UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

FLAVIO ALLES RODRIGUES

Caracterização do Consumo Energético do Hadoop MapReduce

Trabalho de Conclusão de Curso.

Prof. Dr. Claudio Fernando Resin Geyer
Orientador

Prof. Msc. Pedro de Botelho Marcos
Co-orientador

Porto Alegre, dezembro de 2013

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Rodrigues, Flavio Alles

Caracterização do Consumo Energético do Hadoop MapReduce / Flavio Alles Rodrigues. – Porto Alegre: II da UFRGS, 2013.

59 f.: il.

Trabalho de Conclusão – Universidade Federal do Rio Grande do Sul. Curso de Engenharia de Computação, Porto Alegre, BR–RS, 2013. Orientador: Claudio Fernando Resin Geyer; Co-orientador: Pedro de Botelho Marcos.

1. Computação Intensiva em Dados. 2. MapReduce. 3. Green Computing. 4. Consumo de Energia. I. Geyer, Claudio Fernando Resin. II. Marcos, Pedro de Botelho. III. Título.

“Nunca ninguém se torna mestre em um domínio em que não conheceu a impotência, e, quem aceita esta ideia, saberá também que tal impotência não se encontra nem no começo nem antes do esforço empreendido, mas sim no seu centro.”

— WALTER BENJAMIM

AGRADECIMENTOS

A todos que me querem bem. Muito obrigado.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CERN	European Organization for Nuclear Research
IPCC	Intergovernmental Panel on Climate Change
ONU	Organização das Nações Unidas
HDFS	Hadoop Distributed File System
FIFO	First in, First out
HFS	Hadoop Fair Scheduler
CPU	Central Processing Unit
TDP	Thermal Design Power
AIS	All-in Strategy
CS	Covering Set
DVFS	Dynamic Voltage Frequency Scaling
RAM	Random Access Memory

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O crescimento exponencial do poder computacional, das fontes de geração de dados e da capacidade de comunicação em tecnologias recentes criou uma nova categoria de aplicações computacionais: aplicações intensivas em dados. O aumento dos conjuntos de dados é verificado em diversas áreas do conhecimento e atuação humanas. Deste contexto surge a necessidade do desenvolvimento de *frameworks* capazes de armazenar e processar dados em larga escala em um tempo aceitável. MapReduce, desenvolvido pelo Google, é um modelo de programação paralelo com uma implementação associada criado para o processamento de grandes quantidades de dados. O usuário deste *framework* precisa definir somente duas funções (*map* e *reduce*) e o *runtime* se encarrega de lidar de forma transparente ao programador com questões advindas da paralelização da computação, como a distribuição dos dados, escalonamento de tarefas, comunicação entre processos e tolerância a falhas. Porém, esta demanda pelo processamento de quantidades crescentes de dados tem como consequência uma demanda maior por recursos computacionais para processar uma mesma aplicação. O grande problema que esta demanda crescente por recursos computacionais gera é um - também - crescente consumo energético. Esta situação é crítica por duas razões - uma de motivação financeira e outra de motivação ambiental. Por estas razões, é imperativo que sistemas computacionais sejam projetados para serem cientes do consumo energético. A partir destas considerações, este trabalho tem como objetivo caracterizar o consumo energético de um sistema de processamento de grandes quantidades de dados. Hadoop - implementação de código aberto do modelo de programação MapReduce - é o sistema escolhido para a caracterização. A caracterização do consumo de energia deste sistema é acompanhada de considerações sobre o desempenho do *framework* para que o consumo de energia não seja considerado de maneira isolada e, sim, sob uma perspectiva mais ampla.

Palavras-chave: Computação Intensiva em Dados, MapReduce, Green Computing, Consumo de Energia.

Characterization of Hadoop's MapReduce Energetic Consumption

ABSTRACT

The exponential growth of computing power, sources of data generation and communication capabilities in recent technologies created a new category of computer applications: data-intensive applications. The increase of data sets can be found in inumerous areas of human activities and knowledge. In such a context, arises a necessity of developing frameworks capable of storing and processing large-scale data in an acceptable time. MapReduce, developed by Google, is a parallel programming model with an associated implementation created for the the processing of vasts amounts of data. The user of this framework has only to define two functions (map and reduce) and the runtime deals with issues that arise due to the parallelization of the computation, such as data distribution, task scheduling, process communication and fault tolerance. However, this demand for the processing of growing amounts of data results in a higher demand for computing resources to process a single application. The major problem that this growing demand for computing resources generates is an increasing energy consumption. This is critical for two reasons - one of financial motivation and the other of environmental motivation. For these reasons, it is imperative that computer systems are designed to be aware of energy consumption. Motivated by these considerations, this study aims to characterize the power consumption of a system for processing large amounts of data. Hadoop - an open source implementation of the MapReduce programming model - is the chosen system for the proposed characterization. The energy consumption characterization of this system is accompanied by performance considerations of the framework so that energy consumption is not considered as an isolated issue, but in a broader perspective.

Keywords: Data-Intensive Computing, MapReduce, Green Computing, Energy Consumption.

1 INTRODUÇÃO

Este capítulo inicial tem como objetivo contextualizar o trabalho apresentado através deste texto - a avaliação do consumo de energia de um framework para o processamento de grandes quantidades de dados. No primeiro momento são apresentadas as motivações que originaram o estudo. Em seguida, os objetivos e a utilidade do trabalho são discutidas. Ao final do capítulo, a estrutura do texto é detalhada.

1.1 Motivação

1.1.1 Computação Intensiva em Dados

O crescimento exponencial do poder computacional, das fontes de geração de dados e da capacidade de comunicação em tecnologias recentes criou uma nova categoria de aplicações computacionais: aplicações intensivas em dados (GORTON et al., 2008).

Um paralelo pode ser traçado para definir tal categoria de aplicações comparando-a a categoria de aplicações intensivas em computação - uma classificação menos recente e mais comum na literatura científica. Aplicações intensivas em computação são aquelas onde o limitador da velocidade de processamento das mesmas é o poder computacional disponível para tais tarefas. Em constraste, aplicações intensivas em dados são aquelas onde o fator que limita a velocidade de execução das mesmas é a capacidade de gerenciar, analisar e disponibilizar quantidades massivas de dados (GORTON et al., 2008).

O aumento dos conjuntos de dados é verificado em diversas áreas do conhecimento e atuação humanas. Este fenômeno pode ser explicado por três fatores predominantes: diminuição do custo dos dispositivos de armazenamento; aumento do uso de sensores digitais em campos como astronomia, física, química, biologia e ciência climática; e a migração de atividades ecônomicas e sociais para meios eletrônicos, tais como a Internet.

Alguns exemplos de aplicações intensivas em dados ilustram a ubiquidade e relevância desta categoria de problemas, bem como as razões para o crescimento nos conjuntos de dados armazenados eletronicamente - citadas acima:

- O sistema de geração e distribuição de energia elétrica Norte Americano gera aproximadamente 15 TB de dados anualmente (KOUZES et al., 2009);

- Uma nova geração de modelos climáticos que têm uma resolução maior quando comparada aos modelos utilizados anteriormente, produz uma quantidade de dados 1000 vezes maior (8 PB em comparação aos 8 TB gerados pelo modelo antigo) para uma área simulada 30 vezes menor (3 km contra 100 km simulados anteriormente) durante o mesmo período de tempo (KOUZES et al., 2009);

- Avanços teóricos em biologia, mais precisamente em genética, produzem conjunto de dados massivos em experimentos realizados com equipamentos de sequenciamento genético (RECKZIEGEL FILHO, 2013);

- O sistema de indexação de páginas *web* do Google processa aproximadamente 20 TB em documentos a cada iteração (DEAN; GHEMAWAT, 2008);

- Facebook armazena aproximadamente 700 TB em bancos de dados relacionais, bem como 1 PB em fotografias (THUSOO et al., 2009);

- Experimentos em física de partículas em um dos quatro equipamentos que compõem o acelerador de partículas Large Hadron Collider no CERN (Organização Européia para a Pesquisa Nuclear) geram 2 PB/s em cada experimento realizado (KOUZES et al., 2009).

Então, deste contexto, surge a necessidade do desenvolvimento de *frameworks* capazes de armazenar e processar dados em larga escala em um tempo aceitável - i.e. modelos de programação para aplicações intensivas em dados.

1.1.2 MapReduce

MapReduce (DEAN; GHEMAWAT, 2008), desenvolvido pelo Google, é um modelo de programação paralelo com uma implementação associada criado para o processamento de grandes quantidades de dados. Este se baseia em duas primitivas comuns em linguagens de programação funcional: *map* e *reduce*. Desta forma, o programador precisa definir somente estas duas funções e o ambiente de execução se encarrega de lidar de forma transparente ao programador com questões advindas da paralelização da computação, como a distribuição dos dados, escalonamento de tarefas, comunicação entre processos e tolerância a falhas. O MapReduce foi desenvolvido para ser executado em *clusters* homogêneos de grande escala composto por máquinas comuns. A implementação deste modelo e o sistema de execução do mesmo não foram disponibilizados pelo Google.

O MapReduce pode ser considerado um trabalho seminal na área de computação intensiva em dados por ter inspirado o desenvolvimento de diversas plataformas para processamento de quantidades massivas de dados. Hadoop (APACHE, 2013) é um projeto de código aberto desenvolvido com base no artigo que apresentou o modelo MapReduce de computação - e, assim como o sistema original, foi desenvolvido para executar em *clusters* homogêneos de máquinas comuns. O modelo de programação MapReduce também tem implementações que executam sobre sistemas *multicore* com memória compartilhada (YOO; ROMANO; KOZYRAKIS, 2009), placas gráficas (HE et al., 2008) e ambientes de computação em nuvem (LIU; ORBAN, 2011).

Entre todas as implementações MapReduce, Hadoop é a que tem o maior número de usuários, desenvolvedores e estudos relacionados (WHITE, 2012).

1.1.3 *Green Computing*

A necessidade de processar uma quantidade crescente de dados tem como consequência uma demanda maior por recursos computacionais para processar uma mesma aplicação - uma computação típica MapReduce processa muitos *terabytes* de dados em milhares de máquinas (DEAN; GHEMAWAT, 2008). Desta forma, ambientes computacionais que envolvem milhares de máquinas comuns (*commodity machines*) são cada

vez mais comuns entre grandes serviços de Internet - e.g. Google, Facebook e Twitter. Este tipo de infraestrutura computacional é conhecida como *Warehouse-sized Computer* (BARROSO; HÖLZLE, 2009).

O grande problema que tal demanda crescente por recursos computacionais gera é um - também - crescente consumo energético. Esta situação é crítica por duas razões - uma de motivação financeira e outra de motivação ambiental.

Tendências de longo prazo indicam que o aumento de desempenho de sistemas computacionais tem como consequência maior consumo de energia dos mesmos. Então, como o custo de equipamentos computacionais tem se mantido estável nos últimos anos, o custo para energizar estas máquinas tem se tornado cada vez mais importante. Se tal tendência se manter, o custo da energia consumida por um servidor durante sua vida útil pode ultrapassar o custo do equipamento (BARROSO, 2005).

Outro ponto fundamental é a degradação ambiental - e as mudanças climáticas geradas por tal fenômeno. O quarto relatório do Painel Intergovernamental sobre Mudanças Climáticas (IPCC) (IPCC, 2007) - corpo científico estabelecido pela Organização das Nações Unidas (ONU) para prover análises sobre as informações científicas correntes com relação ao risco das mudanças climáticas causadas pelo ser humano -, publicado em 2007, estabelece que "grande parte do aumento médio da temperatura global dos últimos 50 anos é muito provavelmente (probabilidade maior do que 90%) causado devido a atividades humanas". O quinto relatório (IPCC, 2013) do IPCC, a ser completado no próximo ano, afirma que "o aquecimento do clima terrestre é inequívoco e que, desde 1950, muitas das alterações climáticas observadas não tem precedentes na história terrestre"e, também, que "é extremamente provável (95-100% de probabilidade) que a influência humana foi a causa dominante do aquecimento global entre 1951 e 2010".

A queima de combustíveis fósseis é um dos fatores chave para as mudanças climáticas descritas acima (IPCC, 2007). E, segundo a *United States Energy Information Administration*, cerca de 68% da eletricidade gerada nos Estados Unidos em 2012, foi gerada a partir de combustíveis fosséis (EIA, 2013). No ano 2006, os aproximadamente 6000 *data centers* localizados nos Estados Unidos foram responsáveis pelo consumo de aproximadamente 1.5% da eletricidade do país - com uma tendência de crescimento neste consumo de 12% ao ano (KURP, 2008).

Por estas razões, é imperativo que sistemas computacionais (*hardware* e *software*) sejam projetados para serem cientes do consumo energético - ou seja, é fundamental que sistemas computacionais sejam eficientes energeticamente.

1.2 Objetivos

Tendo como base os aspectos destacados anteriormente, este trabalho tem como objetivo caracterizar o consumo energético de um sistema de processamento de grandes quantidades de dados. Hadoop - implementação de código aberto do modelo de programação MapReduce - é o sistema escolhido para a caracterização. A caracterização será acompanhada de considerações sobre o desempenho do *framework* para que o consumo de energia não seja considerado de maneira isolada e, sim, sob uma perspectiva mais ampla.

Espera-se que os resultados expostos neste estudo informe os usuários do Hadoop para que o utilizem de maneira mais eficiente do ponto de vista energético, bem como informe os desenvolvedores do *framework* na tarefa de tornar o sistema ciente de seu consumo de energia.

1.3 Organização

O restante do texto está estruturado da seguinte forma. O segundo capítulo apresenta conceitos necessários para a compreensão do trabalho. O terceiro capítulo aborda os trabalhos relacionados a este estudo. No capítulo seguinte, a metodologia utilizada para atingir o objetivo deste trabalho é descrita. Os experimentos e seus resultados são apresentados no quinto capítulo. Por fim, o último capítulo contém as conclusões do trabalho.

2 CONCEITOS

Este capítulo apresenta os conceitos fundamentais para a compressão deste estudo. Na primeira seção, o modelo de programação MapReduce é detalhado em seus aspectos mais importantes - interface de programação, arquitetura do sistema, metodologia de escalonamento e mecanismo de tolerância a falhas. Na segunda parte do capítulo, diferentes abordagens para mensuração do consumo energético são discutidas, apresentando seus pontos positivos e negativos.

2.1 MapReduce

O MapReduce é um modelo de programação criado pelo Google para o processamento de grandes quantidades de dados de maneira paralela e distribuída. Motivado pelas necessidades computacionais da empresa e inspirado pelas primitivas map e reduce - comuns em linguagens de programação funcional como Lisp - Google propôs o modelo MapReduce com o objetivo principal de libertar o programador do tratamento de situações comuns a sistemas distribuídos, possibilitando que este possa expressar computações paralelas e distribuídas de maneira simples e rápida. Desta forma, o usuário do modelo de programação MapReduce deve prover ao sistema de execução somente duas funções - *map* e *reduce* - e o sistema MapReduce automatiza as tarefas relacionadas à distribuição e paralelização da computação (distribuição de dados, escalonamento de tarefas, comunicação entre nós e tolerância a falhas) (DEAN; GHEMAWAT, 2008).

O modelo de programação é genérico, permitindo que problemas em diversas áreas possam ser resolvidos com o mesmo. O uso do MapReduce ocorre em áreas como processamento de grafos, tradução estatística, aprendizado de máquina e processamento de texto (DEAN; GHEMAWAT, 2010). Outros exemplos de uso do MapReduce podem ser encontrados em áreas como metereologia (XUE; PAN, 2012), biologia (RECKZIEGEL FILHO, 2013) e finanças (ZHANG et al., 2011).

2.1.1 Visão Geral

O sistema MapReduce tem uma arquitetura mestre/escravo. O nó mestre tem como responsabilidade coordenar a execução da aplicação MapReduce - isto é, realizar o escalonamento das tarefas e detectar falhas. O conjunto de nós escravo, por sua vez, é responsável por executar as tarefas *map* e *reduce* nas quais a aplicação é divida pelo mestre.

O sistema de execução MapReduce é composto, também, por um sistema de arquivos distribuído (GHEMAWAT; GOBIOFF; LEUNG, 2003) [Ghemawat et al]. Este sistema de arquivos também tem uma arquitetura mestre/escravo. O mestre (NameNode) é responsá-

vel por armazenar metadados sobre as informações contidas nos arquivos. Os nós escravo (DataNodes) são os nós que efetivamente armazenam os dados contidos no sistema de arquivos. As mesmas máquinas utilizadas pelo sistema de execução MapReduce como nós escravo podem ser utilizadas como DataNodes pelo sistema de arquivos distribuído com o objetivo de reduzir a transferência de dados e, assim, aumentar o desempenho do sistema como um todo.

2.1.2 Modelo de Programação

Uma computação MapReduce tem como entrada um conjunto de tuplas chave-valor e como saída um conjunto de tuplas chave-valor. O programador deve expressar a computação através de duas funções: *map* e *reduce*.

A função *map* tem como entrada uma tupla chave-valor e como saída valores intermediários expressados, também, como tuplas chave-valor. O sistema de execução MapReduce agrupa todas as tuplas que tem uma mesma chave.

A função *reduce* tem como entrada um grupo de tuplas chave-valor e, então, processa este grupo usualmente produzindo um conjunto menor de valores (conforme a computação determinada pelo usuário).

A inicialização de uma computação MapReduce se dá através do nó mestre. O primeiro passo no fluxo de execução é o repartimento e distribuição dos dados no sistema de arquivos distribuído. Os blocos de dados nos quais a entrada é dividida têm o mesmo tamanho e cada um destes blocos de dados é executado por uma tarefa *map* diferente. A execução das tarefas *map* onde os dados de entrada são mapeados para tuplas chave-valor (conforme determinado pela função *map*) é o próximo passo. Em seguida, os dados intermediários (saídas das tarefas *map*) são transferidos entre os nós escravo para a execução pelas tarefas *reduce*. Esta fase de tranferência é conhecida como *shuffle*. Por fim, o último passo consiste na execução das tarefas *reduce* - onde as tuplas que contém a mesma chave são processadas (conforme determinado pela função *reduce*). A Figura 2.1 (MARCOS, 2013) ilustra o fluxo de execução MapReduce.

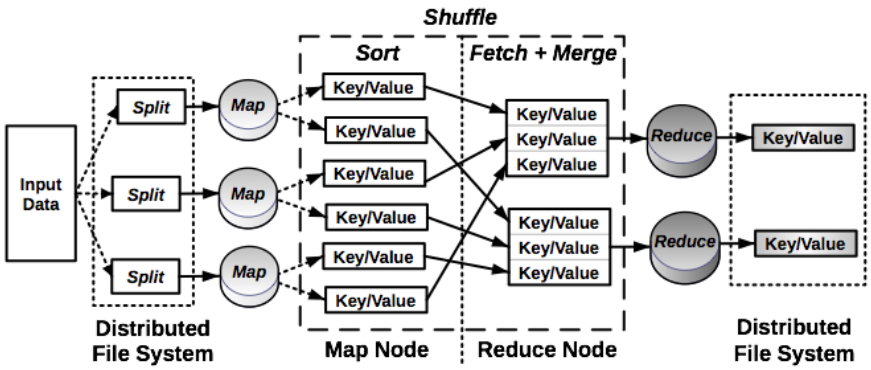

Figura 2.1: Fluxo de execução MapReduce

Alguns comentários são necessários para o melhor entendimento deste fluxo de execução. Ao contrário do que se pode inferir, os nós mestre (no MapReduce e no sistema de arquivos distribuído) não são gargalos na comunicação de dados. Isto ocorre pois em ambos os sistemas a comunicação entre mestre/escravo ocorre somente para o escalonamento de tarefas (MapReduce) e para a troca de informações a cerca da localização de blocos de dados (sistema de arquivos). A transferência de dados no início da execução de tarefas *map* ou entre tarefas *map* e *reduce* ocorre diretamente entre os nós escravo.

A fase *reduce* somente é executada após o término da fase *map*. Ainda, diferentemente do que ocorre na fase *map* - onde o número de tarefas é determinado pelo tamanho da entrada da aplicação - na fase *reduce* o número de tarefas é definido pelo usuário do sistema. Assim cada tupla intermediária é mapeada para uma tarefa *reduce* através de uma tabela *hash*. Isto significa que uma tarefa *reduce* pode processar mais do que uma chave. Por fim, os dados intermediários (saídas da fase *map*, entradas da fase *reduce*) são armazenados localmente, e não no sistema de arquivos distribuídos.

Como mencionado anteriormente, existe um estágio intermediário na computação MapReduce, chamado *shuffle*. Neste estágio o objetivo principal é a transferência de dados entre os nós escravos. Apesar da aparente simplicidade deste estágio, existem detalhes importantes durante a execução do *shuffle* que merecem atenção.

Esta transferência de dados pode ser dividida em duas fases: *sort* e *merge*. Antes do envio dos dados aos nós que executarão a função *reduce*, os nós que executaram o *map* realizam um ordenamento (*sort*) de suas tuplas chave-valor. O objetivo deste ordenamento é agrupar, em cada nó escravo, os dados que devem ser enviados para cada nó onde o *reduce* vai ser executado. Assim que o ordenamento é realizado, a transferência propriamente dita ocorre: os nós responsáveis pela execução das tarefas *reduce* copiam os dados de seu interesse. Finalmente, depois de receber os dados intermediários - e antes de iniciar o processamento - os nós *reduce* executam uma junção (*merge*) dos dados copiados de diversos nós.

A Figura 2.2 apresenta um exemplo de uma aplicação MapReduce para a contagem de palavras em documentos de texto. Na aplicação exemplo, a função *map* recebe como entrada um par chave-valor. A chave representa o nome do documento de texto e o valor os conteúdos deste documento. Primeiramente, o conteúdo é separado em palavras. Depois, para cada palavra um par chave-valor é emitido - onde a chave é a palavra e o valor o número inteiro 1. Os dados intermediários são agrupados por chave e repassados às tarefas *reduce*. A função *reduce* irá processar todos os valores que tem a mesma chave. O processamento consiste no somatório de todos os valores associados à chave de entrada. Ao final de sua execução, a função irá produzir uma tupla chave-valor que representa a contagem total da palavra chave nos dados de entrada.

```
def map(key, value):
    """Emits each word found on value together with a 1 count"""
    # key: document name
    # value: document contents
    wordList = value.split()
    for word in wordList:
        emit(word, 1)

def reduce(key, values):
    """Sums together all counts for a particular word (key)"""
    # key: a word
    # values: list of counts for key
    wordCount = 0
    for v in values:
        wordCount += 1
    emit(key, wordCount)
```

Figura 2.2: Pseudo-código para uma aplicação de contagem de palavras

2.1.3 Escalonamento MapReduce

O nó mestre, responsável pelo escalonamento de tarefas do MapReduce, ao receber a informação de que existe um nó escravo livre para execução, decide qual tarefa *map* executar com base na localizacão dos dados de cada tarefa. Então, no primeiro momento, o escalonador tenta escolher para a execução uma tarefa que tem o bloco de dados que deve ser processado armazenado na máquina livre. Se isto não é possível, o escalonador tenta escolher para a execução uma tarefa *map* que possui seus dados de entrada próximos à máquina livre (e.g. armazenados em uma máquina escrava que se encontra conectada ao mesmo switch onde a máquina livre está conectada).

Este mecanismo de escalonamento, baseado na localidade dos dados a processar, busca diminuir a comunicação via rede entre nós, pois tal recurso - banda de rede - é escasso em um ambiente distribuído com inúmeras máquinas (DEAN; GHEMAWAT, 2008). Como os dados de entrada das tarefas *reduce* estão, potencialmente, espalhados pelas máquinas onde as tarefas *map* executaram, o escalonamento de tarefas *reduce* não é feito com base neste raciocínio de localidade dos dados de entrada.

2.1.4 Tolerância a Falhas

MapReduce é um sistema para processamento de grandes quantidades de dados em um grande número de máquinas que permite expressar uma computação de maneira simples, escondendo os detalhes da paralelização e do gerenciamento da computação do programador. Entre estes detalhes está a tolerância a falhas.

O *framework* MapReduce possui mecanismo de tolerância a falhas dos nós escravo. Porém, MapReduce não tolera falhas do nó mestre. Caso haja falha nesta máquina, a computação é abortada.

O mecanismo de tolerância a falhas nos nós escravo funciona da seguinte forma. Periodicamente, o mestre envia mensagens *echo request* aos nós escravo. Caso não haja resposta em um certo período de tempo, o mestre remove a máquina da computação e re-escalona as tarefas que estavam em execução na máquina faltosa, bem como quaisquer tarefa *map* que já tenham sido executadas neste nó. As tarefas *map* completas precisam ser re-executas devido ao fato de que sua saída é armazenada no sistema de arquivos local. Em constraste, as tarefas *reduce* já executadas não precisam ser executadas em outra máquina pois suas saídas são armazenadas no sistema de arquivos distribuído.

2.1.5 Tarefas Especulativas

Tarefas especulativas consistem em uma otimização do sistema MapReduce que tem o objetivo de evitar que nós escravos mais lentos (uma máquina que está com algum problema de *hardware*, por exemplo) tenham impacto negativo sobre o desempenho do sistema. Portanto, quando o mestre percebe que uma tarefa está com progresso abaixo das outras, uma segunda instância da tarefa é escalonada em outro nó escravo. Assim que uma das duas tarefas terminar, o escalonador abortará aquela que ainda está processando. A partir da criação deste mecanismo de execução especulativa, o desempenho de aplicações MapReduce melhorou significativamente. No algoritmo *sort*, por exemplo, o tempo de execução diminui aproximadamente 44% (DEAN; GHEMAWAT, 2008).

2.1.6 Hadoop

Gerenciado pela Apache Software Foundation e com suporte de grandes companhias - como Facebook, LinkedIn, Microsoft e Yahoo! - Hadoop (APACHE, 2013) é uma im-

plementação de código aberto do modelo de programação MapReduce, sendo a mais popular entre as diferentes opções (WHITE, 2012). Hadoop é fortemente inspirado pelo artigo que apresentou o modelo (DEAN; GHEMAWAT, 2008) - inclusive tendo sido projetado para execução sobre o mesmo tipo de ambiente computacional (i.e. *clusters* de máquinas homogêneas). A arquitetura do sistema é idêntica ao original, tendo como base um modelo de execução mestre/escravo. Assim como MapReduce desenvolvido pelo Google, Hadoop também determina que o usuário do sistema deve programar somente duas funções - *map* e *reduce* - e o sistema de execução automaticamente paraleliza a computação. MapReduce original foi desenvolvido em C++, enquanto que Hadoop foi escrito em Java.

Hadoop também utiliza um sistema de arquivos distribuídos, chamado Hadoop Distributed File System (HDFS). HDFS foi projetado conforme a descrição do Google File System (GHEMAWAT; GOBIOFF; LEUNG, 2003) - sistema de arquivos distríbuido do MapReduce - e por esta razão é equivalente a este - suportando as mesmas funcionalidades e seguindo a mesma arquitetura de armazenamento.

O escalonamento padrão e o mecanismo de tolerância a falhas no Hadoop são iguais aos do MapReduce. Em resumo, Hadoop tem o mesmo modo de operação que MapReduce, descrito nas seções anteriores.

2.1.6.1 Escalonamento Hadoop

Como afirmado anteriormente, o escalonamento de tarefas padrão no Hadoop é identico ao escalonamento baseado em localidade realizado no MapReduce. No Hadoop este escalonador recebe o nome de FIFO (i.e. *First-in*, *First-out*). A razão para este nome se deve ao fato de que o escalonador, em uma situação onde mais de uma aplicação está a espera de execução, ao selecionar uma tarefa para execução dá prioridade à tarefas da aplicação submetida primeiro. Ou seja, em um caso hipotético onde três aplicações A, B e C são submetidas para execução - primeiro A, seguida da aplicação B, por sua vez seguida da aplicação C - as tarefas nas quais B foi dividida serão escalonadas para execução somente quando A não possuir tarefas na fila de espera. A aplicação C, por sua vez, terá que esperar que B não tenha mais tarefas a serem escalonadas para que suas tarefas possam ser executadas. Em resumo, a aplicação com o tempo de submissão mais antigo tem controle total do *cluster* enquanto estiver em execução.

Este escalonamento baseado na ordem de submissão das aplicações funciona bem em um ambiente onde não há compartilhamento de recursos - ou seja, em um ambiente onde somente uma aplicação é executada por vez - ou em um contexto onde a latência na resposta das aplicações MapReduce não é importante.

Porém, o compartilhamento de recursos é uma situação comum pois possibilita uma maior utilização dos mesmos, bem como a consolidação do armazenamento - levando a um custo menor de instalação e manutenção de *clusters* de computadores. E em um ambiente compartilhado, onde vários usuários submetem aplicações para execução, a filosofia de escalonamento FIFO leva a tempos de resposta muito altos.

Para lidar com esta situação um escalonador que aloca tarefas segundo um princípio de eqüidade (em conjunto com considerações sobre localidade) foi proposto (ZAHARIA et al., 2010). Este escalonador é chamado Hadoop Fair Scheduler (HFS). HFS foi projetado com dois objetivos principais:

- Compartilhamento justo dos recursos entre diferentes aplicações e usuários;
- Escalonamento local de tarefas para maximizar o desempenho do sistema (assim

como no escalonador padrão).

Em uma situação hipotética idêntica a apresentada acima, onde três aplicações *A*, *B* e *C* são submetidas para execução, nesta ordem, quando houver um nó escravo disponível, HFS selecionará para execução uma tarefa daquela aplicação que tiver menos tarefas em execução no momento da decisão de escalonamento. Ou seja, se *A* possui 10 tarefas em execução, *B* possui 8 tarefas em execução e *C* possui 7 tarefas em execução, uma tarefa de *C* será escolhida para ser escalonada. Porém, como foi comentado anteriormente, HFS objetiva não somente o compartilhamento justo dos recursos. A localidade dos dados é outro fator de decisão. Portanto, voltando ao exemplo, caso não seja possível escalonar uma tarefa local para *C* na máquina disponível, HFS tentará escalonar uma tarefa local da aplicação seguinte na lista de prioridade. Ou seja, a segunda que possuir menos tarefas em execução - a aplicação *B*, no exemplo ilustrativo. Para evitar que uma aplicação fique indefinidamente à espera de recursos, existe um limite no número de vezes que uma aplicação pode ser rejeitada para o escalonamento em virtude de não possuir dados a serem processados armazenados na máquina livre. Quando este limite é atingido, HFS escalona uma tarefa não-local.

Os resultados da experimentação demonstram que HFS diminui em até 44% o tempo médio de execução de aplicacões quando comparado ao FIFO em uma situação onde diversas aplicações estão executando simultaneamente - atingindo uma taxa de execução de tarefas locais próxima a 100% (ZAHARIA et al., 2010).

2.2 Metodologia de Testes

Antes de iniciar a discussão sobre metodologia de testes, algumas definições são importantes para o melhor entendimento dos conceitos (JAIN, 1991):

- Variável de Resposta: a saída do experimento é chamada variável de resposta. Ou seja, variável de resposta é a métrica em avaliação a ser mensurada nos testes;
- Fatores: parâmetros que têm influência sobre a variável de resposta e possuem mais de um valor possível são chamados fatores;
- Níveis: valores que um fator pode assumir são chamados níveis;
- Fatores Primários: fatores cujo impacto na variável de resposta serão quantificados são os fatores primários;
- Fatores Secundários: fatores que possuem influência sobre a variável de resposta mas que não terão seu impacto quantificado são fatores secundários;
- Replicação: a repetição de experimentos é chamada replicação;
- Interação: fatores interagem quando o efeito de um dos fatores depende do nível que se encontra outro fator;
- Projeto Experimental: consiste na especificação do número de experimentos a ser realizado, a combinação de fatores para cada experimento e o número de replicações efetuadas em cada experimento.

A metodologia de testes conhecida como projeto experimental (JAIN, 1991) tem como objetivo discriminar os efeitos de vários fatores que podem ter influência sobre a variável de resposta de interesse. Desta forma, a metodologia determina se um fator tem efeito significativo ou se as diferenças observadas na variável de resposta ocorrem devido a variações aleatórias ocasionadas por erros experimentais ou parâmetros não controlados.

Existem três variações de projeto experimental: projeto simples, projeto fatorial completo e projeto fatorial fracionado.

Em um projeto fatorial simples, um fator é variado a cada experimento. Esta opção não é estatisticamente eficiente, pois exige um grande número de experimentos para obter uma pequena quantidade de informações. Ainda, este projeto de experimentos não é capaz de determinar se os fatores possuem interações.

O projeto fatorial completo testa todas as combinações possíveis de configuração. A vantagem deste projeto de experimentos é que todas as possibilidades são examinadas e, assim, o efeito de todos os fatores e todas as interações entre os fatores pode ser determinado. O grande problema do projeto fatorial completo é o custo da experimentação devido ao grande número de experimentos necessários.

Caso o custo da experimentação torne o projeto fatorial completo impraticável, uma outra opção é o projeto fatorial fracionado - onde algumas combinações são descartadas. Porém, apesar de ser menos custoso para ser executado, projetar os experimentos determinando quais combinações não serão testadas é uma tarefa complexa, pois exige que se saiba de antemão quais fatores não interagem de maneira significativa. Ainda, devido ao descarte de experimentos, o projeto fatorial fracionado provê menos informação do que o projeto experimental completo - algumas interações entre fatores não podem ser quantificadas.

Outra possibilidade para reduzir o número de experimentos no projeto fatorial completo é a redução do número de níveis em cada fator - utilizando somente dois níveis. Isto é possível pois comumente o efeito de um fator é unidirecional. Ou seja, a variável de resposta decresce continuamente ou aumenta continuamente conforme o fator é modificado do valor mínimo para o valor máximo. Assim, para determinar os efeitos que cada fator tem sobre a variável de resposta é possível realizar a experimentação utilizando os níveis mínimo e máximo do fator.

Um projeto experimental com dois níveis e K fatores é conhecido como 2^K e é utilizado para determinar o efeito sobre a variável de resposta de K fatores. Este projeto experimental tem um número reduzido de experimentos. Porém não é possível calcular os efeitos que erros experimentais e fatores não controlados tem sobre a variável de resposta. Para que isto seja possível - estimar os erros experimentais dos testes e os efeitos que fatores que não foram controlados tem sobre a variável de resposta - é necessário realizar um projeto experimental com replicações, repetindo cada combinação de experimentos. Ao repetir cada experimento R vezes em um conjunto de experimentos 2^K, serão realizados $2^K R$ testes. Este projeto experimental com dois níveis, K fatores e R replicações é chamado projeto experimental $2^K R$. Esta metodologia permite isolar os erros experimentais e os efeitos de fatores não controlados devido às múltiplas (R) replicações de cada combinação de níveis para os K fatores.

2.3 Modelos para Consumo Energético

Existem três abordagens distintas para mensurar o consumo de equipamentos computacionais. São elas: medições de potência no nível de *hardware* (CHANG; FARKAS;

RANGANATHAN, 2003), modelagem da potência consumida através de técnicas de simulação (GURUMURTHI et al., 2002) (WANG et al., 2002) e modelagem em tempo-real através do monitoramento dos níveis de utilização dos recursos (ECONOMOU et al., 2006) (FAN; WEBER; BARROSO, 2007).

A primeira das três abordagens é a mais precisa e rápida. Porém, é necessário algum tipo de equipamento de instrumentação capaz de medir a potência dissipada para que esta abordagem seja utilizada. A segunda abordagem, modelagem através de simulação, é precisa, podendo adicionalmente prover uma análise detalhada dos padrões de consumo energético. O problema desta abordagem é que ela é lenta e muito custosa, sendo inviável utiliza-lá em tempo de execução para aplicações com longa duração e grandes conjuntos de dados a processar (ECONOMOU et al., 2006).

A última das três opções, modelagem do consumo energético em tempo-real através do monitoramento dos níveis de utilização dos recursos, pode ser dividida em duas abordagens distintas: modelos analíticos detalhados e modelos de alto nível. Modelos analíticos detalhados são, de fato, modelos em tempo-real e tem níveis de precisão na mensuração do consumo energético altos. Porém, esta abordagem é limitada no seu escopo - por modelar somente um componente de um sistema computacional - e portabilidade - por depender de informações microarquiteturais muito detalhadas para determinar o consumo energético (e.g. (ZEDLEWSKI et al., 2003) (ISCI; MARTONOSI, 2003)).

A outra abordagem para modelagem de consumo em tempo-real com base na utilização dos recursos - modelos de consumo energético de alto nível - se revela como uma opção interessante. Este tipo de modelo não provê o mesmo nível de precisão que as opções anteriores ao evitar depender de informações detalhadas do *hardware* sendo monitorado, porém é preciso o suficiente (cerca de 10% para a maioria das *workloads*) para ser considerado como uma opção viável para mensurar o consumo de energia de sistemas computacionais (RIVOIRE; RANGANATHAN; KOZYRAKIS, 2008). Ainda, trata-se de uma metodologia de predição de consumo energético não-intrusiva, de baixo custo (não precisa de qualquer instrumentação adicional), rápida e de alta portabilidade (ao utilizar como métricas de utilizacão dos recursos computacionais comumente disponíveis em sistemas operacionais). Por estas razões, constitui uma possibilidade para utilização em tempo de execução.

Assim, seguindo a última abordagem considerada, a potência total dissipada em um sistema computacional poderia ser determinada através de um modelo como este:

$$P_{TOTAL} = U_{CPU} \cdot TDP_{CPU} + U_{MEM} \cdot TDP_{MEM} + U_{DISK} \cdot TDP_{DISK} + U_{NET} \cdot TDP_{NET}$$

Onde U_{CPU}, U_{MEM}, U_{DISK} e U_{NET} representam, respectivamente, o nível de utilização do processador, do sistema de memória, do disco rígido e da interface de rede da máquina. Estes níveis de utilização são multiplicados pelos respectivos valores de *Thermal Design Power* (TDP) de cada componente. TDP representa a potência máxima que o componente dissipa ao executar aplicações reais.

O níveis de utilização podem ser obtidos através do monitoramento de interfaces presentes em sistemas operacionais que reportam os níveis de utilização para os principais componentes de *hardware* do sistema (MOUW, 2001). A potência máxima atingida por cada componente pode ser facilmente obtida através dos manuais de especificação destes.

A energia consumida em um certo período de tempo é obtida através da multiplicação da potência média dissipada - calculada através do modelo - pelo período de tempo de interesse. Utilizando esta metodologia baseada no monitoramento contínuo das métricas

de utilização é possível determinar predições precisas do consumo energético total do sistema (ECONOMOU et al., 2006).

3 ESTADO DA ARTE

Este capítulo apresentará trabalhos que se relacionam de alguma forma a este. Portanto, serão apresentados trabalhos que envolvem a avaliação do consumo energético ou a avaliação do desempenho do modelo de programação MapReduce. Todos os trabalhos citados neste capítulo utilizam a implementação do modelo de programação MapReduce de código aberto Hadoop. Por esta razão, os termos MapReduce e Hadoop são utilizados sem que haja diferenciação entre eles.

3.1 Avaliação do Consumo Energético do MapReduce

Em (CHEN; GANAPATHI; KATZ, 2010) o objetivo do estudo é determinar como a compressão de dados pode melhorar o desempenho e a eficiência energética de aplicações MapReduce. Para isto, uma avaliação sistemática dos compromissos entre computação (compressão gera mais computação) e comunicação (ausência de compressão leva a uma comunicação de dados maior) é realizada. A partir desta análise, um algoritmo que determina se a compressão sobre os dados deve ou não ser executada para que uma determinada aplicação consuma menos energia foi desenvolvido. A decisão do algoritmo (comprimir ou não) é feita com base em características dos dados de entrada e nos padrões de comunicação da aplicação. Os resultados demonstram que o algoritmo atinge entre 35-60% de redução no consumo energético para aplicações com dados com alto índice de compressibilidade e alta comunicação de dados entre as máquinas.

(CHEN; GANAPATHI; KATZ, 2010) tem objetivo parcialmente similar ao do trabalho apresentado nesta monografia. Ambos avaliam se a compressão de dados em uma computação MapReduce tem influência sobre o consumo de energia. Porém, enquanto em (CHEN; GANAPATHI; KATZ, 2010) somente a compressão é avaliada, neste estudo a compressão é avaliada em meio a outros parâmetros - determinando assim não somente se esta tem efeito sobre o consumo energético, bem como que magnitude este efeito tem quando comparado a outros em um subconjunto do universo de parâmetros de configuração e operação do MapReduce.

(LANG; PATEL, 2010) objetiva melhorar a eficiência energética de *clusters* que executam uma implementação do modelo de programação MapReduce através da exploração de períodos de baixa utilização das máquinas que compõem o ambiente computacional. A proposta dos autores para atingir este objetivo consiste em uma estratégia de gerenciamento de energia para *clusters* chamada All-In-Strategy (AIS) - uma solucão que seletivamente desliga/liga nodos que participam de um *cluster* MapReduce em períodos de baixa utilização. Esta estratégia consiste em executar a carga de trabalho necessária em todos os nós disponíveis no *cluster*. E em períodos de baixa utilização em que o *cluster* está ocioso, o ambiente inteiro é colocado em um modo de baixo consumo energé-

tico. Portanto, em resumo, quando surge uma requisição para execução de uma aplicação MapReduce, o *cluster* sofre uma transição do modo de baixo consumo para seu estado de operação normal, executa a aplicação e depois é colocado novamente em modo de espera.

Um *framework* que implementa tal mecanismo (AIS) foi desenvolvido e os efeitos sobre o tempo de resposta de uma aplicação MapReduce e sobre o consumo de energia total do *cluster* foram estudados (LANG; PATEL, 2010). Os resultados dos experimentos demonstraram que, em comparação a um *cluster* sobre o qual nenhum tipo de mecanismo de gerenciamento de energia opera, AIS tem uma eficiência energética até 60% maior com uma degradação no tempo de resposta de até 12%.

(LANG; PATEL, 2010) não consiste em uma caracterização do consumo energético do MapReduce. Nenhuma consideração específica sobre MapReduce é levantada - o consumo energético não é explicado de forma alguma através de parâmetros de configuração ou modos de operação do *framework*. O MapReduce é apenas uma carga de trabalho para validar o mecanismo de gerenciamento energético de clusters proposto pelos autores. Qualquer outro modelo de programação paralelo poderia ser utilizado para validar o mecanismo proposto do artigo sem que este precisasse ser alterado.

(LEVERICH; KOZYRAKIS, 2010) propõe um mecanismo chamado Covering Set (CS) para melhoria da eficiência energética do *framework* MapReduce. Este mecanismo é uma solução para um problema imposto pelo sistema de arquivos distribuídos utilizado pelo MapReduce: não é possível utilizar técnicas de gerenciamento de energia que desligam parte das máquinas devido ao armazenamento dos dados de forma distribuída sobre os nós - desligar as máquinas significaria potencialmente tornar parte dos dados indisponíveis.

Por isto, o mecanismo proposto (CS) atua sobre a distribuição dos dados: pelo menos uma réplica (existem três réplicas para cada bloco de dados por padrão) é armazenada em um subconjunto dos nós (i.e. Covering Set). Então, os nós que não fazem parte do CS podem ser desligados (ou colocados em um modo de operação de baixo consumo energético) em períodos de baixa utilização, pois CS contém um número suficiente de nós para que a disponibilidade de todo o conjunto de dados seja garantida.

Os resultados demonstraram que o mecanismo pode levar a reduções de consumo energético de até 51%. Porém, existe um impacto sobre o desempenho do MapReduce ao desligar os nós que não fazem parte do CS: até 71% de aumento no tempo de resposta foi verificado. Logo, o mecanismo proposto diminui o consumo de energia, porém com aumento do tempo de execução.

GreenHDFS (KAUSHIK; BHANDARKAR, 2010) é uma proposta de alteração do sistema de arquivos distribuído do MapReduce que separa os nodos de armazenamento em duas zonas: *hot* e *cold*. Esta nomenclatura separa os dados conforme suas necessidades de processamento e disponibilidade (de acordo com uma análise de frequência de acesso de cada bloco de dados). A partir desta classificação, os nodos que pertencem à zona *cold* - com dados menos frequentemente acessados - podem ser colocados em modos de baixo consumo de energia ou desligados. Portanto, o GreenHDFS altera o esquema de distribuição dos dados inserindo uma política de classificação de dados com o objetivo de obter maior eficiência energética do ambiente computacional. Resultados de simulação demonstraram que GreenHDFS é capaz de atingir 26% de redução do consumo energético em um período simulado de três meses.

Como (LANG; PATEL, 2010), a proposta destes artigos consiste em uma solucão para seletivamente desligar/ligar nodos que participam de um cluster MapReduce para lidar com períodos de baixa utilização. Porém, diferentemente de (LANG; PATEL, 2010), estes

trabalhos operam diretamente sobre o MapReduce - alterando o método de distribuição dos dados sobre o sistema de arquivos distribuídos. Assim, (LEVERICH; KOZYRAKIS, 2010) e (KAUSHIK; BHANDARKAR, 2010) avaliam o impacto que a distribuição não ciente de energia tem sobre o consumo do MapReduce - e propõem um mecanismo alternativo, ciente do consumo energético. Esta monografia não considera aspectos do sistema de arquivos distribuídos na caracterização do MapReduce proposta.

(WIRTZ; GE, 2011) estuda o comportamento do consumo energético do MapReduce quando este opera em um ambiente com reconfiguração no número de nós escravos (com base em política de desligamento de nós para reduzir o consumo de energia) e com ajuste da frequência dos processadores (*Dynamic Voltage and Frequency Scaling*) com base nas necessidades computacionais correntes.

(WIRTZ; GE, 2011) não avalia o consumo energético do MapReduce em termos de características próprias do mesmo. O trabalho verifica como o consumo de energia se comporta quando o ambiente computacional é gerenciado com políticas para redução do consumo (reconfiguração e DVFS) externas ao MapReduce são utilizadas.

(MAHESHWARI; NANDURI; VARMA, 2012) trata, também, da questão do gerenciamento do consumo de energia em um ambiente computacional onde MapReduce é executado. Os autores argumentam que os mecanismos de tolerância a falhas e balanceamento de carga incorporados no *framework* tem um impacto sobre a eficiência energética, pois mesmo nós ociosos devem permanecer em operação normal (consumindo energia elétrica) para que estejam garantidas a confiabilidade e disponibilidade de dados. Com base nesta avaliação, o artigo aborda a questão de conservação de energia em um ambiente MapReduce.

A principal contribuição do trabalho é um algoritmo que reconfigura dinamicamente o *cluster* MapReduce com base na carga atual do mesmo. Somente um subconjunto de nós inicia o serviço ligado e a partir das requisições para execução de aplicações MapReduce novos nós são adicionados (ligados) caso o subconjunto inicial não seja suficiente para atender a demanda. Então, quando os nós que não fazem parte da computação se tornam ociosos eles são desligados novamente. Alterações na distribuição de dados do sistema de arquivos foram efetuadas para que a reconfiguração dinâmica fosse possível - eliminando a possibilidade de dados estarem indisponíveis devido ao desligamento de nós de armazenamento.

Simulações do algoritmo proposto determinaram que este - quando comparado ao Hadoop original - apresenta uma redução no consumo energético de 33% - em aplicações com altas demandas computacionais - à 54% - em aplicações com baixas demandas de computação.

(MAHESHWARI; NANDURI; VARMA, 2012) é outro trabalho de gerenciamento de energia em *clusters*. Alterações no sistema de arquivos distribuído que possibilitam ligar e desligar nós conforme a demanda são propostas. Não consiste em uma análise do comportamento do MapReduce com enfoque em parâmetros de mesmo. Os próprios autores afirmam que a abordagem proposta pode ser aplicada a outro *framework*.

GreenHadoop (GOIRI et al., 2012) é um *framework* proposto com o objetivo de investigar o gerenciamento da carga computacional de um *datacenter* onde existem dois tipos distintos de fonte de energia: fontes de energia "verdes"(e.g. vento, luz solar) e fontes de energia "marrom"(e.g. usinas termoelétricas). O estudo considera a possibilidade de atrasar, se possível, certas computações para períodos onde a energia "verde"estará disponível - minimizando o uso da energia "marrom".

Para isto, GreenHadoop, um *framework* MapReduce para ambientes computacionais

energizados através de paineis solares e, como segunda opção, pelo sistema elétrico convencional, foi desenvolvido. O *framework* proposto busca maximizar a utilização de energia limpa (i.e. "verde") e minimizar o custo da energia "marrom", executando a requisição em um tempo de resposta máximo. GreenHadoop depende de previsões da disponibilidade de energia solar. A partir de tais previsões, o algoritmo ciente de consumo energético escalona as tarefas que podem ser atrasadas para o período onde a energia verde está disponível. Se não houver disponibilidade de energia solar dentro do período máximo de execução da aplicação, GreenHadoop busca minimizar o custo da energia convencional. Os resultados demonstraram que o framework proposto pode aumentar o consumo de energia solar em até 31% e diminuir os custos de energia convencional em até 39% - quando comparado ao Hadoop original.

(GOIRI et al., 2012) não tem como objetivo uma caracterização do consumo de energia do MapReduce - este pode ser visto como um caso de uso da metodologia de gerenciamento de energia proposta pelos autores.

3.2 Avaliação de Desempenho do MapReduce

(JIANG et al., 2010) apresenta um estudo do desempenho do MapReduce no contexto da área de bancos de dados. Sob esta perspectiva, são identificados cinco fatores que afetam o desempenho do MapReduce: modo de entrada e saída, mecanismo de indexação dos dados, mecanismo de leitura de registros, esquemas de agrupamento e o tamanho do bloco de dados do sistema de arquivos distribuídos.

Os fatores estudados em (JIANG et al., 2010) diferem daqueles estudados neste trabalho. Todos os fatores considerados no artigo tem relação com o sistema de arquivos distribuído - devido ao contexto da pesquisa ser a área de bancos de dados. Neste trabalho o sistema de arquivos distribuído não faz parte da caracterização proposta - a caracterização tem como objetivo melhor entender o comportamento do sistema de execução MapReduce.

(HUANG et al., 2010) apresenta uma plataforma de *benchmarks* para o MapReduce. A plataforma é composta de aplicações sintéticas e reais. Os autores caracterizam as diferentes aplicações presentes na plataforma de benchmarks em termos do tempo de execução, throughput (i.e. o número de tarefas completadas por minuto), a banda do sistema de arquivos distribuídos e os padrões de acesso de dados.

A caracterização difere deste estudo por que não explica os resultados em termos de características próprias do sistema de execução MapReduce - e sim em termos das diferentes aplicações existentes na plataforma de *benchmarks* proposta.

Através de técnicas de aprendizado de máquina, (AGGARWAL; PHADKE; BHANDARKAR, 2010) caracteriza diversas execuções MapReduce a partir de traços de execução de um *cluster* MapReduce do Yahoo!. A caracterização realizada busca classificar cada execução em termos do universo de execuções analisadas - determinando se a execução em questão representa uma execução comum ou não. Diversas métricas são utilizadas para caracterização: número de tarefas *map*, número de tarefas *reduce*, formato da entrada, formato da saída - entre outros.

Outra vez, a caracterização não tem como objetivo relacionar as métricas avaliadas com características do MapReduce. Ainda, o estudo não é propriamente um estudo de desempenho, pois não existem considerações sobre o tempo de execução das aplicações.

4 METODOLOGIA

Neste capítulo serão apresentadas a metodologia de testes e os modelos de consumo energético utilizados para realização do trabalho.

4.1 Metodologia de Testes

Esta seção apresentará a metodologia de testes utilizada para atingir o objetivo deste estudo - esclarecendo as escolhas feitas. Serão apresentados, nesta ordem: os objetivos da experimentação a ser realizada e o sistema a ser avaliado; as variáveis de resposta que serão utilizadas para a avaliação proposta; a técnica de avaliação empregada; os parâmetros que potencialmente têm efeito sobre as variáveis de resposta de interesse; dentre os parâmetros listados, os fatores selecionados para a caracterização proposta; as aplicações utilizadas para avaliar o sistema; o ambiente computacional utilizado nos testes; e, por fim, a estrutura dos experimentos realizados.

4.1.1 Objetivos e Sistema

O objetivo do estudo é a caracterização do consumo energético de um *framework* para desenvolvimento de aplicações intensivas em dados. Uma caracterização do desempenho do mesmo sistema para computação intensiva em dados também será realizada - de forma que o consumo de enegia seja considerado em um contexto mais amplo e, assim, os resultados possam ser compreendidos de maneira mais profunda.

O sistema de computação intensiva em dados a ser estudado é o Hadoop (APACHE, 2013) - implementação de código aberto do modelo de programação MapReduce (DEAN; GHEMAWAT, 2008). O trabalho considerará somente características do sistema de execução Hadoop. O sistema de arquivos distribuído não será considerado no estudo. Os fatores considerados para avaliação, bem como as métricas escolhidas, serão apresentadas adiante.

No decorrer do capítulo, não é feita distinção entre MapReduce e Hadoop - portanto, ambos termos devem ser consideradas sinônimos.

4.1.2 Variáveis de Resposta e Métricas

O sistema a ser estudado - como qualquer outro sistema - pode se comportar de três maneiras distintas quando requisitado para executar o serviço para o qual foi projetado: o sistema pode realizar o serviço corretamente; o sistema pode realizar o serviço incorretamente; e, também, o sistema pode não realizar o serviço requisitado (JAIN, 1991). As duas últimas situações - execução errônea e não execução - ocorrem em condições onde há falhas no sistema avaliado.

A caracterização do MapReduce neste estudo acontecerá somente na primeira das três possibilidades - execução correta do serviço. Ou seja, o *framework* não será caracterizado em situações de falha - onde não executa o serviço requisitado corretamente ou em situações onde o serviço requisitado não é executado.

As métricas utilizadas para caracterizar o sistema são o consumo energético - medido em Joules - e o tempo de execução - medido em segundos.

4.1.3 Técnica de Avaliação

Existem três técnicas de avaliação de sistemas (JAIN, 1991). São elas: modelagem analítica, simulação e experimentação. Uma das principais considerações para escolha da técnica adequada é o estágio de desenvolvimento do sistema a ser avaliado. Experimentação somente pode ser realizada caso o sistema exista e esteja operacional. Portanto, no caso de o sistema em avaliação ser somente conceitual no momento da experimentação, modelagem analítica e simulação são as únicas opções possíveis.

Outro ponto importante na decisão da técnica de avaliação é a complexidade do sistema a ser analisado. Modelagem analítica e simulação exigem um esforço grande por parte do analista, sendo inviável avaliar um sistema intricado com diversos parâmetros de operação. Neste sentido, a experimentação é a técnica menos custosa. Ainda, diferentemente do que ocorre com a modelagem analítica e com simulações, a experimentação não impõe nenhum tipo de simplificação: todos os parâmetros do sistema farão parte da avaliação. O grau de aceitação das técnicas também contribuiu para decidir qual técnica de avaliação empregar neste estudo. Por se basear em experimentos do sistema real - e, também, pela menor complexidade envolvida no desenvolvimento da avaliação - experimentação é comumente a técnica de avaliação mais aceita (JAIN, 1991).

Então, ao selecionar - entre as opções mencionadas - a técnica de avaliação que permite atingir os objetivos do estudo com maior eficácia, as questões levantadas acima foram consideradas - estágio de desenvolvimento do sistema, o esforço necessário para realizar a avaliação e o nível de aceitação da técnica. Primeiramente, como o sistema a ser avaliado é um sistema existente, qualquer uma das técnicas pode ser utilizada. Ainda, o esforço exigido para avaliar um sistema distribuído como MapReduce limita os resultados que podem ser obtidos através da simulação e modelagem analítica. Somando-se a isto, a maior aceitação da experimentação do sistema em um ambiente real, esta foi a técnica de avaliação escolhida para realizar este trabalho.

4.1.4 Parâmetros

Esta subseção lista os parâmetros do sistema que têm efeito sobre as métricas que serão analisadas (i.e. consumo de energia e tempo de execução). Esta listagem é importante pois determina sob quais aspectos do sistema MapReduce a caracterização ocorrerá e quais parâmetros devem ser controlados durante a experimentação para que os resultados sejam precisos e confiáveis. A variável de resposta de interesse principal é o consumo energético. Porém, para determinar quais parâmetros têm potencialmente maior influência sobre o consumo de energia, utilizou-se como base parâmetros que têm potencialmente influência sobre o desempenho do sistema.

Os parâmetros podem ser divididos em duas categorias: parâmetros de sistema (características de *software* e *hardware*) e parâmetros de aplicação (características das requisições dos usuários ao sistema) (JAIN, 1991).

4.1.4.1 Parâmetros de Sistema

Hadoop possui cerca de 190 parâmetros de configuração - entre parâmetros do sistema de execução e parâmetros do sistema de arquivos distribuído. Estes parâmetros podem ser divididos em parâmetros que são configuráveis por execução e parâmetros que são configuráveis por instalação. Aqueles configuráveis por execução podem ser alterados pelo usuário a cada aplicação executada. A outra categoria - parâmetros configuráveis por instalação - configuram o sistema como um todo e é necessário ter privilégios de administrador para que sejam alterados. A caracterização levará em conta tal categorização dos parâmetros de configuração. Mais uma vez, ressalta-se que parâmetros de configuração do sistema de arquivos distribuídos não serão considerados.

Como base para determinar quais parâmetros tem influência sobre o desempenho de uma aplicação MapReduce, foram utilizados dois estudos: (HERODOTOU; BABU, 2011) e (ZAHARIA et al., 2010).

(HERODOTOU; BABU, 2011) - em um artigo que propõe uma técnica para otimizar a configuração de uma aplicação MapReduce - lista 14 parâmetros de configuração do Hadoop com maior influência sobre o desempenho do *framework*. A Tabela 4.1 apresenta estes parâmetros - descrevendo-os rapidamente e expondo qual o valor padrão caso o usuário opte por não configurar o parâmetro (todos os 14 parâmetros listados são configuráveis por execução de aplicação MapReduce):

Outro importante parâmetro de configuração do MapReduce em termos de desempenho é aquele que controla qual escalonador de tarefas será utilizado (*mapred.jobtracker.taskScheduler*). Existem duas opções distintas de escalonador de tarefas: FIFO e Hadoop Fair Scheduler (HFS). FIFO é o escalonador padrão, enquanto HFS é o escalonador alternativo. Ambos apresentam diferenças significativas de desempenho (ZAHARIA et al., 2010). Este parâmetro de configuração não pode ser modificado a cada aplicação executada - sem que o sistema MapReduce seja reinicializado - e para ser alterado é necessário possuir permissões de administrador do sistema.

Evidentemente, o sistema de hardware sobre o qual é executado o MapReduce também tem influência sobre as variáveis de resposta de interesse. Porém, este não apareceu listado acima pois não constitui em um parâmetro do sistema MapReduce - e sim uma característica externa ao framework. O trabalho não tem objetivo de analisar questões externas ao framework - o objetivo do estudo é a caracterização do MapReduce a partir de características internas do sistema.

De toda forma, é importante ter a consciência de que este parâmetro deve ser controlado e isolado para que os resultados sejam confiáveis. Por isto, todos os testes serão executados sobre o mesmo cluster. Mais sobre a plataforma de hardware utilizada na seção 4.1.7.

4.1.4.2 Parâmetros de Aplicação

Os parâmetros de aplicação (ou seja, as características de uma requisição do usuário ao sistema MapReduce) têm influência sobre o desempenho do *framework*, também. Estas características são importantes e devem ser controladas nos testes. São elas: a aplicação a ser executada e o tamanho da entrada que a aplicação deve executar.

4.1.5 Fatores

Nesta seção serão definidos os parâmetros que serão estudados - dentre aqueles descritos na seção anterior - e, portanto, variados durante os testes. Seria impraticável con-

Tabela 4.1: Parâmetros de configuração Hadoop com influência sobre o desempenho

PARÂMETRO	DESCRIÇÃO	DEFAULT
io.sort.mb	Tamanho [MB] do *buffer* para o armazenamento em memória dos pares chave-valor produzidos pela função *map*	100
io.sort.record.percent	Fração do *buffer* definido por **io.sort.mb** para armazenamento de metadados para cada par chave-valor	0.05
io.sort.spill.percent	Limite de uso do *buffer map* (definido em **io.sort.mb**) para que os conteúdos armazenados sejam movidos para o disco	0.80
io.sort.factor	Número de arquivos que devem ser agrupados para o *sort* (ordenamento das saídas *map* pré-transferência dos dados para as máquinas que irão executar a fase de *reduce*)	10
mapreduce.combine.class	Classe da função *combiner* (opcional) para pré-agregar as saídas *map* antes da tranferência para as tarefas *reduce*	null
min.num.spills.for.combine	Número mínimo de arquivos (*spill files*) para iniciar o *combiner*	3
mapred.compress.map.output	Valor booleano que determina se os dados de saída das tarefas *map* serão comprimidos antes do envio às tarefas *reduce*	false
mapred.reduce.slowstart.completed.maps	Proporção do total de tarefas *map* que deve ser concluído antes do início do escalonamento de tarefas *reduce*	0.05
mapred.reduce.tasks	Número de tarefas *reduce*	1
mapred.job.shuffle.input.buffer.percent	Fração do *buffer* das tarefas *reduce* utilizado para armazenar os dados copiados das tarefas *map* durante o *shuffle*	0.7
mapred.job.shuffle.merge.percent	Limite de uso do *buffer* das tarefas *reduce* para início do *merge* (agrupamento das tuplas copiadas das máquinas que executaram a fase *map*)	0.66
mapred.inmem.merge.threshold	Limite no número de pares chave-valor produzidas pela fase *map* copiadas para início do *merge*	1000
mapred.job.reduce.input.buffer.percent	Fração do *buffer* das tarefas *reduce* utilizado para armazenar os dados copiados das tarefas *map* durante *o reduce*	0
mapred.output.compress	Valor booleano que determina se a saída final da aplicação deve ser comprimida antes de ser armazenada no HDFS	false

siderar todos os parâmetros listados anteriormente no estudo. Os parâmetros de interesse para a caracterização são chamados fatores primários (JAIN, 1991). Os parâmetros listados anteriormente que não serão escolhidos para estudo serão mantidos constantes durante todos os experimentos.

Antes de apresentar os fatores primários, os fatores secundários - parâmetros que não farão parte do estudo - são listados, bem como as razões para não incluí-los.

- *io.sort.record.percent*: este é um parâmetro considerado de difícil configuração. Não está claro o que são os metadados e nem em que quantidade tais metadados são produzidos;

- *io.sort.factor*: este parâmetro não será considerado na caracterização devido a escolhas de valores feitas para alguns dos fatores primários (io.sort.mb e io.sort.spill.percent). Tais escolhas tornam este parâmetro irrelevante pois determinam que no máximo dois arquivos serão armazenados em disco para cada tarefa map - e comumente este parâmetro é configurado para que 100 arquivos

sejam agrupados (WHITE, 2012);

- *mapreduce.combine.class*: este parâmetro indica qual - se existir - classe implementa a função *combiner*. *Combine* é um passo adicional opcional na computação MapReduce executado durante o *shuffle*. Consiste na aplicação de uma função (i.e. *combiner*) que agrupa os dados intermediários que têm a mesma chave e, então, envia-os como uma única mensagem à tarefa *reduce*. Tipicamente, a função *combiner* é idêntica a função *reduce*. Se trata de uma otimização importante por reduzir a comunicação entre os nós escravo e, consequentemente, diminuir o uso da banda de rede (DEAN; GHEMAWAT, 2008). Porém, apesar de ser uma otimização importante, a função *combiner* não pode ser utilizada em qualquer aplicação, pois o sistema de execução não especifica quantas vezes esta função será executada sobre uma saída particular da função map. Assim, é esperado que a função combiner seja idempotente (WHITE, 2012) - ou seja, espera-se que a aplicação repetida do *combiner* sobre as saídas tenha o mesmo resultado após a aplicação inicial. Como isto não é possível para toda e qualquer aplicação, o uso do combiner não fará parte da caracterização proposta no estudo.

- *min.num.spills.for.combine*: ver *mapreduce.combine.class*. Este parâmetro somente tem sentido caso uma função *combiner* seja utilizada pela aplicação.

- *mapred.inmem.merge.threshold*: este parâmetro controla o uso do *buffer* das tarefas *reduce* para armazenamento das saídas das tarefas *map* - o parâmetro impõe um limite no número de tuplas que podem permanecer em memória. Existe outro parâmetro que exerce o mesmo controle, porém utilizando outra métrica. *mapred.job.shuffle.input.buffer.percent* determina a fração do *buffer* que pode ser utilizado para armazenar as saídas das tarefas *map*. Portanto, estes parâmetros são redundantes e ao configurar *mapred.inmem.merge.threshold* como zero, o comportamento do uso do *buffer* é exclusivamente determinado por *mapred.job.shuffle.input.buffer.percent* (WHITE, 2012). Devido a redundância, este parâmetro não fará parte da caracterização.

- *mapred.job.reduce.input.buffer.percent*: este parâmetro não será considerado pela impossibilidade de se realizar um número maior de experimentos. Foi escolhido para não fazer parte da caracterização por que os outros parâmetros foram considerados mais importantes.

- *mapred.output.compress*: a compressão ao final da computação (antes do armazenamento no sistema de arquivos distribuídos) não será considerada por não ser necessária do ponto de vista do fluxo de execução do MapReduce. Comprimir os dados neste estágio - ao final da computação - é uma decisão única e exclusivamente feita com base em considerações sobre o sistema de arquivos - que, como dito anteriormente, não faz parte do estudo.

Em resumo, parâmetros foram descartados por serem difíceis de configurar (*io.sort.record.percent*), irrelevantes (*io.sort.factor*), limitados a certas aplicações (*mapreduce.combine.class* e *min.num.spills.for.combine*), redundantes (*mapred.inmem.merge.threshold*) e por serem configurações que não controlam características referentes exclusivamente ao sistema de execução (*mapred.output.compress*). Ainda, no caso do parâmetro *mapred.job.reduce.input.buffer.percent*, este foi descartado

pela limitação na quantidade de experimentos que podem ser realizados. Todos os fatores secundários serão configurados com os valores padrão (exceto *mapred.inmem.merge.threshold* que será configurado em zero - devido às razões explicadas anteriormente).

Os parâmetros de interesse de estudo, a partir dos quais a caracterização será realizada, são listados a seguir - acompanhados de uma breve explicação da importância dos mesmos:

- Parâmetros de sistema
  - *io.sort.mb*: ao definir o tamanho do *buffer* para os valores que devem ser ordenados antes da transferência, este parâmetro determina se os pares chave-valor residem em memória ou devem ser armazenados em disco (e quantas escritas em disco devem acontecer);
  - *io.sort.spill.percent*: outro controle de memória das tarefas *map* que tem o mesmo efeito do parâmetro anterior (determinar o uso do disco e a frequência que as escritas ao disco devem acontecer).
  - *mapred.compress.map.output*: este parâmetro define um compromisso entre computação e armazenamento. O uso da compressão dos dados diminui a comunicação entre nós e aumenta o uso do processador;
  - *mapred.reduce.slowstart.completed.maps*: este parâmetro controla a concorrência entre as fases principais da computação MapReduce (*map* e *reduce*);
  - *mapred.reduce.tasks*: controle da concorrência na fase reduce;
  - *mapred.job.shuffle.input.buffer.percent*: controle de memória durante o *shuffle* nas tarefas *reduce*. Influência o uso do disco e a frequência com que este uso ocorre;
  - *mapred.job.shuffle.merge.percent*: controle de memória durante o shuffle nas tarefas reduce. Influência o uso do disco e a frequência com que ocorre;
    - (Os parâmetros que exercem controle sobre a memória das tarefas *map* - *io.sort.mb* e *io.sort.spill.percent* - e *reduce* - *mapred.job.shuffle.input.buffer.percent* e *mapred.job.shuffle.merge.percent* - são importantes, também, porque representam um compromisso entre o espaço para o armazenamento dos pares chave-valor intermediários e o espaço para a execução das tarefas propriamente ditas)
  - *mapred.jobtracker.taskScheduler*: importante pois define o escalonamento das tarefas, tendo influência direta sobre todo o fluxo de execução.

- Parâmetros de aplicação
  - Aplicação: é evidente a influência da aplicação no desempenho MapReduce - diferentes aplicações tem diferentes demandas computacionais. As aplicações que serão utilizadas na caracterização serão descritas na seção 4.1.6.
  - Tamanho da entrada: o tamanho da entrada também tem uma influência evidente nas variáveis de resposta do estudo - o tempo e a energia para o processamento de 1 GB de dados pela aplicação *X* é inferior ao tempo e energia para o processamento de 10 GB pela aplicação *X*.

Os valores utilizados em cada um dos parâmetros de sistema e os tamanhos das entradas utilizadas na experimentação serão detalhados na seção que define a estrutura dos testes (seção 4.1.8). As aplicações testadas são discutidas na próxima seção.

4.1.6 Aplicações

Devido a impossibilidade de realizar experimentos com um grande número de programas MapReduce, as aplicações utilizadas no estudo devem representar um grupo homogêneo para que os resultados tenham maior relevância e possam ser interpretados como resultados para uma classe de aplicações - e não resultados que caracterizam somente as aplicações selecionadas.

Por essa razão é fundamental definir uma classificação de aplicações MapReduce para depois selecionar as aplicações que representarão todo o espectro de aplicações possível - escolhendo uma aplicação para cada categoria da classificação.

Uma classificação habitual para aplicações é aquela que as divide em duas categorias: *CPU bound* e *IO bound*. Aplicações *CPU bound* são aquelas onde o tempo para completar a computação é determinado principalmente pela velocidade do processador. Estas aplicações têm alta utilização da CPU e, comparativamente, baixa utilização do disco e dos serviços de rede. A outra categoria, aplicações *IO bound*, consiste em aplicações onde o tempo para completar a computação é determinado predominantemente pelos períodos de comunicação - em outras palavras, o tempo gasto requisitando dados é superior ao tempo gasto processando dados. Aplicações *IO bound* têm baixa utilização da CPU e alta utilização do disco e serviços de rede.

(HUANG et al., 2010) caracteriza aplicações MapReduce conforme a utilização de recursos. O trabalho apresenta Sort (APACHE, 2009) e WordCount (APACHE, 2011) como aplicações representativas de dois grandes conjuntos de programas MapReduce reais: aplicações que transformam dados de entrada de uma representação para outra e aplicações que extraem uma pequena quantidade de informações de um grande conjunto de dados.

Sort representa o conjunto de aplicações que transforma os dados de entrada de uma representação para outra. Desta forma, a quantidade de dados transferidos entre a fase map e a fase reduce tem o mesmo tamanho que a entrada da aplicação. A saída da aplicação também tem o mesmo tamanho da entrada da aplicação. Então, a aplicação Sort é uma aplicação IO bound - com baixa utilização da CPU e alta utilização de disco e banda de rede (HUANG et al., 2010).

WordCount representa o segundo conjunto de aplicações MapReduce, por ser uma aplicação que extrai uma pequena quantidade de informações de um grande conjunto de dados de entrada. Assim, os dados transferidos no *shuffle* têm tamanho muito inferior aos dados de entrada. Consequemente, WordCount é uma aplicação CPU bound - tendo alta utilização da CPU e baixa demanda por disco e banda de rede (HUANG et al., 2010).

Por representarem as duas categorias de aplicações MapReduce definidas anteriormente, as aplicações que serão utilizadas na caracterização são Sort e WordCount.

4.1.7 Ambiente de Execução

Todos os experimentos foram executados sobre o mesmo ambiente computacional. Este ambiente consiste em um *cluster* composto por 20 máquinas. Existem três tipos diferentes de processadores presentes no cluster. Abaixo eles são listados - acompanhados pela quantidade de máquinas que possuem tal processador e as configurações de memória de cada grupo de máquinas:

- Intel Core i3-2100 (3MB cache, 3.10 GHz, 2 *cores* (2 *threads*/*core*)), 4 GB RAM: 1 máquina
- Intel Pentium 4 (1 MB cache, 2.80 GHz, 1 *core*), 2 GB RAM: 5 máquinas
- Intel Pentium 4 HT (512 KB cache, 2.80 GHz, 1 *core*), 2 GB RAM: 14 máquinas

Todas as máquinas possuem um disco e conexões de 1 Gbps. O sistema operacional das máquinas é o CentOS 5. A versão Hadoop instalada no *cluster* é 1.0.4. O nó mestre é o que possui o processador Intel Core i3. Os nós escravos estão configurados para executar simultaneamente, no máximo 2 tarefas *map* e 2 tarefas *reduce*. Estas informações estão resumidas na Tabela 4.2.

Tabela 4.2: Ambiente de execução

PROCESSADOR	#MÁQUINAS	MEMÓRIA	DISCOS	*LINKS*	*SISTEMA OPERACIONAL*	*HADOOP*
i3	1	4 GB	1	1 Gbps	CentOS 5	1.0.4
Pentium 4	5	2 GB				
Pentium 4 HT	14	2 GB				

4.1.8 Estrutura dos Experimentos

A caracterização foi dividida em três etapas de experimentos. Os testes realizados e os objetivos de cada uma destas etapas são descritos a seguir.

4.1.8.1 $2^K R$ (Parâmetros configuráveis por execução)

Como dito anteriormente, existem aproximadamente 190 parâmetros de configuração no Hadoop - sendo impraticável configurar todos os parâmetros de maneira satisfatória. Por isto, com a finalidade de orientar o usuário MapReduce, esta etapa de experimentos busca determinar quais parâmetros (entre os configuráveis por execução) são importantes e devem ser devidamente configurados para que tanto o consumo energético como o desempenho do sistema sejam ótimos. Para isto foram realizados testes seguindo a metodologia $2^K R$.

A Tabela 4.3 apresenta os fatores primários que farão parte do projeto experimental, detalhando os níveis utilizados para cada fator. Os níveis foram escolhidos de forma que representassem os valores mínimo e máximo dentro da gama de valores que se encontra em uma situação real.

Tabela 4.3: Fatores primários e níveis utilizados no projeto experimental

FATOR	NÍVEL MÍN.	NÍVEL MAX.
io.sort.mb	64	128
io.sort.spill.percent	0.75	0.95
mapred.compress.map.output	false	true
mapred.reduce.slowstart.completed.maps	0.05	1.0
mapred.reduce.tasks	1	32
mapred.job.shuffle.input.buffer.percent	0.3	0.9
mapred.job.shuffle.merge.percent	0.66	0.9

Assim, os efeitos destes parâmetros - todos provenientes de (HERODOTOU; BABU, 2011) - sobre o consumo de energia e sobre o desempenho foram determinados para

ambas aplicacões em separado (*CPU bound* e *IO bound*) e, também, para ambos escalonadores de tarefas em separado (FIFO e HFS). Todos os experimentos foram executados com uma entrada de 1 GB. Os experimentos foram repetidos 10 vezes. O nível de replicação dos experimentos não foi maior por consequência do tempo disponível para realizar os testes e do fato de que o ambiente de execução ser utilizado por vários outros usuários. Ainda, em outro trabalho realizado sobre o mesmo ambiente computacional e com a mesma instalação Hadoop, o sistema apresentou baixa variabilidade nos testes executados (MARCOS, 2013).

4.1.8.2 Caracterização dos Escalonadores (uma aplicação)

A segunda etapa de experimentos tem como objetivo caracterizar os escalonadores de tarefas (FIFO e HFS) em um contexto onde existe somente uma aplicação em execução. Estes experimentos foram executados para ambas aplicações (*CPU bound* e *IO bound*) e utilizando três tamanhos de entrada diferentes: 256 MB, 4 GB e 10 GB. Os experimentos foram repetidos 10 vezes devido às razões citadas anteriormente.

4.1.8.3 Caracterização dos Escalonadores (múltiplas aplicações)

Por último, foram realizados experimentos para avaliar os escalonadores de tarefas (FIFO e HFS) em uma situação onde existem múltiplas aplicações em execução. Em cada experimento foram executadas 20 aplicações. Foram realizados testes para três workloads diferentes: uma onde todas as 20 aplicações submetidas eram a aplicação CPU bound; a segunda onde todas as 20 aplicações executadas eram a aplicação IO bound; e, por último, uma combinação entre a aplicação CPU bound (10 aplicações executadas) e a aplicação IO bound (10 aplicações executadas). Assim como na segunda etapa de experimentos, três tamanhos de entrada diferentes foram utilizados: 256 MB, 4 GB e 10 GB. Para as workloads homogêneas, 14 execuções tinham como entrada 256 MB, 5 execuções tinham como entrada 4 GB e 1 execução tinha como entrada 10 GB. A workload heterogênea tinha 14 execuções com entrada de 256 MB (7 CPU bound, 7 IO bound), 4 execuções com entrada de 4 GB (2 CPU bound, 2 IO bound) e 2 execuções tinham como entrada 10 GB de dados (1 CPU bound, 1 IO bound). Novamente, os experimentos foram repetidos 10 vezes. Estes experimentos foram planejados de acordo com os testes realizados no artigo que apresentou o escalonador HFS (ZAHARIA et al., 2010).

4.2 Modelos de Consumo Energético

Entre as opções de metodologia para mensurar o consumo energético, optou-se pela utilização de um modelo de consumo de energia de alto nível baseado no monitoramento dos níveis de utilização dos recursos. Antes de detalhar as decisões que levaram ao modelo final, são discutidas as razões para que não fosse utilizada nenhuma das outras opções de mensuração do consumo de energia de sistemas computacionais.

Apesar de ser a opção mais precisa, a falta de equipamentos para medir o consumo de energia das 20 máquinas do *cluster* impediu que fossem realizadas medições de potência no nível de *hardware*. A abordagem que modela o consumo energético através de técnicas de simulação não é adequada para o tipo de sistema que se quer caracterizar - aplicações com longa duração e grandes conjuntos de dados a processar - por ser uma técnica lenta e intrusiva (com alto *overhead*).

A última opção descartada, utilizar um modelo analítico detalhado para determinar o consumo de energia do sistema inteiro, é inviável devido ao fato de que tais modelos são

muito custosos para serem desenvolvidos - com alto grau de detalhamento e conhecimento de características microarquiteturais dos componentes.

Portanto, um modelo de alto nível, baseado exclusivamente em métricas de utilização disponíveis em interfaces do sistema operacional foi utilizado para mensurar o consumo energético. Um monitor distribuído (DUSSO, 2012) foi utilizado para coletar as estatísticas de uso acessíveis, em sistemas *UNIX-like*, atráves do procfs (MOUW, 2001). O monitor segue uma arquitetura mestre/escravo. Os nós escravo coletam as estatísticas de uso do processador, sistema de memória, disco rígido e placa de rede e enviam estas informações ao nó mestre. O nó mestre armazena as informações em um banco de dados relacional. A coleta das estatísticas ocorre a cada 3 segundos.

Definida a metodologia, inicialmente o modelo teria a seguinte forma:

$$P_{TOTAL} = U_{CPU} \cdot TDP_{CPU} + U_{MEM} \cdot TDP_{MEM} + U_{DISK} \cdot TDP_{DISK} + U_{NET} \cdot TDP_{NET}$$

Este modelo assume que os principais componentes tem um comportamento de consumo energético proporcional a seus níveis de utilização. Ou seja, este modelo supõe que os componentes possuem uma alta variação na sua demanda por energia: consomem níveis muito baixos de energia quando estão ociosos e, conforme a utilização dos recursos aumenta, o consumo energético aumenta linearmente até o máximo, quando os componentes estão operando na sua capacidade total. Um componente com este comportamento é chamado *energy-proportional* (BARROSO; HÖLZLE, 2007).

Porém, este comportamento de consumo de energia proporcional à utilização não é realidade para a maioria dos componentes. A CPU é o único componente, entre os presentes no modelo, que realmente possui um comportamento de consumo de energia dinâmico dependente da utilização (FAN; WEBER; BARROSO, 2007). Os outros componentes possuem baixa variação no consumo energético, consumindo praticamente a mesma quantidade de energia quando ociosos e na sua capacidade máxima de utilização. Este comportamento se verifica para o sistema de memória (ECONOMOU et al., 2006) (HÄRDER et al., 2011), disco (FAN; WEBER; BARROSO, 2007) e, também, para a interface de rede (SOHAN et al., 2010).

Isto não significa que todo o consumo de energia deve ser explicado pela CPU. A CPU deve representar todo o consumo energético variável - dependente da utilização da mesma. Os outros componentes tem influência no consumo total representado até 40% do consumo de energia de uma máquina (ECONOMOU et al., 2006). Mas todo este consumo é estático, independente da carga sobre o sistema. Assim, a partir destas considerações, o modelo de energia pode ser reescrito da seguinte forma:

$$P_{TOTAL} = U_{CPU} \cdot TDP_{CPU} + K_{MISC}$$

Neste modelo, a CPU possui uma variação na potência dissipada dependente da taxa de utilização - $U_{CPU} \cdot TDP_{CPU}$ - e o restante representa a potência estática consumida por todos os outros componentes - K_{MISC}. Este modelo, apesar da aparente simplicidade, produz resultados precisos (FAN; WEBER; BARROSO, 2007) (RIVOIRE; RANGANATHAN; KOZYRAKIS, 2008).

O termo que explica o consumo de energia da CPU determina que quando esta estiver em utilização máxima o consumo de energia será igual ao TDP especificado pelo processador. Porém, estes valores normalmente são conservadores, calculados pelos fabricantes a partir do pior cenário possível - com todas as unidades da CPU consumindo o

máximo de potência (FAN; WEBER; BARROSO, 2007). Estes valores normalmente não são atingidos pelos processadores e, por esta razão, o TDP não se configura como uma boa opção para o modelo. Assim, outra vez, o modelo deve ser alterado:

$$P_{TOTAL} = U_{CPU} \cdot K_{CPU} + K_{MISC}$$

K_{CPU} e K_{MISC} são os parâmetros do modelo que representam o consumo máximo do processador e o consumo fixo da máquina, respectivamente. Para determinar os valores destes parâmetros foi utilizada uma metodologia proposta em (ECONOMOU et al., 2006). Este método utiliza um processo de calibragem com o suporte de um medidor de energia para extrair as características básicas de consumo energético da máquina e relacioná-las com as métricas de utilização coletadas junto ao sistema operacional. A partir desta calibragem é possível calcular os parâmetros do modelo e este pode ser utilizado para determinar a potência consumida com base nas métricas de utilização extraídas do sistema operacional.

A fase de calibragem consiste em estressar os componentes que devem ser modelados enquanto um medidor de energia captura o consumo e as estatísticas de uso são armazenadas. Para o modelo proposto anteriormente, a calibragem consiste em estressar o processador através de um programa sintético (OSTATIC, 2013) - ou seja, utilizar o processador na sua capacidade máxima - e um período onde a máquina deve ficar ociosa para que a energia estática consumida pela máquina independente da utilização possa ser capturada.

Então, as métricas de utilização são relacionadas ao consumo energético mensurado pelo equipamento através de um programa linear que tem como objetivo minimizar o erro absoluto do modelo de consumo durante as fases de calibragem.

Este programa linear produz como solução um vetor (s) que contém os parâmetros do modelo de consumo (K_{CPU} e K_{MISC}). Os valores de utilização mensurados durante a fase de calibragem são compilados em uma matriz M (U_{CPU} e a constante 1). A potência capturada pelo medidor de energia é armazenada em um vetor P_{MEAS}. Outro vetor, P_{PRED}, contendo os valores calculados pelo modelo para cada instante de tempo durante todas as fases de calibragem é obtido através da multiplicação da matriz M pela solução do programa s.

$$P_{PRED} = M \cdot s$$

Então, um vetor que contém os erros para cada instante de tempo da potência calculada pelo modelo em relação a potência mensurada pelo equipamento é determinado através da seguinte fórmula (i representa o índice dos elementos dos diferentes vetores):

$$\in_i = \frac{P_{PREDi} - P_{MEASi}}{P_{PREDi}}$$

$\in_n$ representa o erro médio para cada fase de calibragem. Como dito antes, o programa linear tem como objetivo minimizar o erro absoluto do modelo de consumo energético. O erro absoluto é definido como o somatório dos erros médios em cada fase da calibragem. Assumindo que existem N fases de calibragem, a função objetivo pode ser descrita como:

$$\min \left(\sum_{1}^{N} (\in_n) \right)$$

Então, em resumo, cada fase de calibragem é executada uma vez. A partir dos valores obtidos em cada fase (consumo energético e estatísticas de utilização) e com o objetivo de minimizar o erro médio em ambas as fases, a solução do programa (s) é variada até que o erro absoluto esteja abaixo de um valor pré-determinado.

Esta metodologia foi empregada para determinar modelos de consumo energético para as máquinas do ambiente de execução dos testes. O erro absoluto máximo foi definido como 5%. Como dito anteriormente, o *cluster* possui 20 máquinas, separadas em três grupos de configurações. Portanto, foram gerados modelos de consumo para cada um destes três grupos. Os modelos são apresentados na Tabela 4.4.

Tabela 4.4: Modelos de consumo

MÁQUINA	*P*
i3	$98.7*U_{CPU} + 87.5$
Pentium 4	$97.0*U_{CPU} + 83.4$
Pentium 4 HT	$107.3*U_{CPU} + 78.9$

Então, durante a caracterização do sistema MapReduce, o consumo energético para cada máquina foi determinado a partir da potência média (P_{AVG}) - calculada a partir de um dos modelos acima - multiplicada pelo tempo de duração do experimento (t). E o consumo de energia total do experimento foi obtido pela soma do consumo energético de cada uma das máquinas. A fórmula abaixo descreve o cálculo total de energia.

$$\mathrm{E} = \sum_{i=1}^{20} (P_{AVG_i} \cdot t)$$

5 TESTES E RESULTADOS

Este capítulo apresenta os resultados obtidos em cada etapa de experimentos executados neste trabalho. A primeira seção relata os resultados para os testes realizados segundo a metodologia $2^K R$. Na segunda seção, os resultados dos experimentos realizados para caracterização dos escalonadores com uma aplicação são detalhados. Por último são apresentados os resultados dos experimentos com os escalonadores no contexto onde múltiplas aplicações executam simultaneamente. Novamente, neste capítulo MapReduce e Hadoop são considerados sinônimos.

5.1 $2^K R$

No primeiro estágio de testes, foram executados experimentos contendo todas as combinações possíveis de configuração para os fatores listados na Tabela 4.3. O objetivo dos experimentos era determinar, para cada combinação aplicação-escalonador de tarefas, quais fatores têm maior influência sobre o consumo de energia e o desempenho do *framework* MapReduce através da metodologia $2^K R$. Todos os experimentos desta etapa tiveram uma entrada de 1 GB. O nível de replicação dos experimentos foi 10. Portanto, para cada combinação aplicação-escalonador, 1280 ($2^7 10$) execuções MapReduce foram realizadas, totalizando 5120 (1280*4) experimentos neste estágio de testes. Primeiramente, serão apresentados os resultados para a aplicação *CPU bound* e em seguida serão apresentados os resultados para a aplicação *IO bound*.

5.1.1 Aplicação *CPU Bound*

A Tabela 5.1 apresenta os resultados produzidos através da metodologia $2^K R$ para a aplicação *CPU bound* (APACHE, 2011). Lado a lado, nesta ordem, são exibidos os efeitos de cada fator primário sobre o consumo de energia e sobre o desempenho do MapReduce com o escalonador de tarefas FIFO, e os efeitos de cada fator primário sobre o consumo de energia e desempenho com o escalonador HFS. Somente estão expostas as interações entre os fatores que têm efeitos sobre o comportamento da variável de resposta próximos ou superiores a 1% - as interações que não estão presentes na Tabela têm efeito sobre as variáveis de resposta nulo ou muito próximo de zero.

Os resultados para a aplicação *CPU bound* com o escalonador FIFO demonstram que o parâmetro de configuração *mapred.compress.map.output* - responsável por determinar se os dados intermediários devem ser comprimidos antes da transferência para os nós responsáveis pela execução da fase *reduce* - tem efeito predominante na variação do consumo de energia (67.396%) e do desempenho (54.284%). Considerando em separado o consumo energético da aplicação *CPU bound* com o escalonador FIFO, somente a com-

Tabela 5.1: Efeitos aplicação *CPU bound*

CPU Bound	FIFO		HFS	
Parâmetro	**Efeito (Energia)**	**Efeito (Desempenho)**	**Efeito (Energia)**	**Efeito (Desempenho)**
io.sort.mb	0.263%	0.103%	3.914%	3.345%
io.sort.spill.percent	0.001%	5.441%	1.343%	0.153%
mapred.reduce.tasks	0.007%	5.023%	0.135%	1.928%
mapred.reduce.slowstart.completed.maps	0.462%	2.078%	0.000%	0.216%
mapred.job.shuffle.input.buffer.percent	0.003%	0.002%	0.001%	0.003%
mapred.job.shuffle.merge.percent	0.000%	0.002%	0.000%	0.000%
mapred.compress.map.output	67.396%	54.284%	85.442%	88.33%
...				
io.sort.mb | mapred.compress.map.output	1.714%	1.374%	1.365%	1.489%
mapred.reduce.tasks | mapred.reduce.slowstart.completed.maps	0.992%	0.972%	0.530%	1.037%
...				
erros experimentais	27.975%	29.301%	6.215%	2.963%

pressão de dados tem influência significativa. Nenhum dos outros parâmetros é responsável por mais do que 0.5% do comportamento desta variável de resposta. O tempo de execução, por sua vez, possui outros parâmetros com uma influência significativa - ainda que em grau muito menor - sobre sua variação além de *mapred.compress.map.output*, ainda que em uma escala muito inferior. São eles: *io.sort.spill.percent* (5.441%), *mapred.reduce.tasks* (5.023%) e *mapred.reduce.slowstart.completed.maps* (2.078%).

Ainda sobre os resultados do escalonador FIFO, erros experimentais - influência de erros de experimentação e parâmetros não controlados - tiveram uma influência considerável na variação do consumo energético (27.975%) e do desempenho (29.301%). Os parâmetros do MapReduce foram devidamente controlados e isolados. Erros de experimentação também estão descartados: os testes foram realizados conforme define a metodologia (JAIN, 1991) - com todas as combinações de níveis para todos os fatores. Por isto, acredita-se que os resultados podem ter sido influenciados por outras aplicações que foram disparadas durante o experimento - por outros usuários do *cluster* ou pelo próprio sistema operacional - ou por falhas de *hardware*.

Os resultados da aplicação *CPU bound* com escalonamento realizado pelo HFS demonstram novamente influência predominante da compressão de dados entre as diferentes fases da computação sobre o consumo energético (85.442%) e o desempenho (88.33%). Destaca-se ainda o tamanho do *buffer* de saída das tarefas *map* (io.sort.mb) com uma influência relevante, ainda que pequena, sobre as variáveis de resposta - energia (3.914%) e desempenho (3.345%). Erros experimentais tiveram baixa influência sobre os resultados da aplicação *CPU bound* com o escalonador HFS - 6.215% para o consumo energético e 2.963% sobre o desempenho.

Com intuito de melhor ilustrar os resultados apresentados acima, a Tabela 5.2 exibe os dez menores e os dez maiores consumos de energia para ambos escalonadores - discriminados pela combinação dos níveis dos diferentes fatores e acompanhados dos coeficientes de variação para cada experimento. Então, a coluna *Experimento* deve ser lida da seguinte forma: nível *io.sort.mb* | nível *io.sort.spill.percent* | nível *mapred.reduce.tasks* | nível *mapred.reduce.slowstart.completed.maps* | nível *mapred.job.shuffle.input.buffer.percent* | nível *mapred.job.shuffle.merge.percent* | nível *mapred.compress.map.output*.

A partir da Tabela 5.2 destacam-se três situações relevantes:

- Sob ambas políticas de escalonamento, a aplicação *CPU bound* apresenta menor consumo de energia quando a compressão de dados (*mapred.compress.map.output*), fator mais influente para ambas variáveis de resposta avaliadas, não é realizada. Este fato pode ser explicado devido ao modelo de consumo energético empregado.

Tabela 5.2: Melhores e piores resultados para consumo de energia da aplicação *CPU bound*

CPU Bound | FIFO			CPU Bound | HFS		
Experimento	**Energia [kJ]**	**COV**	**Experimento**	**Energia [kJ]**	**COV**
64|0.75|1|0.05|0.3|0.66|false	276.841	0.018	**128|0.95|1|0.05|0.9|0.9|false**	402.744	0.017
64|0.75|1|0.05|0.9|0.66|false	277.009	0.015	**128|0.95|1|0.05|0.3|0.66|false**	404.344	0.014
64|0.75|1|0.05|0.9|0.9|false	277.444	0.022	**128|0.95|1|0.05|0.3|0.9|false**	405.988	0.026
64|0.75|1|0.05|0.3|0.9|false	277.577	0.012	**128|0.95|1|0.05|0.9|0.66|false**	407.917	0.033
64|0.95|1|0.05|0.3|0.66|false	277.859	0.011	**128|0.75|1|0.05|0.9|0.66|false**	409.158	0.025
64|0.95|1|0.05|0.9|0.9|false	280.132	0.017	**64|0.95|1|0.05|0.9|0.66|false**	409.263	0.027
64|0.95|1|0.05|0.3|0.9|false	281.474	0.014	**128|0.75|1|0.05|0.3|0.9|false**	410.07	0.027
128|0.75|1|0.05|0.9|0.66|false	281.625	0.029	**64|0.95|1|0.05|0.3|0.66|false**	411.572	0.019
64|0.95|1|0.05|0.9|0.66|false	281.767	0.024	**64|0.95|1|1.0|0.9|0.9|false**	412.142	0.023
128|0.75|1|0.05|0.9|0.9|false	283.213	0.030	**128|0.95|1|1.0|0.3|0.9|false**	412.492	0.013
...			**...**		
64|0.95|32|0.05|0.3|0.9|true	342.827	0.060	**64|0.75|32|1.0|0.3|0.9|true**	540.746	0.021
64|0.95|1|1.0|0.3|0.9|true	343.107	0.077	**64|0.75|1|1.0|0.3|0.66|true**	542.219	0.018
64|0.75|1|1.0|0.9|0.9|true	343.480	0.081	**64|0.75|1|0.05|0.9|0.66|true**	542.723	0.033
64|0.95|1|1.0|0.9|0.66|true	343.712	0.077	**64|0.75|32|0.05|0.3|0.66|true**	543.775	0.056
64|0.95|1|1.0|0.9|0.9|true	344.104	0.076	**64|0.75|1|1.0|0.3|0.9|true**	545.900	0.020
64|0.95|1|1.0|0.3|0.66|true	344.566	0.080	**64|0.75|32|1.0|0.3|0.66|true**	548.587	0.025
64|0.75|32|0.05|0.9|0.9|true	344.893	0.070	**64|0.75|32|0.05|0.9|0.66|true**	549.348	0.024
64|0.75|1|1.0|0.3|0.66|true	345.865	0.085	**64|0.75|32|0.05|0.9|0.9|true**	549.543	0.024
64|0.75|1|1.0|0.3|0.9|true	346.020	0.083	**64|0.75|32|0.05|0.3|0.9|true**	552.340	0.027
64|0.75|1|1.0|0.9|0.66|true	346.570	0.089	**64|0.75|1|1.0|0.9|0.66|true**	560.069	0.028

A compressão implica em uma utilização maior do processador e em menor comunicação entre nós. Como o processador é o único componente a exibir um comportamento de consumo dinâmico, dependente de sua utilização, uma maior carga de trabalho sobre ele resulta em maior consumo de energia. E este maior consumo do processador não é compensado pela menor comunicação entre os nós, visto que a interface de rede possui um comportamento estático no consumo energético - consumindo sempre a mesma quantidade de energia.

- Os experimentos com escalonador de tarefas FIFO, especialmente aqueles com os piores resultados, apresentam coeficiente de variação significativamente superior aos experimentos realizados com HFS. Este fato melhor ilustra os resultados anteriores, onde os erros experimentais foram importantes fatores no comportamento das variáveis de resposta sob a política de escalonamento FIFO.

- Comparativamente, o consumo de energia do escalonador FIFO é inferior ao consumo de energia do HFS. O menor resultado para HFS é aproximadamente 45% superior ao menor consumo energético registrado para o FIFO. E o pior resultado do escalonador HFS é cerca de 61% maior do que o pior resultado do escalonador padrão do MapReduce. A segunda etapa de experimentos (seção 5.2) provê um melhor entendimento do comportamento comparado dos dois escalonadores.

5.1.2 Aplicação *IO Bound*

A Tabela 5.3 apresenta os resultados produzidos através da metodologia $2^K R$ para a aplicação *IO bound* (APACHE, 2009). Os efeitos sobre a variação do consumo de energia e desempenho para os escalonadores FIFO e HFS estão expostos lado a lado. Novamente, somente estão expostas as interações entre os fatores que têm efeitos sobre o comportamento da variável de resposta próximos ou superiores a 1% - as interações que não estão presentes na tabela têm efeito sobre as variáveis de resposta nulo ou muito próximo de zero.

Tabela 5.3: Efeitos aplicação *IO bound*

IO Bound	FIFO		HFS	
Parâmetro	**Efeito (Energia)**	**Efeito (Desempenho)**	**Efeito (Energia)**	**Efeito (Desempenho)**
io.sort.mb	0.094%	0.015%	0.482%	0.147%
io.sort.spill.percent	0.042%	0.030%	0.007%	0.005%
mapred.reduce.tasks	87.411%	92.631%	77.765%	87.595%
mapred.reduce.slowstart.completed.maps	0.373%	0.365%	0.442%	0.438%
mapred.job.shuffle.input.buffer.percent	0.001%	0.001%	0.000%	0.001%
mapred.job.shuffle.merge.percent	0.006%	0.003%	0.001%	0.000%
mapred.compress.map.output	9.156%	4.216%	15.527%	7.006%
...				
mapred.reduce.tasks | mapred.compress.map.output	0.910%	1.262%	0.709%	1.150%
...				
erros experimentais	1.580%	1.208%	4.295%	3.223%

Os resultados determinam que para ambas políticas de escalonamento somente dois fatores são relevantes. O número de tarefas reduce (*mapred.reduce.tasks*) é o fator com maior influência sobre as variáveis de resposta - sendo responsável por aproximadamente 87% da variação no consumo energético e 92% da variação no desempenho quando a política de escalonamento é FIFO; e, também, respondendo por 77.765% do consumo de energia e 87.595% do desempenho quando o escalonador de tarefas é o HFS. O outro fator relevante em aplicações *IO bound* é compressão de dados (*mapred.compress.map.output*) - 9.156% de efeito sobre o consumo de energia e 4.216% de efeito sobre o desempenho no escalonador FIFO; e, para o escalonador HFS, 15.527% de efeito sobre o consumo energético e aproximadamente 7% de efeito sobre o tempo de execução da aplicação. Os erros experimentais não contribuíram significativamente para definir o comportamento das variáveis de resposta da aplicação *IO bound* em nenhum dos escalonadores de tarefas.

Mais uma vez, com o objetivo de melhor compreender os resultados, os dez menores consumos de energia e os dez maiores consumos de energia da aplicação *IO bound* são apresentados na Tabela 5.4. Os experimentos são designados pela combinação de níveis para cada fator. Outra vez, a coluna *Experimento* deve ser lida da seguinte forma: nível *io.sort.mb* | nível *io.sort.spill.percent* | nível *mapred.reduce.tasks* | nível *mapred.reduce.slowstart.completed.maps* | nível *mapred.job.shuffle.input.buffer.percent* | nível *mapred.job.shuffle.merge.percent* | nível *mapred.compress.map.output*. O coeficiente de variação de cada experimento é apresentado, também.

Os resultados apresentados na Tabela 5.4 permitem levantar os seguintes pontos:

- Assim como ocorreu com a aplicação CPU bound, a ausência de compressão apresenta melhores resultados para ambos escalonadores. Porém, o efeito deste parâmetro é muito inferior.

- O número de tarefas *reduce*, fator responsável por grande parte da variação do consumo de energia e tempo de execução nos experimentos, apresenta melhores resultados com limite superior de configuração (32). Ou seja, para a aplicação *IO bound*, o processamento paralelo das saídas das tarefas *map*, realizado por 32 tarefas *reduce* concorrentemente, proporciona uma diminuição do consumo de energia quando comparado ao processamento sequencial realizado por somente uma tarefa *reduce*.

- Diferentemente do que foi constatado nos experimentos com a aplicação *CPU bound*, não existe uma diferença significativa no consumo de energia de ambos escalonadores (o menor consumo com o escalonador HFS é aproximadamente 4% superior ao menor consumo obtido com o escalonador FIFO; e o maior consumo com

Tabela 5.4: Melhores e piores resultados para consumo de energia da aplicação *IO bound*

IO Bound | FIFO			IO Bound | HFS		
Experimento	**Energia [kJ]**	**COV**	**Experimento**	**Energia [kJ]**	**COV**
128|0.75|32|0.05|0.9|0.66|false	153.591	0.035	**128|0.75|32|0.05|0.9|0.66|false**	160.013	0.034
64|0.75|32|0.05|0.9|0.66|false	154.072	0.033	**128|0.95|32|0.05|0.9|0.66|false**	162.066	0.038
64|0.95|32|0.05|0.9|0.66|false	154.656	0.040	**128|0.95|32|0.05|0.9|0.9|false**	162.573	0.034
64|0.75|32|0.05|0.3|0.9|false	155.037	0.037	**128|0.75|32|0.05|0.9|0.9|false**	162.648	0.024
128|0.95|32|0.05|0.9|0.66|false	155.073	0.048	**128|0.75|32|0.05|0.3|0.9|false**	163.444	0.033
128|0.95|32|0.05|0.9|0.9|false	155.572	0.032	**128|0.95|32|0.05|0.3|0.9|false**	165.180	0.027
128|0.75|32|0.05|0.9|0.9|false	156.620	0.045	**128|0.75|32|0.05|0.3|0.66|false**	166.729	0.050
128|0.75|32|0.05|0.3|0.9|false	157.204	0.056	**64|0.75|32|0.05|0.9|0.66|false**	168.105	0.018
64|0.75|32|0.05|0.9|0.9|false	157.321	0.059	**128|0.95|32|0.05|0.3|0.66|false**	169.345	0.040
128|0.95|32|0.05|0.3|0.9|false	157.740	0.039	**64|0.95|32|0.05|0.3|0.9|false**	169.677	0.022
...			**...**		
128|0.75|1|1.0|0.3|0.9|true	523.680	0.029	**128|0.75|1|0.05|0.9|0.66|true**	541.651	0.253
64|0.75|1|1.0|0.3|0.9|true	525.485	0.018	**64|0.95|1|1.0|0.3|0.66|true**	550.277	0.021
128|0.95|1|1.0|0.9|0.66|true	525.894	0.037	**64|0.95|1|1.0|0.3|0.9|true**	557.225	0.023
128|0.95|1|1.0|0.3|0.66|true	527.764	0.047	**64|0.75|1|1.0|0.3|0.9|true**	557.262	0.015
128|0.75|1|1.0|0.3|0.66|true	531.485	0.038	**128|0.75|1|1.0|0.9|0.66|true**	558.974	0.244
64|0.95|1|1.0|0.9|0.66|true	546.302	0.040	**64|0.95|1|1.0|0.9|0.66|true**	560.354	0.026
64|0.95|1|1.0|0.9|0.9|true	547.031	0.053	**64|0.95|1|1.0|0.9|0.9|true**	560.386	0.023
64|0.95|1|1.0|0.3|0.9|true	549.671	0.043	**64|0.75|1|1.0|0.9|0.66|true**	561.514	0.026
64|0.95|1|1.0|0.3|0.66|true	556.586	0.041	**64|0.75|1|1.0|0.9|0.9|true**	566.781	0.020
64|0.75|1|1.0|0.3|0.66|true	564.038	0.244	**64|0.75|1|1.0|0.3|0.66|true**	597.562	0.226

o escalonador HFS é aproxidamente 6% superior ao maior consumo obtido com o escalonador FIFO).

5.1.3 Discussão

Ao analisar o conjunto de resultados desta primeira etapa de testes é possível perceber duas tendências. Primeiro, existe uma similaridade muito evidente entre os efeitos de cada fator para o consumo de energia e os efeitos de cada fator para o desempenho do sistema MapReduce. Porém, algumas pequenas diferenças são visíveis. Isto se deve ao fato de que o consumo de energia é determinado por duas variáveis: tempo de execução (responsável pela grande similaridade) e utilização dos processadores que compõem o ambiente de execução (que explica as pequenas variações existentes entre os efeitos de ambas variáveis de resposta).

A segunda tendência diz respeito a similaridade dos resultados para os efeitos sobre as variáveis de resposta entre os dois escalonadores de tarefas. Ainda que existam diferenças, em linhas gerais os parâmetros mais influentes para cada escalonador em ambas aplicações testadas - e, também, o grau de influência destes parâmetros - são semelhantes. Assim, mesmo que os valores mensurados para o consumo energético sejam diferentes - como as Tabelas 5.3 e 5.4 evidenciam - a avaliação determina que os fatores considerados impõem variações parecidas para ambas políticas de escalonamento do MapReduce.

5.2 Testes dos Escalonadores com Uma Aplicação

Na segunda etapa de testes, foram executados experimentos para melhor caracterizar os escalonadores de tarefas do MapReduce em um contexto onde somente uma aplicação está em execução. O objetivo deste estágio é estabelecer se existem diferenças no comportamento dos escalonadores em uma situação como esta. Os experimentos foram executados para ambas aplicações (CPU bound e IO bound) e com três diferentes tamanhos de entrada: 256 MB, 4 GB e 10 GB. Estes diferentes tamanhos de entrada serão referenciados nesta seção como pequeno (256 MB), médio (4 GB) e grande (10 GB). Cada

experimento foi repetido 10 vezes. Portanto foram executados 120 testes nesta etapa ((2 escalonadores)*(2 aplicações)*(3 tamanhos de entrada)*(10 replicações)).

No primeiro momento, serão apresentados resultados para a aplicação *CPU bound*. Em seguida, os resultados para a aplicações *IO bound* são exibidos e discutidos.

5.2.1 Aplicação *CPU bound*

Os resultados de consumo energético para aplicação *CPU bound* (APACHE, 2011) são apresentados na Tabela 5.5. A tabela contém os valores médios do consumo de energia para os três tamanhos de entrada testados em ambos escalonadores. São apresentados a média de consumo, o coeficiente de variação do experimentos e o intervalo de confiança para a média (calculado em um nível de confiança de 95%).

Tabela 5.5: Consumo de energia da aplicação *CPU bound*

CPU BOUND	FIFO			HFS		
ENTRADA	ENERGIA [kJ]	COV	INTERVALO DE CONFIANÇA (@95%)	ENERGIA [kJ]	COV	INTERVALO DE CONFIANÇA (@95%)
PEQUENA	203.647	0.074	[192.933, 214.359]	236.068	0.202	[201.956, 270.179]
MÉDIA	1242.921	0.021	[1224.024, 1261.817]	1155.395	0.022	[1137.013, 1173.775]
GRANDE	2495.484	0.026	[2449.282, 2541.685]	2543.712	0.022	[2503.563, 2583.860]

Comparando os valores médios de consumo para cada tamanho de entrada percebe-se que existem diferenças nos consumos de energia de ambos escalonadores. Com as entradas pequena e grande, HFS consome mais energia do que o escalonador FIFO - aproximadamente 15% a mais com a entrada pequena e somente 1.93% com a entrada grande. Apesar das diferenças de médias de consumo de energia para estas entradas, estatisticamente os consumos energéticos devem ser considerados equivalentes, por que os intervalos de confiança de ambos escalonadores se sobrepõem. No caso da entrada pequena, os 15% de diferença entre as médias não confirmam uma real desigualdade estatística devido a alta variação no experimentos com o escalonador HFS.

A entrada média registra uma inversão no comportamento médio do consumo de energia: o escalonador HFS tem consumo médio inferior ao FIFO ($\approx 7.5\%$). E para este tamanho de entrada é possível afirmar estatisticamente que os escalonadores tem comportamento diferente - não há sopreposição dos intervalos de confiança para as médias de consumo de energia. Portanto, o escalonador HFS consome menos energia do que o escalonador FIFO com o tamanho de entrada médio.

A Tabela 5.6 apresenta os resultados para o tempo de execução das aplicações *CPU bound* neste conjunto de testes.

Tabela 5.6: Desempenho da aplicação *CPU bound*

CPU BOUND	FIFO			HFS		
ENTRADA	DESEMPENHO [s]	COV	INTERVALO DE CONFIANÇA (@95%)	DESEMPENHO [s]	COV	INTERVALO DE CONFIANÇA (@95%)
PEQUENA	108.100	0.077	[102.119, 114.08]	128.400	0.208	[109.319, 147.48]
MÉDIA	375.500	0.014	[371.848, 379.151]	381.700	0.024	[375.065, 388.334]
GRANDE	734.200	0.011	[728.363, 740.036]	770.500	0.035	[751.336, 789.663]

O desempenho dos escalonadores é estatisticamente equivalente para as entradas pequena e média - apesar do tempo de execução média do escalonador FIFO ser inferior nos dois casos, os intervalos de confiança se confudem, refutando a possibilidade de afirmar

que um dos escalonadores tem desempenho superior ao outro com estas entradas. Porém, com o tamanho de entrada maior é possível fazer tal afirmação. O escalonador FIFO tem média de tempo de execução inferior ao escalonador HFS sem que os intervalos de confiança se sobreponham. Portanto, neste caso - tamanho de entrada grande, o desempenho FIFO foi superior ao HFS por uma pequena margem ($\approx 4.9\%$).

A proposta do escalonador HFS tem dois objetivos: criar uma política de escalonamento justa - dividindo os recursos computacionais entre as diferentes aplicações e usuários; e aumentar a taxa de tarefas executadas localmente. Por esta razão, para avaliar e melhor caracterizar as políticas de escalonamento, a Tabela 5.7 apresenta os valores para a taxa de tarefas *map data-local* - ou seja, a taxa de tarefas map executadas em uma máquina onde os dados a serem processados estão armazenados.

Tabela 5.7: Localidade das tarefas *map* da aplicação *CPU bound*

CPU BOUND	FIFO			HFS		
ENTRADA	TAREFAS *map* DATA-LOCAL	COV	INTERVALO DE CONFIANÇA (@95%)	TAREFAS *map* DATA-LOCAL	COV	INTERVALO DE CONFIANÇA (@95%)
PEQUENA	17.50%	0.967	[5.396%, 29.603%]	98%	0.061	[93.707%, 100%]
MÉDIA	85.52%	0.023	[84.130%, 86.908%]	93.51%	0.028	[91.652%, 95.359%]
GRANDE	95.36%	0.010	[94.654%, 96.057%]	96.41%	0.012	[95.613%, 97.206%]

A partir destes dados, conclui-se que o escalonador HFS atinge uma maior taxa de tarefas executadas localmente com os tamanhos de entrada pequeno e médio. Em especial, no caso da entrada pequena, a diferença é muito significativa (mesmo considerando a alta dispersão das taxas dos experimentos com o escalonador FIFO) - 17.5% com o escalonador FIFO e 98% com o escalonador HFS. A diferença entre a taxa média de tarefas locais decai conforme o tamanho da entrada aumenta. Ainda existe uma diferença significativa para o tamanho de entrada médio - 85.52% com o escalonador FIFO e 93.51% no HFS. Porém, com o tamanho de entrada maior esta diferença se reduz ao ponto de se tornar insignificante estatisticamente - a taxa média de tarefas *map* executadas localmente no HFS é superior, porém, os intervalos de confiança para as médias se intercalam.

5.2.2 Aplicação *IO bound*

Os resultados de consumo de energia da aplicação IO bound (APACHE, 2009) para ambas políticas de escalonamento são apresentados na Tabela 5.8. Os dados estão estruturados seguindo a mesma lógica da seção anterior.

Tabela 5.8: Consumo de energia da aplicação *IO bound*

IO BOUND	FIFO			HFS		
ENTRADA	ENERGIA [kJ]	COV	INTERVALO DE CONFIANÇA (@95%)	ENERGIA [kJ]	COV	INTERVALO DE CONFIANÇA (@95%)
PEQUENA	96.801	0.072	[91.796, 101.805]	100.350	0.065	[95.708, 104.992]
MÉDIA	530.700	0.023	[522.083, 539.316]	526.852	0.033	[514.422, 539.280]
GRANDE	1128.011	0.030	[1103.562, 1152.460]	1166.139	0.030	[1141.350, 1190.926]

Os resultados apontam que não existe diferença estatística para o consumo de energia da aplicação *IO bound* entre os escalonadores de tarefas em nenhuma das entradas testadas. Os intervalos de confiança para as médias de consumo energético se sobrepõem

nas três situações. Por isto, o consumo de energia dos escalonadores para a aplicação *IO bound* é equivalente nas condições testadas.

A caracterização prossegue com os valores obtidos para o desempenho da aplicação *IO bound* nesta etapa de testes. Os resultados estão expostos na Tabela 5.9.

Tabela 5.9: Desempenho da aplicação *IO bound*

IO BOUND	FIFO			HFS		
ENTRADA	DESEMPENHO [s]	COV	INTERVALO DE CONFIANÇA (@95%)	DESEMPENHO [s]	COV	INTERVALO DE CONFIANÇA (@95%)
PEQUENA	42.000	0.089	[39.323, 44.676]	45.400	0.088	[42.552, 48.247]
MÉDIA	205.600	0.033	[200.788, 210.411]	211.400	0.036	[205.884, 216.915]
GRANDE	435.600	0.024	[428.157, 443.042]	467.400	0.028	[457.946, 476.853]

Os resultados para os tempos de execução da aplicação *IO Bound* são semelhantes aos apresentados pela aplicação *CPU bound*. Somente com a maior entrada houve diferenças estatísticas no comportamento do desempenho entre os escalonadores FIFO e HFS. Outra vez, para este tamanho de entrada, FIFO teve desempenho superior (HFS foi, em média, 7.3% mais lento). Com a entrada pequena e média, os tempo de execução médios registrados para ambos escalonadores não são estatisticamente diferentes.

Por fim, a Tabela 5.10 apresenta os valores para a taxa de tarefas *map* executadas em uma máquina onde os dados a serem processados estão armazenados.

Tabela 5.10: Localidade das tarefas *map* da aplicação *IO bound*

IO BOUND	FIFO			HFS		
ENTRADA	TAREFAS *map* DATA-LOCAL	COV	INTERVALO DE CONFIANÇA (@95%)	TAREFAS *map* DATA-LOCAL	COV	INTERVALO DE CONFIANÇA (@95%)
PEQUENA	77.06%	0.105	[71.273%, 82.843%]	100%	0.000	[100%, 100%]
MÉDIA	90.77%	0.037	[88.387%, 93.142%]	97.70%	0.019	[96.388%, 99.01%]
GRANDE	95.14%	0.014	[94.209%, 96.059%]	98.91%	0.005	[98.563%, 99.246%]

O escalonador HFS tem uma taxa de localidade de tarefas map estatisticamente superior para todos tamanhos de entrada testados. HFS mantém um alto índice de tarefas map locais - a aplicação IO bound que processa a entrada de 256 MB teve 100% de tarefas locais em todas as replicações do experimento e, ainda, a menor média de tarefas locais foi de 97.5% (com um baixo coeficiente de variação - 0.019). O escalonador FIFO, outra vez, obteve taxas crescentes conforme o tamanho da entrada aumentou.

5.2.3 Discussão

Os experimentos realizados aqui determinam que no contexto apresentado - escalonamento de somente uma aplicação para o processamento de entradas com 256 MB, 4 GB e 10 GB - os escalonadores de tarefas não possuem comportamentos estatisticamente diferentes quanto ao consumo energético e ao desempenho na maioria das situações.

O consumo energético difere somente com a entrada média na aplicação *CPU bound*. Em todas as outras combinações aplicação-entrada, os escalonadores apresentam resultados equivalentes para o consumo de energia. O desempenho difere em duas situações: ambas aplicações possuem melhor desempenho com o escalonador FIFO ao processar a entrada de 10 GB. Porém, é importante ressaltar que a diferença em ambos os casos não é de magnitude muito elevada - FIFO tem desempenho 4.9% superior com a aplicação *CPU bound* e aproximadamente 7.3% para a aplicação *IO bound*.

A terceira variável de resposta analisada nesta etapa - taxa de tarefas *map* executadas localmente - estabeleceu que para esta métrica existe uma diferença significativa entre as políticas de escalonamento do MapReduce. HFS obteve altos índices de tarefas *map* executadas localmente - com médias superiores a 93% para todas as combinações de aplicação-entrada. Somente em um dos casos - escalonamento da aplicação *CPU bound* para processamento da entrada de 10 GB - o escalonador FIFO teve uma taxa de tarefas *map* locais estatisticamente equivalente à taxa obtida pelo escalonador HFS. Porém, apesar de alcançar um de seus objetivos (maior localidade no escalonamento de tarefas *map*), HFS não obtém melhor desempenho ou menor consumo energético no contexto avaliado nesta seção.

Por estas razões, não é possível estabelecer que um escalonador é superior ao outro para o escalonamento de uma aplicação que executa sem concorrência com outras requisições ao sistema MapReduce.

Estes resultados contradizem os valores encontrados na primeira etapa de experimentos, onde nos testes com a aplicação *CPU bound*, os escalonadores apresentaram diferenças consideráveis no consumo energético ($\approx 60\%$, em alguns casos). Esta situação, combinada com a alta influência dos erros experimentais no projeto experimental realizado anteriormente, reforçam o argumento de que os testes $2^K R$ da aplicação *CPU bound* sofreram influências de variações externas não controladas.

5.3 Testes dos Escalonadores com Múltiplas Aplicações

Na terceira e última etapa de testes, foram executados experimentos para caracterizar os escalonadores de tarefas do MapReduce em um contexto onde diversas aplicações são submetidas para execução. O objetivo deste estágio é estabelecer se existem diferenças no comportamento dos escalonadores em uma situação onde diversos usuários (ou diversas aplicações) concorrem pelos recursos.

Os experimentos consistiam na submissão simultânea de 20 aplicações MapReduce com diferentes tamanhos de entrada (novamente, os tamanhos utilizados foram 256 MB, 4 GB e 10 GB). As aplicações executadas separam esta etapa em três partes. Primeiramente, as 20 aplicações submetidas eram a aplicação *CPU bound*. Na segunda parte, as 20 aplicações submetidas eram a aplicação MapReduce *IO bound*. Para estas duas *workloads*, a distribuição das entradas a serem executadas foi a seguinte: 14 aplicações processavam a entrada de 256 MB; 5 aplicações processavam a entrada de 4 GB; e 1 aplicação processava a entrada de 10 GB. A terceira parte desta etapa de experimentação consisitia na execução de 10 aplicações MapReduce *CPU bound* e 10 aplicações MapReduce *IO bound*. Esta *workload* heterogênea, teve a seguinte distribuição das entradas a serem processadas: 14 aplicações processavam a entrada de 256 MB (7 *CPU bound*, 7 *IO bound*); 4 aplicações processavam a entrada de 4 GB (2 *CPU bound*, 2 *IO bound*); e 2 aplicações processavam a entrada de 10 GB (1 *CPU bound*, 1 *IO bound*). A ordem de submissão das aplicações em todos os experimentos foi a seguinte: primeiro foram submetidas as aplicações que devem processar as entradas de 10 GB; em seguida as aplicações que devem processar as entradas de 4 GB; e, por último, as aplicações que devem processar as entradas de 256 MB.

Cada uma destas três *workloads* foi executada para ambos escalonadores de tarefas em avaliação. Os experimentos para cada combinação workload-escalonador foram repetidos 10 vezes. Assim foram executadas 1200 aplicações MapReduce neste estágio de experimentos ((2 escalonadores)*(3 *workloads*)*(20 aplicações)*(10 replicações)).

O restante desta seção está estruturado da seguinte forma: no primeiro momento, são exibidos os resultados para a *workload CPU bound*; em seguida, os resultados para a *workload IO bound* são apresentados; os resultados para a *workload heterogênea* estão expostos na sequência; por fim, uma breve discussão a cerca dos resultados obtidos nesta etapa de experimentos encerra esta seção.

5.3.1 *Workload CPU Bound*

A Tabela 5.11 apresenta os resultados do consumo energético, tempo de execução e taxa de tarefas *map* executadas localmente para a *workload CPU bound*. Os resultados para cada escalonador de tarefas MapReduce estão dispostos lado a lado. Para cada métrica avaliada são apresentadas a média, o coeficiente de variação e o intervalo de confiança (calculados com nível de confiança de 95%) para a média nas 10 replicações realizadas.

Tabela 5.11: Resultados da *workload CPU bound*

CPU Bound	FIFO			HFS		
MÉTRICA	MÉDIA	COV	INTERVALO DE CONFIANÇA (@95%)	MÉDIA	COV	INTERVALO DE CONFIANÇA (@95%)
ENERGIA [kJ]	8892.224	0.015	[8793.632, 8990.814]	8028.026	0.003	[8008.528, 8047.523]
DESEMPENHO [s]	2545.800	0.016	[2516.144, 2575.455]	2332.100	0.007	[2320.479, 2343.720]
TAREFAS *map* DATA-LOCAL	86.20%	0.011	[85.55%, 86.85%]	97.40%	0.012	[96.59%, 98.28%]

Os resultados para consumo de energia definem que HFS teve um consumo médio inferior ao consumo do escalonador FIFO. HFS consumiu aproximadamente 8028 kJ em cada repetição do experimento, enquanto que o escalonador FIFO consumiu uma média de 8892.224 kJ em cada replicação - ou seja, uma diferença de 10.76%. O desempenho médio - tempo para execução de todas as 20 aplicações submetidas - também foi menor quando o escalonamento era realizado pelo HFS (2332.1 segundos) do que quando o escalonamento foi realizado pelo escalonador FIFO (2545.8 segundos) - aproximadamente, um tempo de processamento 9.16% maior para o escalonador de tarefas FIFO. E, ainda, o menor consumo de energia e melhor desempenho do escalonador HFS foram obtidos com uma taxa de localidade nas tarefas *map* superior. 97.4% das tarefas *map* escalonadas pelo HFS foram escalonadas em máquinas que armazenavam o bloco de dados a ser processado pela tarefa, ao passo que o escalonador FIFO obteve uma taxa de localidade no escalonamento das tarefas *map* de 86.2%.

O coeficiente de variação das três métricas teve resultados baixos nos experimentos de ambos escalonadores (o maior coeficiente de variação registrado foi de 0.016), o que permite determinar - devido a intervalos de confiança estreitos, que não se sobrepõem em nenhum dos casos - que houve diferença estatística entre os dois escalonadores nas três métricas avaliadas. Portanto, em resumo, nas condições definidas para esta etapa de experimentos, o escalonador de tarefas HFS faz melhor uso dos recursos disponíveis para executar as requisições feitas ao MapReduce - consumindo menos energia e finalizando a computação em menor tempo.

5.3.2 *Workload IO Bound*

Os resultados do consumo energético, tempo de execução e taxa de tarefas *map* executadas localmente para a *workload IO bound* para ambos escalonadores de tarefas MapRe-

duce estão exibidos na Tabela 5.12. Os resultados estão dispostos conforme a estrutura da seção anterior.

Tabela 5.12: Resultados da *workload IO bound*

IO Bound	FIFO			HFS		
MÉTRICA	MÉDIA	COV	INTERVALO DE CONFIANÇA (@95%)	MÉDIA	COV	INTERVALO DE CONFIANÇA (@95%)
ENERGIA [kJ]	4462.439	0.022	[4393.445, 4531.431]	4369.253	0.025	[4291.533, 4446.973]
DESEMPENHO [s]	1574.500	0.032	[1538.020, 1610.979]	1661.500	0.010	[1649.388, 1673.611]
TAREFAS *map* DATA-LOCAL	81.00%	0.012	[80.267%, 81.7%]	98.70%	0.004	[98.387%, 98.945%]

Para a *workload IO bound*, não é possível diferenciar estatiscamente o consumo de energia dos escalonadores. Apesar da diferença de 2%, aproximadamente, os intervalos de confiança de ambos escalonadores para esta métrica se sobrepõem, impossibilitando determinar que o escalonador HFS consome menos energia nesta situação. Por esta razão, estatisticamente, o consumo de energia dos escalonadores é equivalente. O desempenho do escalonador FIFO, ao contrário do que ocorreu com a *workload CPU bound*, foi superior ao desempenho do escalonador HFS, porém em uma escala menor ($\approx 5.5\%$). E, novamente, HFS (98.7%) obteve uma taxa de localidade de tarefas *map* mais alta do que o escalonador FIFO (81%).

Então, diferentemente do observado com a *workload CPU bound*, existe um desacoplamento entre os resultados obtidos para cada métrica. A taxa superior de tarefas *map* executadas localmente não foi acompanhada de um desempenho superior do HFS. E o desempenho superior do escalonador FIFO não resultou em um menor consumo de energia.

5.3.3 *Workload* Heterogênea

Os resultados dos testes realizados com a workload heterogênea são apresentados na Tabela 5.13. A lógica de apresentação, outra vez, é idêntica a das duas seções anteriores.

Tabela 5.13: Resultados da *workload* heterogênea

Mixed	FIFO			HFS		
MÉTRICA	MÉDIA	COV	INTERVALO DE CONFIANÇA (@95%)	MÉDIA	COV	INTERVALO DE CONFIANÇA (@95%)
ENERGIA [kJ]	7075.611	0.014	[7004.725, 7146.495]	6571.242	0.008	[6533.545, 6608.937]
DESEMPENHO [s]	2064.700	0.024	[2029.641, 2099.7581]	1929.000	0.009	[1916.191, 1941.808]
TAREFAS *map* DATA-LOCAL	84.20%	0.010	[83.572%, 84.783%]	95.10%	0.010	[94.394%, 95.805%]

Nesta *workload*, os resultados exibem diferenças estatísticas nas três métricas avaliadas entre os escalonadores - ou seja, não há sobreposição dos intervalos de confiança e, assim, é possível determinar qual escalonador teve resultados melhores. HFS consumiu em média 6571 kJ durante as 10 replicações, enquanto que FIFO apresentou consumo médio de 7075 kJ. Assim, o consumo de energia do escalonador FIFO foi 7.6% superior. HFS também teve desempenho melhor do que FIFO, e em uma escala similar a superioridade apresentada no consumo de energia ($\approx 7\%$). E como verificado nas *workloads* homogêneas, a taxa de localidade no escalonamento das tarefas map foi superior sob a política de escalonamento HFS (95.1%, comparados aos 84.2% obtidos com o escalonador FIFO).

Logo, outra vez o escalonador de tarefas HFS faz melhor uso dos recursos disponíveis para executar as requisições feitas ao MapReduce - consumindo menos energia e executando a *workload* submetida ao sistema em tempo menor do que o escalonador FIFO.

5.3.4 Discussão

Os resultados apresentados anteriormente permitem afirmar que para a *workload CPU bound* e a *workload* composta por aplicações das duas categorias de aplicação consideradas HFS é a opção de escalonamento de tarefas que apresenta melhores resultados de consumo de energia e desempenho para a situação de escalonamento de múltiplas aplicações apresentadas nesta seção.

Portanto, o escalonamento justo e com preferência para tarefas locais - realizado pelo escalonador HFS - produz resultados melhores do ponto de vista de eficiência energética (executando a mesma quantidade de trabalho com menos energia) nas *workloads* citadas.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou uma caracterização do consumo de energia do Hadoop, um *framework* para desenvolvimento de aplicações intensivas em dados. O consumo de energia foi determinado através de um modelo baseado em estatísticas de uso providas pelo sistema operacional. Para que o consumo de energia pudesse ser considerado sob uma perspectiva mais ampla, paralelamente à caracterização do Hadoop com relação ao consumo de energia, um estudo do desempenho do sistema também foi apresentado. A caracterização foi realizada através de uma grande quantidade de experimentos: foram executadas 6440 aplicações MapReduce, totalizando 6755 GB processados pelo *framework*.

Os experimentos foram divididos em três etapas distintas: na primeira etapa foram realizados testes para determinar quais parâmetros de configuração do Hadoop têm influência sobre o consumo de energia e, também, sobre o desempenho do *framework*; em seguida uma comparação entre os dois escalonadores de tarefas do Hadoop (FIFO e HFS) foi realizada apontando os comportamentos destes em um cenário onde existe uma única aplicação a ser escalonada; e, por último, os escalonadores de tarefas foram comparados em uma situação onde múltiplas aplicações são executadas simultaneamente. Os experimentos foram realizados com duas aplicações distintas: uma aplicação *CPU bound* (WordCount) e uma aplicação *IO bound* (Sort).

A primeira etapa de experimentos apontou que, para aplicações MapReduce *CPU bound*, o parâmetro que tem maior efeito sobre o consumo de energia é aquele que controla a compressão de dados intermediários (*mapred.compress.map.output*). Para aplicações MapReduce *IO bound*, o número de tarefas reduce (*mapred.reduce.tasks*) foi o parâmetro com maior influência sobre o consumo energético. Ainda, os testes desta etapa permitiram determinar que existe uma similaridade muito evidente entre os efeitos de cada fator sobre o consumo de energia e os efeitos de cada fator sobre o desempenho do sistema MapReduce. Também foi possível estabelecer que existe uma similaridade entre os efeitos sobre as variáveis de resposta entre os dois escalonadores de tarefas. Portanto, em linhas gerais, os parâmetros mais influentes para cada escalonador em ambas aplicações testadas - e o grau de influência destes parâmetros - são semelhantes.

A segunda etapa de experimentos apontou que não é possível estabelecer que um escalonador é superior ao outro para o escalonamento de uma aplicação que executa sem concorrência com outras requisições ao sistema MapReduce.

Os resultados obtidos na última etapa de testes permitem afirmar que, para a *workloads CPU bound* e a *workload* composta por aplicações das duas categorias de aplicação consideradas, HFS é a opção de escalonamento de tarefas que apresenta melhores resultados de consumo de energia e desempenho para a situação de escalonamento de múltiplas aplicações simultâneas.

Acredita-se que os objetivos definidos para este trabalho foram atingidos: a caracteri-

zação do consumo energético de um framework para computação intensiva em dados. Os resultados relatados neste estudo podem servir como base para que usuários utilizem o Hadoop com maior eficiência energética. Os resultados também podem informar desenvolvedores do sistema que desejam tornar o *framework* ciente de seu consumo energético.

Como trabalhos futuros que podem surgir a partir deste estudo, destacam-se as seguintes possibilidades. O desdobramento mais evidente é o desenvolvimento de um novo escalonador de tarefas para o MapReduce ciente do consumo energético a partir do resultados apresentados neste trabalho. As informações contidas no estudo podem, também, impulsionar o desenvolvimento de novas funcionalidades relacionadas ao consumo de energia no simulador MRSG (KOLBERG et al, 2013). Ainda, uma caracterização do consumo de energia a partir de parâmetros do sistema de arquivos distribuído do MapReduce poderia complementar o conhecimento adquirido com este trabalho.

7 AGRADECIMENTO

Este trabalho foi realizado com apoio do projeto GREEN-GRID: Computação de Alto Desempenho Sustentável, financiado pela Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS).

REFERÊNCIAS

AGGARWAL, S.; PHADKE, S.; BHANDARKAR, M. Characterization of hadoop jobs using unsupervised learning. In: CLOUD COMPUTING TECHNOLOGY AND SCIENCE (CLOUDCOM), 2010 IEEE SECOND INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.748–753.

APACHE, F. S. F. **http://wiki.apache.org/hadoop/sort**. 2009.

APACHE, F. S. F. **http://wiki.apache.org/hadoop/wordcount**. 2011.

APACHE, F. S. F. **http://hadoop.apache.org**. 2013.

BARROSO, L. A. The price of performance. **Queue**, [S.l.], v.3, n.7, p.48–53, 2005.

BARROSO, L. A.; HÖLZLE, U. The case for energy-proportional computing. **Computer**, [S.l.], v.40, n.12, p.33–37, 2007.

BARROSO, L. A.; HÖLZLE, U. The datacenter as a computer: an introduction to the design of warehouse-scale machines. **Synthesis Lectures on Computer Architecture**, [S.l.], v.4, n.1, p.1–108, 2009.

CHANG, F.; FARKAS, K. I.; RANGANATHAN, P. Energy-driven statistical sampling: detecting software hotspots. In: **Power-Aware Computer Systems**. [S.l.]: Springer, 2003. p.110–129.

CHEN, Y.; GANAPATHI, A.; KATZ, R. H. To compress or not to compress-compute vs. IO tradeoffs for mapreduce energy efficiency. In: ACM SIGCOMM WORKSHOP ON GREEN NETWORKING. **Proceedings...** [S.l.: s.n.], 2010. p.23–28.

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. **Communications of the ACM**, [S.l.], v.51, n.1, p.107–113, 2008.

DEAN, J.; GHEMAWAT, S. MapReduce: a flexible data processing tool. **Communications of the ACM**, [S.l.], v.53, n.1, p.72–77, 2010.

DUSSO, P. M. A monitoring system for WattDB: an energy-proportional database cluster. **Trabalho de Graduação**, [S.l.], 2012.

ECONOMOU, D. et al. Full-system power analysis and modeling for server environments. In: IN PROCEEDINGS OF WORKSHOP ON MODELING, BENCHMARKING, AND SIMULATION. **Anais...** [S.l.: s.n.], 2006. p.70–77.

EIA. **http://www.eia.gov/tools/faqs/faq.cfm?id=427&t=3**. 2013.

FAN, X.; WEBER, W.-D.; BARROSO, L. A. Power provisioning for a warehouse-sized computer. **ACM SIGARCH Computer Architecture News**, [S.l.], v.35, n.2, p.13–23, 2007.

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The Google file system. In: ACM SIGOPS OPERATING SYSTEMS REVIEW. **Anais...** [S.l.: s.n.], 2003. v.37, n.5, p.29–43.

GOIRI, Í. et al. GreenHadoop: leveraging green energy in data-processing frameworks. In: ACM EUROPEAN CONFERENCE ON COMPUTER SYSTEMS, 7. **Proceedings...** [S.l.: s.n.], 2012. p.57–70.

GORTON, I. et al. Data-intensive computing in the 21st century. **Computer**, [S.l.], v.41, n.4, p.30–32, 2008.

GURUMURTHI, S. et al. Using complete machine simulation for software power estimation: the softwatt approach. In: HIGH-PERFORMANCE COMPUTER ARCHITECTURE, 2002. PROCEEDINGS. EIGHTH INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2002. p.141–150.

HÄRDER, T. et al. Energy efficiency is not enough, energy proportionality is needed! In: **Database Systems for Adanced Applications**. [S.l.]: Springer, 2011. p.226–239.

HE, B. et al. Mars: a mapreduce framework on graphics processors. In: PARALLEL ARCHITECTURES AND COMPILATION TECHNIQUES, 17. **Proceedings...** [S.l.: s.n.], 2008. p.260–269.

HERODOTOU, H.; BABU, S. Profiling, what-if analysis, and cost-based optimization of MapReduce programs. **Proc. of the VLDB Endowment**, [S.l.], v.4, n.11, p.1111–1122, 2011.

HUANG, S. et al. The HiBench benchmark suite: characterization of the mapreduce-based data analysis. In: DATA ENGINEERING WORKSHOPS (ICDEW), 2010 IEEE 26TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.41–51.

IPCC. **http://www.ipcc.ch/publications_and_data/ar4/syr/en/contents.html**. 2007.

IPCC. **http://www.ipcc.ch/report/ar5/**. 2013.

ISCI, C.; MARTONOSI, M. Runtime power monitoring in high-end processors: methodology and empirical data. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE, 36. **Proceedings...** [S.l.: s.n.], 2003. p.93.

JAIN, R. **The art of computer systems performance analysis**. [S.l.]: John Wiley & Sons Chichester, 1991. v.182.

JIANG, D. et al. The performance of mapreduce: an in-depth study. **Proceedings of the VLDB Endowment**, [S.l.], v.3, n.1-2, p.472–483, 2010.

KAUSHIK, R. T.; BHANDARKAR, M. GreenHDFS: Towards an Energy-Conserving Storage-Efficient, Hybrid Hadoop Compute Cluster **Proceedings of the USENIX Annual Technical Conference**, [S.l.], p.1-9, 2010.

KOLBERG, W. et al. MRSG: a MapReduce simulator over SimGrid. **Parallel Computing**, [S.l.], v.39, n.4-5, p.233–244, 2013.

KOUZES, R. T. et al. The changing paradigm of data-intensive computing. **Computer**, [S.l.], v.42, n.1, p.26–34, 2009.

KURP, P. Green computing. **Communications of the ACM**, [S.l.], v.51, n.10, p.1–13, 2008.

LANG, W.; PATEL, J. M. Energy management for mapreduce clusters. **Proceedings of the VLDB Endowment**, [S.l.], v.3, n.1-2, p.129–139, 2010.

LEVERICH, J.; KOZYRAKIS, C. On the energy (in) efficiency of hadoop clusters. **ACM SIGOPS Operating Systems Review**, [S.l.], v.44, n.1, p.61–65, 2010.

LIU, H.; ORBAN, D. Cloud mapreduce: a mapreduce implementation on top of a cloud operating system. In: CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2011 11TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2011. p.464–474.

MAHESHWARI, N.; NANDURI, R.; VARMA, V. Dynamic energy efficient data placement and cluster reconfiguration algorithm for MapReduce framework. **Future Generation Computer Systems**, [S.l.], v.28, n.1, p.119–127, 2012.

MARCOS, P. d. B. Maresia: an approach to deal with the single points of failure of the mapreduce model. **Dissertação de Mestrado**, [S.l.], 2013.

MOUW, E. Linux kernel procfs guide. **Faculty of Information Technology and Systems**, [S.l.], 2001.

OSTATIC. **http://ostatic.com/stress**. 2013.

RECKZIEGEL FILHO, B. Aplicação do MapReduce na análise de mutações gênicas de pacientes. **Trabalho de Graduação**, [S.l.], 2013.

RIVOIRE, S.; RANGANATHAN, P.; KOZYRAKIS, C. A Comparison of High-Level Full-System Power Models. **HotPower**, [S.l.], v.8, p.3–3, 2008.

SOHAN, R. et al. Characterizing 10 Gbps network interface energy consumption. In: LOCAL COMPUTER NETWORKS (LCN), 2010 IEEE 35TH CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.268–271.

THUSOO, A. et al. Hive: a warehousing solution over a map-reduce framework. **Proceedings of the VLDB Endowment**, [S.l.], v.2, n.2, p.1626–1629, 2009.

WANG, H.-S. et al. Orion: a power-performance simulator for interconnection networks. In: MICROARCHITECTURE, 2002.(MICRO-35). PROCEEDINGS. 35TH ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2002. p.294–305.

WHITE, T. **Hadoop: the definitive guide**. [S.l.]: O'Reilly, 2012.

WIRTZ, T.; GE, R. Improving MapReduce energy efficiency for computation intensive workloads. In: GREEN COMPUTING CONFERENCE AND WORKSHOPS (IGCC), 2011 INTERNATIONAL. **Anais...** [S.l.: s.n.], 2011. p.1–8.

XUE, S.-J.; PAN, W.-B. Parallel PK-means Algorithm on Meteorological Data Using MapReduce. **Wuhan Ligong Daxue Xuebao (Journal of Wuhan University of Technology)**, [S.l.], v.34, n.12, p.139–142, 2012.

YOO, R. M.; ROMANO, A.; KOZYRAKIS, C. Phoenix rebirth: scalable mapreduce on a large-scale shared-memory system. In: WORKLOAD CHARACTERIZATION, 2009. IISWC 2009. IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2009. p.198–207.

ZAHARIA, M. et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: EUROPEAN CONFERENCE ON COMPUTER SYSTEMS, 5. **Proceedings...** [S.l.: s.n.], 2010. p.265–278.

ZEDLEWSKI, J. et al. Modeling Hard-Disk Power Consumption. In: FAST. **Anais...** [S.l.: s.n.], 2003. v.3, p.217–230.

ZHANG, Y. et al. Parallel option pricing with BSDEs method on MapReduce. In: COMPUTER RESEARCH AND DEVELOPMENT (ICCRD), 2011 3RD INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. v.1, p.289–293.