de linguagem natural.

- ***ZooKeeper* (Apache Software Foundation, 2008b)**

  O Apache ZooKeeper é um serviço centralizado de coordenação, capaz de proporcionar serviços de grupos de maneira sincronizada e com alta atomicidade e disponibilidade. Este é bastante utilizado para solucionar condições de corrida inevitáveis.

- ***Hive* (Apache Software Foundation, 2008c)**

  O Hive é uma infra-estrutura para armazenamento de dados construído no topo de camadas do Hadoop, que proporciona ferramentas para indexação de dados, consultas *ad-hoc* e análise de grandes quantidades de informações armazenadas.

- ***Chukwa* (Apache Software Foundation, 2008d)**

O Chukwa é um sistema de coleta de dados para monitoramento de sistemas distribuídos de larga escala. Este sistema também possui ferramentas que permitem a apresentação, monitoramento e análise dos dados coletados.

Na figura 2.2, os sub-projetos do Hadoop são apresentados em camadas, de acordo com as relações entre eles:

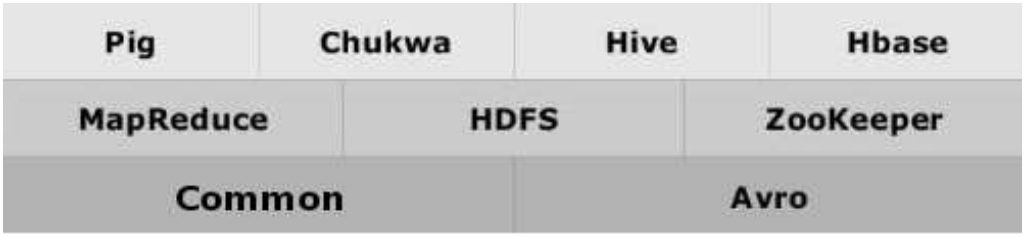

Figura 2.2: Sub-projetos do Hadoop em Camadas

Atualmente, as ferramentas deste projeto são utilizadas por diversas empresas de grande porte em inúmeras tarefas cotidianas, como por exemplo, análises e processamentos de *logs* dos serviços prestados. Destacam-se entre essas, Yahoo!, Facebook, Last.fm, Amazon, Adobe, AOL, Joost, Ning, Cloudera Inc, entre outras (Apache Software Foundation, 2009b).

2.2.1 Hadoop MapReduce

O Hadoop MapReduce (Apache Software Foundation, 2007a) é uma implementação de MapReduce em forma de *framework*, desenvolvido na linguagem de programação Java, e que visa o processamento de grandes quantidades de dados em *clusters* computacionais de larga escala de maneira confiável, escalável, distribuída e tolerante a falhas. Este *framework* é baseado nas premissas do modelo de programação MapReduce (DEAN; GHEMAWAT, 2004), porém ao contrário da concretização do Google, é um *software* livre e de código aberto.

Como já apresentado na figura 2.1, o MapReduce utiliza um *pipeline* especial para representar o fluxo de dados. Esse mesmo *pipeline* é utilizado na concretização de MapReduce do Hadoop, como pode ser visto na figura 2.3. Nesta, é possível perceber que os dados de entrada são inicialmente divididos em N pedaços (*splits*), sendo N um número proporcional ao número de *maps*. A entrega dos pedaços aos *maps* é um processo determinístico que ocorre através de uma função de *hash*. Cada *map* realiza sua computação sobre os pedaços que lhe foram fornecidos e repassam às funções de *reduce* os novos

dados. Novamente, a entrega dos dados é determinística, porém dessa vez a função considerada é a de *reduce*. Cada função de *reduce* realiza sua computação e grava os dados no seu próprio arquivo de saída, localizado no HDFS.

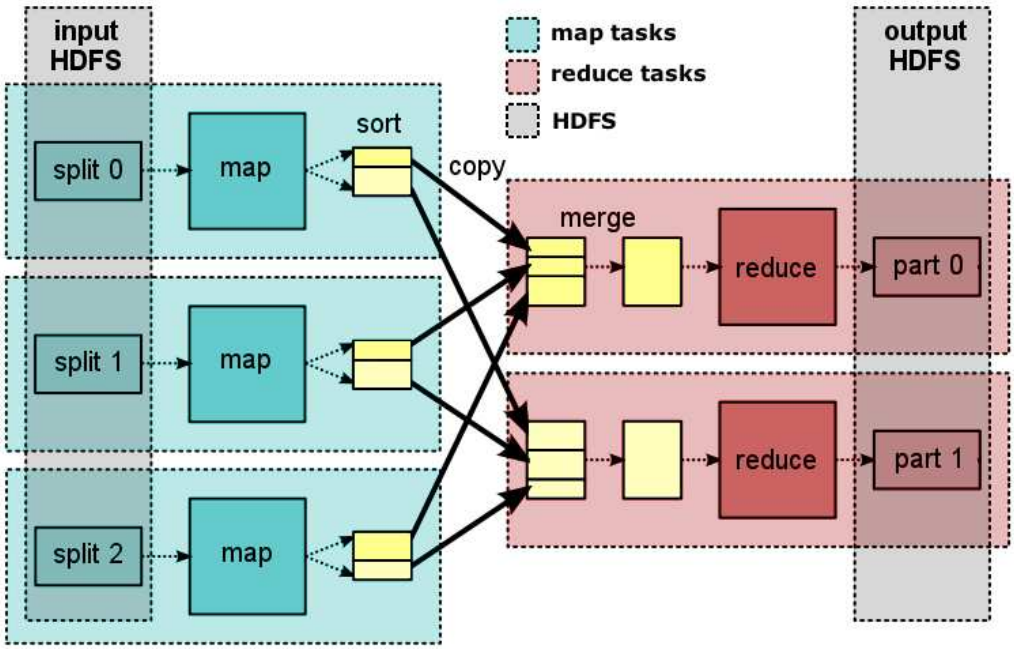

Figura 2.3: Exemplo de Fluxo de Dados no Hadoop MapReduce

Neste *framework*, um *job* representa o trabalho como um todo, ou seja, é formado pelos dados de entrada, pela aplicação MapReduce e pelos arquivos de configuração. O Hadoop executa um *job* através da divisão deste em diversas tarefas (*tasks*), que podem ser *map tasks* ou *reduce tasks*. Na figura 2.3, são representadas três tarefas de *map* e duas de *reduce*. Uma *map task* consiste do recebimento dos dados de entrada, realização da computação de *map* e ordenação do resultado desta tarefa, de tal forma que, através da operação *shuffle* os dados encontrem-se separados de acordo com as *reduce tasks* que os receberão. Cada *reduce task* recebe um conjunto de dados de diversos *maps* e realiza uma mescla desses dados, mantendo-os ordenados. Por fim, esses dados são utilizados na entrada da função *reduce*, onde é realizada a redução dos dados, a fim de gerar a saída do *job* gravada no HDFS.

A operação *shuffle* está presente tanto nas *map tasks*, quanto nas *reduce tasks*. Esta operação é considerada uma das mais importantes para o bom desempenho do Hadoop (WHITE, 2009), pois ela é responsável por garantir que a entrada de todo e qualquer *reduce* estará ordenada pelas chaves dos pares de informações. Além disso, a transferência dos dados de saída das funções de *map* para as entradas das funções de *reduce* também

fazem parte do *shuffle*.

2.3 *Benchmarking*

O neologismo *benchmarking* surgiu na área de Administração de Empresas em 1988 (Kaiser Associates, 1988) e consiste de um processo sistemático e contínuo de avaliação de produtos, serviços e processos de trabalhos de organizações, cuja finalidade é, através da comparação de desempenhos com outras empresas, indicar oportunidades de melhorias nesta em que se está realizando o *benchmarking*.

Em computação, o termo *benchmarking* é utilizado para o ato de executar determinados programas, a fim de obter comparações justas de desempenho entre diferentes sistemas computacionais e/ou seus componentes. Estes programas, normalmente chamados de *benchmarks*, utilizam conjuntos de aplicações ou operações reais ou sintéticas e tentam fazer com que os resultados obtidos sejam os mais próximos da realidade (WEICKER, 2002).

Existem três formas principais de classificar os *benchmarks*, sendo elas: quanto ao escopo de atuação, quanto à composição do *benchmark* e quanto à sua execução. No primeiro caso, a classificação divide-se de acordo com o foco de avaliação, ou seja, o nível de atuação a ferramenta de *benchmarking* irá analisar, podendo ser então dividido em nível de sistema e nível de componente (ZELKOWITZ, 2009).

- Nível de Sistema: Avalia o desempenho geral do sistema em que se está realizando o *benchmark*, onde normalmente é calculado apenas o tempo de execução de uma determinada aplicação.

- Nível de Componente: Avalia o desempenho de um componente específico do sistema em que se está realizando o *benchmark*, a fim de verificar qual a influência que este possui no desempenho geral.

A classificação baseada na composição da ferramenta de *benchmarking* consiste em cinco divisões que se distinguem pelas aplicações que compõem o *benchmark* (HENNESSY; PATTERSON, 2006), sendo elas:

- Aplicações Reais: São aplicações reais utilizadas pelos usuários do sistema, onde o intuito é recolher informações próximas à realidade destes.

- Aplicações Modificadas (ou instrumentadas): São aplicações reais, porém sem a presença do usuário do sistema, visto que são gerados scripts que realizam tarefas específicas nas quais se deseja realizar o *benchmarking*.

- Kernels: São programas formados por trechos de códigos extraídos de aplicações reais, com o intuito de analisar o desempenho das principais funcionalidades do sistema, separadamente.

- *Toy Benchmarks*: São pequenas funções, com resultados conhecidos e foco na portabilidade do *benchmark*.

- *Benchmarks* Sintéticos: São códigos criados artificialmente, que tentam imitar as principais funcionalidades das aplicações reais.

A terceira classificação tem como objetivo classificar os *benchmarks* quanto ao momento em que os testes são executados (HENNESSY; PATTERSON, 2006), sendo que as quatro divisões são:

- *Application-Based*: Basicamente, executam aplicações reais e apenas armazenam o tempo de execução.

- *Playback*: Consiste da análise dos *logs* gerados por uma determinada aplicação, e sucessivas tentativas de reprodução dos testes com configurações distintas às apresentadas nos *logs*.

- Execução sintetizada: A execução destina-se a aproximar-se de uma determinada funcionalidade do sistema ou componente a ser avaliado, a fim de obter resultados específicos daquele sub-sistema.

- Modo de inspeção: Não tenta imitar a execução da aplicação, mas sim executá-la e inspecioná-la em tempo de execução. Nestes casos, normalmente necessita-se de alguma instrumentação de código para apresentar os resultados em tempo real.

Existe também uma classificação para as métricas analisadas pelos *benchmarks*, a qual consiste em dividi-las quanto à abordagem em relação ao sistema ou componente que se deseja realizar o *benchmarking*. As duas divisões existentes são:

- Métricas Caixa-Preta (*black-box*): As métricas deste tipo referem-se a informações aparentes externamente ao sistema ou componente, ou seja, elas possuem a granularidade maior do que a granularidade do sistema ou componente em questão.

- Métricas Caixa-Branca (*white-box*): Essas métricas referem-se especificamente ao sistema ou componente em questão, ou seja, para atingir esta granularidade fina, é necessário conhecer o funcionamento interno do sistema ou componente.

Para melhor exemplificar métricas caixa-preta e caixa-branca, podem ser consideradas métricas caixa-branca as quantidades de tarefas *map* e *reduce* geradas, a quantidade de dados analisados por cada *map*, entre diversas outras. Já métricas caixa-preta podem ser consideradas aquelas fornecidas pelo Sistema Operacional como por exemplo o tempo total de execução de uma aplicação, o uso de memória e processadores, entre outras.

Tanto *benchmarks* para MapReduce, quanto para Hadoop são temas recentes no meio científico, porém muitos deles utilizam conceitos e propostas de *benchmarking* para diversas outros modelos de programação e *frameworks*.

Outro tema de pesquisa recente é sobre os *benchmarks* para Cloud-Computing, visto que as ferramentas existentes atualmente não são capazes de calcular os desempenhos relativos em plataformas semelhantes às propostas nas *clouds* (BINNIG et al., 2009). Isto deve-se principalmente porque os *benchmarks* atuais para sistemas distribuídos consideram a execução de aplicações em uma plataforma estática durante os testes, porém nas plataformas para Cloud-Computing, a quantidade de máquinas utilizadas para a execução de um serviço pode variar de acordo com a demanda.

2.4 Grid'5000

O Grid'5000 (GRID5000 Funding, 2003) é uma plataforma computacional científica destinada à pesquisa de sistemas distribuídos e paralelos em larga escala. Um dos principais objetivos do Grid'5000 é proporcionar uma plataforma altamente reconfigurável, controlável e monitorável aos seus usuários (CAPPELLO et al., 2005). A meta inicial do projeto era atingir 5000 processadores, porém esta foi alterada para 5000 núcleos, e então alcançada no final de 2008.

A plataforma foi inicialmente distribuída em nove cidades da França, sendo elas: Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia e Toulouse. A cidade de Porto Alegre (Brasil) foi a décima cidade a integrar este projeto, em Julho de 2009. A

seguir, na Tabela 2.1, é possível verificar a quantidade de nós, processadores e núcleos que cada cidade possui, bem como o total do Grid'5000:

Tabela 2.1: Número de Nós, Processadores e Núcleos em cada cidade do Grid'5000

Cidade	Nós	Processadores	Núcleos
Bordeaux	202	424	650
Grenoble	34	68	272
Lille	145	290	618
Lyon	135	252	252
Nancy	259	518	1310
Orsay	342	684	684
Rennes	161	322	900
Sophia	178	356	568
Toulouse	137	274	434
Porto Alegre	14	28	112
Total	1607	3216	5800

A infra-estrutura de rede utilizada pelo Grid'5000 é proporcionada pelo RENATER (*French National Telecommunication Network for Technology, Education and Research*) (Groupement d'Intérêt Public RENATER, 1993), sendo que este consiste de cerca de 30 POPs (*Points Of Presence*). A interconexão entre as cidades do Grid'5000 na França é feita através de cabos de fibra ótica, que permitem comunicações a 10Gbit/s, cuja topologia de rede pode ser vista na figura 2.4.

A estrutura e organização dos componentes de rede, bem como a disposição e configuração dos componentes de processamento em cada cidade do Grid'5000 pode variar. Mesmo com essa autonomia, todas as cidades da França possuem as mesmas características como interface de entrada, ou seja, possuem um roteador ótico com suporte à tecnologia de Multiplexagem Densa em Comprimento de Onda (DWDM) e ao menos um roteador com suporte à Ethernet 1Gbit/s. Um exemplo dessa estrutura pode ser visto na figura 2.5.

Ainda na figura 2.5, é possível verificar a existência de alguns componentes que mantém diversos serviços destinados aos usuários e administradores do Grid'5000. Mesmo podendo existir diferenças entre os serviços prestados, os principais são: NAT, NTP, DNS, VTun, Web Server, SMTP, NFS, Ganglia, Nagios, Backup, MySQL, OAR-Server, DHCP, OAR-Submission, Kadeploy, Xen, Compilation Server, LDAP.

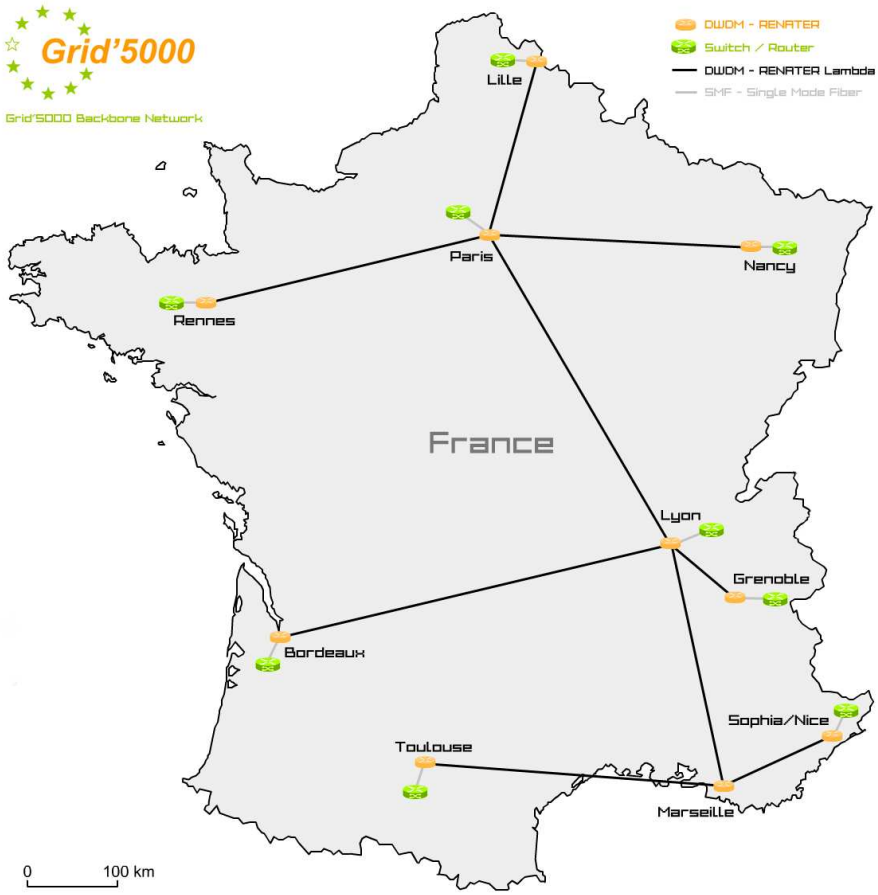

Figura 2.4: Topologia de rede do Grid'5000

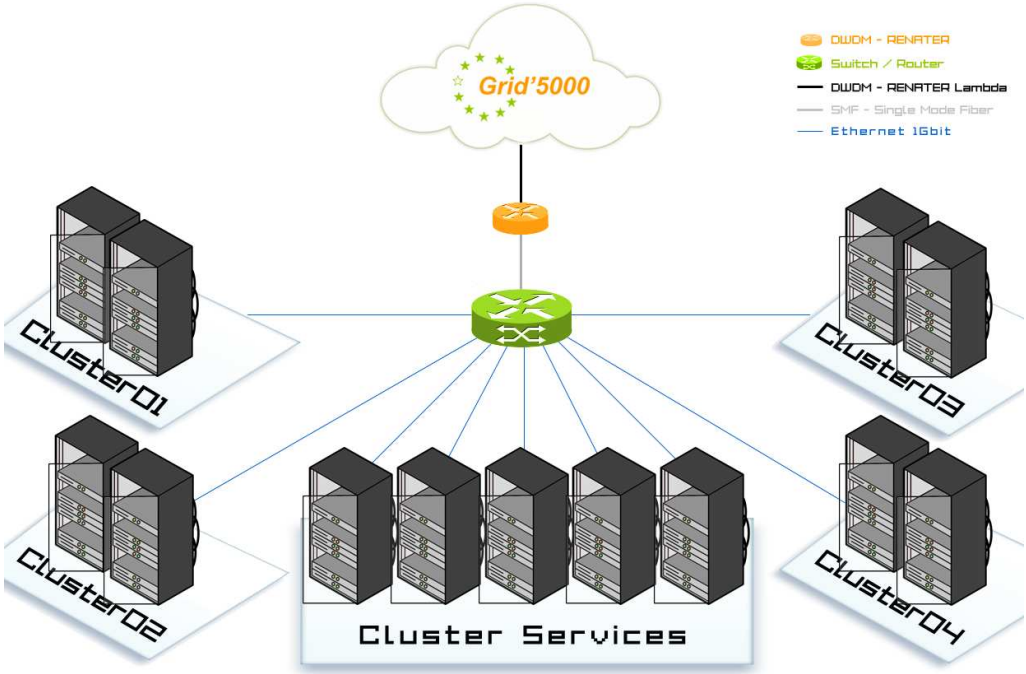

Figura 2.5: Estrutura de Rede interna a um site do Grid'5000

Dentre todos estes serviços, dois deles merecem destaque para este trabalho: o OAR e o Kadeploy. O OAR (Laboratoire d'Informatique de Grenoble, 2008; CAPIT, 2008) é um gerenciador de recursos para *clusters* de larga escala, através do qual são submetidas reservas de componentes computacionais no Grid'5000. Uma vez realizada a reserva com o OAR e recebido o retorno com os componentes selecionados, utiliza-se o Kadeploy (LEDUC; PEAQUIN, 2007) para realizar a implantação de um determinado Sistema Operacional nestas máquinas. Sendo que o ato de implantar o Sistema Operacional em um determinado computador é chamado de *deploy*, e no Grid'5000, estão disponíveis para tal os seguintes sistemas:

- Debian Sid *{base, nfs e big}*
- Debian Etch *{base, nfs e big}*
- Debian Lenny *{base, nfs e big}*
- FreeBSD
- Solaris
- Microsoft Windows

3 AMBIENTE DE TESTES PARA HADOOP NO GRID'5000

O projeto de um ambiente de testes consiste na descrição detalhada do ambiente de *hardware* e *software* no qual os testes serão executados, bem como a descrição de quaisquer outras ferramentas que venham a interagir com o *software* sob teste (British Computer Society, 1998). No presente capítulo, serão descritos os pré-requisitos e a preparação do ambiente de testes para Hadoop no Grid'5000, utilizado para os testes com os benchmarks neste trabalho.

3.1 Pré-requisitos do Hadoop

Como acontece com todo e qualquer *software*, existem alguns pré-requisitos necessários para o seu correto funcionamento. O Hadoop possui poucos pré-requisitos, porém estes devem ser cuidadosamente preparados, para obter uma execução funcional. Basicamente, eles estão relacionados ao conjunto de computadores interligados em rede, ao sistema operacional e à Máquina Virtual Java (VENNER, 2009).

Quanto ao primeiro item, o Hadoop está projetado para utilizar um único computador ou um conjunto desses, sendo que a primeira opção é o padrão inicial e a segunda é o caso mais útil e empregado. A comunicação entre os componentes deve ser feita através do protocolo de comunicação SSH e a forma de autenticação deve ser baseada no modelo de chaves públicas, para que não seja necessária a autenticação interativa baseada em senhas.

O Hadoop foi desenvolvido para ser utilizado tanto em Sistemas Operacionais baseados em GNU/Linux, quanto baseados em Win32. Enquanto a primeira opção é indicada para casos de desenvolvimento e produção, a segunda apenas é indicada para fases de desenvolvimento.

Devido ao fato do núcleo do Hadoop ter sido desenvolvido utilizando a linguagem de programação Java, é necessário que os componentes computacionais possuam uma Máquina Virtual Java (JVM). A principal recomendação para este item é de se utilizar as própria JVMs fornecidas pela Sun, a partir da versão 1.6.00, pois as versões anteriores podem não possuir todas as funcionalidades necessárias ao Hadoop.

3.2 Pré-requisitos do FTH-Grid

Além de atender os pré-requisitos do Hadoop, descritos anteriormente, o ambiente de testes deve também buscar atender alguns requisitos desejáveis, interessantes ao projeto FTH-Grid e a outros projetos que venham utilizar o Hadoop no Grid'5000. Alguns destes são:

- Permitir a realização de testes em diferentes conjuntos computacionais;
- Automatizar a configuração do Hadoop de acordo com o conjunto de máquinas de cada teste;
- Proporcionar repetições determinísticas dos testes;
- Suportar diferentes formas de interação com o sistema (ativa e passiva);
- Inicializar o Hadoop, automaticamente após a sua configuração;
- Coletar e armazenar informações e *logs* sobre os testes realizados;

3.3 Concretização no Grid'5000

Existem alguns casos de propostas de ambientes de testes para Hadoop disponíveis na internet (NOLL, 2009a,b; BEROSIK, 2009), nos quais são apresentados os passos necessários para a correta instalação e configuração do Hadoop, bem como discussões sobre os problemas encontrados e as soluções adotadas. Nenhuma delas considera plataformas computacionais de grande escala, nem formas de automatizar os processos de configuração e inicialização do Hadoop. Os objetivos destas descrições são apenas apresentar como executar o Hadoop em *clusters* estáticos, sem composições de conjuntos distintos de computadores.

Nesta seção será apresentada a concretização de um ambiente de testes dinâmico para Hadoop, que no caso, considera como plataforma computacional o Grid'5000, e está di-

vidida em três sub-seções, sendo elas: o processo de instalação, configuração do Hadoop e utilização do ambiente.

3.3.1 Instalação

Para esta concretização do ambiente de testes, é considerado que o processo de instalação dos componentes deste deverá ocorrer apenas uma vez, visto que será criada uma imagem do sistema operacional. Esta imagem conterá todos os aplicativos a serem utilizados pelo sistema operacional e os arquivos do Hadoop localizados nos devidos diretórios, sendo que em todos os testes a serem realizados, as configurações de instalação são as mesmas.

Para o processo de instalação deve-se seguir os passos apresentados na seção A.1 do apêndice, os quais consistem principalmente da reserva de um único nó no Grid'5000, da implementação do ambiente lenny-x64-nfs neste nó, instalação das aplicações úteis ao ambiente e a colocação dos arquivos da distribuição do Hadoop em uma pasta específica, externa ao NFS.

A única modificação necessária na etapa de instalação, nos arquivos de configuração do Hadoop é a configuração da variável $JAVA_HOME. O passo final consiste em criar uma nova imagem do sistema, através de um script específico para o Grid'5000 (chamado *tgz-g5k*).

3.3.2 Configuração

O processo de configuração faz-se necessário neste ambiente de testes, pois para cada novo teste, existem propriedades do Hadoop que podem ser diferentes, principalmente ao que se refere ao conjunto de máquinas reservadas para o trabalho.

Desta forma, os scripts de configuração executam tarefas de atualização dos arquivos da pasta *conf* do Hadoop. Após esta atualização, os scripts devem ainda inicializar a execução do Hadoop e deixar tudo pronto para que o usuário apenas precise indicar qual aplicação deseja executar neste *framework*. Os scripts criados para a configuração e inicialização do Hadoop no Grid'5000 encontram-se na seção A.2 do apêndice.

3.3.3 Utilização

Para utilizar o ambiente de testes proposto, o usuário apenas necessita realizar os mesmos passos solicitados para quaisquer outros ambientes do Grid'5000, porém apenas

com os parâmetros distintos. Na seção A.3 do apêndice, encontra-se um exemplo dessa sequência de passos.

Desta forma, o primeiro deles consiste acessar via SSH o *frontend* da cidade em que se deseja executar o Hadoop. O segundo consiste em reservar no Grid'5000, através do OAR (Laboratoire d'Informatique de Grenoble, 2008), a quantidade de componentes em que se deseja implantar o ambiente de testes. Neste comando, deve-se passar como argumentos o tipo de deploy, a quantidade de nós e o tempo limite de execução da aplicação (*walltime*).

Enquanto o terceiro passo consiste em realizar o *deploy* do ambiente que se deseja utilizar, através da ferramenta Kadeploy (LEDUC; PEAQUIN, 2007). Neste comando, deve-se passar como argumentos o arquivos descritor do ambiente, os nós reservados para o trabalho, a chave pública do usuário e o script a ser executado após o *deploy*.

4 SOLUÇÕES DE *BENCHMARKING* PARA HADOOP

No presente capítulo serão apresentados os requisitos desejáveis à escolha dos *benchmarks*, bem como as soluções encontradas para tal, seja através da simulação de carga de trabalho, análise de componentes ou análise de *logs*. Por fim, serão apresentados quadros comparativos entre os *benchmarks* analisados, as ferramentas selecionadas para o projeto FTH-Grid, bem como algumas sugestões de melhoria para um dos *benchmarks* selecionados.

4.1 Requisitos

Dentre os requisitos existentes para a análise e seleção de *benchmarks*, neste trabalho é considerado primordial, que uma ou mais dessas ferramentas sejam capazes de comparar duas versões distintas do Hadoop (original e modificado), a fim de suprir as necessidades do projeto FTH-Grid, como apresentado em capítulos anteriores.

Além deste requisito, existem alguns fatores que são considerados desejáveis às ferramentas escolhidas, sendo eles:

- Que o *benchmark* seja popular e utilizado em grande escala.
- Caso ainda não seja popular, que o *benchmark* possa vir a se tornar.
- Que o *benchmark* seja referência científica, citado e utilizado em artigos.
- Que a ferramenta seja de fácil acesso e execução.
- Que o *benchmark* possua boa documentação interna e ou na internet.
- Que apresente resultados consistentes de forma clara e objetiva.

Nas seções que seguem, são apresentadas as principais propostas existentes para *benchmarking* do Hadoop, sejam elas baseadas em simulação de carga de trabalho, análise

de componentes ou análise de *logs*. Além disso, também serão apresentadas, em maior profundidade, as soluções definidas como ideais para os requisitos anteriormente citados.

4.2 *Benchmarks* com Simulação de Carga de Trabalho

A carga de trabalho consiste de um conjunto de aplicações e comandos que os usuários utilizam frequentemente (HENNESSY; PATTERSON, 2006). Simular esta, significa executar tais operações em determinadas sequências, a fim de obter análises de desempenho diretamente relacionadas às necessidades dos usuários de um programa. Por sua vez, os *benchmarks* que simulam essa carga de trabalho levam em consideração as principais atividades e funcionalidades utilizadas na maioria das aplicações feitas com Hadoop.

Uma das principais cargas de trabalho consideradas pelos *benchmarks* para Hadoop é a operação de ordenação de dados, ou seja, a partir de uma entrada desordenada, obtém-se uma saída ordenada de acordo com as chaves de cada par de informações. Existe uma quantidade relativamente grande de algoritmos de ordenação, que normalmente são apresentadas nas disciplinas de Pesquisa e Ordenação de Dados.

O mais conhecido dos *benchmarks* de ordenação de dados para Hadoop é o chamado **Sort** (WHITE, 2009). Esta é uma aplicação MapReduce, que realiza uma ordenação parcial dos dados de entrada e que além desta ordenação, fornece um programa que gera os dados de entrada aleatoriamente (RandomWriter) e outro que valida os dados de saída do processo de ordenação (SortValidator).

Outra implementação de destaque para ordenação de dados com Hadoop é a chamada **TeraSort**, criada por Owen O' Malley (O'MALLEY; MURTHY, 2009), com o intuito de participar da competição de ordenação chamada GraySort (GRAY et al., 1998). Em 2009, o TeraSort foi o campeão da edição em duas categorias, ao ordenar 500 GB em 1 minuto (MinuteSort) e 100 TB em 173 minutos (GraySort). Para provar a escalabilidade da solução, ainda demonstrou-se os resultados da ordenação de 1 PB em 975 minutos (equivalente a 16,25 horas).

Ao invés de representarem a carga de trabalho em um único tipo de operação, existem *benchmarks* que buscam resumir esta carga na forma de várias operações importantes aos usuários. Uma destas propostas é o **GridMix**, um *benchmark* que simula uma carga de trabalho real, a fim de aproximar as aplicações com a maior quantidade possível de casos de uso do Hadoop (WHITE, 2009). As aplicações internas ao GridMix utilizam além

de aplicações de ordenação, casos de funções de combinação (entre o *map* e o *reduce*), leituras de dados não-locais, entre outros. Este *benchmark* é considerado um dos mais próximos da realidade (WHITE, 2009), porém ainda não é o mais popular dos *benchmarks*, mesmo este já tendo sido utilizado como base de avaliações (SANDHOLM; LAI, 2009).

Outra implementação que foi além das funções de ordenação, foi a apresentada por Shengkai Zhu (ZHU et al., 2009) e que consiste da adaptação de duas aplicações do *benchmark* Stanford **SPLASH-2** (WOO et al., 1995), a fim de verificar a viabilidade na execução de aplicações típicas para supercomputação utilizando o modelo de programação MapReduce, bem como tentar encontrar gargalos neste. Para isto, as aplicações transportadas foram o Radix Sort e o Water Spatial. O primeiro consiste do método de ordenação Radix Sort e é utilizada principalmente por necessitar de comunicação entre todos os processos. O segundo consiste de um modelo que avalia forças e tendências que ocorrem ao longo do tempo em um sistema de moléculas de água e foi utilizado devido a existir necessidade de comunicação entre células vizinhas.

Por fim, o **MRBench** é uma proposta de *benchmarking* com foco no processamento de consultas orientadas a negócios e modificações concorrentes de dados (KIYOUNG et al., 2008). Para isso, o *benchmark* trabalha com grandes volumes de dados relacionais e executa consultas altamente complexas. Foram implementadas em Hadoop MapReduce todas as 22 consultas existentes no TPC-H (Transaction Processing Performance Council, 2009). Os resultados são gerados utilizando as unidades de medida *Query-per-Hour (QphH)* e consequentemente Price/QphH. Existe uma aplicação homônima na distribuição do Hadoop, porém esta não possui as mesmas características do que as apresentadas no artigo, ao passo que a aplicação do projeto Hadoop apenas executa trabalhos pequenos, por repetidas vezes (WHITE, 2009).

4.3 *Benchmarks* para Análise de Componentes

Enquanto os *benchmarks* a nível de sistema buscam avaliar o desempenho geral deste, os que trabalham a nível de componente avaliam o desempenho ou funcionamento apenas de determinadas partes do sistema (ZELKOWITZ, 2009). Nesta seção são apresentadas soluções de *benchmarking* a nível de componente ou funcionalidade, referente aos seguintes tópicos: HDFS, MapReduce, Confiabilidade, Comunicação entre processos e

Arquivos de Entrada e Saída.

4.3.1 Hadoop Distributed File System

Visto que o Hadoop Distributed File System (HDFS) é um sistema distribuído para armazenamento de arquivos, os *benchmarks* para este componente consideram como principal fator de análise a vazão de operações sobre os arquivos localizados no HDFS.

O principal *benchmark* para HDFS é o **TestDFSIO**, que calcula a vazão (*throughput*) de operações sobre os arquivos do HDFS, como por exemplo: leitura, escrita e remoção. Os parâmetros controláveis deste *benchmark* são basicamente as operações, quantidade e tamanho dos arquivos (WHITE, 2009). Com esta mesma finalidade, também existe o *benchmark* chamado **DFSThroughput**, porém os contadores apresentados no resultado do TestDFSIO são mais completos (VENNER, 2009).

Para as opções de entrada e saída distribuída de arquivos no HDFS existe a ferramenta de *benchmarking* **TestFileSystem**, que também possui como parâmetros a quantidade e tamanho dos arquivos (VENNER, 2009). Para finalizar os *benchmarks* para HDFS, existe a ferramenta chamada **DFSCIOTest** (VENNER, 2009), que é direcionada à entrada e saída distribuída da *libhdfs*, a qual é uma biblioteca que promove os serviços de arquivos HDFS para aplicações escritas em C e C++.

4.3.2 Hadoop MapReduce

Os *benchmarks* baseados em simulação de carga de trabalho preocupam-se com o desempenho do Hadoop de maneira geral, analisando somente o tempo de execução das aplicações. Nesta seção, busca-se uma outra forma de medir o desempenho do Hadoop MapReduce, através da análise baseada nos conceitos que compõem o modelo de programação MapReduce.

Um dos principais fatores desejáveis à análise por parte dos *benchmarks* de MapReduce é justamente o cálculo da quantidade ideal de *maps* e *reduces* em uma determinada aplicação sobre um conjunto de dados de tamanho fixo. Porém não existe nenhuma ferramenta que realize esse processo de forma automática, sendo necessário executar a aplicação, manualmente, por diversas vezes com diferentes quantidades de *maps* e *reduces*, a fim de se obter os valores que resultaram em um menor tempo de execução.

Outro fator importante de se analisar é a correta execução das funções de *map* e *reduce*, por parte do *framework*. O **MapRedTest** é uma ferramenta que se aproxima aos

testes de software, visto que ela é capaz de definir se o Hadoop está executando as funções de *map* e *reduce* corretamente. Esta ferramenta utiliza o Hadoop para calcular um valor correspondente a uma soma, se o valor for o esperado, significa que o Hadoop está executando corretamente suas funções quanto ao modelo de programação MapReduce (VENNER, 2009). Outra ferramenta nesta mesma linha, porém apenas para casos de ordenação é o **TestMapRedSort**, que baseado na entrada, saída, número de *maps* e *reduces* da computação, é capaz de definir se a execução foi propriamente correta (VENNER, 2009).

Por fim, o **ThreadedMapBench** é um *benchmark* que analisa a influência da quantidade de *spils* que ocorrem no *shuffle*, sobre o desempenho de uma aplicação MapReduce (VENNER, 2009).

4.3.3 Comunicação entre Processos

Por tratar-se de um modelo de programação baseado em sistemas distribuídos, o MapReduce necessita de uma grande quantidade de comunicações entre os processos de cada trabalho. Dessa forma, existem dois *benchmarks* para Hadoop que abordam este tema. O primeiro deles é o **TestRPC**, que como o próprio nome sugere, realiza alguns testes relativos às chamadas remotas de procedimentos, enquanto o outro é o **TestIPC**, que trata de realizar testes acerca das comunicações entre processos, internamente ao Hadoop Common (VENNER, 2009).

4.3.4 Arquivos, Entrada e Saída

Existem diversos formatos de entrada e saída criados especificamente para o Hadoop e os *benchmarks* relacionados à estes itens, preocupam-se em verificar se eles estão corretamente formados e preenchidos. As principais ferramentas para estes casos são: FileBench, TestArrayFile, TestSequenceFile, TestSetFile, TestBigMapOutput e TestSequenceFileInputFormat (VENNER, 2009).

O *benchmark* **FileBench** analisa os formatos SequenceFileInputFormat e SequenceFileOutputFormat, com compressão por blocos, por registros e sem compressão; além dos formatos TextInputFormat e TextOutputFormat comprimidos e não comprimidos. O **TestArrayFile**, **TestSequenceFile** e **TestSetFile** são utilizados para verificar se os arquivos estão corretamente preenchidos com pares chave/valor binários.

O **TestBigMapOutput** é um *benchmark* que trabalha em um arquivo realmente grande

não-divisível, enquanto o **TestSequenceFileInputFormat** é um *benchmark* para testar o formato dos arquivos de entrada sequenciais. Por fim o **TestTextInputFormat** analisa os arquivos de entrada no formato de entrada de texto.

4.3.5 Confiabilidade do Hadoop

O último componente a ser considerado neste estudo, porém não menos importante, é a realização de testes baseados em injeção de falhas, para se verificar a confiabilidade do Hadoop, para execução de aplicações MapReduce em sistemas considerados suscetíveis a falhas.

O **MRReliabilityTest** é um *benchmark* que testa se o Hadoop MapReduce pode ser considerado uma ferramenta confiável ou não (VENNER, 2009). Isto ocorre baseado no fato de que este *benchmark* executa alguns exemplos de aplicações sintéticas no Hadoop, ao passo que tenta finalizar a execução através de injeção de falhas nos nós que estão executando as tarefas. As aplicações que são executadas internamente a este teste são SleepJob, RandomWriter, Sort e ValidatorSort.

4.4 *Benchmarks* Baseados em Análise de Logs

Em uma parte considerável dos casos apresentados, é possível além de verificar os valores das métricas através de acompanhamento da execução, também verificá-las através da análise de *logs*. O Hadoop possui um sistema bastante completo de geração de *logs*, o que facilita a análise da execução baseada nesses registros. Como a análise de *logs* pode abranger diversos pontos de avaliação, o número de casos de ferramentas para *benchmarking* através de análise dos *logs* é relativamente grande.

As principais iniciativas de análise de *logs* para MapReduce e Hadoop, surgem do projeto Fingerpointing (Carnegie Mellon University, 2009), orientado pela professora Priya Narasimhan, da Universidade Carnegie Mellon. Seguindo a ordem cronológica das iniciativas, em Maio de 2008, a primeira consistiu em relacionar métricas *white-box* dos *logs* baseados em eventos do Hadoop com as métricas *black-box* do sistema operacional na iniciativa chamada **BlackSheep** (TAN; NARASIMHAN, 2008), juntamente à iniciativa **RAMS** (TAN; NARASIMHAN, 2008) que consistiu em relacionar as atividades no modo de operações privilegiadas do sistema operacional e o modo de usuário sem privilégios em condições sem falhas, a fim de verificar suas influências no resultado.

A segunda iniciativa, apresentada em Dezembro de 2008, é uma ferramenta para análise de fluxo de dados e de controle através de máquinas de estado da execução de aplicações com Hadoop, baseadas nos *logs* gerados pelo próprio Hadoop. Esta iniciativa ficou conhecida como **SALSA**: *Analyzing Logs as StAte Machines* (TAN et al., 2008). Diferentemente de propostas anteriores, não pertencentes a este projeto, como por exemplo o **Magpie** (BARHAM et al., 2004), o SALSA não necessita de instrumentação de código para recolher as informações necessárias para a criação da máquina de estados da execução da aplicação.

Em Maio de 2009, surge a iniciativa **BliMeE**: Blind Men and the Elephant (PAN, 2009), que é um *framework* com algoritmos para análise baseados na combinação das métricas *black-box* do sistema operacional, métricas *white-box* do Hadoop, juntamente com os *hearthbeats* a nível de aplicação (PAN et al., 2009a).

A iniciativa chamada **Ganesha** (PAN et al., 2008, 2009b) utiliza métricas *black-box* a nível de sistema operacional, a fim de diagnosticar faltas em sistemas MapReduce, de maneira transparente ao usuário. No mesmo mês de Junho de 2009, foi criada uma ferramenta visual para *debug* das aplicações feitas com Hadoop, que utiliza os *logs* gerados pelo próprio Hadoop, e ficou conhecida como **Mochi** (TAN et al., 2009).

Uma outra iniciativa que não pertence ao projeto Fingerpointing é o **X-Tracing Hadoop** (KONWINSKI et al., 2008), que consiste de uma iniciativa para tentar coletar rastros de eventos deixados pela execução do Hadoop utilizando o *framework* X-Trace (FONSECA et al., 2007). Esta coleta é então apresentada através de páginas Web e promovem estatísticas de desempenho, gráficos de utilização e análise dos pontos críticos da execução.

4.5 Seleção dos *Benchmarks*

4.5.1 Quadros Comparativos

Nesta subseção serão apresentados quadros comparativos dos *benchmarks* analisados, separados de acordo com a classificação destes. Nestes encontram-se o título de cada um e o seu ponto de atuação ou métricas informadas.

Tabela 4.1: Tabela Comparativa dos *Benchmarks* baseados em Simulação de Carga de Trabalho

Benchmark	**Foco**
Sort	Tempo de execução de ordenação de dados
TeraSort	Tempo de execução de ordenação de dados
GridMix	Tempo de execução de carga de trabalho real
SPLASH-2	Tempo de execução de ordenação de dados e Simulador de Particulas
MRBench	Consultas orientadas a negócio e modificações concorrentes de dados

Tabela 4.2: Tabela Comparativa dos *Benchmarks* baseados em Análise do HDFS

Benchmark	**Foco**
TestDFSIO	Vazão (throughput) de operações em arquivos
DFSThroughput	Vazão (throughput) de operações em arquivos
TestFileSystem	Vazão de escrita e leitura
DFSCIOTest	Vazão de operações em C e C++

Tabela 4.3: Tabela Comparativa dos *Benchmarks* baseados em Análise do MapReduce

Benchmark	**Foco**
MapRedTest	Eficácia da execução das funções de *map* e *reduce*
TestMapRedSort	Eficácia da ordenação de dados
ThreadedMapBench	Influência dos *spils* na saída do *map*

Tabela 4.4: Tabela Comparativa dos *Benchmarks* baseados em Análise de Comunicação

Benchmark	**Foco**
TestRPC	RPC (*Remote Procedure Call*)
TestIPC	IPC (*Inter Process Communication*)

Tabela 4.5: Tabela Comparativa dos *Benchmarks* baseados em Análise de Arquivos

Benchmark	**Foco**
FileBench	Análise de SequenceFile(Input|Output)Format e Text(Input|Output)Format
TestArrayFile	Análise de arquivos com formatos binários de pares chave/valor
TestSequenceFile	Análise de arquivos com formatos binários de pares chave/valor
TestSetFile	Análise de arquivos com formatos binários de pares chave/valor
TestBigMapOutput	Teste com um arquivo de saída grande não-divisível.
TestSequenceFileInputFormat	Análise de SequenceFileInputFormat

Tabela 4.6: Tabela Comparativa dos *Benchmarks* baseados em Análise de Logs

Benchmark	**Foco**
BlackSheep	Métricas White-box e Black-box
RAMS	Diferença das métricas no modo administrador e modo usuário do SO
SALSA	Análise de fluxo de dados e geração de máquinas de estado da execução da aplicação
Magpie	Máquinas de estado da execução, sem instrumentação de código
BliMeE	Métricas white-box, black-box e *hearthbeats*
Ganesha	Métricas black-box
Mochi	Ferramenta Visual para debug de aplicações
X-Tracing Hadoop	Coleta de eventos para análise da execução

Tabela 4.7: Tabela Comparativa dos *Benchmarks* baseados em Análise de Confiabilidade

Benchmark	**Foco**
MRReliabilityTest	Confiabilidade da execução do Hadoop

4.5.2 *Benchmarks* Selecionados ao FTH-Grid

Conforme apresentado na seção de requisitos deste capítulo, o principal objetivo desta atividade é selecionar uma ou mais ferramentas de *benchmarking*, com as quais o programador seja capaz de comparar duas versões distintas do Hadoop.

Dessa forma, o primeiro fator a ser comparado é o tempo de execução de um programa que simule a carga de trabalho de aplicações que utilizem o Hadoop. Para este quesito foi selecionado o *benchmark* **GridMix** por ser o que possui maior abrangência quanto à quantidade de casos de uso do Hadoop. Outros fatores que influenciaram essa escolha foram: (1) a potencialidade deste *benchmark*, visto que ele é reconhecido como completo, porém não é ainda o mais utilizado no meio científico; (2) a facilidade de acesso ao *benchmark*, visto que este é distribuído juntamente com o pacote do Hadoop Common. Uma das principais desvantagens deste *benchmark* é que ele necessita de uma configuração não-trivial anterior à execução propriamente dita.

A análise de confiabilidade do Hadoop foi considerada outro fator importante ao conjunto de ferramentas para *benchmarking*, principalmente pelo fato do projeto FTH-Grid prever alterações na tolerância a falhas bizantinas no Hadoop. Sendo assim, o *benchmark* escolhido para este quesito foi o **MRReliabilityTest**, por ser o único que possui foco em confiabilidade e por ter um sistema de injeção de falhas implementado internamente.

Como prevê-se no projeto FTH-Grid a alteração do código da distribuição do Hadoop, a fim de torná-la tolerante a falhas bizantinas, é importante precaver-se de que após essas modificações serem realizadas, o Hadoop continue executando as funções de *map* e *reduce* devidamente. Desta forma, o *benchmark* selecionado para suprir esta necessidade foi o **MapRedTest**, que como explicado anteriormente, realiza o cálculo de uma soma e retorna um determinado valor. Se este valor corresponder ao esperado, o Hadoop segue executando corretamente as funções de *map* e *reduce*, caso contrário, houve alguma alteração na forma de implementação do modelo de programação MapReduce.

Por fim, como não estava definido o exato local onde o Hadoop seria modificado pelo projeto FTH-Grid, optou-se por selecionar um *benchmark* que alertasse para mudanças de desempenho em operações no HDFS. O *benchmark* selecionado, nesse caso, é o **TestDF-**

SIO, devido à facilidade de execução e clareza nos resultados retornados ao programador.

Estes *benchmarks* são apresentados na tabela 4.8, que resume a área de atuação de cada um destes.

Tabela 4.8: Tabela Comparativa dos *Benchmarks* selecionados ao FTH-Grid

Benchmark	**Foco**
GridMix	Simulação de Carga de Trabalho Real
MRReliabilityTest	Confiabilidade da execução do Hadoop
MapRedTest	Eficácia das tarefas de *map* e *reduce*
TestDFSIO	Vazão de operações em arquivos localizados no HDFS

4.5.3 Considerações Sobre o MRReliabilityTest

A ferramenta MRReliabilityTest, selecionada na seção anterior para teste da confiabilidade do Hadoop, possui uma boa forma controlável de injeção de falhas, porém poderia retornar uma quantidade de informações maior do que as atualmente retornadas. O retorno final do programa apenas são os contadores do Hadoop e se a execução foi bem sucedida. Em outras palavras, poderia gerar informações úteis ao contexto de tolerância a falhas, como por exemplo as seguintes métricas:

- Taxa de Defeitos (Defeitos por unidade de tempo)
- MTTF - Mean Time To Failure (Tempo esperado até a primeira ocorrência de defeito)
- MTTR - Mean Time To Repair (Tempo médio para reparo do sistema)
- MTBF - Mean Time Between Failures (Tempo médio entre duas falhas)

5 RESULTADOS E AVALIAÇÕES

Neste capítulo serão apresentados alguns dados coletados sobre as principais atividades deste trabalho, bem como uma breve avaliação sobre estes, sendo a primeira seção destinada ao ambiente de testes e a segunda à exploração e seleção de *benchmarks* para Hadoop.

5.1 Ambiente de Testes

A proposta e concretização do Ambiente de Testes foi realizado anteriormente ao início da exploração das ferramentas de *benchmarking* para Hadoop, visto que tal ambiente serviria como plataforma de suporte para a realização dos testes, coleta e análise das informações.

Os resultados obtidos nesta etapa do projeto foram:

- Criação de uma imagem de sistema adaptada a partir do Debian Lenny para funcionar com o Hadoop
- Desenvolvimento do script de configuração e inicialização do Hadoop
- Disponibilização do Ambiente de Testes para Hadoop a todos os usuários do Grid'5000
- Criação de um tutorial sobre como utilizar o Ambiente Hadoop no Grid'5000, publicado na Wiki da plataforma, como pode ser visto na Figura 5.1
- Desenvolvimento de novos scripts para coleta de informações sobre o Hadoop e aplicações desenvolvidas com este *framework*

Foram realizados alguns testes relativos ao tempo de implantação do ambiente de testes nos componentes reservados para um determinado trabalho. Estes têm como objetivo verificar a viabilidade da utilização do ambiente proposto, visto que o tempo de *deploy* é

Figura 5.1: Captura de Tela do Tutorial na Wiki do Grid'5000.

considerado como tempo de utilização da plataforma. No caso do Grid'5000, esse tempo apenas é descontado do limite de horas que cada usuário tem direito diariamente, porém em infra-estruturas de Cloud-Computing esse tempo pode significar custo monetário real. Para os testes foram utilizados os clusters *bordeplage* (93 nós com 1 processador de 4 núcleos cada) e *bordereau* (51 nós com 1 processador de 2 núcleos cada), localizados na cidade de Bordeaux/FRA.

As informações obtidas nestes testes representam o tempo total de *deploy*, sendo que o ambiente de testes proposto é comparado ao tempo de três outros ambientes amplamente utilizados no Grid'5000 (*lenny-x64-base*, *lenny-x64-nfs* e *lenny-x64-big*). No total, foram realizados sete testes, que diferem quanto à quantidade de máquinas alocadas para o trabalho, sendo esta: 1, 2, 5, 10, 25, 50 e 100 máquinas. A taxa de repetição dos testes foi de cinco amostras, sendo que os resultados presentes na tabela 5.1, representam a média destas amostras.

Sabe-se que a quantidade de repetições dos testes utilizada neste trabalho é pequena para se obter uma grande taxa de confiabilidade dos dados obtidos, porém esta foi necessária visto que os usuários do Grid'5000 possuem diáriamente apenas cerca de 500 horas

computacionais a disposição. Essas horas computacionais utilizadas por cada usuário são calculadas através do somatório das horas computacionais de cada teste, que por sua vez é calculado da seguinte forma:

HorasComp = Nº de Nós x Nº de Processadores x Nº de Núcleos x Tempo do teste (h)

Para melhor exemplificar os efeitos deste cálculo, em uma reserva com 50 nós, compostos por 1 processador com 4 núcleos cada, resulta na permissão de utilização de apenas 2 horas e meia por dia de testes. A situação fica ainda pior se considerarmos testes com 100 máquinas, o que nas mesmas condições resulta na permissão de uso de apenas 1 hora e 15 minutos diários.

Tabela 5.1: Tempo Médio de *Deploy* (em segundos)

Nº de Nós	*lenny-x64-base*	*lenny-x64-nfs*	*lenny-x64-big*	*lenny-x64-hadoop*
1	256,6	282,8	311,4	307,2
2	258,8	285,8	312,2	314,4
5	264,4	294,4	316,8	325,8
10	279,1	298,6	353,7	362,2
25	311,2	332,3	363,4	395,1
50	696,5	699,7	731,9	1916,3
100	700,8	706,5	760,2	2880,6

Desde o início era esperado que o ambiente de testes proposto (*lenny-x64-hadoop*) obtivesse, em todos os casos, um tempo maior de implantação do que os ambientes *lenny-x64-base* e *lenny-x64-nfs*, visto que o mesmo foi criado a partir do *lenny-x64-nfs*, que por sua vez é baseado no *lenny-x64-base*.

Porém foi possível observar que o ambiente *lenny-x64-hadoop* é viável para casos de testes com até 25 máquinas, pois este manteve-se com o tempo de implantação muito próximo aos outros ambientes. Quando os testes exigem escalabilidade, o ambiente proposto é capaz de atender qualquer demanda, independente da quantidade de nós que o teste possui, porém a partir de 50 nós, existe um *overhead* causado pelas comunicações e transferências de arquivos necessárias à configuração do Hadoop, o qual torna o uso do ambiente proposto uma opção onerosa.

Essa sobrecarga no tempo de implantação pode ser amenizada através da utilização de ferramentas que permitam a paralelização do envio dos arquivos de configuração do

Hadoop. Um exemplo de ferramenta disponível é o Parallel SSH (CHUN, 2003), com o qual cada nó poderia requisitar os arquivos e copiá-los paralelamente.

Outro ponto importante de ser observado é o aumento excessivo do tempo de deploy em todas as imagens na passagem de 25 para 50 nós. Porém, o motivo deste não é trivial de se encontrar e seriam necessários novos testes para que se possa indicar um único motivo real para tal. Ainda assim, um fator que pode contribuir para tal aumento, é o fato dos componentes de rede dos dois *clusters* não estarem no mesmo nível hierárquico e para os testes com 25 máquinas ser possível alocar todos nós em um único cluster, enquanto no teste com 50 poder ocorrer de se utilizar os dois *clusters* em conjunto.

Além da sobrecarga no tempo de implantação, outra desvantagem desta proposta de ambiente de teste, consiste no fato de a cada nova versão do Hadoop criada, é necessário também criar uma nova imagem de sistema, a fim de ser utilizada como ambiente no Grid'5000.

Esta proposta traz diversas facilidades na configuração e inicialização do Hadoop no Grid'5000, visto que o programador deve se preocupar apenas com as aplicações que este desenvolver, enquanto o Hadoop estará disponível automaticamente. Além desta vantagem, é importante destacar o fato de não haver acréscimo na quantidade de passos a serem realizados até a implantação do ambiente em todas as máquinas reservadas para um trabalho, ou seja, segue sendo necessário apenas reservar os nós e indicar qual ambiente deseja implantar nestes componentes.

5.2 Benchmarks

Após a implementação do Ambiente de Testes ter sido concluída foi possível iniciar os testes com os *benchmarks* para o Hadoop. Estes testes foram importantes para verificar o funcionamento interno dos *benchmarks*, bem como iniciar as coletas de informações e dados, a fim de observar quais métricas eram geradas pelas ferramentas.

Os resultados obtidos nesta etapa do projeto foram:

- Exploração e apresentação de cerca de trinta e cinco ferramentas capazes de realizar *benchmarking* para Hadoop
- Seleção de quatro *benchmarks* para o projeto FTH-Grid
- Apresentação dos resultados da exploração e seleção em dois *workshops* internos

ao projeto

A partir da seleção das ferramentas GridMix, MRReliabilityTest, TestDFSIO e MapRedTest, é possível comparar duas versões distintas do Hadoop em diversos aspectos que vão desde o tempo de execução de aplicações que simulam cargas reais de trabalho até testes para verificar se o Hadoop está executando corretamente funções *map* e *reduce*.

6 CONCLUSÕES

Neste trabalho foi apresentada uma proposta e concretização de um ambiente de teste para Hadoop, bem como uma busca e seleção de *benchmarks* para Hadoop, de acordo com os requisitos propostos pelo projeto FTH-Grid, ao qual este trabalho está vinculado.

Outros importantes indicadores deste trabalho são: o tutorial sobre o ambiente, publicado na Wiki da plataforma Grid'5000; uma apresentação em um *workshop* interno ao LaSIGE/POR e uma apresentação em um *workshop* Interno do Projeto FTH-Grid, em Paris/FRA.

As contribuições científicas deste trabalho foram direcionadas aos participantes do projeto FTH-Grid quanto à seleção dos *benchmarks*, aos usuários do Hadoop quanto à revisão de ferramentas para *benchmarking* e aos usuários do Grid'5000 quanto ao ambiente de testes concretizado nesta plataforma e sua documentação.

Como trabalhos futuros, existe a possibilidade de complementar o *benchmark* MR-ReliabilityTest, tais como as métricas citadas na seção de avaliação, pois este possui um grande potencial para além do sistema de injeção de falhas existente neste, também gerar métricas importantes para a tolerância a falhas e a intrusões. Além disso, pretende-se submeter uma proposta de sessão prática sobre o Hadoop na plataforma Grid'5000 à *Grid'5000 Spring School 2010*, que ocorrerá em Lille/FRA, dentre os dias 6 a 9 de Abril de 2010.

REFERÊNCIAS

Apache Software Foundation. **Welcome to Hadoop MapReduce!** Disponível em http://hadoop.apache.org/mapreduce/. Acessado em 18/01/2010.

Apache Software Foundation. **Welcome to Apache Hadoop!** Disponível em http://hadoop.apache.org/. Acessado em 18/01/2010.

Apache Software Foundation. **Welcome to Hadoop Common!** Disponível em http://hadoop.apache.org/common/. Acessado em 18/01/2010.

Apache Software Foundation. **Welcome to Hadoop Distributed File System!** Disponível em http://hadoop.apache.org/hdfs/. Acessado em 18/01/2010.

Apache Software Foundation. **Welcome to Pig!** Disponível em http://hadoop.apache.org/pig/. Acessado em 18/01/2010.

Apache Software Foundation. **Welcome to Apache Avro!** Disponível em http://hadoop.apache.org/avro/. Acessado em 18/01/2010.

Apache Software Foundation. **Welcome to Apache ZooKeeper!** Disponível em http://hadoop.apache.org/zookeeper/. Acessado em 18/01/2010.

Apache Software Foundation. **Welcome to Hive!** Disponível em http://hadoop.apache.org/hive/. Acessado em 18/01/2010.

Apache Software Foundation. **Welcome to Chukwa!** Disponível em http://hadoop.apache.org/chukwa/. Acessado em 18/01/2010.

Apache Software Foundation. **Welcome to HBase!** Disponível em http://hadoop.apache.org/hbase/. Acessado em 18/01/2010.

Apache Software Foundation. **Applications and organizations using Hadoop**. Disponível em http://wiki.apache.org/hadoop/PoweredBy. Acessado em 18/01/2010.

BARHAM, P.; DONNELLY, A.; ISAACS, R.; MORTIER, R. Using magpie for request extraction and workload modelling. In: OSDI'04: PROCEEDINGS OF THE 6TH CONFERENCE ON SYMPOSIUM ON OPEARTING SYSTEMS DESIGN & IMPLEMENTATION, 2004, Berkeley, CA, USA. **Anais...** USENIX Association, 2004. p.18–18.

BashReduce. **BashReduce - Mapreduce Bash Script**. Disponível em http://blog.last.fm/2009/04/06/mapreduce-bash-script. Acessado em 18/01/2010.

BEROSIK, J. L. L. . G. **Building and Installing a Hadoop/MapReduce Cluster from Commodity Components**. [S.l.]: Thomson Reuters Corporation, 2009.

BINNIG, C.; KOSSMANN, D.; KRASKA, T.; LOESING, S. How is the weather tomorrow?: towards a benchmark for the cloud. In: DBTEST '09: PROCEEDINGS OF THE SECOND INTERNATIONAL WORKSHOP ON TESTING DATABASE SYSTEMS, 2009, New York, NY, USA. **Anais...** ACM, 2009. p.1–6.

British Computer Society. **BS 7925-1 Glossary of Software Testing terms**. Disponível em http://www.testingstandards.co.uk/living_glossary.htm. Acessado em 18/01/2010.

CAPIT, N. **OAR Documentation**. Montbonnot-Saint-Martin, FRA: [s.n.], 2008. 67p. Disponível em http://oar.imag.fr/docs/OAR-DOCUMENTATION.pdf. Acessado em 18/01/2010.

CAPPELLO, F.; CARON, E.; DAYDE, M.; DESPREZ, F.; JEANNOT, E.; JEGOU, Y.; LANTERI, S.; LEDUC, J.; MELAB, N.; MORNET, G.; NAMYST, R.; PRIMET, P.; RICHARD, O. Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform. In: GRID'2005 WORKSHOP, 2005, Seattle, USA. **Anais...** [S.l.: s.n.], 2005.

Carnegie Mellon University. **Fingerpointing**: problem diagnosis in distributed systems. Disponível em http://www.ece.cmu.edu/ fingerpointing/. Acessado em 18/01/2010.

CHANG, F.; DEAN, J.; GHEMAWAT, S.; HSIEH, W. C.; WALLACH, D. A.; BURROWS, M.; CHANDRA, T.; FIKES, A.; GRUBER, R. E. Bigtable: a distributed storage system for structured data. In: IN PROCEEDINGS OF THE 7TH CONFERENCE ON

USENIX SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION - VOLUME 7, 2006. **Anais...** [S.l.: s.n.], 2006. p.205–218.

CHUN, B. N. **pssh - Parallel SSH**. Disponível em http://www.theether.org/pssh/. Acessado em 18/01/2010.

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. **OSDI'04: Sixth Symposium on Operating System Design and Implementation**, San Francisco, CA, USA, 2004.

Disco. **Disco - Massive Data - Minimal Code**. Disponível em http://discoproject.org/. Acessado em 18/01/2010.

FONSECA, R.; PORTER, G.; KATZ, R. H.; SHENKER, S.; STOICA, I. X-trace: a pervasive network tracing framework. In: IN NSDI, 2007. **Anais...** [S.l.: s.n.], 2007.

GRAY, J.; NYBERG, C.; SHAH, M.; GOVINDARAJU, N. **Sort Benchmark Home Page**. Disponível em http://sortbenchmark.org/. Acessado em 18/01/2010.

Greenplum. **Driving The Future Of Data Warehousing And Analytics**. Disponível em http://www.greenplum.com/technology/mapreduce/. Acessado em 18/01/2010.

GRID5000 Funding. **Grid5000:home** - http://www.grid5000.fr/. Acessado em 18/01/2010.

GridGain. **GridGain - Cloud Development Platform**. Disponível em http://www.gridgain.com/. Acessado em 18/01/2010.

Groupement d'Intérêt Public RENATER. **Site web du GIP RENATER**. Disponível em http://www.renater.fr/. Acessado em 18/01/2010.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Architecture**: a quantitative approach. Palo Alto, CA, USA: Morgan Kaufmann, 2006.

Holumbus. **Holumbus - Sailing Your Documents**. Disponível em http://holumbus.fh-wedel.de/. Acessado em 18/01/2010.

Intel Corporation. **Excerpts from A Conversation with Gordon Moore**: moore s law. 2005.

Kaiser Associates. **Beating the Competition**: a pratical guide to benchmarking. Washington, DC, USA: Kaiser Associates, 1988.

KIYOUNG, K.; KYUNGHO, J.; HYUCK, H.; SHIN-GYU, K.; HYUNGSOO, J.; YEOM, H. MRBench: a benchmark for mapreduce framework. **ICPADS '08. 14th IEEE International Conference on Parallel and Distributed Systems, 2008.**, Melbourne, VIC, Australia, 2008.

KONWINSKI, A.; ZAHARIA, M.; KATZ, R.; STOICA, I. **Monitoring Hadoop using X-Trace**. 2008.

Laboratoire d'Informatique de Grenoble. **OAR - Resource Management System for High Performance Computing**. Disponível em http://oar.imag.fr/. Acessado em 18/01/2010.

LäMMEL, R. Google's MapReduce programming model — Revisited. **Sci. Comput. Program.**, Amsterdam, The Netherlands, The Netherlands, v.68, n.3, p.208–237, 2007.

LEDUC, J.; PEAQUIN, G. **Kadeploy**. Disponível em http://kadeploy.imag.fr/. Acessado em 18/01/2010.

LIP6 and LaSIGE. **FTH-Grid Project**: fault-tolerant hierarchical grid scheduling. Disponível em http://fth-grid.di.fc.ul.pt/. Acessado em 18/01/2010.

MapSharp. **MapSharp**. Disponível em http://mapsharp.codeplex.com/. Acessado em 18/01/2010.

MOORE, G. E. Cramming more components onto integrated circuits. **Electronics**, [S.l.], v.38, n.8, 1965.

NOLL, M. G. **Running Hadoop On Ubuntu Linux (Single-Node Cluster)**. Disponível em http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_(Single-Node_Cluster). Acessado em 18/01/2010.

NOLL, M. G. **Running Hadoop On Ubuntu Linux (Multi-Node Cluster)**. 2009.

OLSTON, C.; REED, B.; SRIVASTAVA, U.; KUMAR, R.; TOMKINS, A. Pig latin: a not-so-foreign language for data processing. In: SIGMOD '08: PROCEEDINGS OF

THE 2008 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2008, New York, NY, USA. **Anais...** ACM, 2008. p.1099–1110.

O'MALLEY, O.; MURTHY, A. C. **Winning a 60 Second Dash with a Yellow Elephant**. Disponível em http://sortbenchmark.org/Yahoo2009.pdf. Acessado em 18/01/2010.

PAN, X. **The Blind Men and the Elephant**: piecing together hadoop for diagnosis. 2009. Tese (Doutorado) — Carnegie Mellon University.

PAN, X.; TAN, J.; KAVULYA, S.; GANDHI, R.; NARASIMHAN, P. **Ganesha**: black-box fault diagnosis for mapreduce systems. [S.l.]: Carnegie Mellon University, 2008.

PAN, X.; TAN, J.; KAVULYA, S.; GANDHI, R.; NARASIMHAN, P. Blind Men and the Elephant (BLIMEy): piecing together hadoop for diagnosis. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING (ISSRE), 2009, Mysore, India. **Anais...** [S.l.: s.n.], 2009.

PAN, X.; TAN, J.; KAVULYA, S.; GANDHI, R.; NARASIMHAN, P. Ganesha: black-box fault diagnosis for mapreduce systems. In: WORKSHOP ON HOT TOPICS IN MEASUREMENT AND MODELING OF COMPUTER SYSTEMS (HOTMETRICS 2009), 2009, Seattle, WA, US. **Anais...** [S.l.: s.n.], 2009.

Phoenix. **The Phoenix System for MapReduce Programming**. Disponível em http://mapreduce.stanford.edu/. Acessado em 18/01/2010.

RANGER, C.; RAGHURAMAN, R.; PENMETSA, A.; BRADSKI, G.; KOZYRAKIS, C. Evaluating MapReduce for Multi-core and Multiprocessor Systems. In: HPCA '07: PROCEEDINGS OF THE 2007 IEEE 13TH INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE COMPUTER ARCHITECTURE, 2007, Washington, DC, USA. **Anais...** IEEE Computer Society, 2007. p.13–24.

SANDHOLM, T.; LAI, K. MapReduce optimization using regulated dynamic prioritization. In: SIGMETRICS '09: PROCEEDINGS OF THE ELEVENTH INTERNATIONAL JOINT CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 2009, New York, NY, USA. **Anais...** ACM, 2009. p.299–310.

SCHMIDT, S. **The Holumbus Framework**: distributed computing with mapreduce in haskell. 2009. Dissertação (Mestrado) — Fachhochschule Wedel.

Skynet. **Skynet - A Ruby MapReduce Framework**. Disponível em http://skynet.rubyforge.org/. Acessado em 18/01/2010.

TAN, J.; NARASIMHAN, P. **RAMS and BlackSheep**: inferring white-box application behavior using black-box techniques. 2008. Tese (Doutorado) — Carnegie Mellon University.

TAN, J.; PAN, X.; KAVULYA, S.; GANDHI, R.; NARASIMHAN, P. SALSA: analyzing logs as state machines. In: WASL, 2008. **Anais...** USENIX Association, 2008.

TAN, J.; PAN, X.; KAVULYA, S.; GANDHI, R.; NARASIMHAN, P. Mochi: visual log-analysis based tools for debugging hadoop. In: USENIX WORKSHOP ON HOT TOPICS IN CLOUD COMPUTING (HOTCLOUD), 2009, San Diego, CA, US. **Anais...** [S.l.: s.n.], 2009.

Transaction Processing Performance Council. **TPC BenchmarkTM H Standard Specification Revision 2.9.0**. San Francisco, CA, USA: [s.n.], 2009. 135p.

VENNER, J. **Pro Hadoop**. Berkeley, CA, USA: Apress, 2009.

WEICKER, R. **Benchmarking**. Berlin, GER: Springer Berlin / Heidelberg, 2002.

WHITE, T. **Hadoop**: the definitive guide. Sebastopol, CA, USA: O'reilly and Yahoo! Press, 2009.

WOO, S. C.; OHARA, M.; TORRIE, E.; SINGH, J. P.; GUPTA, A. The SPLASH-2 programs: characterization and methodological considerations. In: ISCA '95: PROCEEDINGS OF THE 22ND ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 1995, New York, NY, USA. **Anais...** ACM, 1995. p.24–36.

ZELKOWITZ, M. Models and Metrics for Energy-Efficient Computing. In: ADVANCES IN COMPUTERS: COMPUTER PERFORMANCE ISSUES, 2009. **Anais...** Academic Press, 2009. p.164–165.

ZHU, S.; XIAO, Z.; CHEN, H.; CHEN, R.; ZHANG, W.; ZANG, B. Evaluating SPLASH-2 Applications Using MapReduce. In: APPT '09: PROCEEDINGS OF THE 8TH INTERNATIONAL SYMPOSIUM ON ADVANCED PARALLEL PROCESSING TECHNOLOGIES, 2009, Berlin, Heidelberg. **Anais...** Springer-Verlag, 2009. p.452–464.

APÊNDICE A GUIA DE REFERÊNCIA PARA O AMBIENTE DE TESTES

Este capítulo de apêndice é apresentado como um guia de referêcia de passos e comandos utilizados para a criação do Ambiente de Testes, bem como para a configuração e utilização do mesmo. O intuito com este capítulo, é registrar diretamente os comandos a serem utilizados para reprodução da criação, configuração e utilização do ambiente.

A.1 Instalação e criação do Ambiente de Testes

Nesta primeira seção são apresentados os passos realizados para a criação do ambiente de testes no Grid'5000, desde a reserva do nó onde se criará a imagem, até o passo da utilização do script que gera a imagem compactada.

Listing A.1: Passos para a criação da imagem de sistema do ambiente de testes

```
1) oarsub -I -t deploy -l nodes=1,walltime=03:00:00
2) kadeploy3 -e lenny-x64-nfs -f \$OAR\_FILE\_NODES
3) ssh root@node.site.grid5000.fr
4) apt-get update
5) apt-get upgrade
6) apt-get install sun-java6-jdk
7) mkdir /opt/hadoop/
8) wget http://link-to-hadoop-release/hadoop-version.tar.gz
9) tar -xf  hadoop-version.tar.gz
10) Modificar a variável $JAVA_HOME para /usr/lib/jvm/java
   -6-sun, no arquivo hadoop-env.xml da pasta conf
11) tgz-g5k user@frontend:~/path/image.tgz
```

A.2 Configuração do Ambiente de Testes

Nesta seção é apresentado o script de configuração do ambiente de testes, para evitar que a cada execução do Hadoop seja necessário o usuário configurar manualmente os itens neste apresentado.

Listing A.2: Script de configuração e inicialização do Hadoop no Grid'5000

```
#!/bin/bash
USER_LOGIN_ID=`id -u`
USER_LOGIN_NAME=`id -un`
USER_GROUP_ID=`id -g`
USER_GROUP_NAME=`id -gn`
DATE=`date +%s`
HADOOP_PATH='/opt/hadoop'
HADOOP_CONF_PATH=$HADOOP_PATH/'conf'
TMP_DIR=/tmp/$DATE'-'$$'-hadoop'
NUM_HOSTS=`cat $OAR_FILE_NODES | uniq | wc -l`
SLAVES=`cat $OAR_FILE_NODES | uniq`
MASTERS=`head -n 1 $OAR_FILE_NODES`
mkdir $TMP_DIR
cat > $TMP_DIR/masters << FIM
$MASTERS
FIM
cat > $TMP_DIR/slaves << FIM
$SLAVES
FIM
#Create/Modify "hdfs-site.xml" file
cat > $TMP_DIR/hdfs-site.xml << FIM
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
    <description>Default block replication.
      The actual number of replications can be specified
         when the file is created.
      The default is used if replication is not specified
         in create time.
    </description>
  </property>
</configuration>
FIM
#Create/Modify "core-site.xml" file
cat > $TMP_DIR/core-site.xml << FIM
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/tmp/hadoop-\${user.name}</value>
  </property>
  <property>
    <name>fs.default.name</name>
```

```
    <value>hdfs://$MASTERS:54310</value>
  </property>
</configuration>
FIM
#Create/Modify "mapred-site.xml" file
cat  > $TMP_DIR/mapred-site.xml << FIM
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>hdfs://$MASTERS:54311</value>
    <description>The host and port that the MapReduce job
       tracker runs
      at.  If "local", then jobs are run in-process as a
         single map
      and reduce task.
    </description>
  </property>
</configuration>
FIM
if [ -d /home/$USER_GROUP_NAME/$USER_LOGIN_NAME/.ssh ]
   then
      cd /home/$USER_GROUP_NAME/$USER_LOGIN_NAME/.ssh/
   else
      mkdir /home/$USER_GROUP_NAME/$USER_LOGIN_NAME/.ssh
      cd /home/$USER_GROUP_NAME/$USER_LOGIN_NAME/.ssh/
fi
eval `/usr/bin/ssh-agent -s`
if [ -f id_rsa -a -f id_rsa.pub ]
   then
      echo "/**"
      echo " * The user has a RSA key pair."
      echo " */"
      ssh-add id_rsa
      echo `cat id_rsa.pub` >> authorized_keys
      echo "StrictHostKeyChecking no" >> config
   else
      if [ -f id_dsa -a -f id_dsa.pub ]
         then
            echo "/**"
            echo " * The user has a DSA key pair."
            echo " */"
            ssh-add id_dsa
            echo `cat id_dsa.pub` >> authorized_keys
            echo "StrictHostKeyChecking no" >> config
         else
            echo "/**"
```

```
            echo " * Creating a RSA key pair."
            echo " */"
            ssh-keygen -t rsa -f /home/$USER_GROUP_NAME/
               $USER_LOGIN_NAME/.ssh/id_rsa -P ""
            ssh-add id_rsa
            echo `cat id_rsa.pub` >> authorized_keys
            echo "StrictHostKeyChecking no" >> config
      fi
fi
for SLAVE in $SLAVES
do
   ssh root@$SLAVE chown -R $USER_LOGIN_NAME:
      $USER_GROUP_NAME $HADOOP_PATH
   scp $TMP_DIR/* $USER_LOGIN_NAME@$SLAVE:$HADOOP_CONF_PATH
done
rm -rf $TMP_DIR/
ssh $USER_LOGIN_NAME@$MASTERS $HADOOP_PATH/bin/stop-all.sh
ssh $USER_LOGIN_NAME@$MASTERS $HADOOP_PATH/bin/hadoop
   namenode -format
ssh $USER_LOGIN_NAME@$MASTERS $HADOOP_PATH/bin/start-dfs.sh
ssh $USER_LOGIN_NAME@$MASTERS $HADOOP_PATH/bin/start-mapred
   .sh
echo 'Masternode: '$MASTERS
```

A.3 Utilização do Ambiente de Testes

Nesta seção serão apresentados os passos para se utilizar o ambiente de testes no Grid'5000, de tal forma que ao final da execução destes passos, se obtenha o Hadoop sendo executado em todos os nós reservados para um determinado trabalho.

Listing A.3: Passos para a utilização da imagem de sistema do ambiente de testes

```
1) ssh user@frontend.site.grid5000.fr
2) oarsub -I -t deploy -l nodes=50,walltime=12:00:00
3) kadeploy3 -a ~vvielmocogo/hadoop/0.20.1/lenny-x64-nfs-
   hadoop.dsc3 -f \$OAR\_FILE\_NODES -k ~/.ssh/id\_rsa.pub
    -s ~vvielmocogo/hadoop/0.20.1/config.sh
```