Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Configurando o Hadoop Através de um Processo Empírico Flexível

Geraldo Abrantes Sarmento Neto

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sistemas Distribuídos

Lívia Maria Rodrigues Sampaio Campos
Raquel Vigolvino Lopes
(Orientadoras)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

S246c Sarmento Neto, Geraldo Abrantes.
Configurando o Hadoop através de um processo empírico flexível / Geraldo Abrantes Sarmento Neto. – Campina Grande, 2012.
83 f. : il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

Orientadoras: Profª. Drª. Lívia Maria Rodrigues Sampaio Campos, Profª. Drª. Raquel Vigolvino Lopes.
Referências.

1. MapReduce. 2. Hadoop. 3. Configuração. 5. Eficiência. I. Título.

CDU 004.75(043)

"CONFIGURANDO O HADOOP ATRAVÉS DE UM PROCESSO EMPÍRICO FLEXÍVEL"

GERALDO ABRANTES SARMENTO NETO

DISSERTAÇÃO APROVADA EM 23/04/2012

LÍVIA MARIA RODRIGUES SAMPAIO CAMPOS , D.Sc
Orientador(a)

RAQUEL VIGOLVINO LOPES, D.Sc
Orientador(a)

ANDREY ELÍSIO MONTEIRO BRITO, Dr.
Examinador(a)

ALEXANDRE NÓBREGA DUARTE, D.Sc
Examinador(a)

CAMPINA GRANDE - PB

Resumo

A geração de grandes volumes de dados, também conhecidos com Big Data, vem se tornando muito comum em ambientes acadêmicos e corporativos. Nesse contexto, é essencial que as aplicações que processam Big Data explorem da melhor forma possível as infraestruturas distribuídas de alto desempenho (como *clusters*), possivelmente presentes nesses ambientes, através da implantação dessas aplicações sobre sistemas de computação intensiva de dados tais como o popular Hadoop. No que diz respeito à configuração desta plataforma, observa-se uma quantidade considerável de parâmetros que devem ser ajustados e os quais os usuários normalmente não têm noção de como fazê-los, resultando em um Hadoop mal configurado e com um desempenho aquém do seu real potencial. Este trabalho propõe um processo para auxiliar a configuração eficiente do Hadoop através do uso de técnicas empíricas que utilizam subespaços de parâmetros dessa plataforma, e da aplicação de análises estatísticas para verificar a relevância dos mesmos, extraindo os valores otimizados em função do subespaço de parâmetros considerado. Visando instanciar o processo, foi realizado um estudo de caso de forma a obter uma configuração com impacto positivo sobre o tempo de resposta de uma aplicação representativa para esse contexto. A validação foi feita através de uma comparação do processo proposto com soluções existentes na qual foi possível observar que o processo teve uma significativa vantagem, levando em consideração o mesmo ambiente e *workload* utilizados na etapa de instanciação. Apesar do tempo médio de conclusão do processo ter sido maior que o das outras soluções, foram levantados cenários em que o uso do processo proposto é mais vantajoso (e possivelmente mais viável) que o uso das outras soluções. Isso ocorre devido à sua flexibilidade, uma vez que ele não apresenta restrições quanto ao subespaço de parâmetros selecionado e métricas possíveis de serem analisadas.

Abstract

The generation of large amounts of data, also known as Big Data, is becoming very common both in the academy and in the enterprises environments. In that context, it is essential that applications responsible for processing Big Data exploit high-performance distributed infrastructures (such as *cluster*), commonly present in those environments, through the deploying of such applications on data-intensive scalable supercomputing (DISC) systems such as the popular Hadoop. Regarding the configuration of that platform, there is a considerable amount of parameters to be adjusted by users who do not know how to set them, resulting in a Hadoop poorly configured and performing below of its real potential. This work proposes a process to help in Hadoop efficient configuration by using empirical techniques to analyze subspaces of parameters of this platform, and the application of statistical foundations to verify the relevance of such parameters, obtaining the optimized values according to the subspace of parameters considered. Aiming the process instantiation, we performed a case study in order to obtain proper settings with a positive impact on the response time of a representative application in this context. The validation was performed through a comparison between the proposed process and some existing solutions in which we observed that the former had a significant advantage regarding same environment and *workload* used in the instantiation stage. Although the average completion time of the process has been higher than the other solutions, we presented scenarios which the use of the proposed process is more advantageous (and feasible) than the use of other solutions. This happens due to its flexibility, since it has no constraints on the subspace of selected parameters and metrics possible to be analyzed.

É melhor atirar-se em luta, em busca de dias melhores, do que permanecer estático como os pobres de espírito, que não lutaram, mas também não venceram. Que não conheceram a glória de ressurgir dos escombros. Esses pobres de espírito, ao final de sua jornada na Terra, não agradecem à Deus por terem vivido, mas desculpam-se diante dele, por simplesmente, haverem passado pela vida.

Bob Marley

Agradecimentos

Agradeço inicialmente a Deus, por ter me dado forças durante todos esses anos. Aos meus pais (Espedito e Iris) e à minha irmã (Nathalia) pelos momentos em família e pelos valores e ensinamentos básicos que carrego comigo até hoje.

Agradeço às minhas orientadoras Lívia Sampaio e Raquel Lopes por toda a paciência e dedicação, além das sugestões e críticas que me ajudaram a crescer. E não poderia esquecer de agradecer a Fubica (a.k.a. Francisco Brasileiro) por ter compartilhados tantos conselhos e ensinamentos.

A todos os companheiros do LSD; de uma forma especial a Pryscilla, que sempre me deu muita força desde minha chegada; a Ianna e Ana Cristina, pelas incansáveis tardes de estudo; a Ricardo, David, Manel e Abmar pelo apoio e pelo ambiente amistoso do LSD; a Brian (a.k.a. Adabriand) e Manoel pelos serviços de impressão e correio da VolúpiaExpress®. E a Lesandro e Edigley, fiéis companheiros dos finais de semana LSDanos; pelos momentos de boas conversas regadas a café e filosofia.

Agradeço também a todos os meus amigos; pessoas que me acompanharam por essa caminhada e que me proporcionaram grande momentos de descontração: Leandro Sobreira (vulgo Pipoca), Pedro Vitor (vulgo Pateta), Isabelle Maia e Thiago Menezes (vulgo Coveiro), além de Jaclason Machado (e sua digníssima esposa Lilian).

A todos os amigos que me acolheram lá "pras bandas do sertão": Kléber e Klepler Trajano, Wilker, Pablo, Vilmar, Everton, entre outros; grandes personagens de uma história que está começando a ser contada.

Enfim, agradeço a todos aqueles que me apoiaram e me incentivaram durante esta caminhada. Aprendi que quando se quer muito uma coisa, você deve lutar por ela. Desistir é abrir mão de sonhos; de projetos de vida.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Este capítulo apresenta os aspectos introdutórios sobre esta dissertação. Inicialmente são apresentadas a motivação e a definição do problema estudado. Em seguida são descritos os objetivos gerais e discutidas as contribuições da pesquisa realizada. Para finalizar o capítulo é feita uma descrição sobre a organização do restante da dissertação.

1.1 Motivação

A evolução dos computadores pessoais aliada à melhoria nas tecnologias das redes de computadores levaram à criação de infraestruturas alternativas aos onerosos supercomputadores. Apoiada no processamento paralelo e distribuído, essas infraestruturas utilizam agregados de computadores convencionais interligados tanto por redes de altíssima velocidade, a exemplo dos *clusters*, como por redes convencionais (locais ou geograficamente distantes), a exemplo das grades computacionais [1][15][25].

Atualmente essas infraestruturas estão sendo bastante empregadas, tanto no meio acadêmico quanto no meio corporativo, para executar aplicações distribuídas que geram ou processam grandes massas de dados, chamadas de aplicações intensivas de dados (do inglês *data intensive*) ou aplicações de *Big Data* [7][3][4].

Alguns exemplos dessas aplicações podem ser observados no compartilhamento de mídias de alta qualidade [41], mecanismos complexos de busca e recomendação [54] e no número crescente de sensores e outros equipamentos geradores de dados presentes em muitos lugares [13]. O Facebook®, por exemplo, coleta cerca de 15 terabytes de dados diaria-

mente [4]. Outro importante gerador de dados é o Large Hadron Collider (LHC) que tem uma estimativa de produção de 10 petabytes de dados por ano [20].

Visando alcançar um maior grau de utilização dessas infraestruturas distribuídas e, ao mesmo tempo, explorar seu poder computacional, foram desenvolvidos uma série de modelos de programação, tais como: MPI (do inglês *Message Passing Interface*) [31], PVM (do inglês *Parallel Virtual Machine*) [29], FilterStream [5], DataflowGraph [2] e MapReduce [16][17].

Esses modelos ditaram o desenvolvimento dos sistemas escaláveis de computação intensiva de dados (DISC, do inglês *Data Intensive Scalable Computing*), também chamados de plataformas de alto desempenho, tais como: DataCutter (modelo FilterStream) [5][6], Anthill (modelos PVM e FilterStream) [24], Open-MPI (modelo MPI) [30] , Dryad (modelo DataflowGraph) [37] e Hadoop (modelo MapReduce) [16].

Devido à facilidade de programação, o modelo MapReduce tornou-se muito atrativo [52]. A ideia básica por trás desse modelo consiste na implementação de duas funções: *map* e *reduce*. Essas funções representam a divisão de uma grande tarefa em tarefas menores que são executadas paralelamente em diferentes nodos[1] [16].

O Hadoop é a plataforma MapReduce mais utilizada atualmente [20] [65]. É possível elencar uma série de áreas nas quais há aplicações implantadas sobre essa plataforma, a saber: indexação na Web, mineração de dados, geração de relatórios, análise de arquivos de *log*, análise financeira, simulação científica e pesquisa em bioinformática [38].

Apesar da grande disseminação do Hadoop, a maioria de seus usuários utilizam-no com uma eficiência aquém a seu real potencial [4]. Isso ocorre devido à grande quantidade de parâmetros a serem configurados manualmente e que exigem conhecimento sobre o perfil da aplicação, especificação do *hardware* ou detalhes de implementação da própria plataforma [65]. Desse modo, toda a responsabilidade de ajustes dos parâmetros de configuração recai sobre o próprio usuário da plataforma, que na maioria das vezes é completamente leigo e enfrenta dificuldades para configurá-la adequadamente [35].

[1]Esse termo será utilizado, neste trabalho, para designar máquinas ou estações pertencentes a um *cluster* ou grade computacional.

1.2 Definição do Problema

O Hadoop dispõe de aproximadamente 200 parâmetros de configuração, dentre os quais mais de 25 interferem significativamente no desempenho da aplicação [4]. Nesse contexto de desempenho (que é muito enfatizado na literatura), os parâmetros estão relacionados à quantidade de núcleos de CPU e memória RAM disponíveis, quantidade de tarefas (*maps* e *reduces*) por nodo, tamanho de *buffers* alocados para as tarefas, número de *streams* utilizados para gravar dados em arquivos, etc.

O impacto desses parâmetros varia de acordo com a aplicação utilizada, com os dados de entrada e com os recursos de *hardware* disponíveis. Se tais parâmetros forem configurados adequadamente, o tempo de execução pode ser reduzido em mais de 50% [4][40].

Apesar do número reduzido de parâmetros que impactam no desempenho, em relação ao número total (aproximadamente 25 de 200), eles devem ser ajustados manualmente, o que torna a tarefa de configuração bastante complexa. Além do mais, essa complexidade é ainda maior se forem consideradas as interações cruzadas que ocorrem entre alguns parâmetros, ou seja, quando o efeito de um parâmetro p_1 depende do valor ajustado para o parâmetro p_2 [39].

Há uma série de pesquisas que procuraram soluções para melhorar o desempenho do Hadoop. Algumas delas não exploram o impacto dos parâmetros de configuração dessa plataforma, mas criam mecanismos visando melhorar algumas políticas existentes como o escalonamento de tarefas [70] ou os mecanismos de comunicação entre *maps* e *reduces* [20], por exemplo. Outras pesquisas analisam ou propõem técnicas para a configuração eficiente dos parâmetros dessa plataforma [4][40], mas se limitam a um subespaço de parâmetros ou métricas específicas de desempenho.

Manipulando ou não os parâmetros de configuração, muitas soluções do estado da arte (como essas que foram supracitadas) se focam na análise do tempo de resposta. Contudo, a execução de aplicações nessa plataforma pode estar associada a objetivos que vão além do tempo de resposta, como é o caso da vazão de *jobs* por unidade de tempo [32] ou do consumo de energia [42]. É importante levar isso em consideração, uma vez que diferentes métricas podem estar associadas a diferentes conjuntos de parâmetros.

1.3 Objetivos

O objetivo deste trabalho é descrever uma solução flexível para facilitar a configuração do Hadoop, visando melhorar a eficiência dessa plataforma em função de uma métrica de interesse definida pelo usuário.

Especificamente é possível destacar os seguintes objetivos:

- Analisar técnicas do estado da arte sobre a configuração do Hadoop;
- Discutir alguns modelos analíticos que podem auxiliar na configuração do Hadoop;
- Analisar como técnicas empíricas existentes podem ser organizadas de tal forma a se obter um processo para auxiliar na configuração de plataformas MapReduce e de sistemas cuja eficiência dependa do ajuste adequado de seus vários parâmetros de configuração;
- Avaliar o processo proposto para fins de desempenho do Hadoop;
- Validar a solução apresentada através de uma análise comparativa com soluções relacionadas;

1.4 Resultados e Contribuições

Neste trabalho é proposto o uso de um processo empírico para auxiliar usuários do Hadoop a configurar adequadamente essa plataforma visando otimizar uma métrica de interesse. Para tanto, foram descritas as atividades desse processo de forma detalhada e foi realizado um estudo de caso envolvendo uma aplicação representativa da computação intensiva de dados.

O processo é baseado em técnicas empíricas e necessita que o usuário defina tanto a métrica de interesse quanto os parâmetros candidatos que poderão ser relevantes de acordo com essa métrica estabelecida. A natureza empírica do processo pode impactar em um tempo relativamente alto de conclusão do mesmo, quando comparado a soluções relacionadas. No entanto os resultados mostram que a abordagem empírica conseguiu uma otimização melhor que a das outras soluções. Desse modo, o processo proposto pode se mostrar vantajoso nos casos em que o cenário de execução do Hadoop não é alterado a curto ou médio prazo.

Uma das soluções que é utilizada na validação do processo proposto é um conjunto de regras genéricas conhecidas como *rules of thumbs*. Algumas dessas regras são imprecisas e careciam de uma formalização adequada. Outra contribuição desse trabalho foi especificar equações simples para o ajuste de alguns parâmetros relevantes que são cobertos por *rules of thumbs*.

A documentação do processo proposto feita neste trabalho também pode ser considerada como uma contribuição. Apesar das técnicas empíricas e estatísticas empregadas já existirem, até onde foi possível comprovar há poucos trabalhos que instanciam tais técnicas tendo vista a configuração de plataformas de computação intensiva como o Hadoop.

Além disso, é importante destacar a contribuição do processo sob a visão da flexibilidade que ele proporciona. Diferente de outras soluções [36], o processo proposto não limita os subespaços de parâmetros que podem ser considerados. Reduzindo o espaço de busca, diminui-se a probabilidade de se encontrar um bom ótimo local ou possivelmente um ótimo global [28].

O outro aspecto de flexibilidade considerado é que a maioria dos trabalhos relacionados se limitam a métricas de desempenho, como o tempo de resposta [20][4][40]. O processo proposto pode ser estendido à qualquer métrica passível de ser avaliada no contexto de aplicações sobre o Hadoop. Essa flexibilidade é útil quando se fala em métricas alternativas, como o consumo de energia [42] por exemplo, que vem ganhando destaque em trabalhos que envolvem a análise de sistemas complexos, inclusive o próprio Hadoop [42].

1.5 Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma:

Capítulo 2 - O Modelo MapReduce e a Plataforma Hadoop. Nesse capítulo é descrito o modelo de programação MapReduce, o funcionamento de sua arquitetura e como são desenvolvidas as aplicações que seguem esse modelo. A seguir é apresentada a plataforma Hadoop, seus principais componentes e suas principais características. Ainda nesse capítulo serão discutidos fatores relacionados aos mecanismos de configuração da plataforma e como alguns parâmetros podem ser corretamente configurados através de informações de

hardware.

Capítulo 3 - Trabalhos Relacionados. Nesse capítulo são apresentados os trabalho relacionados não apenas à configuração adequada de aplicações sobre o Hadoop, mas também relacionados às pesquisas que envolvem a configuração de sistemas complexos e àquelas que envolvem o uso de técnicas empíricas em diferentes domínios.

Capítulo 4 - Processo Empírico Proposto. Esse capítulo apresenta o processo proposto. Nele são descritas as atividades desse processo e discutidas as técnicas empíricas e estatísticas utilizadas para a configuração de aplicações sobre o Hadoop.

Capítulo 5 - Instanciação do Processo. Esse capítulo descreve um estudo de caso realizado visando instanciar o processo proposto e analisar os resultados do mesmo. Para o cenário considerado utilizou-se a métrica tempo de resposta. Através dessa escolha foi possível fazer uma análise comparativa com algumas soluções relacionadas. Ainda nesse capítulo é discutido como o tempo de conclusão para se executar cada solução pode ter impacto sobre o desempenho.

Capítulo 6 - Conclusões. Nesse capítulo são apresentadas as conclusões desta pesquisa com ênfase nas contribuições para a comunidade. Elencou-se, ainda, alguns trabalhos futuros que podem aperfeiçoar o processo proposto.

Capítulo 2

O MapReduce e a Plataforma Hadoop

Este capítulo apresenta o modelo de programação MapReduce e uma de suas implementações mais conhecidas, a plataforma Hadoop.

Sobre o MapReduce são apresentadas suas principais características e é descrito como implementar aplicações que seguem esse modelo. Sobre o Hadoop são mostrados seus principais componentes e são discutidos alguns pontos relacionados à sua configuração.

Ainda neste capítulo será discutida a configuração eficiente de alguns parâmetros dessa plataforma considerando as chamadas *rules of thumbs*.

2.1 MapReduce

Originalmente desenvolvido pela Google® [16], o MapReduce é um modelo de programação projetado para aplicações intensivas de dados executadas em ambientes distribuídos. Para se desenvolver uma aplicação seguindo esse modelo o usuário deve implementar duas funções: *map* e *reduce*. No MapReduce, a computação envolve a manipulação de dados no formato $<chave, valor>$. A função *map* toma pares de entrada do tipo $<k_1, v_1>$ e produz um conjunto de pares intermediários do tipo $<k_2, v_2>$ que são agrupados sob a mesma chave k_2 e passados para a função *reduce*. Essa função por sua vez recebe um conjunto do tipo $<k_2, lista(v_2)>$ e junta (mescla) esses dados para gerar um conjunto $<k_3, v_3>$. Vale ressaltar que tanto as chaves k_1, k_2 e k_3 quanto os valores v_1, v_2 e v_3 podem ser de diferentes tipos [35].

Para que o leitor tenha uma melhor compreensão dessa computação, é exibido a seguir

um algoritmo de um programa contador de palavras que é simbólico para trabalhos que envolvem o modelo Map Reduce: o WordCount [16][4]. O Algoritmo 1 exibe o pseudocódigo para a função *map*. Como instâncias de chave de entrada (*input_key*) e valores de entrada (*input_value*) podem ser considerados o nome de um documento e seu conteúdo, respectivamente.

Algoritmo 1: Pseudocódigo de uma função *map* para o algoritmo WordCount

```
Data: String: input_key;
Data: String: input_value;
begin
    for each word w in input_value do
        emitIntermediate(w, "1");
end
```

O Algoritmo 2, por sua vez, exibe o pseudocódigo para a função *reduce*, que recebe uma chave (key) e uma lista de valores ($values$) que representam, para o algoritmo considerado, a contagem de palavras em função dessa chave.

Algoritmo 2: Pseudocódigo de uma função *reduce* para o algoritmo WordCount

```
Data: String: key;
Data: List: values;
begin
    int : result = 0;
    for each word v in values do
        result = result + parseToInt(v);
    emitAsString(result);
end
```

A Figura 2.1 mostra a representação da execução de uma aplicação MapReduce. Inicialmente a biblioteca MapReduce, invocada pelo programa do usuário, divide os dados de entrada em M segmentos, tipicamente em tamanhos que compreendem uma faixa de 16 MB a 64 MB, por segmento. A seguir, o programa é replicado no *cluster*. Uma das cópias do programa é chamada de *master* e as demais são chamadas de *workers*. O *master* é responsável por distribuir as M tarefas *map* e as R tarefas ***reduce*** aos *workers*, doravante denominados

de *workers map* e *workers reduce*, respectivamente

Cada *worker map* carrega uma porção dos dados, passando para sua tarefa *map* seus correspondentes pares $<chave, valor>$. Os pares intermediários produzidos pela tarefa *map* são armazenados em um *buffer*. Periodicamente os dados desse *buffer* são escritos no disco local, particionados em R regiões. As informações referentes às localizações dessas partições são enviadas ao *master*, que as encaminha aos *workers reduce*.

Quando um *worker reduce* é notificado sobre essas localizações, ele carrega os dados a partir do disco local onde os *workers map* escreveram seus respectivos dados intermediários. Uma vez que esses dados são carregados, o *worker reduce* ordena-os a partir de suas chaves intermediárias, agrupando juntas todas as ocorrências sob a mesma chave. A partir de então o *worker reduce* itera sobre os dados ordenados passando-os para a função *reduce*, que executa as devidas computações. Após esse processo a saída obtida é anexada a um arquivo de saída.

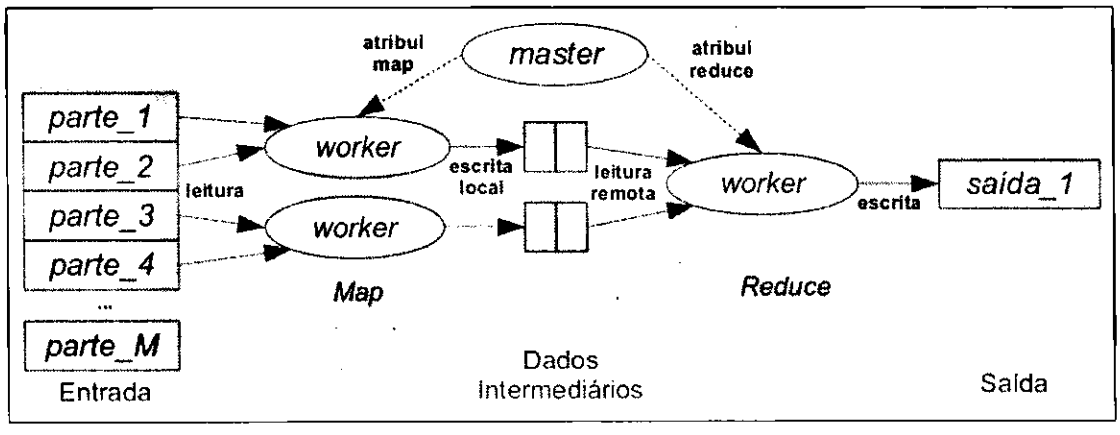

Figura 2.1: Arquitetura do MapReduce

A arquitetura do MapReduce foi projetada para ser tolerante a falhas. Portanto, periodicamente o *master* envia um sinal para cada *worker* solicitando um sinal de retorno para indicar seu estado. Se um *worker* não responde até determinado tempo limiar, o *master* considera que a execução desse *worker* falhou.

Quando ocorre a falha de uma tarefa *map*, ela deve ser reexecutada em outro nodo; pois se há uma falha, os dados intermediários armazenados no sistema de arquivos local podem não estar disponíveis para acesso por parte dos *reduces*. O mesmo não acontece para as tarefas *reduce*, uma vez que suas saídas são escritas de forma replicada diretamente no sistema de arquivos distribuído (DFS, do inglês *Distributed File System*).

A preocupação com os recursos não se restringe ao armazenamento; a largura de banda da rede também é considerada. Por se tratar de um recurso caro, o MapReduce administra-o tirando proveito do fato de que, ao iniciar uma aplicação MapReduce, os dados de entrada são particionados e replicados em diferentes nodos. Munido de informações sobre a localização desses dados, o *master* tenta escalonar as tarefas *map* nos nodos que contém réplicas dos dados correspondentes às suas respectivas entradas, evitando mais transferências de dados e, consequentemente, reduzindo o consumo de largura de banda. Se esse escalonamento não for possível, o *master* tenta escalonar as tarefas *map* o mais próximo possível dessas réplicas.

Apesar de ter documentado o modelo de programação, o código-fonte do sistema MapReduce da Google® não é aberto. Isso motivou muitas empresas a desenvolverem seus próprios sistemas e plataformas MapReduce, seguindo todas as características e especificações desse modelo. Uma plataforma MapReduce de código-fonte aberto muito conhecida é o Hadoop [65], que é um dos focos desta pesquisa e será discutido na seção a seguir.

2.2 Hadoop

O Hadoop é um *framework* e uma plataforma MapReduce de código aberto orientados ao desenvolvimento e implantação de aplicações distribuídas apropriadas para computação intensiva de dados.

Atualmente essa plataforma tem sido muito difundida [4]. Além de ser diretamente utilizada por empresas como Facebook®, Yahoo!®, Twitter® e IBM®, há empresas como a Cloudera® que fornecem serviços e ferramentas baseadas no Hadoop para serem executadas em provedores de computação na nuvem como o Amazon® EC2 [65].

A Apache®, atual mantenedora do Hadoop, disponibiliza projetos relacionados [60] a essa plataforma. Dentre eles destacam-se:

- *Core*: um conjunto de componentes e interfaces para sistemas de arquivos distribuídos e E/S[1] em geral;
- *HDFS*: (do inglês *Hadoop Distributed FileSystem*) um sistema de arquivos distribuído apropriado para cenários de computação intensiva de dados. Esse é o sistema de arquivos padrão do Hadoop;

- *Pig*: linguagem e ambiente de execução em *data flow*[2] para explorar grandes conjuntos de dados;
- *HBase*: um sistema de banco de dados instalado sobre o HDFS para suporte tanto de computação em lote como consultas através de uma linguagem própria;
- *Hive*: um sistema de *data warehouse* distribuído. Ele gerencia dados armazenados no HDFS e fornece uma linguagem de consulta similar ao SQL para consulta nos dados;

A Seção 2.2.1 apresenta os componentes do Hadoop e que papel eles exercem dentro do modelo MapReduce. A Seção 2.2.2 descreve a execução de um *job* MapReduce no Hadoop. A seguir, a Seção 2.2.3 discute as particularidades que envolvem a configuração dessa plataforma.

2.2.1 Componentes

O Hadoop dispõe de alguns processos que executam como *daemon* [59] nas máquinas que compõem sua arquitetura. Tais processos são: o *jobtracker*, o *tasktracker*, o *namenode* e o *datanode*. Esse processos se integram de uma forma similar ao que é exibido na Figura 2.2.

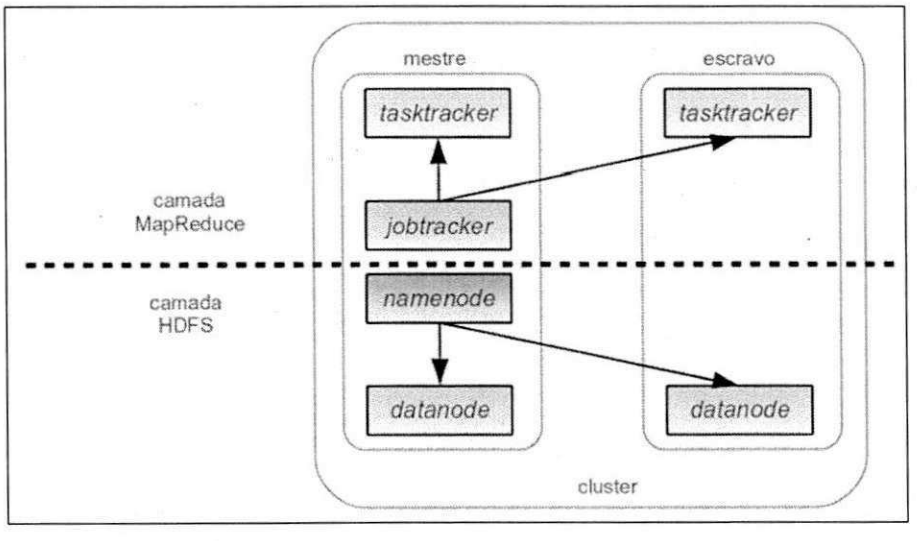

Figura 2.2: Modelo de um *cluster* com os principais componentes do Hadoop

[1]Entrada e Saída.

[2]Um paradigma de programação que modela um fluxo de dados através de um grafo dirigido.

O *jobtracker* é o gerenciador de *jobs* no Hadoop. A máquina que executa esse processo torna-se o mestre da arquitetura. Esse componente coordena a execução de todos os *jobs*, escalona as tarefas (*map* e *reduce*) sobre os nodos escravos e monitora o progresso de execução da aplicação. Se uma tarefa chegar a falhar, o *jobtracker* pode reescaloná-la em um nodo diferente [65].

O *tasktracker* é um processo que é executado nos nodos escravos. Ele é o responsável por executar as tarefas *map* e *reduce* e reportar informações sobre a execução das mesmas ao *jobtracker*.

Como mencionado anteriormente, o HDFS é o sistema de arquivos distribuído padrão do Hadoop. Vinculados a ele, há outros processos *daemon*, a citar: o *namenode* e o *datanode*.

O *namenode* é o responsável por gerenciar o espaço de nomes do sistema de arquivos. Ele mantém a árvore do sistema de arquivos e armazena metadados sobre todos os arquivos e diretórios [65].

O *datanode* é executado em um nó escravo e é responsável por armazenar blocos de dados e recuperá-los à medida que forem solicitados. Periodicamente, o *datanode* se reporta ao *namenode* atualizando informações sobre os blocos que ele está armazenando correntemente [65].

2.2.2 Execução de um *job* MapReduce

Apesar de já ter sido abordada a execução de um *job* MapReduce na Seção 2.1, é importante especificar as particularidades dessa execução no Hadoop.

Através da linha de código: `JobClient.runJob(conf)`, o desenvolvedor explicita a criação de um *job* em seu programa principal, conforme especificado no *framework* do Hadoop [64].

O método `runJob` cria uma instância de `JobClient` que, por sua vez, submete um *job* através da invocação do método `submitJob()`. Mais especificamente esse método executa os seguintes passos:

- Solicita um novo identificador de *job* (*jobID*) ao *jobtracker*;
- Checa a especificação do diretório de saída, uma vez que o *job* não é executado caso esse diretório já exista ou não seja especificado;

- Calcula os segmentos de dados de entrada correspondes ao *job*;
- Copia os recursos necessários para executar o *job*, incluindo o arquivo JAR contendo o código da aplicação, o arquivo de configuração (discutido na Seção 2.2.3) e o cálculo dos seus segmentos de dados de entrada;
- Informa ao *jobtracker* que o *job* está pronto para ser executado;

A partir de então o *job* é inserido em uma fila na qual o escalonador de *jobs* pode selecioná-lo para execução (segundo algum algoritmo de escalonamento), o que envolve a criação de um objeto para representar o *job* em execução, registrando as informações de status e progresso de suas tarefas.

Para criar a lista de tarefas *map* e *reduce* a serem executadas, o escalonador de *jobs* recupera o cálculo dos segmentos de dados de entrada, criando uma tarefa *map* por segmento. O número de tarefas *reduce* é especificado pela propriedade *mapred.reduce.tasks* de `JobConf`. As propriedades de `JobConf` serão melhor explicadas na Seção 2.2.3.

Os *tasktrackers* têm um número fixo de *slots* para executar tarefas *map* e *reduce* simultaneamente. Ele escalona essas tarefas de modo a preencher todos os *slots*, dando prioridade para as tarefas *map*. Regularmente o *tasktracker* envia um sinal ao *jobracker* para sinalizar que está vivo, esse sinal é chamado de *heartbeat*.

Finalmente, o *tasktracker* localiza o arquivo JAR correspondente, descompacta-o em um diretório local, copia quaisquer arquivos necessários para a execução para o disco local e cria uma instância de `TaskRunner`: a entidade que vai realizar a execução da tarefa propriamente dita. Para isso, essa instância lança uma nova máquina virtual Java (JVM, do inglês *Java Virtual Machine*) para comportar a execução dessa tarefa, podendo reutilizar uma JVM já existente conforme especificado pela propriedade *mapred.job.reuse.jvm.num.tasks*.

Todos esses passos de execução de um *job* estão resumidos na Figura 2.3.

Lado Map e Lado Reduce

Quando uma tarefa *map* é executada ela começa a produzir dados de saída que não são diretamente escritos no disco, mas em um *buffer* circular cujo tamanho é especificado através da propriedade *io.sort.mb*. Quando os dados contidos no *buffer* atingem um certo limiar, definido pela propriedade *io.sort.spill.percent*, eles são ordenados e escritos (derramados,

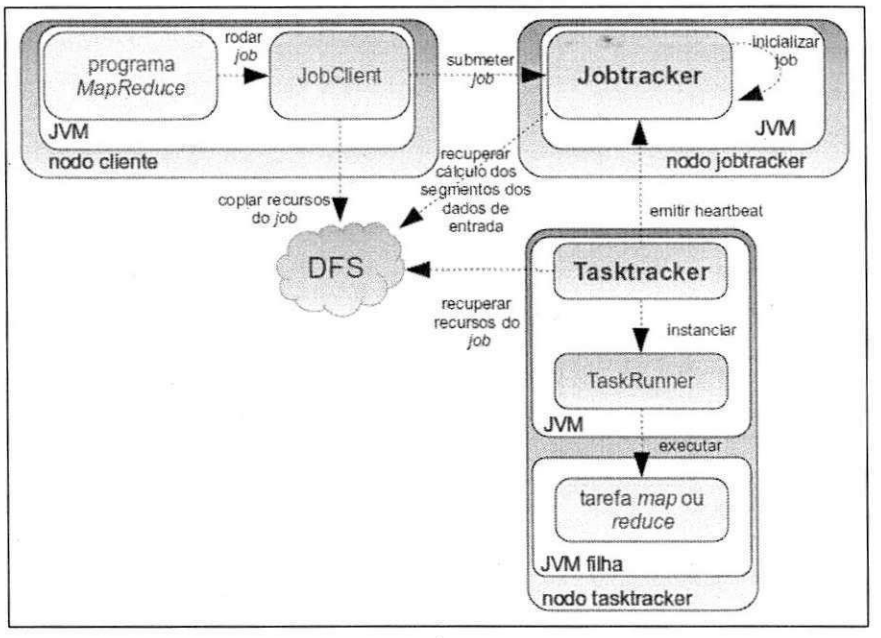

Figura 2.3: Execução de um *job* MapReduce sob uma granularidade mais fina

do inglês *spilled*) no disco. Se o desenvolvedor especificou algum função para combinar os dados, ela é utilizada antes da escrita em disco.

Antes que a tarefa *map* seja finalizada, os arquivos escritos em disco são mesclados (unidos) em um único arquivo que é delimitado de acordo com a quantidade de tarefas *reduces* associadas, conforme mostra a Figura 2.4. A propriedade *io.sort.factor* controla o número máximo de fluxos de dados (*streams*) para mesclar esses dados [65].

Quando uma tarefa *reduce* é escalonada é iniciada uma fase chamada de *shuffle* [35] ou *copy* [65]: busca das saídas de diferentes tarefas *maps* associadas a essa tarefa *reduce*. O número de *threads* utilizadas para realizar essa busca em paralelo é especificado pela propriedade *mapred.reduce.parallel.copies*.

Normalmente esses dados são armazenados em um *buffer* no lado *reduce* controlado pela propriedade *mapred.job.shuffle.input.buffer.percent*. Se não houver espaço suficiente, os dados são armazenados no disco. Quando os dados contidos nesse *buffer* atingem o limiar definido pela propriedade *mapred.job.shuffle.merge.percent*, eles são mesclados e derramados no disco.

À medida que as cópias dos dados vão se acumulando no disco, elas vão sendo mescladas por uma *thread* em segundo plano. Quando todas as saídas das tarefas *map* são copiadas,

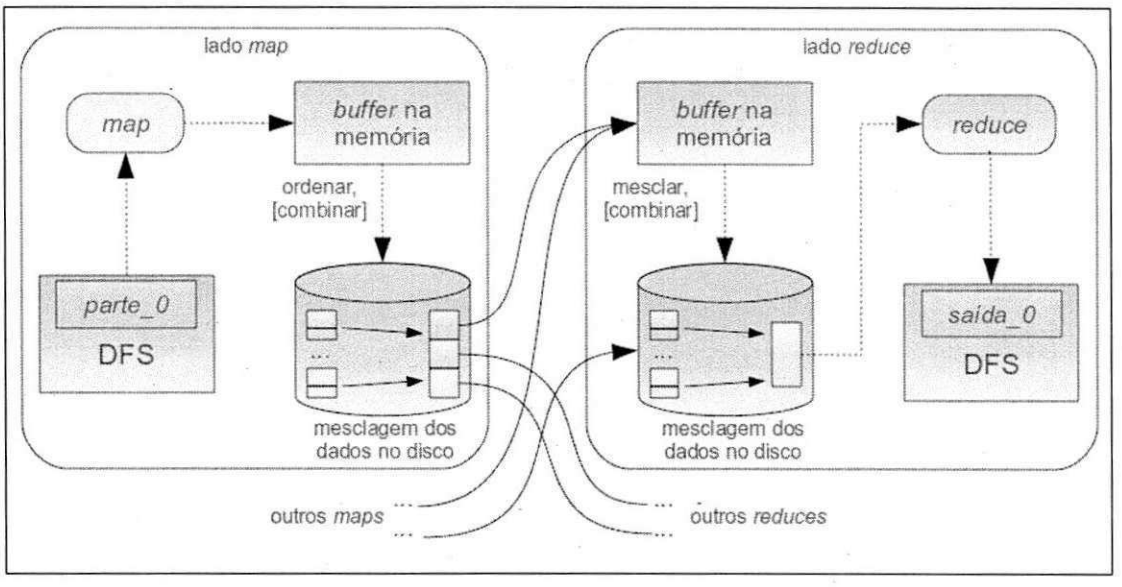

Figura 2.4: Execução das tarefas *map* e *reduce* sob uma granularidade mais fina

a tarefa *reduce* inicia a fase de ordenação desses dados de forma a obter um único arquivo que é finalmente passado para a função *reduce*, cuja saída é diretamente escrita no sistema de arquivos distribuído.

2.2.3 Mecanismos de Configuração

Quando um *job* é submetido à execução no Hadoop, é criado uma instância da classe `JobConf` responsável, entre outras coisas, pela definição das informações de configuração desse *job*. Essa instância tem como atributos uma série de propriedades doravante denominadas de *parâmetros de configuração*: características tanto do *job* quanto do ambiente onde ele está sendo executado.

Os parâmetros de configuração podem ser ajustados pela interface do programa do usuário ou através dos arquivos: *core-site.xml*[3], *hdfs-site.xml*[4] e *mapred-site.xml*[5]. Outros serviços como Pig e Hive dispõem de linguagens através das quais também é possível especificar os parâmetros de configuração dos *jobs* [4].

Ao todo, o Hadoop dispõe de aproximadamente 200 parâmetros de configuração que

[3]Detalhes em: *http://hadoop.apache.org/common/docs/r0.20.2/core-default.html*.
[4]Detalhes em: *http://hadoop.apache.org/common/docs/r0.20.2/hdfs-default.html*.
[5]Detalhes em: *http://hadoop.apache.org/common/docs/r0.20.2/mapred-default.html*.

podem impactar em diferentes aspectos, tais como desempenho e eficiência energética. Os parâmetros que não interferem na eficiência da plataforma (ou que apresentam uma interferência desprezível) são aqueles responsáveis por definir informações secundárias, como por exemplo: localização de diretórios do sistema de arquivos, definição do número máximo de arquivos de *log*, definição de classes de serialização, *codecs*, etc. Também não foi encontrado nenhum registro (na literatura relacionada) sobre o impacto desses parâmetros na eficiência do Hadoop [65][4][35].

O ajuste dos parâmetros que interferem na eficiência da plataforma depende de alguns fatores como: os dados de entrada, a métrica que se deseja observar e a configuração do *cluster*. O grau de relevância de um subconjunto de parâmetros está associado a uma determinada métrica de interesse, ou seja, um parâmetro p_1 pode ser relevante para análise das métricas m_1 e m_2, mas pode não ser relevante para uma métrica m_3. Desse modo, a Tabela 2.1 lista os parâmetros que podem impactar em 3 métricas diferentes: *tempo de resposta*, *vazão de jobs* e *consumo de energia*, levando em consideração diferentes níveis de confiança sobre o impacto. Os parâmetros rotulados por $***$ têm um alto nível de confiança devido às evidências empíricas encontradas na literatura [4][35]. Os parâmetros rotulados por $**$ têm um nível de confiança médio devido à falta de evidências empíricas, mas com suporte em *rules of thumbs* [65][11][62] e conhecimento teórico. Finalmente os parâmetros rotulados por $*$ têm um nível de confiança baixo devido ao suporte fornecido exclusivamente pelo conhecimento teórico adquirido através da literatura [65] e experiência pessoal no uso do Hadoop.

Vale ressaltar que essa lista foi baseada tanto na literatura relacionada [65][4][62] como no conhecimento teórico a respeito dos parâmetros dos Hadoop; por este último motivo algumas das associações entre parâmetros e métricas ainda precisam ser evidenciadas empiricamente. Desse modo, a lista exibida na Tabela 2.1 pode ser usada como um passo inicial na identificação dos parâmetros possivelmente relevantes para a execução das aplicações sobre o Hadoop, levando em consideração o aspecto que se deseja explorar para melhorar a eficiência dessa plataforma.

Legenda	
***	Confiança Alta no Impacto
**	Confiança Média no Impacto
*	Confiança Baixa no Impacto
-	Sem Confiança no Impacto

Tabela 2.1: Parâmetros de configuração do Hadoop que podem ter impacto sobre algumas métricas de eficiência

Id	Parâmetro	Arquivo xml	Descrição	*Default*	Métricas Associadas	Nível de Confiança
1	io.sort.mb	mapred-site	Tamanho (em MB) do *buffer* usado pelas tarefas *maps* para ordenação	100	Tempo de resposta	***
					Vazão de *jobs*	***
					Consumo de Energia	*
2	io.sort.spill. percent	mapred-site	Limiar de io.sort.mb (em %) para iniciar a escrita dos dados em disco	0.8	Tempo de resposta	**
					Vazão de *jobs*	**
					Consumo de Energia	*
3	io.sort.record. percent	mapred-site	Porcentagem de *io.sort.mb* reservado para armazenamento de metadados	0.05	Tempo de resposta	***
					Vazão de *jobs*	***
					Consumo de Energia	*
4	io.sort.factor	mapred-site	Número máximo de *streams* para unir as saídas das tarefas *map* em um *tasktracker*	10	Tempo de resposta	***
					Vazão de *jobs*	***
					Consumo de Energia	*
5	mapred.compress. map.output	mapred-site	Habilita o uso da compressão sobre as saídas das tarefas *map*	*false*	Tempo de resposta	***
					Vazão de *jobs*	***
					Consumo de Energia	*
6	mapred.jobtracker. taskScheduler	mapred-site	Seleciona a política de escalonamento dos *jobs* aplicada pelo *jobtracker*	org.apache. hadoop. mapred. jobs.JobQueue TaskScheduler	Tempo de resposta	**
					Vazão de *jobs*	***
					Consumo de Energia	-

Continua na próxima página

Tabela 2.1 – Continuação

Id	Parâmetro	Arquivo xml	Descrição	*Default*	Métricas Associadas	Nível de Confiança
7	mapred.reduce. tasks	mapred-site	Número total de tarefas *reduce* por *job*	1	Tempo de resposta	***
					Vazão de *jobs*	***
					Consumo de Energia	*
8	mapred.tasktracker. map.tasks. maximum	mapred-site	Número máximo de tarefas *map* executadas simultaneamente em um *tasktracker*	2	Tempo de resposta	***
					Vazão de *jobs*	***
					Consumo de Energia	**
9	mapred.tasktracker. reduce.tasks. maximum	mapred-site	Número máximo de tarefas *reduce* executadas simultaneamente em um *tasktracker*	2	Tempo de resposta	***
					Vazão de *jobs*	***
					Consumo de Energia	**
10	mapred.child. java.opts	mapred-site	Quantidade de memória RAM alocada a cada JVM que executa tarefas *map* e *reduce*	-Xmx200m (200MB)	Tempo de resposta	**
					Vazão de *jobs*	***
					Consumo de Energia	*
11	mapred.child. ulimit	mapred-site	Quantidade máxima de memória virtual (em MB) usada para controlar tarefas *map* e *reduce*	*<não especificado>*	Tempo de resposta	*
					Vazão de *jobs*	*
					Consumo de Energia	-
12	mapred.reduce. parallel.copies	mapred-site	Número de *threads* usadas para buscar as saídas das tarefas *map* em paralelo	5	Tempo de resposta	*
					Vazão de *jobs*	*
					Consumo de Energia	-
13	mapred.job. shuffle.input. buffer.percent	mapred-site	Porcentagem do *heap* das tarefas *reduce* para armazenar (em *buffer*) as saídas das tarefas *map*	0.70	Tempo de resposta	**
					Vazão de *jobs*	**
					Consumo de Energia	*
14	mapred.job. shuffle.merge. percent	mapred-site	Limiar (em %) de *mapred.job. shuffle.input.buffer.percent* para desencadear a junção dos dados nos *workers reduce*	0.66	Tempo de resposta	*
					Vazão de *jobs*	*
					Consumo de Energia	*
15	mapred.job. reduce.input. buffer.percent	mapred-site	Porcentagem do *heap* das tarefas *reduce* para armazenar (em *buffer*) as saídas das tarefas *map* enquanto as tarefas *reduce* estão sendo executadas	0.0	Tempo de resposta	*
					Vazão de *jobs*	*
					Consumo de Energia	*

Continua na próxima página

Tabela 2.1 – Continuação

Id	Parâmetro	Arquivo xml	Descrição	*Default*	Métricas Associadas	Nível de Confiança
16	mapred.reduce. slowstart. completed.maps	mapred-site	Fração das tarefas *map* que precisam ser completadas antes que as tarefas *reduce* sejam escalonadas pelo *jobtracker*	0.05	Tempo de resposta	*
					Vazão de *jobs*	*
					Consumo de Energia	*
17	mapred.job. reuse.jvm. num.tasks	mapred-site	Número de tarefas a serem executadas por JVM	1	Tempo de resposta	**
					Vazão de *jobs*	**
					Consumo de Energia	*
18	mapred.map. tasks. speculative. execution	mapred-site	Esse mecanismo executa uma cópia de uma tarefa *map* supostamente retardatária em outro nodo	*true*	Tempo de resposta	**
					Vazão de *jobs*	**
					Consumo de Energia	*
19	mapred.reduce. tasks. speculative. execution	mapred-site	Esse mecanismo executa uma cópia de uma tarefa *reduce* supostamente lenta em outro nodo	*true*	Tempo de resposta	**
					Vazão de *jobs*	**
					Consumo de Energia	*
20	mapred.job. tracker. handler. count	mapred-site	Número de *threads* servidoras para o *jobtracker*. Deve ser ajustado proporcionalmente ao tamanho do *cluster*	10	Tempo de resposta	**
					Vazão de *jobs*	**
					Consumo de Energia	-
21	tasktracker. http.threads	mapred-site	Número total de *threads* para distribuir as dados de saída das tarefas *map* para as tarefas *reduce*	40	Tempo de resposta	**
					Vazão de *jobs*	**
					Consumo de Energia	-
22	dfs.namenode. handler. count	mapred-site	Número de *threads* servidoras para o *namenode*. Deve ser ajustado proporcionalmente ao tamanho do *cluster*	10	Tempo de resposta	**
					Vazão de *jobs*	**
					Consumo de Energia	-
23	dfs.datanode. handler.count	hdfs-site	Número de *threads* servidoras para o *datanode*. Deve ser ajustado proporcionalmente ao tamanho do *cluster*	10	Tempo de resposta	**
					Vazão de *jobs*	**
					Consumo de Energia	-
24	dfs.replication	hdfs-site	Fator de replicação dos blocos do sistema de arquivos distribuído	3	Tempo de resposta	**
					Vazão de *jobs*	**
					Consumo de Energia	*

Continua na próxima página

Tabela 2.1 – Continuação

Id	Parâmetro	Arquivo xml	Descrição	*Default*	Métricas Associadas	Nível de Confiança
25	dfs.block.size	hdfs-site	Tamanho (em *bytes*) do bloco usado pelo sistema de arquivos distribuído	67108864 (64MB)	Tempo de resposta	***
					Vazão de *jobs*	***
					Consumo de Energia	*
26	io.file.buffer.size	core-site	Tamanho de um *buffer* (em *bytes*) para operações de E/S	4096 (4KB)	Tempo de resposta	**
					Vazão de *jobs*	**
					Consumo de Energia	*
27	fs.inmemory. size.mb	core-site	Quantidade máxima de memória alocada para mesclar a saída das tarefas *map* nas tarefas *reduce*	200	Tempo de resposta	*
					Vazão de *jobs*	*
					Consumo de Energia	*

É possível configurar alguns parâmetros da Tabela 2.1 de forma a potencializar as métricas associadas aos mesmos. Isso pode ser feito com o auxílio de ferramentas de ajuste automático [36]. Os parâmetros não cobertos por tais ferramentas podem ser ajustados com o auxílio de algumas regras genéricas chamadas de *rules of thumbs*, discutidas a seguir.

2.2.4 Ajuste de Parâmetros Através de *Rules of Thumbs*

Com o intuito de auxiliar usuários na configuração do Hadoop, algumas empresas passaram a publicar, em *white papers* ou livros, equações matemáticas e regras que fornecem ajustes genéricos para alguns parâmetros baseando-se em informações da infraestrutura (*hardware*) ou apenas na experiência adquirida com o uso dessa plataforma. Essas equações e regras são conhecidas como *rules of thumbs*. Exemplos dessas regras podem ser encontrados em [11][62]. Observe que a maioria dos parâmetros passíveis de serem configurados pelas *rules of thumbs* podem ser encontrados na Tabela 2.1.

Devido a essa generalidade, a maioria das *rules of thumbs* acaba indicando valores imprecisos, uma vez que elas normalmente não levam em consideração que o tipo de aplicação pode ter efeito sobre a configuração, por exemplo. No entanto, há alguns parâmetros que podem ser ajustados considerando apenas as informações da infraestrutura onde a aplicação

é executada. Dentre estes, destacam-se os parâmetros responsáveis pelo ajuste do número máximo de tarefas *map* (a saber, *mapred.tasktracker.map.tasks.maximum*) e *reduce* (a saber, *mapred.tasktracker.reduce.tasks.maximum*) executadas simultaneamente e pelo ajuste da quantidade de memória RAM alocada para cada tarefa (a saber, *mapred.child.java.opts*). Dado que a formalização desses parâmetros é inexistente ou inadequada, é possível estabelecer algumas equações matemáticas para o ajuste dos mesmos, seguindo as orientações da literatura [65][62].

Como é possível conhecer o número de núcleos de CPU disponíveis em um nodo, o número máximo de tarefas MapReduce que podem executar simultaneamente em um *tasktracker*, valor designado por max_{tasks}, pode ser ajustado manualmente de acordo com a métrica de interesse. Por exemplo, para otimizar o *tempo de resposta* deve-se utilizar todos os núcleos de CPU disponíveis. No entanto, para a métrica *consumo de energia*, pode-se optar por uma redução no número de núcleos de CPU utilizados [61]. A Tabela 2.2 sumariza possíveis formas de se ajustar o valor de max_{tasks} baseando-se na métrica que se deseja otimizar; considerando num_{CPU} como o número de núcleos de CPU disponíveis em um nodo.

Tabela 2.2: Ajuste do valor de max_{tasks}

	Número de tarefas simultâneas
Métricas que envolvem eficiência no uso da CPU (e.g. tempo de resposta e vazão de *jobs*)	$max_{tasks} = 2 \cdot num_{CPU}$
Métricas que envolvem eficiência energética (e.g. consumo de energia)	*Depende da estratégia energética utilizada.*

Conforme a Tabela 2.2, para métricas que envolvem eficiência no uso da CPU, o valor de max_{tasks} corresponde ao dobro de num_{CPU}. Desse modo, eleva-se o nível de contenção da CPU e evita-se a ociosidade da mesma, ocasionada pela execução de muitas operações de leitura e escrita comumente realizadas pelas tarefas MapReduce, que têm o perfil orientado a E/S (do inglês, *I/O bound*) [55].

Dentre as max_{tasks} tarefas, define-se quantas delas serão reservadas, de forma independente, para as tarefas *map* e *reduce*. Devido à própria arquitetura do MapReduce [16], incide sobre as tarefas *map* uma carga maior de processamento. Apoiado nesse fato, há *rules*

of thumbs que indicam que o número de tarefas *map* executadas simultaneamente em um *tasktracker* (parâmetro *mapred.tasktracker.map.tasks.maximum*) deve ser maior que o número de tarefas *reduce* (parâmetro *mapred.tasktracker.reduce.tasks.maximum*) [65][4]; ou seja:

$$max_{maps} > max_{reduces} \tag{2.1}$$

onde max_{maps} corresponde ao parâmetro *mapred.tasktracker.map.tasks.maximum* e $max_{reduces}$ corresponde ao parâmetro *mapred.tasktracker.reduce.tasks.maximum*. Consequentemente, é observada a seguinte relação:

$$max_{tasks} = max_{maps} + max_{reduces} \tag{2.2}$$

A configuração de *mapred.child.java.opts*, por outro lado, depende tanto de informações do *hardware*, em particular da memória RAM, quanto da aplicação. Uma forma de configurar adequadamente esse parâmetro é fixá-lo em seu limite superior, que depende da quantidade de memória RAM disponível, pois qualquer uso desse recurso além do limite pode aumentar o número de trocas de processos entre memória e disco (*swapping*) o que incorre em uma séria queda de desempenho [65].

Uma equação que pode ser utilizada para auxiliar esse passo da configuração é exibida a seguir:

$$mem_{task} = \left\lfloor \frac{mem_{total} - mem_{proc}}{max_{tasks}} \right\rfloor \tag{2.3}$$

onde mem_{task} representa o parâmetro *mapred.child.java.opts*, mem_{total} representa a quantidade total de memória RAM da máquina e mem_{proc} representa a quantidade de memória RAM sendo utilizada por outros processos (incluindo os próprios *daemons* do Hadoop).

O parâmetro *mapred.inmem.merge.threshold*, designado por $buffer_{merge}$, controla o tamanho do *buffer* utilizado no lado *reduce* durante a cópia dos dados recebidos dos *maps*. Como esse mesmo controle é exercido pelo parâmetro *mapred.job.shuffle.merge.percent*, define-se:

$$buffer_{merge} = 0 \tag{2.4}$$

onde o valor 0 desabilita o controle desse *buffer*, por parte desse parâmetro. Dessa forma, reduz-se o número de parâmetros que, apesar de usarem diferentes critérios, exercem a mesma função [65].

Assim como os parâmetros discutidos nesta seção, há outros parâmetros que podem ser configurados através de *rules of thumbs*, mas que não puderam ser formalizados adequadamente, a saber: *tasktracker.http.threads*, *mapred.job.tracker.handler.count*, *dfs.namenode.handler.count*, *dfs.datanode.handler.count* e *dfs.replication*.

Configuração Condicionada

Como não há uma documentação formal sobre os valores limites de alguns parâmetros de configuração, o usuário pode realizar uma busca cega que pode conduzir a falhas no *job* em execução devido a uma reserva de recursos acima ou abaixo do potencial da infraestrutura, por exemplo. Para minimizar esse problema, esta subseção exibe a configuração condicionada de alguns parâmetros relacionados àqueles que formalizados pelas *rules of thumbs* supracitadas.

Um desses parâmetros a serem configurados de forma condicionada é o *io.sort.mb* (veja sua descrição na Tabela 2.1). O valor desse parâmetro não pode ser superior à quantidade de memória RAM estabelecida para cada tarefa (*mapred.child.java.opts*). Formalmente tem-se:

$$0 < buffer_{sorting} \leq mem_{task} \tag{2.5}$$

onde $buffer_{sorting}$ corresponde ao parâmetro *io.sort.mb*. Essa restrição é imposta simplesmente porque não é possível reservar para o *buffer* de um *map* uma quantidade maior que a própria memória RAM alocada a ele.

Embora pareça natural maximizar o valor de $buffer_{sorting}$, há registros de que essa medida pode reduzir o desempenho das tarefas *map* devido ao custo da fase de escrita dos dados desta tarefa em disco (do inglês *spill fase*) [35]. Nesse caso, torna-se claro que o ajuste de alguns parâmetros é complexo e exige o auxílio de uma ferramenta que identifique o valor mais eficiente para o cenário corrente.

Um parâmetro que também está relacionado com recursos de *hardware* é *mapred.child.ulimit*, responsável por definir a quantidade máxima de memória virtual para

controlar as tarefas *map* e *reduce*. Se o ajuste desse parâmetro for declarado, seu valor deve ser maior que o definido para *mapred.child.java.opts*. Formalmente, tem-se:

$$mem_{virtual} \geq mem_{task} \tag{2.6}$$

onde $mem_{virtual}$ representa o parâmetro *mapred.child.ulimit*. Essa *rule of thumb* está presente na própria documentação do Hadoop [60].

Outro parâmetro que apresenta a configuração condicionada é o *io.sort.record.percent* (veja sua descrição na Tabela 2.1). Um valor muito grande para *io.sort.record.percent* pode deixar o tamanho da reserva de metadados equivalente ao tamanho do próprio *buffer* de ordenação (*io.sort.mb*). Empiricamente foi mostrado que valores entre 0.05 e 0.45 podem impactar positivamente no *tempo de resposta* da aplicação [4][35]. Portanto, é possível formalizar a seguinte *rule of thumb*:

$$0.05 < buffer_{metadata} \leq 0.45 \tag{2.7}$$

onde $buffer_{metadata}$ corresponde ao parâmetro *io.sort.record.percent*.

O número total de tarefas *map* por *job*, designado por num_{maps}, é definido pela razão entre o tamanho total dos dados de entrada e o tamanho do bloco do sistema de arquivos distribuído (parâmetro *dfs.block.size*). Formalmente, tem-se:

$$num_{maps} = \frac{input_{size}}{block_{size}} \tag{2.8}$$

onde $input_{size}$ representa o tamanho total dos dados de entrada e $block_{size}$ representa o parâmetro *dfs.block.size*. Em *clusters* de pequeno porte (com aproximadamente 5 a 10 nodos), é aconselhável utilizar-se valores maiores que o *default*: 128MB, por exemplo. Essa medida reduz a sobrecarga decorrente da criação de muitas tarefas *map* [62].

Resumindo o que foi discutido nessas últimas seções, a Tabela 2.3 sumariza as *rules of thumbs* formalizadas para o ajuste direto de alguns parâmetros de configuração, assim como algumas restrições sobre tais ajustes.

Tabela 2.3: *Rules of thumbs* formalizadas neste trabalho

Parâmetro Hadoop	Descrição	Variável Simbólica	*Rules of thumbs*
mapred.tasktracker. map.tasks.maximum	Número máximo de tarefas *map* executadas simultaneamente em um *tasktracker*	max_{maps}	Para métricas que envolvem eficiência de CPU: $max_{tasks} = 2 \cdot num_{CPU}$
mapred.tasktracker. reduce.tasks.maximum	Número máximo de tarefas *reduce* executadas simultaneamente em um *tasktracker*	$max_{reduces}$	tal que: $max_{tasks} = max_{maps} + max_{reduces}$ sujeito a: $max_{maps} > max_{reduces}$
mapred.child. java.opts	Quantidade de memória RAM alocada a cada JVM que executa tarefas *map* e *reduce*	mem_{task}	$mem_{task} = \left\lfloor \frac{mem_{total} - mem_{proc}}{max_{tasks}} \right\rfloor$ tal que: mem_{total} corresponde à quantidade total de memória RAM e mem_{proc} corresponde à quantidade de memória RAM sendo utilizada por outros processos sujeito a: $mem_{virtual} \geq mem_{task}$
mapred.inmem. merge.threshold	Tamanho limiar do *buffer* utilizado no lado *reduce* durante a cópia dos dados recebidos dos *maps*	$buffer_{merge}$	$buffer_{merge} = 0$

Continua na próxima página

Tabela 2.3 – Continuação

Parâmetro Hadoop	Descrição	Variável Simbólica	*Rules of thumbs*
dfs.block.size	Tamanho do bloco do sistema de arquivos distribuído	$block_{size}$	$num_{maps} = \frac{input_{size}}{block_{size}}$ tal que: num_{maps} corresponde à quantidade total de tarefas *map* por *job* e $input_{size}$ corresponde ao tamanho total dos dados de entrada sujeito a: $block_{size}$ condicionado ao tamanho do *cluster*

2.3 Consideração Finais do Capítulo

Este capítulo apresentou o MapReduce: um modelo de programação para aplicações intensivas de dados. Foi apresentada uma visão geral sobre a execução de aplicações MapReduce, além de uma discussão envolvendo pontos de tolerância a falhas e gerência de recursos.

A seguir foi apresentado o Hadoop: uma plataforma MapReduce de código aberto muito utilizada atualmente. Após o levantamento de algumas de suas características, foram descritos seus principais componentes e mecanismos de configuração.

Uma das contribuições deste trabalho foi documentada neste capítulo: a formalização de algumas *rules of thumbs* (resumidas na Tabela 2.3), como as que definem a quantidade de *slots* para as tarefas *map* e *reduce* simultaneamente. As equações mostradas podem auxiliar o usuário no ajuste eficiente do Hadoop e serão úteis durante a aplicação do processo proposto, descrito no Capítulo 4.

Capítulo 3

Trabalhos Relacionados

Este capítulo apresenta alguns trabalhos relacionados à configuração eficiente do Hadoop e de sistemas complexos em geral. É feita uma análise buscando destacar semelhanças e diferenças desses trabalhos com o que é proposto nesta dissertação.

Para uma melhor organização deste capítulo, os trabalhos discutidos estão divididos em quatro seções. Na Seção 3.1 são apresentadas algumas pesquisas sobre a configuração de sistemas complexos. Na Seção 3.2 são discutidos trabalhos relacionados à avaliação de desempenho do Hadoop com ênfase nos que abordam o ajuste eficiente de seus parâmetros de configuração. Na Seção 3.3 são discutidos alguns trabalhos envolvendo o uso de técnicas empíricas e estatísticas para solução de problemas em diversos domínios. Finalmente, Seção 3.4 descreve uma ferramenta para *tuning* automático e adaptativo do Hadoop: o Starfish [36]. Essa ferramenta recebe um destaque maior neste trabalho devido à sua relevância no estudo de caso realizado envolvendo a instanciação do processo proposto (vide Capítulo 5).

3.1 Configuração de Sistemas Complexos

A preocupação com a configuração de sistemas complexos não está restrita às plataformas de alto desempenho. Brown et al [10], do Centro de Pesquisas da IBM, defendem que a complexidade de configuração de sistemas é um obstáculo para a adoção de novos produtos para o setor de TI (Tecnologia de Informação) e é responsável pelo aumento de custos em serviços nesse setor. Isso motivou a construção de um modelo para reduzir tal complexidade.

Nesse caso, o modelo representa um sistema como um conjunto de componentes organizados hierarquicamente. Através dessa representação é possível derivar métricas que indicam a complexidade de algumas configurações [10].

Baseados na pesquisa de Brown et al, dois pesquisadores do Laboratório Fujitsu, Kikuchi e Tsuchia, apresentam um método para gerar procedimentos capazes de alterar a configuração de sistemas automaticamente [43]. Este método se baseia em fórmulas de lógica de primeira ordem e teoria dos conjuntos, além de coletar dados sobre informações de configuração e conhecimento do gerenciamento do sistema alvo.

Quando dois sistemas são similares é possível transferir a experiência de configuração de um para o outro; hipótese defendida por Chen et al [12]. Nesse trabalho, eles definem um modelo baseado em Redes Bayesianas e apresentam um algoritmo para descobrir a configuração otimizada de parâmetros de sistemas complexos. A Rede Bayesiana auxilia no registro de dependências de parâmetros para acelerar as buscas de configuração em novos sistemas.

Seguindo uma estratégia diferente desses trabalhos, a proposta levantada por essa dissertação se foca em técnicas empíricas que auxiliam na definição de uma série de ensaios a partir dos quais é possível aferir a métrica de interesse. A partir de então são utilizados recursos estatísticos para guiar a tomada de decisão no que diz respeito a escolha dos parâmetros significativamente relevantes.

3.2 Análise de Desempenho do Hadoop

A pesquisa de Zaharia et al [70] explora a execução especulativa do Hadoop através do escalonador LATE (do inglês *Longest Approximate Time to End*). Através da previsão do tempo de resposta das tarefas do Hadoop, o escalonador é capaz de identificar aquelas tarefas que excedem esse tempo de término e criam cópias das mesmas em outros nodos, "especulando" que estas devem terminar primeiro. Isso adiciona uma camada de tolerância a falhas não permitindo que tarefas supostamente defeituosas apresentam impacto negativo sobre o desempenho global do *job*. Esse foi um dos primeiros trabalhos relevantes encontrados a investigar como melhorar o desempenho do Hadoop.

No que se refere à configuração da plataforma Hadoop, a pesquisa de Ekanayake et al descreve a plataforma CGL-MapReduce [20]. Diferente do Hadoop, que armazena os dados

intermediários em arquivo, o CGL-MapReduce utiliza *streaming* para enviar os resultados intermediários diretamente dos produtores (*maps*) para os consumidores (*reduces*) reduzindo a sobrecarga de comunicação. Ekanayake et al executam um estudo de caso para comparar o desempenho das duas plataformas, no qual se observa que o Hadoop apresenta um desempenho inferior ao CGL-MapReduce.

Uma linha de pesquisa diferente adotada por Pavlo et al foi comparar o Hadoop com DBMS paralelos: Vertica e DBMS-X [52]. Para tanto, eles desenvolveram um *benchmark* composto por cinco tarefas, uma delas retiradas do próprio artigo original sobre MapReduce [16]. Observou-se que o desempenho dos DBMS foi superior ao Hadoop, uma vez que cai sobre o modelo MapReduce uma penalidade devido ao custo da chamada materialização (persistência) dos dados intermediários entre as fases *map* e *reduce*.

Os trabalhos discutidos acima não documentam como o Hadoop foi configurado durante os experimentos. Com isso, não se pode afirmar que tais comparações foram justas, uma vez que não é possível afirmar de que maneira os parâmetros foram ajustados e se tais valores impactaram na eficiência do Hadoop. Na comparação de Ekanayake et al, por exemplo, como o *buffer* usado pelas tarefas *maps* foi ajustado? O ajuste adequado desse parâmetro não poderia trazer uma melhoria de desempenho ao Hadoop?

A partir deste ponto, os trabalhos que serão citados se aproximam mais com a proposta desta dissertação devido à documentação de alguns parâmetros de configuração do Hadoop ajustados nos estudos de caso realizados. A partir dos resultados desses trabalhos é possível perceber que o desempenho das aplicações melhora quando a plataforma é devidamente configurada.

O trabalho de Jiang et al [40] começou a pesquisar profundamente os fatores que podem interferir no desempenho de um *job* no Hadoop, a citar: a técnica de leitura que as tarefas *map* utilizam para carregar os dados do sistema de arquivos, a conversão dos dados brutos em registros e a estratégia de comparação de chaves usada durante a ordenação. Se implementados de forma apropriada, o desempenho dos sistemas baseados em MapReduce pode aumentar na magnitude de 2,5 a 3,5 vezes. Assim como Pavlo et al, Jiang et al também realizaram experimentos comparativos entre o Hadoop e alguns DBMS. Eles documentaram os valores ajustados de alguns parâmetros de configuração do Hadoop utilizados nos experimentos, mas não documentam qualquer critério que utilizam para tal escolha.

Babu [4] publicou um *position paper* que se focou na análise de um conjunto de parâmetros de configuração que interferem no desempenho do Hadoop. O estudo de caso realizado por ele envolveu o ajuste desses parâmetros em diferentes níveis, tornando possível observar como o tempo de resposta das aplicações pode ser reduzido com a configuração eficiente dessa plataforma.

Apesar do tempo de resposta ser a métrica de desempenho mais utilizada na literatura, algumas pesquisas, relativamente recentes, como a de Kaushik e Bhandarkar [42] consideram o consumo de energia como uma importante métrica para a configuração do Hadoop. Nessa pesquisa eles descrevem como o Green HDFS, um sistema de arquivos distribuído para o Hadoop, permite uma redução do consumo de energia através da disposição dos dados em duas diferentes classes de servidores: *Cold Zone*, que garante longos períodos de inatividade sem prejudicar o desempenho do sistema, e a *Hot Zone*, onde há intensa utilização dos mesmos.

O consumo de energia também é uma métrica considerada por Wirtz e Ge [66] para a análise de *jobs* no Hadoop. Para isso eles utilizam, além da alocação de recursos, a tecnologia DVFS (do inglês *Dynamic Voltage and Frequency Scaling*) que permite ajustar dinamicamente a frequência da CPU baseada no tipo de *workload*, explorando configurações que permitam reduzir o consumo de energia sem perda de desempenho.

Outra métrica que vem ganhando espaço é a vazão de *jobs*. Essa métrica vem sendo explorada em ambientes onde há grandes *clusters* e usuários se utilizando de tais infraestruturas para lançar *jobs* (frequentemente pequenos) concorrentemente; tanto no Hadoop como em outras plataformas de computação intensiva. Este cenário é explorado por Isard et al [38], que descrevem um *framework* para o escalonamento de *jobs* concorrentes com compartilhamento de recursos. Eles realizam um estudo de caso com o Dryad, uma plataforma que implementa o modelo DataflowGraph [37].

O trabalho de Grover e Carey [32] explora a vazão de *jobs* no Hadoop através de um cenário onde usuários executam *jobs* concorrentemente através de linguagens de consulta de alto nível (como SQL, por exemplo); segundo eles, um cenário similar ao utilizado pelo Facebook®. Especificamente, eles exploram *jobs* que realizam a amostragem dos dados baseada em predicados, o que não requer que a entrada inteira do *job* seja executada.

É possível notar que os trabalhos descritos nesta seção se concentram em um único as-

pecto, seja *desempenho* ou *eficiência energética*. Não foram encontrados trabalhos que avaliam outras métricas, como por exemplo: *utilização de memória RAM e disco*, *consumo de largura de banda* e *tempo médio entre falhas*.

Desse modo, o processo proposto nesta dissertação é flexível, pois não restringe as métricas que podem ser avaliadas.

3.3 Técnicas Empíricas

Apesar da configuração de sistemas complexos baseada em técnicas estatísticas ainda não ser bem documentada e explorada na prática, é possível destacar algumas pesquisas que se utilizam de tais técnicas tanto para a configuração do Hadoop quanto para a resolução de problemas de diferentes domínios.

A técnica de balanceamento dinâmico de carga (DLB, do inglês *Dynamic Load-Balancing*) pode ser utilizada pela Engenharia de Tráfego para distribuir o tráfego de rede com o propósito de alcançar determinado objetivo usando um modelo de fila $M/M/1$ [47].

Larroca e Rougier [44] citam falhas desse modelo em fila e propõem um *framework* que gera um modelo de regressão múltipla para estimar uma função de tempo de atraso a partir de medições realizadas. Outro trabalho que utiliza modelos de regressão é o de Ekren e Heragu no domínio de sistemas veiculares [21].

Aproximando-se do contexto de plataformas de alto desempenho destaca-se o trabalho de Rizvandi et al [53], que descreve um método para configuração do Hadoop através da construção de séries temporais de uso de CPU com o propósito de calcular a similaridade entre *jobs*. O cálculo proposto por Rizvandi et al é realizado através de coeficientes de correlação dos padrões analisados. Apesar de ser uma boa técnica, os padrões dos *jobs* documentados nesse trabalho são baseados somente no consumo de CPU. Uma análise baseada no consumo de outros recursos (memória, disco, etc.) poderia fornecer informações mais ricas.

Uma área onde a configuração otimizada é muito abordada é a de ajuste de sistemas de gerenciamento de banco de dados. Debnath et al [18] propõem uma metodologia para encontrar parâmetros de configuração que impactam no desempenho dos DBMS. Assim como o processo proposto nesta dissertação, eles utilizam técnicas estatísticas para elencar os parâmetros com impacto mais significativo.

Estendendo uma técnica originalmente criada para análise de desempenho em bancos de dados distribuídos, a KCCA (do inglês *Kernel Canonical Correlation Analysis*) [27], Ganapathi et al [26] descrevem e avaliam um *framework* para predizer o tempo de execução de *jobs* Hadoop através da correlação de características da pré-execução e métricas de desempenho da pós-execução.

3.4 Descrição do Starfish

A utilização de ferramentas para *tuning* de sistemas não é uma solução recente. O trabalho de Chen [14] descreve um otimizador que utiliza algoritmos baseados em regras e busca aleatória para auxiliar no ajuste eficiente de sistemas de entrada e saída paralela.

Essas ferramentas foram sendo refinadas para permitir um ajuste automático e adaptativo, ajustando os parâmetros do sistema alvo e usando técnicas como funções de custo para avaliar os benefícios desses ajustes [34]. Algumas ferramentas com tais melhorias começaram a ser designadas pelo acrônimo STAR, do inglês *Self-Tuning Adaptive Routines* [22].

Nesse contexto, Herodotou et al desenvolveram na Universidade de Duke (Durham, EUA) o Starfish [36]: um sistema de ajuste automático e adaptativo para análise de aplicações *Big Data* no Hadoop.

Arquiteturalmente, o Starfish é implantado sobre a pilha de camadas do Hadoop como mostra a Figura 3.1. Observe que um *job* pode ser gerado diretamente a partir de um programa escrito em Java, Python, C++, etc., a partir de consultas em Pig Latin ou HiveQL ou submetido como parte de um *workflow* [69] por serviços como o Amazon® Elastic MapReduce [36][45].

Como o *Starfish* dispõe de muitas funcionalidades, seus componentes podem ser elencados em três níveis de operação: *job*, *workflow* e *workload* como mostra a Figura 3.2. A seguir será discutido cada um destes componente, a citar: o *profiler*, o *sampler*, o motor *what-if*, o otimizador *just-in-time*, o escalonador *workflow-aware*, o otimizador de *workload* e o *elastisizer*.

No contexto do Starfish, um perfil é uma representação concisa da execução de um *job*. Tecnicamente, ele consiste do fluxo de dados e das estimativas de custo desse *job*. Essa estimativa é calculada através de uma função de custo $F(p, d, r, c)$ que recebe informações da

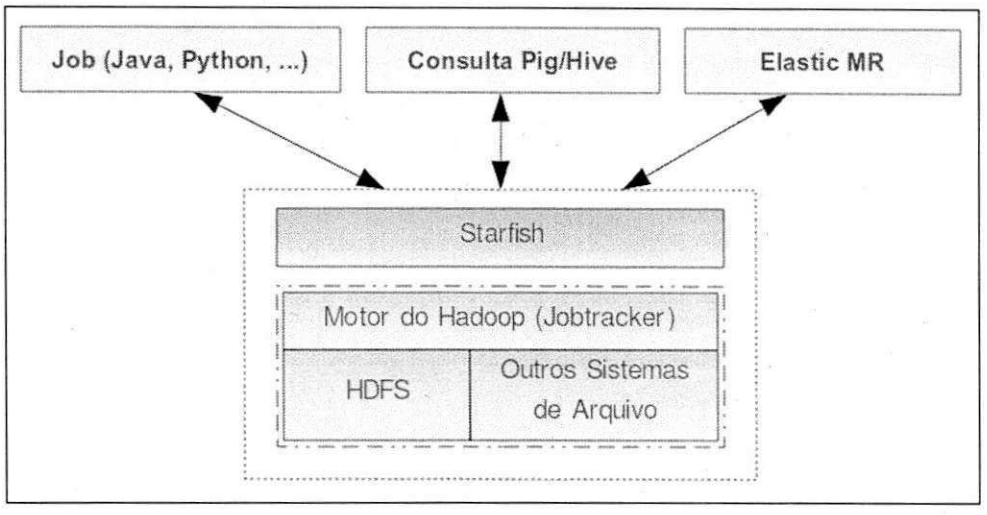

Figura 3.1: O Starfish na pilha de camadas do Hadoop

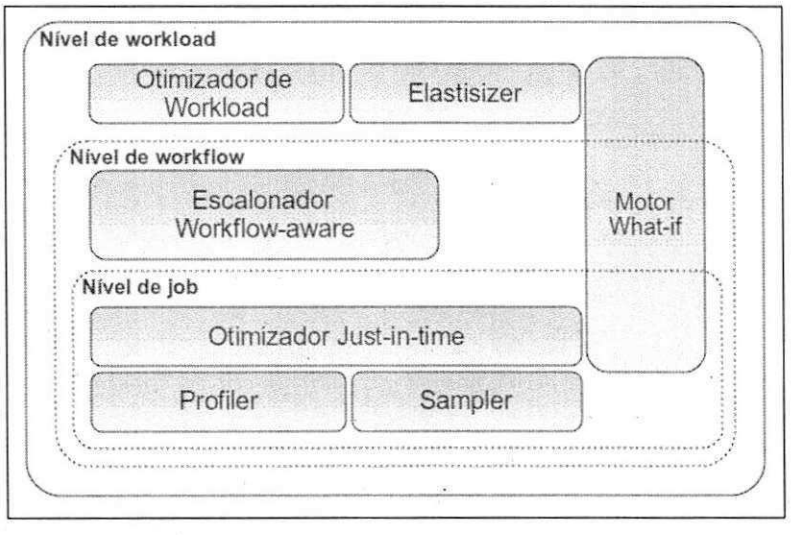

Figura 3.2: Arquitetura do Starfish

aplicação executada p, dos dados de entrada d, dos recursos disponíveis r e de um conjunto de parâmetros de configuração c.

O perfil gerado auxilia na compreensão do comportamento do *job* assim como no diagnóstico de gargalos durante a execução. A criação de um perfil é de responsabilidade do *profiler*. Para isso, este se utiliza de uma técnica de instrumentação dinâmica (através da ferramenta BTrace[1]), coletando as informações do *job* em tempo de execução com o auxílio de outro componente: o *sampler*.

Dado o perfil de um *job* $j = <p, d_1, r_1, c_1>$ executado sobre a aplicação p, dados de entrada d_1, usando os recursos r_1 e um conjunto de configurações c_1 o motor *what-if* informa qual é o desempenho dessa mesma aplicação p se ela for executada sobre os dados de entrada d_2, usando os recursos r_2 e um conjunto de configurações c_2. Desse modo tem-se um novo *job*, representado por $j' = <p, d_2, r_2, c_2>$. Observando a Figura 3.2 é possível notar que o motor *what-if* é utilizado em todos os níveis da arquitetura do Starfish.

Para otimizar os planos de execução em *workflows* e a configuração dos *jobs* o Starfish utiliza as funcionalidades do otimizador *just-in-time*. Este componente recorre ao *profiler* e ao *sampler* para a tomada de decisões de configuração e execução do *job*.

O nível de *workflow* está sujeito a muitas políticas de execução de *jobs* nas quais é possível realizar alguma otimização como, por exemplo, a disposição dos dados feita pelo sistema de arquivos distribuído. Ao invés de tomar decisões considerando um *job*, o escalonador *workflow-aware* toma decisões considerando os *jobs* que escrevem dados (denominados de produtores) e os que lêem dados (denominados de consumidores), a partir de um determinado *workload*. Nesse caso, o escalonador interage com o motor *what-if*.

O otimizador de *workload* representa os *workloads* como grafos dirigidos e aplica algumas transformações, como: grafo-para-grafo [36]. Ele também utiliza o motor *what-if* para fazer uma previsão de desempenho baseada em custos.

O *elastisizer* auxilia usuários de computação na nuvem que utilizam a política *pay-as-you-go*[2]. Esse componente determina automaticamente as melhores configurações para processar um determinado *workload* sujeito a objetivos específicos do usuário como: custos financeiros e tempo de conclusão dos *jobs*.

Para avaliar o Starfish foram realizados estudos de caso envolvendo a execução das apli-

[1]Uma ferramenta de *tracing* para a plataforma Java. Disponível em: *http://kenai.com/projects/btrace.*

cações WordCount e TeraSort processando *workloads* de 30 Gb e 50 Gb. respectivamente. Para ambos os estudos de caso foi realizada uma comparação dos resultados do Starfish com os resultados de *jobs* configurados por meio de *rules of thumbs* (vide Seção 2.2.4). A análise dos resultados foi focada em aspectos de desempenho, como porcentagem de uso do disco, porcentagem de *overhead* e tempo de resposta. No entanto, vale ressaltar que a otimização realizada pelo Starfish leva em consideração somente o tempo de resposta.

3.5 Consideração Finais do Capítulo

Este capítulo apresentou alguns trabalhos relacionados à configuração de sistemas complexos, em particular a plataforma Hadoop. Além disso foi feita uma revisão sobre o uso de técnicas empíricas de estatísticas para a resolução de problemas.

Através dessa revisão é possível concluir que as técnicas e ferramentas existentes para a configuração adequada do Hadoop apresentam limitações relacionadas aos subespaços de parâmetros considerados ou à quantidade de métricas observadas.

Para preencher essa lacuna, esta dissertação propõe um processo flexível para configuração eficiente do Hadoop baseado em técnicas empíricas e estatísticas. Esse processo é capaz de avaliar qualquer subconjunto de parâmetros de configuração sob a observação de qualquer aspecto, como desempenho e eficiência. O Capítulo 4 descreve todas as atividades e características desse processo de forma detalhada.

[2]Termo aplicado a serviços os quais o usuário paga pelos recursos que consome.

Capítulo 4

Processo Empírico Proposto

Este capítulo descreve, de forma detalhada, o processo proposto para auxiliar na configuração eficiente do Hadoop. Este processo é baseado em técnicas empíricas e estatísticas bem conhecidas na literatura e gera como resultado uma configuração eficiente para a métrica observada, considerando todos os parâmetros relevantes selecionados pelo usuário.

Na seção a seguir será definido o processo e nas seções seguintes serão apresentadas cada uma de suas atividades.

4.1 Definição do Processo Empírico

Segundo Sommerville [56], um processo é um conjunto de atividades usado para se atingir uma meta. Nesse sentido, o processo empírico proposto compreende as atividades de *definição de uma métrica de interesse*, *seleção dos parâmetros candidatos* a relevantes, *análise de relevância* desses parâmetros e, finalmente, *refinamento e otimização*.

A Figura 4.1 mostra como as atividades do processo se relacionam. Observe que dadas algumas condições, é possível retroceder às duas atividades iniciais; isso caracteriza o aspecto cíclico do processo.

4.2 Atividade 1: Definição da Métrica de Interesse

No contexto do processo proposto, definir a métrica de interesse compreende escolher um critério de avaliação a ser utilizado durante a execução da aplicação desejada. Baseado nesse

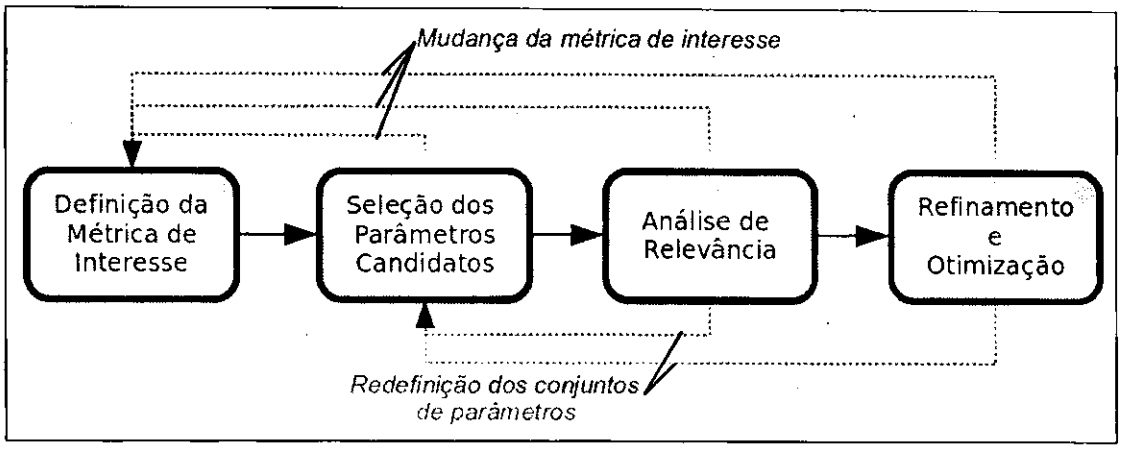

Figura 4.1: Relacionamento das atividades do processo proposto

critério serão selecionados os parâmetros que serão investigados visando otimizar a própria métrica escolhida. São exemplos de métricas: *tempo de resposta, vazão de jobs, utilização (de CPU, de memória RAM, etc.) consumo de largura de banda* e *consumo de energia.*

É importante que essa definição seja feita no início do processo, pois a métrica escolhida deve guiar as decisões a serem tomadas nas atividades posteriores. Se no decorrer da aplicação do processo a métrica precisar ser alterada, o processo inteiro deve ser reiniciado.

4.3 Atividade 2: Seleção dos Parâmetros Candidatos

A Atividade 2 compreende a escolha de um conjunto $L = \{p_1, p_2, ..., p_k\}$ com k parâmetros candidatos a relevantes, considerando a métrica escolhida na Atividade 1. Será chamado de H o espaço de busca disponível ao usuário, $H = \{p_1, p_2, ..., p_{|H|}\}$, em outras palavras, o conjunto H contém todos os parâmetros de configuração do Hadoop. Uma consequência direta dessa definição é que $L \subseteq H$.

A seleção do conjunto L deve ser baseada na métrica de interesse, pois diferentes métricas podem estar associadas a diferentes parâmetros de configuração. Para auxiliar nessa escolha, o usuário pode se basear na Tabela 2.1 (vide Capítulo 2) que lista alguns parâmetros possivelmente impactantes para os aspectos de desempenho e eficiência energética. Baseado nessa tabela, o usuário pode definir seu conjunto L e verificar (em atividades posteriores) quais parâmetros desse conjunto são significativamente relevantes.

Para métricas de desempenho, por exemplo, parâmetros que envolvem o ajuste de *buffers*, a definição da quantidade de tarefas *map* e *reduce* e a alocação de memória RAM e CPU normalmente são relevantes [65][4].

É importante ressaltar que a quantidade de parâmetros selecionados nesta atividade, i.e. o valor de k, irá impactar nas atividades posteriores: quanto mais parâmetros forem selecionados, maior o tempo necessário para realizar a análise de relevância, refinamento e otimização. Essa questão é melhor esclarecida na Seção 4.4.1.

4.4 Atividade 3: Análise de Relevância

Nesta atividade, cada parâmetro pertencente ao conjunto L é analisado considerando a sua relevância para a métrica definida na Atividade 1. Se o impacto de um parâmetro na métrica de interesse for significativamente irrelevante, ele será descartado na atividade seguinte. Para essa análise é necessário realizar um planejamento de experimentos guiado por um *projeto fatorial* [39].

Um planejamento de experimentos tem como objetivo obter o máximo possível de informações com o número mínimo de ensaios experimentais (ou observações). A análise apropriada do experimento seguindo um projeto fatorial permite separar o efeito de vários fatores (variáveis independentes) que podem afetar uma variável de resposta (variável dependente); além de determinar se um fator tem um impacto significativo ou se a diferença observada é atribuída a variações aleatórias causadas por erros experimentais ou fatores não controlados [39]. Os erros experimentais consistem nas diferenças entre os respectivos valores medidos e o valor esperado.

Através de um projeto fatorial 2^k, é possível determinar o efeito de k fatores sobre uma métrica de interesse realizando um número relativamente reduzido de ensaios experimentais, pois considera-se apenas 2 níveis: valores mínimo e máximo não-extremos dentro do domínio do fator, compondo uma faixa relativamente grande e bem delimitada. No projeto fatorial os níveis são codificados para uma escala entre -1 e 1, permitindo que se normalize os valores dos fatores, que se elimine as unidades de medidas de cada um deles e que a análise pós-experimento seja padronizada.

No contexto do processo proposto, os fatores são os parâmetros do Hadoop a serem con-

figurados para uma aplicação específica. Desse modo, vários ensaios experimentais devem ser realizados executando a aplicação em diferentes tratamentos, isto é conseguido variando-se os fatores (parâmetros de configuração), e medindo o resultado dessa variação específica na métrica de interesse, como por exemplo: o consumo de energia. Um tratamento é um conjunto bem definido dos níveis de cada parâmetro para um ensaio experimental específico.

Os níveis codificados podem ser agrupados em um instrumento chamado *tabela de sinais* [39]. A Tabela 4.1 exibe um exemplo de uma tabela de sinais para um cenário com 2 fatores (parâmetros): A =*io.sort.mb* e B =*io.sort.record.percent*, cujos conjuntos de níveis correspondem a $A = \{100, 200\}$ e $B = \{0.1, 0.3\}$[1]. Para esse cenário, assuma que o parâmetro C =*mapred.child.java.opt*[2] é fixado em $-Xmx400m$.

Para o parâmetro A, por exemplo, observe que os níveis -1 e 1 correspondem aos valores 100 e 200, respectivamente. Note que tanto o valor mínimo (-1) quanto o valor máximo (1) não são extremos, uma vez que o intervalo dos valores extremos desse parâmetro é $[0, 400]$ (conforme o valor do parâmetro C). Desse modo, a faixa escolhida para esse experimento ($[100, 200]$) é bem definida e está contida dentro do intervalo dos valores extremos.

Tabela 4.1: Exemplo de uma tabela de sinais para o projeto fatorial

A (Cod.)	B (Cod.)	A	B
-1	-1	100	0.1
1	-1	200	0.1
-1	1	100	0.3
1	1	200	0.3

Uma variação do projeto fatorial 2^k incorpora o conceitos de *grau de replicação* r: número de vezes que o mesmo tratamento é executado. Por este motivo, esse projeto é conhecido como *projeto fatorial com replicação* [39] ou $2^k r$. O experimento realizado nesta atividade será feito utilizando-se o $2^k r$, permitindo-se estimar os erros experimentais. O projeto fatorial com replicação gera um total de $2^k \cdot r$ ensaios experimentais. No contexto do processo, k corresponde à quantidade de elementos de L.

Quando o número de observações ainda é muito grande, o usuário pode optar por sim-

[1] Ajustado conforme a Equação 2.7.

[2] A faixa $[-1, 1]$ do fator A foi definida em função do valor de C conforme a Equação 2.5.

plificar o projeto $2^k r$ em um *projeto fatorial fracionado* $2^{k-p} \cdot r$ tal que $p < k$. No entanto, quanto mais próximo de k for o valor de p, mais informações sobre interações entre os fatores serão perdidas. Por esse motivo, normalmente evita-se utilizar projetos experimentais fracionados. Desse modo, ajustando $p = 0$, retrocede-se ao $2^k r$ do projeto fatorial com replicação.

Os parâmetros de configuração selecionados na Atividade 2 correspondem aos chamados *fatores primários*: aqueles que têm seus níveis alterados durante o experimento. Dentro do planejamento do experimento também é importante definir se há *fatores secundários*: aqueles que têm seus níveis constantes durante o experimento, por não haver interesse em observá-los ou por não ser possível controlá-los. No contexto do processo, os parâmetros pertencentes aos conjuntos T e $H - L - T$ são fatores secundários. Para esses conjuntos, o ajuste dos parâmetros pode ser feito levando em consideração as *rules of thumbs* ou seus valores *default*, respectivamente. Essa conceitualização é esquematizada na Figura 4.2.

No cenário supracitado, os parâmetros A e B são *fatores primários* e o parâmetro C é um *fator secundário* em T (definido por *rules of thumbs*). Como alguns fatores secundários podem determinar os níveis de alguns fatores primários (como por exemplo, a relação entre os fatores A e C), é importante que se defina inicialmente os fatores secundários para, a partir de então, definir os fatores primários e seus respectivos níveis.

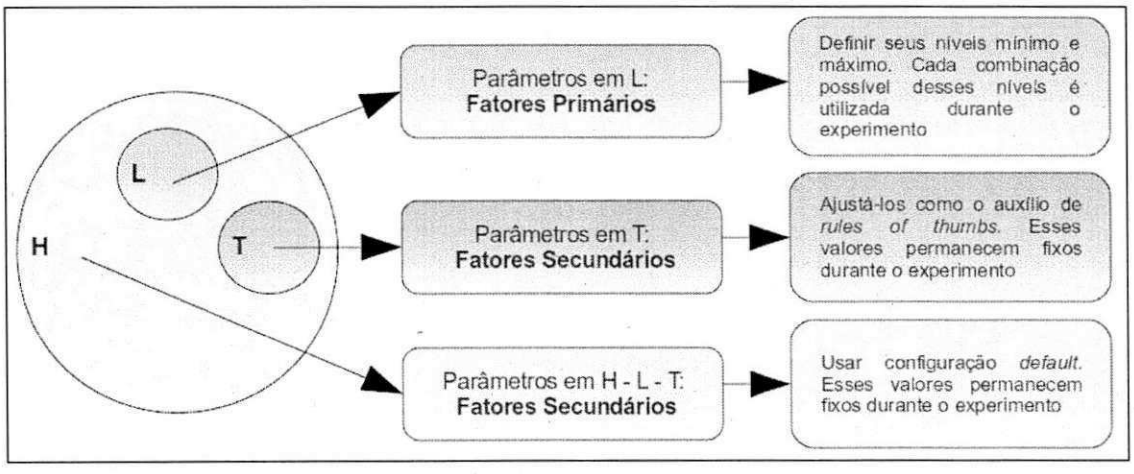

Figura 4.2: Esquema envolvendo a definição dos fatores primários e secundários

A partir do experimento guiado por projeto fatorial, são coletados valores da métrica de interesse os quais podem ser usados para calcular um modelo de *regressão múltipla* [39]:

uma fórmula usada para estimar o valor de uma variável dependente (métrica de interesse, no contexto deste processo) em função de um conjunto de variáveis independentes (parâmetros pertencentes ao conjunto L, no contexto deste processo).

Após o cálculo de modelo do regressão, é possível aplicar algumas técnicas estatísticas para verificar a adequação do próprio modelo e avaliar a relevância de cada parâmetro de configuração. Estas técnicas são o *coeficiente de determinação* e a *análise de variância.*

O *coeficiente de determinação* R^2 [39] é um índice que varia entre 0 e 1, através do qual é possível descobrir a qualidade do modelo estimado. Quanto mais próximo de 1 for R^2, mais adequada é a regressão para estimar a variável independente. No entanto, é importante ressaltar que o coeficiente de determinação não é um instrumento muito preciso para essa finalidade, uma vez que ele não define um limite entre o que seria uma regressão boa e uma ruim. Por esse motivo, esse coeficiente não deve ser usado como uma única ferramenta para a tomada de decisões.

Uma técnica que fornece resultados baseados em níveis de confiança é a *análise de variância* (ANOVA) [39], que consiste basicamente em comparar diferentes tratamentos baseados na soma dos quadrados[3].

A análise de variância é uma técnica que permite avaliar tanto a adequação do modelo de regressão quanto a relevância de cada variável independente. Para o primeiro caso utiliza-se o *teste-F* e para o segundo caso utiliza-se o *teste-t* pareado [39].

Os resultados de ambos os testes podem ser expressos em função de um índice chamado de *p-valor*: valor percentual usado como indicador de aceitação ou rejeição da hipótese nula (H_0), baseando-se em um nível de significância estabelecido para o experimento [67]. No contexto do processo proposto, se o *p-valor* de um parâmetro é inferior ao nível de significância estabelecido, esse parâmetro é considerado relevante.

Para que os resultados de todas as técnicas estatísticas supracitadas sejam confiáveis, os erros experimentais calculados precisam ser normalmente distribuídos, independentes e com variância constante (propriedade da homoscedasticidade) [39].

Para verificar se os erros são normalmente distribuídos, podem ser aplicados os testes de *Shapiro-Wilks* e *Anderson-Darling* [33]. Se os resultados desses dois testes divergirem o

[3]Do inglês *Sums of Squares*: medida baseada no somatório da diferença entre as amostra dos dados e a média da população.

usuário pode aplicar outros testes, como o de *Kolmogorov-Smirnov* ou o teste *Chi-Quadrado (para normalidade)*.

Além desses testes, é possível ter uma noção visual da normalidade através da construção de um *gráfico quantil-quantil* [39] como o que é exibido na Figura 4.3. Observe através dessa figura que quase todos os pontos estão sobre a reta ou muito próximos a ela. Isso indica uma forte evidência de normalidade.

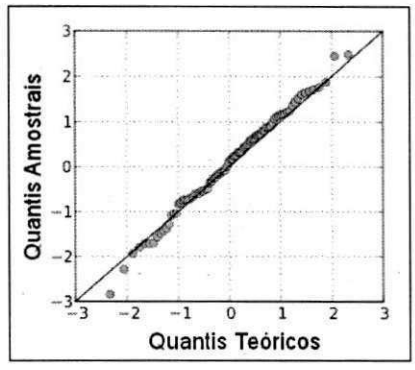

Figura 4.3: Exemplos de um gráfico quantil-quantil para o teste de normalidade

A verificação dos requisitos de independência e homoscedasticidade dos erros experimentais pode ser feita aplicando-se testes específicos como o *Chi-Quadrado (para independência)* [50] e o de *Bartlett* [23], respectivamente. Além desses testes, é possível ter uma noção visual dos requisitos de independência e homoscedasticidade através da análise de um *gráfico de dispersão* [33] considerando os erros experimentais em função das respostas estimadas.

As Figuras 4.4-a) e 4.4-b) mostram gráficos de dispersão em que a distribuição dos erros experimentais aparecem em padrões linear e não-linear, respectivamente. Nesse tipo de análise, qualquer enviesamento da distribuição dos pontos indica tendência dos erros experimentais. Desse modo, como na Figura 4.4-a) não há enviesamento, há indícios que os erros experimentais são independentes.

As Figuras 4.5-a) e 4.5-b) mostram gráficos de dispersão com distribuições de erros experimentais sem espalhamento e com espalhamento, respectivamente. Desse modo os erros experimentais na distribuição da Figura 4.5-a) apresentam uma variância constante, evidenciando que os erros podem estar respeitando o requisito da homoscedasticidade; comportamento não observado na Figura 4.5-b).

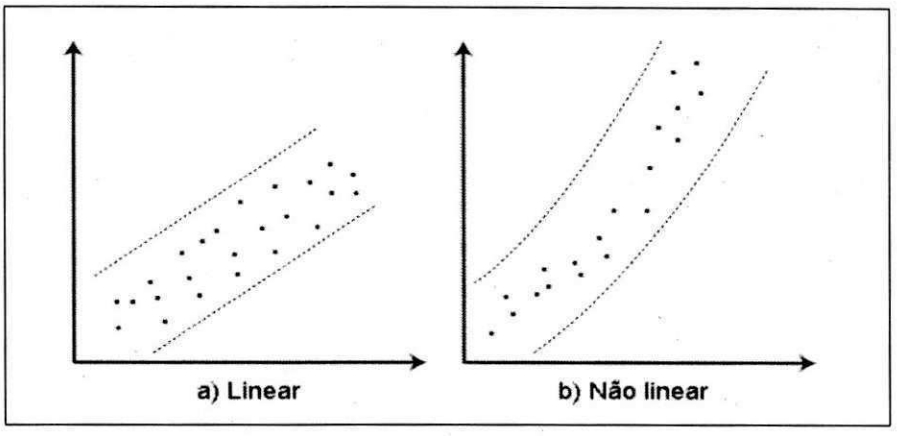

Figura 4.4: Exemplos de gráficos de dispersão para o requisito da independência dos erros

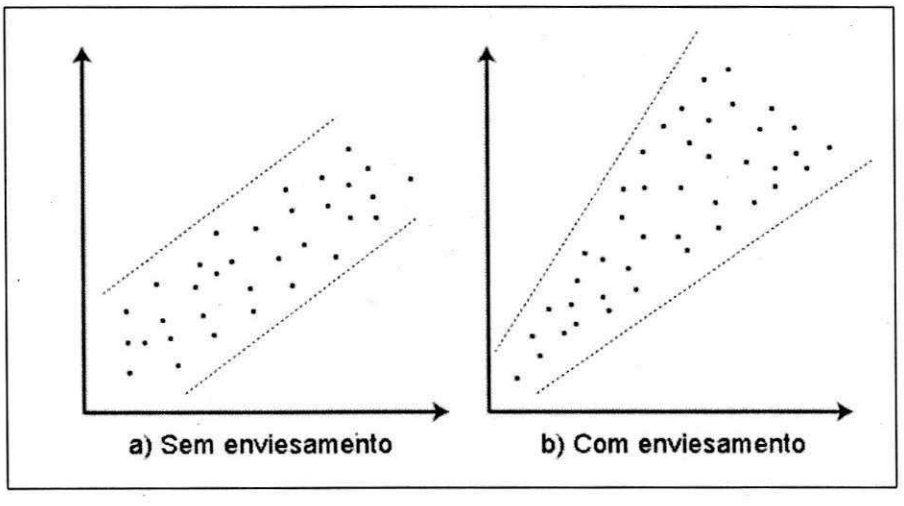

Figura 4.5: Exemplos de gráficos de dispersão para o requisito da homoscedasticidade dos erros

Vale ressaltar que a análise visual (através dos gráficos quantil-quantil e de dispersão) pode ser imprecisa e que qualquer tomada de decisão no que diz respeito aos requisitos de normalidade, independência e homoscedasticidade (dos erros experimentais) deve ser baseada nos testes específicos de cada requisito, resumidos na Tabela 4.2.

Tabela 4.2: Resumo dos testes de normalidade, independência e homoscedasticidade

Objetivo	**Testes**	**Hipótese nula (H_0)**
Normalidade	Shapiro-Wilk e Anderson-Darling	Os erros são normalmente distribuídos
Independência	Chi-Quadrado para independência	Os erros são independentes
Homoscedasticidade	Bartlett	Os erros têm variâncias constantes

Se os requisitos impostos aos erros experimentais não forem satisfeitos, a análise de relevância dos parâmetros pode ser feita com o auxílio do teste não-paramétrico de *Kruskal-Wallis* [58] que verifica se os grupos testados são iguais ou se há pelo menos um grupo diferente dos outros. No caso de haver pelo menos um grupo diferente, utiliza-se o teste pareado de *Mann-Whitney* [68] aplicando (posteriormente) o teste de correção de *Bonferroni* [19] para identificar cada grupo significativamente diferente. No contexto do processo, os grupos são formados pelos valores coletados (da métrica de interesse) em função dos níveis (-1 e 1) de cada parâmetro.

É importante ressaltar que a aplicação dos testes paramétricos (coeficiente de determinação, análise de regressão, teste-t e teste-F), na condição de que pelo menos um dos requisitos (normalidade, independência e homoscedasticidade dos erros experimentais) não seja satisfeito, pode conduzir a resultados não confiáveis.

Independente da técnica utilizada (testes paramétricos não-paramétrico), cada parâmetro significativamente relevante torna-se elemento de um novo conjunto definido como $M = \{p_1, p_2, ..., p_{k'}\}$, tal que $k' \leq k$ e $M \subseteq L$. Para a atividade seguinte, os parâmetros em $M - L$ (considerados irrelevantes nesta atividade) são fixos em seus valores *default*.

Uma vez que o conjunto M é devidamente preenchido, o usuário pode avançar para a Atividade 4 ou pode retornar à Atividade 2 (vide Figura 4.1). Se o conjunto M contém poucos elementos ou está vazio é aconselhável retornar à Atividade 2, pois avançar para a Atividade 4 e otimizar uma pequena quantidade de parâmetros pode gerar uma configuração com pouca melhoria na eficiência do Hadoop. No entanto, retornar à Atividade 2 incorre em um

aumento no tempo de término do processo, pois seria necessário realizar mais experimentos com os novos parâmetros adicionados.

Como é possível notar, a decisão em seguir para a Atividade 4 ou retroceder à Atividade 2 impõem um *trade-off* entre obter uma configuração mais eficiente e ter disponibilidade de tempo para esperar a execução de mais experimentos. Essa decisão deve ser feita pelo usuário que deve priorizar uma dessas duas opções.

Retornando-se à Atividade 2, considera-se M como um novo conjunto L e adiciona-se novos parâmetros associados à métrica de interesse; de preferência aqueles que ainda não foram selecionados em nenhuma passagem durante a seleção dos parâmetros.

Ao final desta atividade, tem-se a informação dos níveis dos que otimizam a métrica de interesse. No entanto, a otimização dos parâmetros em M precisa ser avaliada, uma vez que o experimento feito nesta atividade considerou apenas valores mínimos e máximos em uma faixa relativamente grande; cenário que torna tal otimização imprecisa. Desse modo, os elementos de M (parâmetros considerados relevantes nesta atividade) e seus respectivos níveis otimizados serão utilizados para compor um novo experimento visando refinar os níveis dos parâmetros otimizados nesta atividade de forma a realizar uma nova otimização, desta vez com uma granularidade mais fina.

4.4.1 Impacto dos valores de k e r

Uma questão que precisa ser discutida neste ponto da descrição do processo é o impacto que os valores de k e r têm sobre o tempo de execução do processo. Apesar de quanto maior o número de fatores selecionados em L, maior a probabilidade de que parâmetros relevantes sejam encontrados; o tempo necessário para a execução da atividade de análise de relevância pode ser muito elevado. Então torna-se necessário ter uma noção de como selecionar os valores de k e r.

Para tanto, considere um cenário composto por um *cluster* com 5 nodos (cada nodo com 20 GB de memória RAM e 2 processadores de 4 núcleos) e uma *workload* de 50 GB. Considere também que os *jobs* MapReduce executados nesse ambiente apresentaram tempo médio de resposta de 20 min. A Figura 4.6 exibe uma simulação do tempo necessário (em horas) para se concluir o experimento da Atividade 3 em função da tamanho do conjunto L (k) e do grau de replicação do experimento (r).

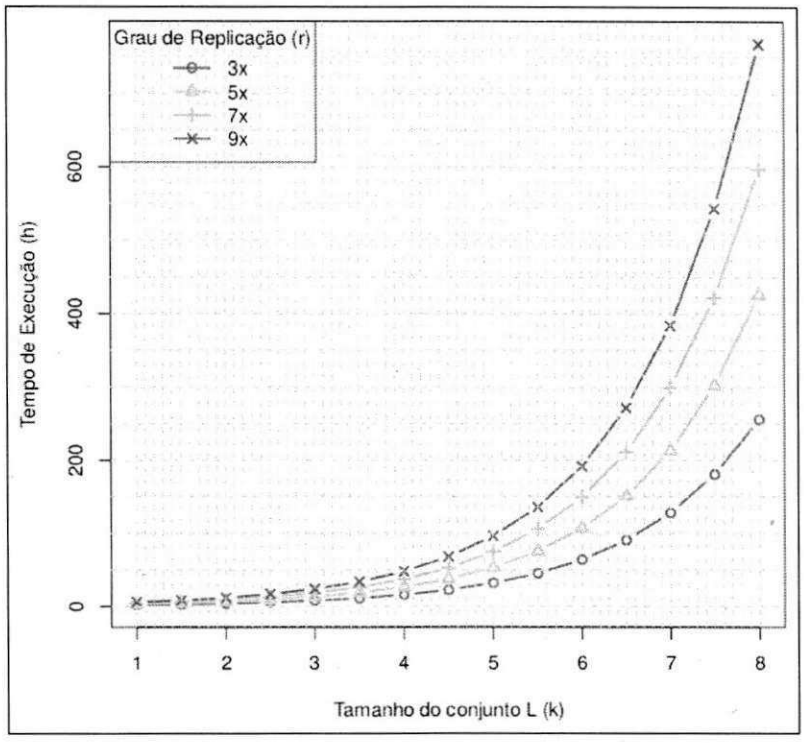

Figura 4.6: Simulação de um cenário sobre o tempo de execução da Atividade 3

Através dessa simulação é possível inferir que, apesar de ser possível incluir muitos parâmetros nos conjuntos L, o tempo de conclusão do experimento que envolve esses conjuntos pode ser muito alto. No cenário considerado na Figura 4.6, observe com o aumento linear dos valores de k e r incorre em um aumento exponencial do tempo de execução do experimento.

A análise do impacto de k e r não pode desprezar as informações do *hardware*. Se mais recursos de *hardware* estivessem disponíveis para a mesma *workload*, é provável que os tempos de execução obtidos na simulação seriam consideravelmente menores.

4.5 Atividade 4: Refinamento e Otimização

Nesta etapa os fatores que pertencem ao conjunto M devem compor um experimento guiado por *superfície de resposta* [48] (do inglês, *response surface*): conjunto de técnicas para analisar a variação da métrica de interesse, bem como encontrar um tratamento que otimize uma determinada métrica com uma granularidade mais fina do que usando um projeto fatorial.

Assim como no projeto fatorial, a superfície de resposta permite planejar um experimento

com um número reduzido de níveis por fator. Para tanto, é necessário se utilizar de projetos experimentais específicos, como por exemplo o conhecido *projeto composto central* (CCD, do inglês *Central Composite Design*) [8][9].

O projeto composto central contém uma série de pontos que podem ser agrupados (blocados) de acordo com seus níveis. O *bloco cubo* agrupa os pontos cujos níveis correspondem ao do projeto fatorial (-1 e 1), por esse motivo esses pontos são chamados aqui de *pontos fatoriais*. O *bloco estrela* agrupa os *pontos de eixo*, cujos níveis correspondem a $-\alpha$ e α. Há ainda os chamados *pontos centrais* cujos níveis são codificados por 0 e estão presentes em ambos os blocos.

A Figura 4.7 exibe um modelo de 2 fatores para o projeto composto central. Note como os blocos (cubo e estrela) são integrados para representar um único projeto experimental.

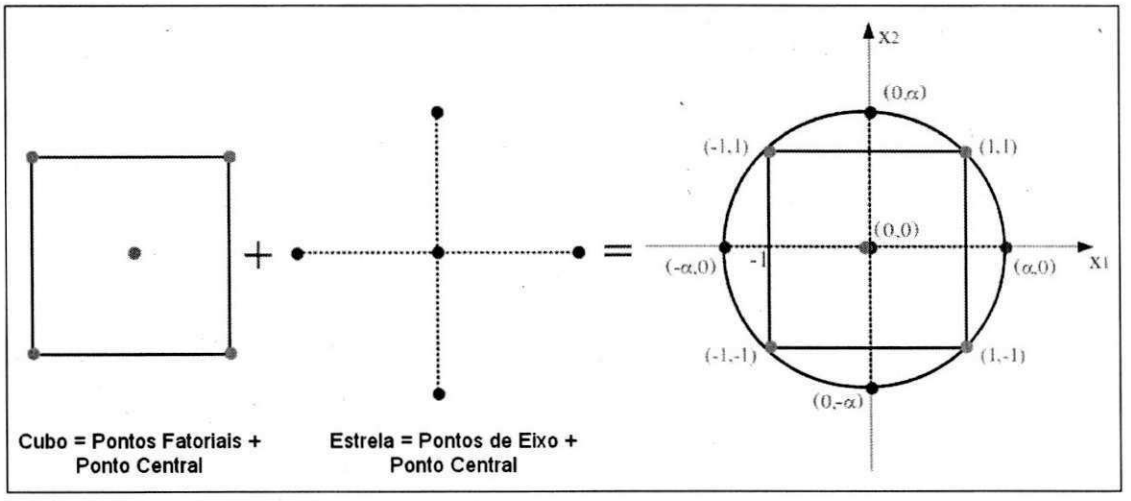

Figura 4.7: Modelo de fatores utilizado pelo CCD para 2 fatores (x_1 e x_2)

Fica visualmente mais intuitivo compreender o significado dos nomes dos blocos (cubo e estrela) quando se analisa um modelo de 3 fatores para o projeto composto central. A Figura 4.8 mostra o modelo de 3 fatores baseado no trabalho original de Box e Wilson [8].

Observe que a codificação dos pontos de estrela correspondem aos níveis $-\alpha$ e α. Esses níveis são definidos em função do número de fatores em M, a saber: $-[2^{k'}]^{\frac{1}{4}}$ e $[2^{k'}]^{\frac{1}{4}}$, respectivamente para $-\alpha$ e α. Todas essas características inerentes ao projeto composto central permitem atender a vários requisitos fundamentais para uma boa predição do modelo de superfície de resposta, com destaque para a *rotabilidade*[4] e a *ortogonalidade dos blocos*[5] [8].

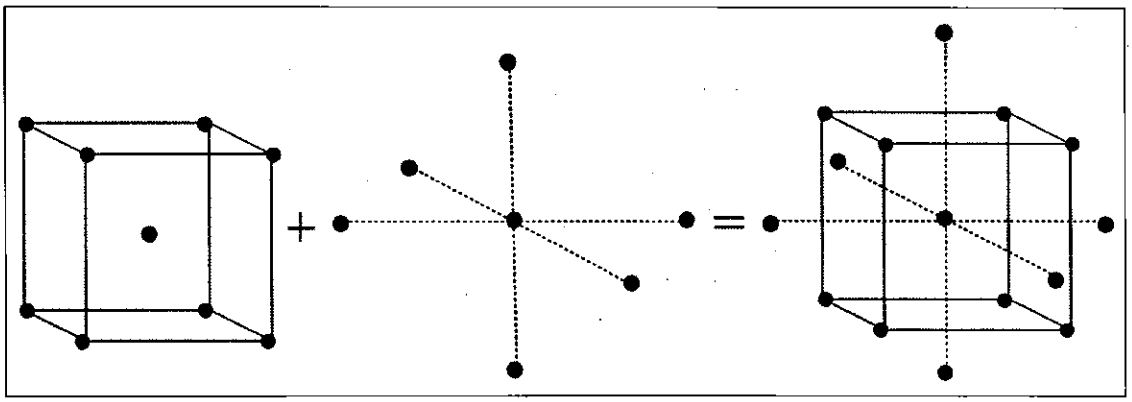

Figura 4.8: Modelo de fatores utilizado pelo CCD para 3 fatores

A Figura 4.9-a) mostra um possível comportamento da métrica de interesse quando se utiliza o projeto fatorial. Observe que, considerando a faixa entre -1 e 1, a métrica de interesse atinge seu valor máximo (ótimo) no nível 1. No entanto, esse modelo não é capaz de capturar a métrica de interesse para valores estritamente próximos a 1. Nesta figura, observe que para o valor hipotético $1 + \delta$ a métrica de interesse seria ainda maior.

As técnicas de superfície de resposta são utilizadas para contornar essa deficiência do projeto fatorial. Utilizando o projeto composto central, o nível anteriormente ótimo (1, no exemplo da Figura 4.9-a)) torna-se nível central do novo projeto (codificado por 0), permitindo a exploração de valores intermediários (e estritamente próximos) e estimando um modelo de curvatura para realizar a otimização dos parâmetros, conforme mostra a Figura 4.9-b). Somente após a definição do nível central é que são definidos os pontos fatoriais (-1 e 1) e os pontos de eixo ($-\alpha$ e α).

No contexto do processo proposto, cada parâmetro em M deve ter seus níveis redefinidos. Aqueles níveis que eram ótimos para o projeto fatorial, tornam-se os pontos centrais no projeto composto central. Como exemplo, considere os mesmos fatores do cenário demonstrado na atividade anterior A =*io.sort.mb*, B =*io.sort.record.percent* e C =*mapred.child.java.opts*, sendo C um fator secundário fixo em $-Xmx400m$. Suponha que os níveis ótimos encontrados com o projeto fatorial foram 200 (1 conforme a Tabela 4.1) para A e 0.1 (-1 conforme a Tabela 4.1) para B. Para o experimento a ser executado nesta

[4] A variação de uma predição depende apenas de sua distância até os pontos centrais.

[5] Os coeficientes da equação de superfície não podem estar correlacionados com os efeitos dos blocos.

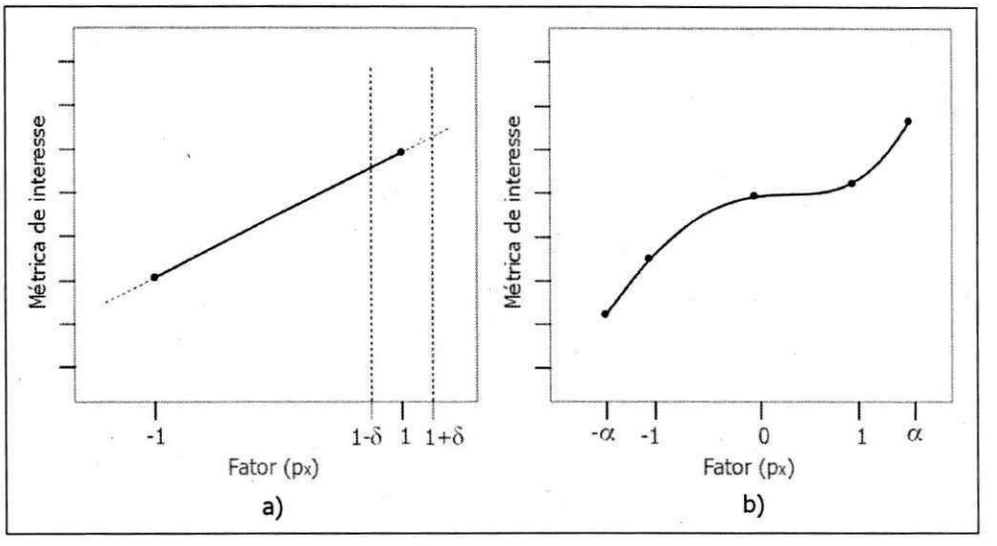

Figura 4.9: Fatores com a) projeto fatorial e b) projeto composto central

atividade (utilizando técnicas de superfície de resposta), esse níveis tornam-se pontos centrais do projeto fatorial, sendo ambos codificados por 0 agora.

A etapa seguinte é definir os novos níveis -1 e 1 (pontos fatoriais do projeto composto central). Como os pontos centrais já foram valores máximos ou mínimos na Atividade 3, a faixa entre -1 e 1 não pode ser relativamente grande, sobretudo porque esse valores podem exceder os limites permitidos para um determinado parâmetro. Uma forma de encontrar esses valores é fixar, para cada parâmetro, um valor δ que determina o incremento (e o decremento) sobre o nível central 0. O valor de δ pode ser calculado a partir da seguinte equação:

$$\delta = 0.2 \cdot (fd_{sup} - fd_{inf}) \tag{4.1}$$

onde fd_{sup} e fd_{inf} correspondem aos níveis superior e inferior utilizados no experimento guiado por projeto fatorial (Atividade 3). Optou-se por utilizar 20% (0.2) sobre a faixa $[fd_{inf}, fd_{sup}]$ para que os valores adjacentes ao nível central sejam explorados com uma granularidade mais fina (vide Figura 4.9).

Para exemplificar o cálculo de δ e, posteriormente, a definição dos novos níveis -1 e 1, considere novamente os fatores A e B supracitados. Dado que os níveis desses parâmetros para o experimento com projeto fatorial (vide Atividade 3) foram $A = \{100, 200\}$ e $B =$

$\{0.1, 0.3\}$, seus respectivos valores δ_A e δ_B são:

$$\delta_A = 0.2 \cdot (200 - 100) = 0.2 \cdot 100 = 20$$

$$\delta_B = 0.2 \cdot (0.3 - 0.1) = 0.2 \cdot 0.2 = 0.04$$

Munido dos valores de δ de cada fator, calcula-se os novos níveis de -1 e 1 em função do valor do nível central (V_{n_0}) aplicando-se: $V_{n_0} - \delta$ e $V_{n_0} + \delta$, para cada fator. A definição desse novos níveis é sumarizada na Tabela 4.3.

Tabela 4.3: Calculando os novos níveis -1 e 1

	Nível -1	**Nível** 0	**Nível** 1
Fator A	$V_{n_0} - \delta_A = 200 - 20 = 180$	200	$V_{n_0} + \delta_A = 200 + 20 = 220$
Fator B	$V_{n_0} - \delta_B = 0.1 - 0.04 = 0.06$	0.1	$V_{n_0} + \delta_B = 0.1 + 0.04 = 0.14$

Para encontrar os valores dos pontos de estrela deve-se inicialmente determinar os valores codificados $-\alpha$ e α. Como $k' = 2$, tem-se $\alpha = [2^2]^{\frac{1}{4}} = 1.41$, ou seja, $-\alpha = -1.41$ e $\alpha = 1.41$. A seguir, aplica-se esses valores à seguinte equação (baseada em [9]):

$$V_d = V_c \cdot \delta + V_{n_0} \tag{4.2}$$

onde V_d representa o valor real (decodificado), V_c representa o valor codificado e V_{n_0} representa o valor do ponto central. Para o exemplo dos parâmetros A e B, os pontos de estrela correspondem a $A = \{171.2, 228.8\}$ e $B = \{0.0424, 0.1576\}$. O cálculo desses valores é sumarizado na Tabela 4.4.

Tabela 4.4: Calculando os níveis $-\alpha$ e α

	Nível $-\alpha$	**Nível** 0	**Nível** α
Fator A	$(-1.41) \cdot 20 + 200 = 171.2$	200	$1.41 \cdot 20 + 200 = 228.8$
Fator B	$(-1.41) \cdot 0.04 + 0.1 = 0.0424$	0.1	$1.41 \cdot 0.04 + 0.1 = 0.1576$

A Tabela 4.5 mostra a tabela de sinais para um planejamento de experimentos envolvendo o projeto composto central. Os parâmetros A e B são os mesmos utilizados nos exemplos anteriores e seus níveis ($-\alpha$, -1 , 0 , 1 , α) correspondem a respectivamente $A = \{171.2, 180, 200, 220, 228.8\}$ e $B = \{0.0424, 0.06, 0.1, 0.14, 0.1576\}$.

Quando se usa a superfície de resposta com projeto composto central, o número de ensaios experimentais é calculado considerando cada bloco. Como o bloco cubo agrupa os

Tabela 4.5: Exemplo de uma tabela de sinais para o projeto composto central

Bloco	**Ponto**	**A (Cod.)**	**B (Cod.)**	**A**	**B**
Cubo	Fatorial	-1	-1	190	0.08
		1	-1	210	0.08
		-1	1	190	0.12
		1	1	210	0.12
	Central	0	0	200	0.1
Estrela	Eixo	$-\alpha$	0	185.6	0.1
		α	0	214.4	0.1
		0	$-\alpha$	200	0.0712
		0	α	200	0.1288
	Central	0	0	200	0.1

pontos do projeto fatorial, tem-se 2^k tratamentos para esse bloco. O bloco estrela, por outro lado, combina níveis dos pontos de eixo ($-\alpha$ e α) e níveis dos pontos centrais (0), gerando $2 \cdot k$ tratamentos. Sendo n_0 o número de replicações dos pontos centrais e r' o grau de replicação total para cada tratamento, define-se a quantidade de ensaios experimentais através da equação:

$$(2^{k'} + 2 \cdot k' + n_0) \cdot r' \tag{4.3}$$

Assim como no experimento com projeto fatorial, é possível ter um projeto experimental fracionado definindo um valor para p', tal que $p' < k'$. Nesse caso, a quantidade de ensaios experimentais é definida por:

$$(2^{k'-p'} + 2 \cdot (k' - p') + n_0) \cdot r' \tag{4.4}$$

Como se evita o uso de um projeto experimental fracionado, para que não se perca informações referentes às interações de alguns parâmetros; é comum definir $p' = 0$. Nesse caso, a Equação 4.4 torna-se a Equação 4.3.

Após o término do experimento é possível criar um modelo de regressão assim como o que foi discutido na Atividade 3 (para o projeto fatorial). Uma diferencial da análise de

variância da superfície de resposta é a aplicação de testes mais detalhados para avaliar a adequação do modelo, como por exemplo o teste *lack to fit* [9].

Para a análise do experimento com superfície de resposta utiliza-se as mesmas técnicas aplicadas ao projeto fatorial no que diz respeito aos requisitos de independência, normalidade e homoscedasticidade dos erros experimentais (vide Tabela 4.2) e os possíveis testes não-paramétricos a serem aplicados (vide Seção 4.4).

Na superfície de resposta, o valor ótimo da métrica de interesse é encontrado por meio da técnica da *busca da máxima inclinação*, que permite "caminhar" sequencialmente sobre a superfície em direção e sentido onde ocorre um aumento ou diminuição máxima da variável de resposta [48]; o que depende do objetivo estabelecido pelo usuário: maximizar ou minimizar uma determinada métrica.

Ao final do experimento guiado por superfície de resposta, obtêm-se os valores otimizados para os parâmetros em M. Integrando esses parâmetros àqueles definidos como fatores secundários ajustados com o auxílio de *rules of thumbs* (conjunto T), forma-se o conjunto P de todos parâmetros ajustados que otimizam a métrica de interesse, considerando a instanciação corrente. Formalmente esse conjunto pode ser definido da seguinte forma: $P = M \cup T$.

Se o usuário optar por incluir mais parâmetros aos conjunto P, ele deve retroceder à atividade de *seleção dos parâmetros candidatos*. No entanto, refazer esses passos (atividades) demanda executar novos experimentos, o que incorre em uma espera maior por resultados.

4.5.1 Impacto dos valores de k' e r'

A Atividade 4 permite um controle muito pequeno sobre o valor de k', pois o mesmo está condicionado à análise de relevância do conjunto L feita na Atividade 3. No entanto é importante que seja levantada a mesma discussão feita na Seção 4.4.1; sendo que, para esta atividade, a discussão envolve os valores de k' e r'.

Levando em consideração que o número de ensaios experimentais realizados com o projeto composto central é maior que o número de ensaios experimentais realizados com o projeto fatorial, os valores de k' e r' apresentam um impacto ainda maior sobre o tempo de execução do processo, quando comparados a k e r.

Para se ter uma maior noção desse impacto, considere o mesmo ambiente utilizado na Seção 4.4.1. A Figura 4.10 exibe uma simulação do tempo necessário (em horas) para se

concluir o experimento da Atividade 4 em função da tamanho do conjunto M (k') e do grau de replicação do experimento (r').

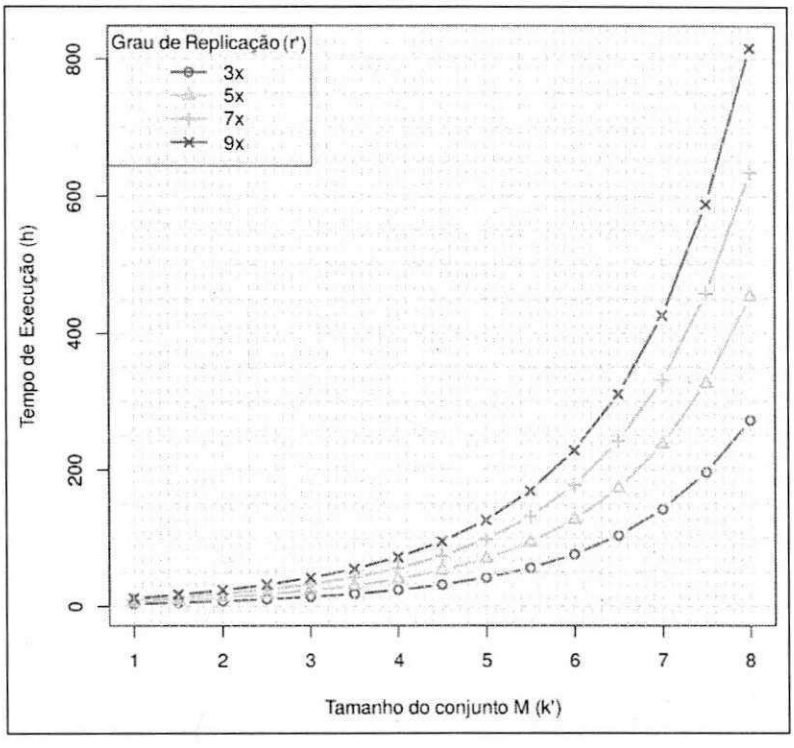

Figura 4.10: Simulação de um cenário sobre o tempo de execução da Atividade 4

Observe que elevando-se linearmente os valores de k' e r' o tempo de execução do processo cresce exponencialmente. É importante repetir que os recursos de *hardware* utilizados e o tamanho da *workload* exercem forte influência sobre esses resultados. Se mais recursos de *hardware* estivessem disponíveis para a mesma *workload*, é provável que os tempos de execução obtidos seriam consideravelmente menores.

4.6 Considerações Finais do Capítulo

Este capítulo descreveu com detalhes o processo proposto para configuração eficiente de aplicações que executam sobre o Hadoop. Ele é baseado em técnicas empíricas e estatísticas bem conhecidas da literatura e explora as lacunas que outros métodos e ferramentas não preenchem (vide Capítulo 3).

É possível notar que as Atividade 3 e 4 têm algumas semelhanças, uma vez que ambas en-

volvem a execução de um experimento e a análise de seus resultados. A Tabela 4.6 sumariza algumas particularidades entre essas duas atividades, discutidas ao longo deste capítulo.

No capítulo a seguir será apresentado um estudo de caso realizado através da instanciação do processo utilizando uma aplicação conhecida na literatura especializada.

Tabela 4.6: Particularidades provenientes das Atividades 3 e 4

	Projeto Experimental	**Número de Ensaios**
Atividade 3	Fatorial com Replicação	$2^{k-p} \cdot r$
Atividade 4	Superfície de Resposta com CCD	$(2^{k'-p'} + 2 \cdot (k' - p') + n_0) \cdot r'$

Capítulo 5

Instanciação do Processo

Este capítulo descreve um estudo de caso realizado consistindo na instanciação do processo empírico proposto. Neste estudo de caso foi considerado um cenário experimental para o qual são detalhadamente descritos todos os passos realizados em cada atividade do processo. Logo após a descrição do estudo de caso é feita a validação dos seus resultados por meio de uma comparação com outras soluções relacionadas, a saber: Starfish, *rules of thumbs* e uma solução híbrida envolvendo as duas primeiras. A última seção deste capítulo faz uma avaliação sobre o impacto do tempo de conclusão do processo.

Na seção a seguir descreve-se as características do ambiente usado para os experimentos. Isso envolve tanto a especificação do *cluster* onde os experimentos foram executados quanto a descrição de todo o ferramental de *software* utilizado.

5.1 Ambiente de Execução

A instanciação do processo envolveu a execução de experimentos realizados em um *cluster* homogêneo com 5 nodos, dos quais 1 foi designado como mestre (executando o *jobtracker* e o *namenode*) e os outros 4 como escravos (executando os *tasktrackers* e os *datanodes*), de acordo com a arquitetura do Hadoop.

Cada nodo do *cluster* é composto por 2 processadores Intel Xeon X5550 2.67GHz com 4 núcleos e suporte a Hyper-Threading (totalizando 16 núcleos virtuais[1] por nodo), 1 disco

[1]Uma distinção feita em relação aos processadores com Hyper-Threading, nos quais cada núcleo físico é reconhecido pelo sistema operacional como dois núcleos lógicos.

rígido de 500 GB SATA 3 GB/s 7.200 RPM e memória RAM de 20 GB. Todos os nodos são conectados através de rede Gigabit Ethernet e dispõem do sistema operacional Debian GNU/Linux com o *kernel* versão 2.6.32-5.

A aplicação usada no estudo de caso foi o TeraSort: um algoritmo adequado para o modelo MapReduce e capaz de ordenar grandes bases de dados [65]. Essa aplicação é representativa para a computação intensiva de dados e sua utilização pode ser observada em trabalhos semelhantes [4][35]. A *workload* usada neste estudo de caso consistiu de uma base de 50 GB gerado por uma ferramenta auxiliar do TeraSort chamada: TeraGen [65].

Para análise dos dados resultantes dos experimentos utilizou-se a ferramenta estatística R [49], especificamente nas fases de *investigação de relevância* e *refinamento e otimização*. A versão utilizada do Hadoop foi a 0.20.203.0. Uma das soluções utilizadas para a validação do processo foi o Starfish [63], disponibilizado através de uma distribuição de *software*. A versão do Starfish implantada foi a 0.3.0.

Os *scripts* utilizados na instanciação do processo foram desenvolvidos em Shell Script, Python e R. Esses *scripts* fornecem o suporte para a execução de cada atividade do processo empírico proposto, cabendo ao usuário configurar algumas variáveis e invocar os arquivos executáveis relacionados. Há também um diretório dedicado para o armazenamento dos resultados dos experimentos.

Os *scripts* estão disponíveis na página Web reservada para esta pesquisa[2]. Neste local é possível encontrar outras informações, inclusive uma lista de parâmetros similar à Tabela 2.1. Essa tabela pode ser utilizada para auxiliar a construção dos conjuntos de parâmetros (L e M) necessários para a execução do processo.

5.2 Estudo de Caso

Esta seção descreve a instanciação do processo proposto utilizando o ambiente especificado na Seção 5.1 e seguindo todas as atividades que compõem o processo, assim como descrito no Capítulo 4.

[2]URL: *http://redmine.lsd.ufcg.edu.br/projects/configHadoop/wiki*.

5.2.1 Atividade 1: Definição da Métrica de Interesse

A métrica selecionada para este cenário foi o *tempo de resposta*. Tal escolha proporcionou validar o processo proposto através da comparação com a ferramenta Starfish [36][35], discutida no Capítulo 3. Além disso, essa métrica é muito utilizada nas pesquisas sobre avaliação de desempenho de plataformas de computação intensiva [20][4][12].

Para medir o tempo de resposta dos experimentos, vários *jobs* foram executados sequencialmente. Cada um deles processou a *workload* de 50 GB gerado pelo TeraGen. Esse tamanho de *workload* é representativo para essa aplicação [4].

5.2.2 Atividade 2: Seleção dos Parâmetros Candidatos

O conjunto L definido para este estudo de caso é exibido na Tabela 5.1. A seleção desses parâmetros foi feita com base na na Tabela 2.1 levando em consideração o tempo de resposta, métrica estabelecida na atividade anterior.

Tabela 5.1: Conjunto L para o estudo de caso que mede o *tempo de resposta*

Nome do Parâmetro
io.file.buffer.size
io.sort.mb
io.sort.record.percent
mapred.reduce.parallel.copies
mapred.reduce.slowstart.completed.maps
mapred.reduce.tasks

5.2.3 Atividade 3: Análise de Relevância

Uma vez que o conjunto L foi definido, é necessário realizar o planejamento de experimentos definindo os fatores secundários e os níveis dos fatores primários (parâmetros em L).

A Tabela 5.2 lista os fatores secundários ajustados com o auxílio de *rule of thumbs* (conjunto T) e seus respectivos valores de ajuste. Lembrando que esses valores permaneceram fixos durante o experimento.

Tabela 5.2: Conjunto T: parâmetros com valores constantes ajustados através de *rule of thumbs*

Parâmetro	Valor
dfs.block.size	134217728
mapred.child.java.opts	$-Xmx515m$
mapred.inmem.merge.threshold	0
mapred.tasktracker.map.tasks.maximum	24
mapred.tasktracker.reduce.tasks.maximum	8

O parâmetro *dfs.block.size* define o tamanho dos blocos do sistema de arquivos distribuído. Por padrão esse parâmetro é ajustado em 67108864 (64 MB). No entanto, devido ao volume de dados optou-se por utilizar um tamanho de bloco maior: 134217728 (128 MB). O tamanho do bloco do sistema de arquivos distribuído está intimamente relacionado com a quantidade de tarefas *maps* gerados para um *job* [65]. Para uma base de 50 GB (51200 MB) e blocos de 128 MB são gerados $51200/128 = 400$ tarefas *maps*. Por outro lado, se fosse usado o valor padrão, o número de tarefas *maps* seria o dobro: $51200/64 = 800$. Em *clusters* de pequeno porte (com aproximadamente 5 a 10 nodos), um número elevado de tarefas *map* pode causar uma considerável sobrecarga de processamento [62]. Essa *rule of thumb* foi formalizada na Equação 2.8.

Conforme discutido na Seção 2.2.4, atribuiu-se o valor 0 (zero) a *mapred.inmem.merge.threshold* para desabilitar o controle, por parte desse parâmetro, sobre o tamanho do *buffer* utilizado no lado *reduce* durante a cópia dos dados recebidos dos *maps*, mesma função exercida pelo parâmetro *mapred.job.shuffle.merge.percent*.

Conforme a Tabela 2.2, os valores de *mapred.tasktracker.map.tasks.maximum* e *mapred.tasktracker.reduce.tasks.maximum* foram ajustados para que houvesse no máximo 2 tarefas executando simultaneamente em cada núcleo virtual, melhorando a eficiência no uso da CPU (vide Capítulo 2). Como havia, para cada nodo, 16 núcleos virtuais disponíveis, totalizou-se 32 tarefas executadas simultaneamente em cada *tasktracker*. Como é aconselhável executar simultaneamente mais tarefas *map* que *reduce* (vide Seção 2.2.4), foi definido $max_{maps} = 24$ e $max_{reduces} = 8$. Inclusive, pode-se usar a Equação 2.2 para confirmar o número total de tarefas executando simultaneamente em cada *tasktracker*:

$max_{tasks} = 24 + 8 = 32$.

Para o parâmetro *mapred.child.java.opts* foi utilizada a Equação 2.3. Como a memória total de cada nodo (mem_{total}) corresponde a 20480 MB (20 GB) e como foi constatado que o consumo médio de memória RAM pelo sistema operacional e outros processos[3] do nodo (mem_{proc}) correspondia a aproximadamente 4000 MB, tem-se:

$$mem_{task} = \left\lfloor \frac{20480 - 4000}{32} \right\rfloor = \lfloor 515 \rfloor = 515 \text{ MB}$$

onde mem_{task} representa o parâmetro *mapred.child.java.opts*.

Apos definir os fatores secundários, definiu-se os níveis dos fatores primários. Tais definições foram feitas nesta ordem porque alguns fatores secundários são usados para se determinar os níveis de alguns fatores primários. A Tabela 5.3 exibe os fatores primários e seus respectivos níveis [4].

Tabela 5.3: Conjunto L: fatores primários e seus respectivos níveis

Identificação	Parâmetro	Níveis
A1	io.file.buffer.size	{4096, 32768}
B1	io.sort.mb	{100, 250}
C1	io.sort.record.percent	{0.1, 0.3}
D1	mapred.reduce.parallel.copies	{5, 20}
E1	mapred.reduce.slowstart.completed.maps	{0.1, 0.6}
F1	mapred.reduce.tasks	{5, 300}

Para os parâmetros que não fazem parte nem do conjunto L nem do conjunto T, foram utilizados seus respectivos ajustes *default*. Por esse motivo, eles não foram listados, uma vez que o Hadoop ajusta como *default* todos os parâmetros cuja configuração é omitida.

Nesta atividade tem-se um experimento com $k = 6$ fatores (conforme a Tabela 5.3). Mesmo utilizando o projeto fatorial (vide Capítulo 4) a quantidade de tratamentos, ou seja, o conjunto dos valores de cada fator em um ensaio experimental, ainda é elevada: $2^k = 64$ tratamentos. Por esse motivo optou-se por utilizar uma taxa de replicação baixa $r = 3$,

[3] Valores coletados através do comando *free -mlot* do GNU/Linux.

[4] Devido às definições do Hadoop, para os níveis dos parâmetros utiliza-se a notação inglesa de ponto decimal em vez de vírgula.

totalizando $2^k \cdot r = 192$ ensaios experimentais. Apesar de ser um valor baixo, essa taxa de replicação permite estimar os erros experimentais. O impacto do valor de r foi discutido na Seção 4.4.1

Após o final desse experimento foi necessário verificar os requisitos de normalidade, independência e homoscedasticidade dos erros experimentais, conforme discutido no Capítulo 4. Os resultados desses testes podem ser observados na na Tabela 5.4. Para todos esses testes, a hipótese nula (H_0) não pôde ser rejeitada devido aos *p-valores* serem maiores que os níveis de significância estatisticamente aceitáveis (entre 0.001 e 0.1).

Tabela 5.4: Resultados dos testes de normalidade, independência e homoscedasticidade

Objetivo	**Testes**	***p-valor***
Normalidade	Shapiro-Wilk	0.3633
	Anderson-Darling	0.3357
Independência	Chi-Quadrado para independência	0.2394
Homoscedasticidade	Bartlett	0.6075

Para se ter uma noção visual da normalidade dos erros, observe o gráfico quantil-quantil exibido na Figura 5.1. Note que a maioria dos pontos está sobre a reta que cruza o gráfico obliquamente.

Além de verificar a normalidade dos dados, foi necessário verificar ainda os requisitos de homoscedasticidade e independência dos erros experimentais. Para isso foi gerado o gráfico de dispersão exibido na Figura 5.2. Note que não há enviesamento nesse gráfico, comprovando que os erros experimentais são independentes e com desvio padrão constante [39].

Como os requisitos de normalidade, independência e homoscedasticidade foram satisfeitos, foi possível calcular um modelo de regressão e aplicar os testes paramétricos para identificar os possíveis parâmetros relevantes, assim como descrito no Capítulo 4. A Figura 5.3 exibe um sumário dos resultados da análise feita a partir do modelo de regressão com o auxílio da ferramenta R. Note que essa figura exibe muitas informações, dentre as quais destaca-se a coluna que mostra os *p-valores* do *teste-t* pareado ($Pr(> |t|)$) e as linhas $Multiple\ R\ squared$ e $F\text{-}statistic$ que exibem o valor do coeficiente de determinação R^2 e do *teste-F* para a análise de regressão (ANOVA), respectivamente.

Observe que o valor de R^2 foi alto $(0,973)$ fornecendo indícios de que o modelo de

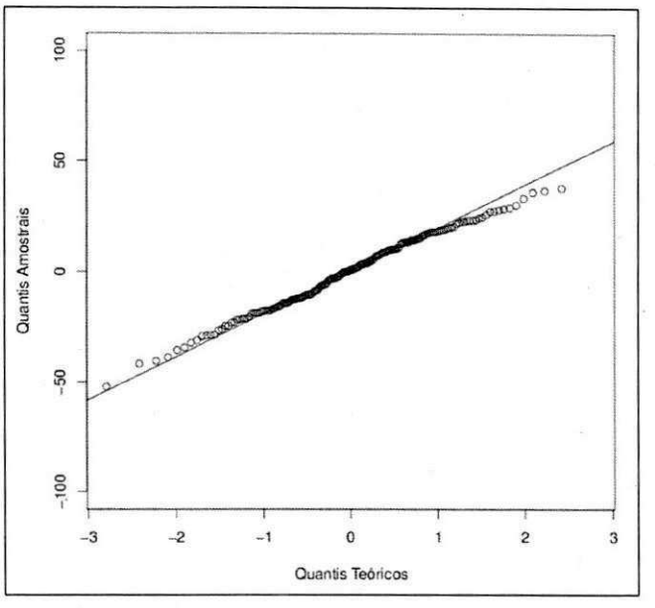

Figura 5.1: Gráfico quantil-quantil dos resultados da Atividade 3

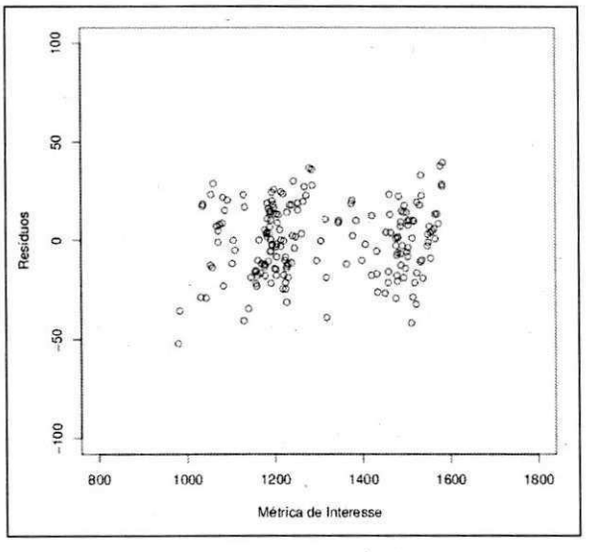

Figura 5.2: Gráfico de dispersão dos resultados da Atividade 3

regressão gerado foi adequado. Para confirmar essa informação é só observar que o *p-valor* do *teste-F* em *F-statistic* é muito menor que $0,01$ (ou 99% de confiança), reforçando o resultado de R^2.

Para definir quais parâmetros são relevantes pode-se consultar a coluna $Pr(>|t|)$ e checar quais deles, sem considerar as interações, têm o *p-valor* inferior a $0,01$. A partir da Figura 5.3 é possível notar que os parâmetros $A1$, $B1$, $C1$ e $F1$ (definidos na Tabela 5.3) são significativamente relevantes. Por esse motivo, eles foram utilizados para construir o conjunto $M = \{A1, B1, C1, F1\}$, que é utilizado na atividade seguinte.

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1309.0269     2.0815  628.88  < 2e-16 ***
A1           -21.7236     2.0815  -10.44  < 2e-16 ***
B1           -41.1107     2.0815  -19.75  < 2e-16 ***
C1           -50.6502     2.0815  -24.33  < 2e-16 ***
D1             3.4006     2.0815    1.63   0.1042
E1             0.5992     2.0815    0.29   0.7738
F1          -142.1335     2.0815  -68.28  < 2e-16 ***
A1:B1         -8.4377     2.0815   -4.05  7.7e-05 ***
A1:C1         -3.7586     2.0815   -1.81   0.0727 .
A1:D1          0.0403     2.0815    0.02   0.9846
A1:E1         -1.5903     2.0815   -0.76   0.4459
A1:F1          6.7022     2.0815    3.22   0.0015 **
B1:C1        -37.0142     2.0815  -17.78  < 2e-16 ***
B1:D1          0.1198     2.0815    0.06   0.9542
B1:E1          0.2872     2.0815    0.14   0.8904
B1:F1         13.0152     2.0815    6.25  3.2e-09 ***
C1:D1          0.1289     2.0815    0.06   0.9507
C1:E1          0.9426     2.0815    0.45   0.6513
C1:F1         12.4830     2.0815    6.00  1.2e-08 ***
D1:E1         -5.1228     2.0815   -2.46   0.0149 *
D1:F1         -5.1990     2.0815   -2.50   0.0135 *
E1:F1         12.7330     2.0815    6.12  6.4e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1

Residual standard error: 28.8 on 170 degrees of freedom
Multiple R-squared: 0.973,      Adjusted R-squared: 0.97
F-statistic:  297 on 21 and 170 DF,  p-value: <2e-16
```

Figura 5.3: Sumário dos resultados da análise de regressão da Atividade 3

Ao final desta atividade o processo informa os níveis que maximizam e minimizam a métrica de interesse. A Tabela 5.5 mostra uma compilação dos resultados desta atividade para o estudo de caso realizado. Essas informações são úteis para a definição dos novos níveis na Atividade 4.

Tabela 5.5: Níveis obtidos na Atividade 3 que minimizam o *tempo de resposta*

Identificação	Parâmetro	Nível codificado	Nível real
A1	io.file.buffer.size	1	32768
B1	io.sort.mb	1	250
C1	io.sort.record.percent	1	0.3
F1	mapred.reduce.tasks	1	300

5.2.4 Atividade 4: Refinamento e Otimização

Os elementos do conjunto M foram utilizados em um experimento guiado com técnicas de *superfície de resposta*. Para tanto, foi necessário definir os novos níveis desses parâmetros. Aplicando-se os passos descritos na Seção 4.4, obteve-se os níveis exibidos na Tabela 5.6. Os níveis dos parâmetros no projeto fatorial (vide Tabela 5.5) tornaram-se os níveis centrais do projeto composto central. A partir de então encontrou-se os níveis fatoriais (-1 e 1) e de eixo ($-\alpha$ e α) em conformidade com os recursos de *hardware* disponíveis e com a quantidade de fatores (k') em M.

Tabela 5.6: Conjunto M para o cenário que envolve o *tempo de resposta*

Identificação	Parâmetro	Níveis
A1	io.file.buffer.size	$\{24576, 28672, 32768, 36864, 40960\}$
B1	io.sort.mb	$\{206.8, 220, 250, 280, 293.2\}$
C1	io.sort.record.percent	$\{0.2424, 0.26, 0.3, 0.34, 0.3576\}$
D1	mapred.reduce.tasks	$\{215, 241, 300, 359, 385\}$

Há uma observação que deve ser feita sobre *io.file.buffer.size*. Esse parâmetro deve ser um múltiplo do tamanho da página de memória para o *hardware* utilizado no experimento[5]. Para a arquitetura Intel®, que é utilizada no ambiente descrito na Seção 5.1, esse tamanho equivale a $4K$. Por esse motivo, os valores encontrados a partir das equações discutidas na Seção 4.5 não puderam ser utilizados no ajuste desse parâmetro. Para este estudo de caso foram utilizados os múltiplos de $4K$ mais próximos do valor central: 32768 $(32K)$.

[5]Orientações da documentação dos parâmetros: *http://hadoop.apache.org/common/docs/current/core-default.html*.

Após o experimento, foram conduzidos os mesmos passos descritos na Atividade 3 para testar os requisitos de normalidade, independência e homoscedasticidade dos erros. Os resultados foram os similares aos encontrados na Seção 5.2.3.

A partir da saída gerada pelo processo foram obtidos valores otimizados para os parâmetros em M. Tomando esses parâmetros e integrando-os aos fatores secundários ajustados com o auxílio de *rules of thumbs* (conjunto T), formou-se o conjunto P de todos parâmetros ajustados que otimizam a métrica de interesse. Esses parâmetros e seus respectivos valores podem ser visualizados na Tabela 5.7.

Tabela 5.7: Conjunto P: parâmetros e valores otimizados sugeridos

Parâmetro	**Valor Otimizado**
dfs.block.size	134217728
io.file.buffer.size	32768
io.sort.mb	235
io.sort.record.percent	0.266
mapred.child.java.opts	$-Xmx515m$
mapred.inmem.merge.threshold	0
mapred.reduce.tasks	294
mapred.tasktracker.map.tasks.maximum	24
mapred.tasktracker.reduce.tasks.maximum	8

Uma das formas de se visualizar os efeitos dos parâmetros relevantes nesta atividade é através de um gráfico de superfície como o que é exibido na Figura 5.4. Observe o efeito acentuado que o número de *reduces* (parameter *mapred.reduce.tasks*) tem sobre o tempo de resposta do *job*. Além disso, note como os parâmetros *io.sort.mb* e *io.sort.record.percent* também se mostraram relevantes. Nesse tipo de gráfico, parâmetros menos relevantes apresentam curvas menos acentuadas.

5.3 Validação dos Resultados

Os parâmetros do conjunto P e seus respectivos valores constituem o produto final do processo e podem ser utilizados diretamente na configuração do Hadoop. Tal configuração pro-

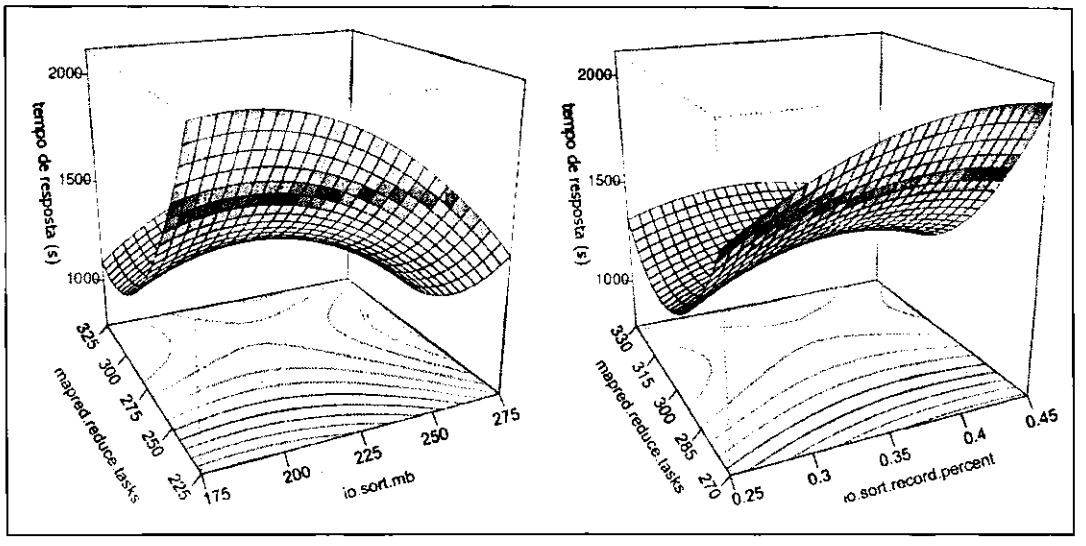

Figura 5.4: Gráfico de superfície para o estudo de caso realizado

verá uma execução eficiente dessa plataforma, considerando a métrica definida e o subespaço de parâmetros selecionados durante a aplicação do processo.

Para a validação do processo proposto foi executado um experimento comparativo entre a saída obtida através do estudo de caso descrito nas seções anteriores (conjunto P) e a saída de outras soluções relacionadas, a saber: *rule of thumbs* [65] e Starfish [36].

Considerando essas soluções, foram utilizados os valores gerados a partir de cinco diferentes cenários: o processo proposto, a configuração *default* do Hadoop, o Starfish, as *rules of thumbs* (RT) e uma configuração híbrida envolvendo o uso de *rules of thumbs* e Starfish, visando ajustar alguns parâmetros não cobertos pelo Starfish. Este último cenário será chamado de Starfish+RT.

Os parâmetros ajustados para os cenários Starfish e RT são exibidos respectivamente nas Tabelas 5.8 e 5.9. Os parâmetros em negrito na Tabela 5.9 representam os parâmetros não cobertos pelo Starfish e utilizados para completar a configuração do mesmo no cenário Starfish+RT.

Os valores gerados para as *rules of thumbs* foram baseados nas equações exibidas na Seção 2.2.4 (vide Tabela 2.3) e em outras fontes [11][62]. Para o Starfish, a geração dos valores exigiu a invocação do seu módulo *profiler* (vide Seção 3.4).

Uma vez que os cenários e seus respectivos valores foram encontrados, executou-se *jobs* Hadoop com a mesma *workload* e ambiente descritos na Seção 5.1 visando aplicar a va-

Tabela 5.8: Parâmetros ajustados no cenário *Starfish*

Parâmetro	**Valor**
io.sort.factor	89
io.sort.mb	114
io.sort.record.percent	0.2037
io.sort.spill.percent	0.8674
mapred.compress.map.output	$false$
mapred.inmem.merge.threshold	682
mapred.job.reduce.input.buffer.percent	0.1104
mapred.job.shuffle.input.buffer.percent	0.7753
mapred.job.shuffle.merge.percent	0.43614
mapred.reduce.tasks	120

lidação propriamente dita. Inicialmente executou-se cada cenário $n_i = 10$ vezes, número arbitrário necessário para se obter uma amostra inicial. A seguir, foi calculado o tamanho da amostra adequada para cada cenário (com 95% de confiança). Para isso, utilizou-se a equação a seguir (baseada em Jain [39]):

$$n = \left(\frac{100zs}{r\bar{x}}\right)^2 \tag{5.1}$$

onde n corresponde ao número de amostras, z corresponde ao valor da distribuição normal para o nível de confiança desejado, s corresponde ao desvio padrão amostral, r corresponde à acurácia da medição e $\bar{x}$ corresponde à media amostral. Por meio da Equação 5.1 foram encontrados os tamanhos das amostras que não superaram $n = 10$ execuções (para todos os cenários). Portanto não foi necessário obter nenhuma amostra a mais, uma vez que $n \leq n_i$.

Após a execução de todos os cenários e a medição de seus respectivos tempos de resposta, calculou-se um valor médio para cada cenário considerando um intervalo de confiança de 95%, cujos valores são exibidos na Figura 5.5. Os valores em percentuais representam o aumento do tempo médio das outros cenários em relação ao tempo médio do processo proposto.

Observe que há uma diferença considerável entre os tempos médios de resposta dos cenários quem envolvem o Starfish, apesar de estarem relacionados à mesma ferramenta.

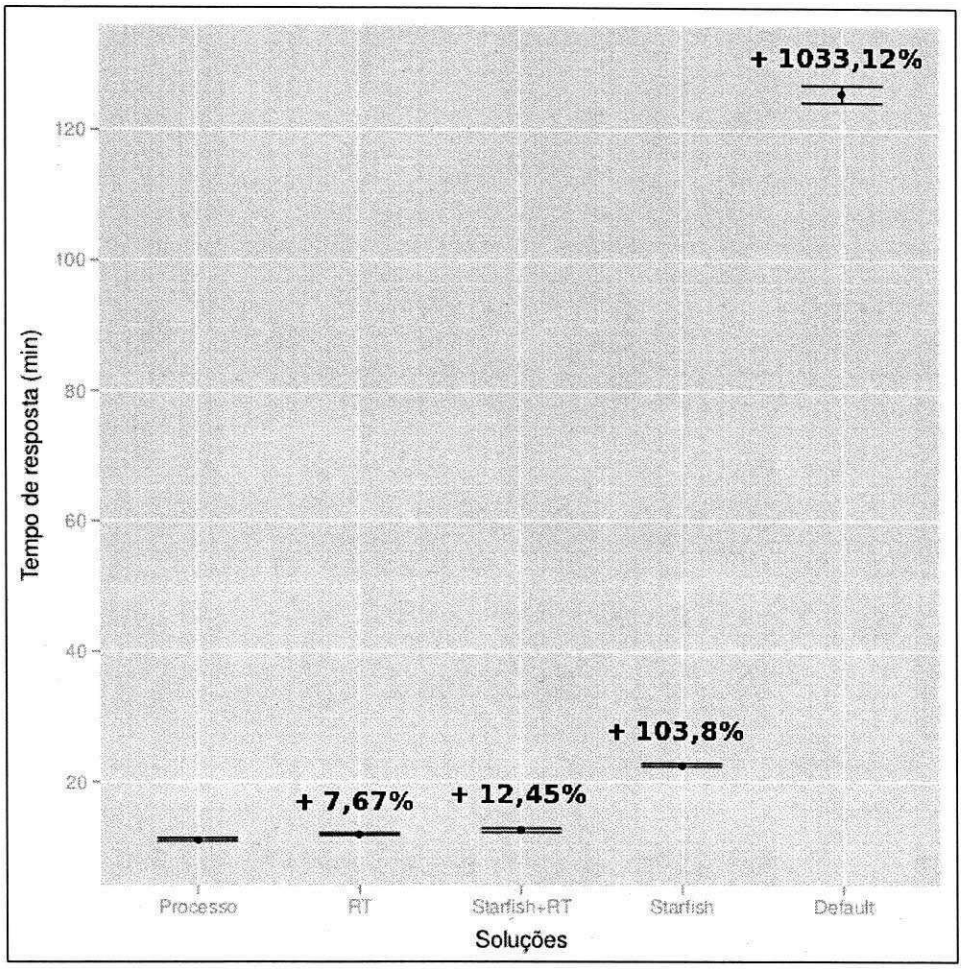

Figura 5.5: Tempos de resposta médios com intervalos de confiança de 95% para os cenários considerados na validação

Tabela 5.9: Parâmetros ajustados no cenário RT

Parâmetro	Valor
dfs.block.size	134217728
mapred.child.java.opts	$-Xmx515m$
mapred.compress.map.output	*true*
mapred.job.reuse.jvm.num.tasks	-1
mapred.reduce.tasks	112
mapred.tasktracker.map.tasks.maximum	24
mapred.tasktracker.reduce.tasks.maximum	8

Essa discrepância é explicada principalmente pelo fato do Starfish não recomendar valores para os parâmetros que ajustam a alocação da CPU e da memória RAM das tarefas, a saber: *mapred.tasktracker.map.tasks.maximum*, *mapred.tasktracker.reduce.tasks.maximum* e *mapred.child.java.opts* (vide Seção 2.2.4). Desse modo, este cenário utiliza valores *default* para o ajuste desses parâmetros, o que deixou a utilização daqueles recursos abaixo da real capacidade do *cluster*.

5.4 Impacto do Tempo de Conclusão

O processo proposto baseia-se no planejamento e execução de experimentos. Apesar de ser efetiva, a estratégia empírica requer a execução de vários ensaios, sendo necessário despender normalmente muito tempo para a conclusão de todos eles. Desse modo, é necessário avaliar como o tempo de conclusão dos experimentos realizados pode ter impacto sobre a execução dos *jobs* do usuário.

A Tabela 5.10 exibe o tempo (em minutos) de conclusão dos experimentos para todas as soluções utilizadas na validação discutida na seção anterior. Apesar dos bons resultados alcançados pelo processo, observe que ele apresentou um tempo de conclusão muito alto em relação às outras soluções.

Dados os resultados exibidos na Tabela 5.10, pode-se supor que o processo não é apropriado, pois o tempo necessário para se concluir os experimentos é muito maior do que a vantagem que ele apresentou sobre as outras soluções, considerando a métrica tempo

Tabela 5.10: Tempo necessário para executar as soluções consideradas na validação

Solução	Tempo de execução (em min)
Processo	5400
Rule of Thumbs (RT)	0
Starfish + RT	45
Starfish	45

de resposta. No entanto, vários *jobs* MapReduce são executados em ambientes que têm pouca alteração com o tempo, como por exemplo em cenários envolvendo Microsoft® [38] e Facebook® [32]. Nesses casos, é provável que haja a necessidade de se executar o processo apenas uma vez, já que a configuração do Hadoop também não se altera.

Em cenários como esses, o tempo necessário para executar todo o processo pode ser desprezado, uma vez que existe um certo tempo t_n após n sucessivas execuções de *jobs* em que a melhora de desempenho alcançada sobrepõe o tempo desprendido para gerar tal configuração através do processo.

Para usuários preocupados somente com o tempo de resposta e que desejam executar seus *jobs* de modo *ad hoc* no Hadoop, ou seja, pontualmente, com poucas execuções; o uso das *rule of thumbs* parece ser mais atrativo. No entanto, se o usuário está preocupado com outros objetivos, não há registros que essas soluções suportem outras métricas. É nesse aspecto que o processo proposto, apesar de ser relativamente oneroso, fornece uma maior flexibilidade para se trabalhar com as métricas de interesse do usuário.

5.5 Considerações Finais do Capítulo

Neste capítulo foi descrito um estudo de caso realizado com o processo proposto tomando uma aplicação e um ambiente que são representativos na área [16][4]. Cada atividade do processo foi detalhadamente instanciada para o estudo de caso e, através do uso de gráficos e tabelas, foi possível apresentar dados relevantes em cada fase dos experimentos, como resultados de testes estatísticos e valores de parâmetros.

Através de uma validação feita por meio de comparação, foi possível observar que o processo apresentou um tempo médio de resposta melhor que outras soluções de configuração

automática. Por outro lado, o que se mostrou ser uma ameaça à validade foi o alto tempo desprendido para se executar o processo em relação aos outros cenários considerados. No entanto, foi discutido que apesar desse custo há cenários em que o uso do processo é mais vantajoso, sendo possível ignorar o tempo necessário para executá-lo.

No próximo capítulo serão feitas as considerações finais e discutidos aspectos referentes a trabalhos futuros.

Capítulo 6

Considerações Finais

O Hadoop é uma implementação MapReduce muito difundida atualmente [20] [65]. Através dele é possível processar de forma distribuída um grande volume de dados. Esta plataforma é utilizada em muitas áreas, que vão desde a indexação Web e mineração de dados até a simulação científica e pesquisas em bioinformática [38].

Uma importante preocupação no uso do Hadoop é a presença de uma grande quantidade de parâmetros de configuração. Apesar de ser uma plataforma muito difundida, seus usuários não dispõem de informações suficientes para configurá-la adequadamente, executando seus *jobs* com uma eficiência aquém do real potencial da plataforma [4]. Apesar de haver soluções direcionadas à configuração eficiente do Hadoop, até onde foi possível comprovar, nenhuma delas se mostra flexível em relação aos subespaços de parâmetros considerados e às métricas de interesse possíveis de serem avaliadas.

Nesta dissertação foi proposto um processo para auxiliar usuários do Hadoop a configurar eficientemente essa plataforma. Nesse sentido, foi utilizado um conjunto de técnicas empíricas que se baseiam na métrica de interesse do usuário (tempo de resposta, vazão, consumo de energia, etc.), nos recursos de *hardware* disponíveis e nas informações sobre possíveis parâmetros de configuração que podem ser significativamente impactantes para a métrica escolhida.

Para demonstrar a instanciação desse processo, foi realizado um estudo de caso envolvendo um cenário e uma aplicação representativos em relação a trabalhos semelhantes [4]. A métrica escolhida para essa instanciação foi o tempo de resposta. Essa escolha é justificada pela necessidade de se realizar uma validação comparando o processo a algumas soluções

existentes, sendo essa a única métrica coberta por essas soluções.

Apesar de ter apresentado bons resultados, levando em consideração o tempo médio de resposta; o processo proposto apresentou um ponto fraco: o tempo necessário para se executar todos os experimentos necessários. No entanto, foram levantados alguns cenários em que as vantagens trazidas pelo processo superam o tempo necessário para executá-lo, como por exemplo: cenários onde o ambiente de execução dos *jobs* não se altera com o tempo e cenários que exigem flexibilidade em relação às métrica que se deseja analisar.

Concluindo toda a parte descritiva, experimental e analítica, pode-se destacar como principais contribuições desse trabalho: 1) a documentação de um processo que apesar de utilizar técnicas empíricas e estatísticas bastante conhecidas [39], foca-se no uso sistemático das mesmas visando aplicá-las à configuração de plataformas de computação intensiva como o Hadoop [51]; 2) a proposta de uma solução para auxiliar a configuração adequada do Hadoop que seja flexível tanto em relação ao número de parâmetros possíveis de serem cobertos nos experimentos quanto em relação às métricas de desempenho e eficiência possíveis de serem avaliadas; e 3) a formalização de algumas *rules of thumbs* que auxiliam na configuração eficiente da plataforma.

6.1 Trabalhos Futuros

Um dos atrativos do processo proposto é a sua flexibilidade. Isto significa que o processo pode ser instanciado para otimizar diferentes aspectos de eficiência e suas respectivas métricas. Para demostrar essa flexibilidade são descritos nesta seção os resultados preliminares de um estudo de caso envolvendo a instanciação do processo para a vazão (*throughtput*) dos *jobs*, uma métrica em que se pretende maximar a quantidade de *jobs* a serem executados por uma determinada unidade de tempo (exemplo: hora).

A vazão de *jobs* tem sido utilizada em trabalhos relativamente recentes [32] que consideram como cenário um *cluster* de grande porte (com magnitude de dezenas de nodos) sendo compartilhado com vários usuários, os quais executam suas aplicações Hadoop de forma concorrente [38].

Desse modo, o cenário utilizado para o estudo de caso envolvendo a vazão de *jobs* consistiu no mesmo *cluster* especificado na Seção 5.1, mas aplicado à execução de *jobs* para

o TeraSort e Naive Bayes, considerando dois usuários diferentes; cada *job* carregando uma *workload* de 10 GB. O Naive Bayes é um algoritmo de mineração de dados largamente usado para classificar objetos normalmente em duas categorias diferentes [57]. Uma de suas aplicações mais conhecidas é a filtragem de *spam* [46].

É importante destacar que as instanciações do processo proposto para novas métricas podem ser de grande valia na descoberta de associações entre essas próprias métricas e parâmetros de configuração. Para este estudo de caso foi construída a Tabela 6.1 que mostra os elementos que compõem o conjunto dos parâmetros candidatos (L) a serem relevantes para a métrica de interesse. Como fatores secundários foram utilizados os mesmos parâmetros exibidos na Tabela 5.2, ajustados através de *rule of thumbs*.

Tabela 6.1: Conjunto L para o cenário que envolve a vazão de *jobs*

Identificação	**Parâmetro**	**Níveis**
A1	io.file.buffer.size	{4096,65536}
B1	io.sort.mb	{100,250}
C1	io.sort.record.percent	{0.05,0.35}
D1	mapred.jobtracker.taskScheduler	{*org.apache.hadoop.mapred.JobQueueTaskScheduler*, *org.apache.hadoop.mapred.FairScheduler*}
E1	mapred.reduce.tasks	{5,300}

Após o primeiro experimento, seguindo o projeto fatorial (vide Seção 4.4), foi constatado que os erros experimentais atenderam aos requisitos de normalidade, independência e homoscedasticidade. Desse modo, foi possível gerar um modelo de regressão através do qual foi feita a análise exibida na Figura 6.1. Observe que os parâmetros $B1 = io.sort.mb$, $D1 = mapred.jobtracker.taskScheduler$ e $E1 = mapred.reduce.tasks$ são significativamente relevantes.

Como apenas 3 parâmetros foram considerados relevantes após o experimento da Atividade 3, optou-se por retornar à Atividade 2, incluir o parâmetro $io.sort.spill.percent$ ao novo conjunto L e reexecutar o experimento da Atividade 3. Após essa reexecução, foi possível observar que o parâmetro $io.sort.spill.percent$ tornou-se significativamente relevante. A seguir, seguiu-se para a Atividade 4, fechando o conjunto M com 4 parâmetros. Utilizando técnicas de superfície de resposta foi possível obter os valores que otimizaram a métrica de

interesse (vide Seção 4.5). O impacto destes parâmetros sobre a vazão de *jobs* pode ser melhor visualizado no gráfico de superfície exibido na Figura 6.2.

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   5.5937     0.1137   49.19   <2e-16 ***
A1            0.0313     0.1137    0.27    0.784
B1           -0.2604     0.1137   -2.29    0.025 *
C1            0.1354     0.1137    1.19    0.237
D1           -2.6562     0.1137  -23.36   <2e-16 ***
E1           -3.0729     0.1137  -27.02   <2e-16 ***
A1:B1         0.1771     0.1137    1.56    0.123
A1:C1         0.1146     0.1137    1.01    0.317
A1:D1         0.0312     0.1137    0.27    0.784
A1:E1        -0.0104     0.1137   -0.09    0.927
B1:C1         0.0729     0.1137    0.64    0.523
B1:D1         0.2396     0.1137    2.11    0.038 *
B1:E1         0.1562     0.1137    1.37    0.173
C1:D1        -0.1146     0.1137   -1.01    0.317
C1:E1        -0.1563     0.1137   -1.37    0.173
D1:E1         1.8854     0.1137   16.58   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1

Residual standard error: 1.11 on 80 degrees of freedom
Multiple R-squared: 0.952,      Adjusted R-squared: 0.942
F-statistic:  105 on 15 and 80 DF,  p-value: <2e-16
```

Figura 6.1: Sumário dos resultados da análise de regressão considerando os parâmetros em P

Vale ressaltar que este estudo de caso foi instanciado de forma independente do estudo de caso descrito na Seção 5.2, mas alguns parâmetros escolhidos para os respectivos conjuntos L foram os mesmos, a citar: *io.file.buffer.size*, *io.sort.record.percent*, *io.sort.mb* e *mapred.reduce.tasks*. Ainda é possível notar diferenças entre alguns parâmetros otimizados nos estudos de caso das métricas tempo de resposta e vazão de *jobs*. Diferente da métrica tempo de resposta, o parâmetro *io.file.buffer.size* se mostrou consideravelmente relevante para a vazão de *jobs*. Por outro lado, o parâmetro *io.sort.record.percent* se mostrou relevante para o tempo de resposta, mas não para a vazão de *jobs*. O único parâmetro que foi consideravelmente relevante para ambas as métricas foi *mapred.reduce.tasks*.

Uma forma de melhorar o estudo de caso discutido nesta seção é considerar aplicações e cenários abordados por outros trabalhos nesse contexto [32]. E para finalizar este estudo de caso é necessário validar os resultados alcançados através do confrontamento com possíveis resultados obtidos através de experimentos realizados em *clusters* de grande porte. Como normalmente é caro realizar experimentos em *clusters* de grande porte, uma validação mais viável seria uma comparação do processo proposto com as configurações das *rules of thumbs*

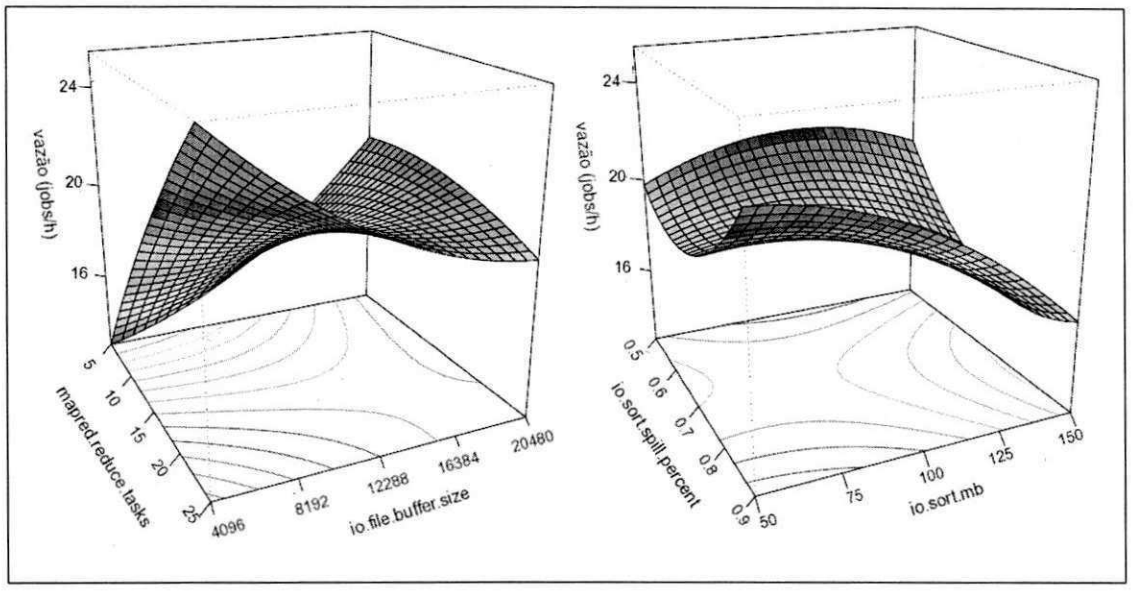

Figura 6.2: Gráfico de superfície para o estudo de caso realizado

e configurações *default* levando em consideração a vazão.

Através da descrição dos resultados preliminares desse estudo de caso, reforça-se a flexibilidade do processo proposto quanto à métrica de interesse. Por outro lado, ainda há uma série de desafios que envolvem essa pesquisa, um deles diz respeito à instanciação do processo para outras métricas, como por exemplo: consumo de energia.

Se for possível aplicar, ao processo, amostras da *workload* alcançando-se resultados que possam ser utilizados (com a mesma eficiência) para executar a *workload* completa, o tempo de execução do processo poderia ser reduzido. Outra contribuição nessa linha é a especificação de mais modelos analíticos para auxiliar na configuração do Hadoop, reduzindo os parâmetros utilizados em experimentos e, consequentemente, tornando o processo mais rápido.

Uma última contribuição futura proposta é a criação de ferramentas que facilitem a aplicação deste processo, automatizando algumas atividades e auxiliando na análise dos dados.

Bibliografia

[1] David P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[2] V.F. Astolfi and J.L. Silva. Execution of algorithms using a dynamic dataflow model for reconfigurable hardware - commands in dataflow graph. In *Proceedings of the 3rd Southern Conference on Programmable Logic*, pages 225–230. IEEE Computer Society, 2007.

[3] Ren B., G. Agrawal, B. Chamberlain, and S. Deitz. Proceedings of the ieee international symposium on parallel and distributed processing workshops and phd forum. In *Translating Chapel to Use FREERIDE A Case Study in Using an HPC Language for Data-Intensive Computing*, pages 1242–1249, sept. 2011.

[4] Shivnath Babu. Towards automatic optimization of mapreduce programs. *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, page 137, 2010.

[5] M. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz. Datacutter: Middleware for filtering very large scientific datasets on archival storage systems. In *Proceedings of the IEEE Symposium on Mass Storage Systems*, pages 119–133. IEEE Computer Society Press, 2000.

[6] Michael D. Beynon, Tahsin Kurc, Umit Catalyurek, Chialin Chang, Alan Sussman, and Joel Saltz. Distributed processing of very large datasets with datacutter. *Parallel Computing Clusters and computational grids for scientific computing*, pages 1457–1478, nov. 2001.

[7] T. Bicer, D. Chiu, and G. Agrawal. A framework for data-intensive computing with cloud bursting. In *Proceedings of the IEEE International Conference on Cluster Computing*, pages 169–177, oct. 2011.

[8] G. E. P. Box and K. B. Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society*, 13(1):1–45, 1951.

[9] Nuran Bradley. The resposne surface methodology. Master's thesis, Indiana University, South Bend, 2007.

[10] A.B. Brown, A. Keller, and J.L. Hellerstein. A model of configuration complexity and its application to a change management system. In *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management, 2005. IM 2005*, pages 631–644. IEEE Computer Society Press, may 2005.

[11] Cloudera. Configuration Parameters: What can you just ignore? Disponível em: <http://www.cloudera.com/blog/2009/03/configuration-parameters-what-can-you-just-ignore/>, Último acesso em: 07 mar. 2012.

[12] H. Chen, W. Zhang, and G. Jiang. Experience transfer for the configuration tuning in large-scale computing systems. *Proceedings of the IEEE Transactions on Knowledge and Data Engineering*, 23(3):388 –401, march 2011.

[13] Xiangqian Chen, K. Makki, Kang Yen, and N. Pissinou. Sensor network security: a survey. *Communications Surveys Tutorials, IEEE*, 11(2):52 –73, 2009.

[14] Ying Chen. Automated tuning of parallel i/o systems: An approach to portable i/o performance for scientific applications. *IEEE Trans. Softw. Eng.*, 26:362–383, April 2000.

[15] W. Cirne, F. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, and M. Mowbray. Labs of the world, unite!!! *Journal of Grid Computing*, 4:225–246, March 2006.

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

[17] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51:107–113, 2008.

[18] B.K. Debnath, D.J. Lilja, and M.F. Mokbel. Sard a statistical approach for ranking database tuning parameters. In *Proceedings of the IEEE 24th International Conference on Data Engineering Workshop*, pages 11 –18, april 2008.

[19] Hyunsook Do and Gregg Rothermel. Using sensitivity analysis to create simplified economic models for regression testing. In *Proceedings of the 2008 international symposium on Software testing and analysis*, ISSTA '08, pages 51–62, New York, NY, USA, 2008. ACM.

[20] J Ekanayake, S Pallickara, and G Fox. Mapreduce for data intensive scientific analyses. *Proceedings of the IEEE Fourth International Conference on eScience*, pages 277–284, 2008.

[21] B.Y. Ekren and S.S. Heragu. Simulation based regression analysis for rack configuration of autonomous vehicle storage and retrieval system. In *Proceedings of the Winter Simulation Conference*, pages 2405–2413, dec. 2009.

[22] Ahmad Faraj, Xin Yuan, and David Lowenthal. Star-mpi: self tuned adaptive routines for mpi collective operations. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 199–208, New York, NY, USA, 2006. ACM.

[23] Siamak Faridani, Ephrat Bitton, Kimiko Ryokai, and Ken Goldberg. Opinion space: a scalable tool for browsing online comments. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1175–1184, New York, NY, USA, 2010. ACM.

[24] Renato A. Ferreira, Wagner Meira, Jr., Dorgival Guedes, Lucia M. A. Drummond, Bruno Coutinho, George Teodoro, Tulio Tavares, Renata Araujo, and Guilherme T. Ferreira. Anthill: A scalable run-time environment for data mining applications. In *Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing*, pages 159–167, Washington, DC, USA, 2005. IEEE Computer Society.

[25] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *Proceedings of the IFIP International Conference on Network and Parallel Computing*, pages 2–13. Springer-Verlag LNCS 3779, 2006.

[26] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson. Statistics-driven workload modeling for the cloud. pages 87–92. IEEE Computer Society Press, 2010.

[27] A. Ganapathi, H. Kuno, U. Daval, J. Wierner, A. Fox, M. Jordan, and D. Patterson. Predicting multiple performance metrics for queries: Better decisions enabled by machine learning. IEEE Computer Society Press, 2009.

[28] Marco Gaviano, Dmitri E. Kvasov, Daniela Lera, and Yaroslav D. Sergeyev. Algorithm 829: Software for generation of classes of test functions with known local and global minima for global optimization. *ACM Trans. Math. Softw.*, 29(4):469–480, 2003.

[29] A. Geist, Beguelin A., Dongarra J., Jiang W., R. Manchek, and V. S. Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing (Scientific and Engineering Computation)*. MIT Press, 1st edition, 1994.

[30] Richard L. Graham, Timothy S. Woodall, and Jeffrey M. Squyres. Open mpi: a flexible high performance mpi. In *Proceedings of the 6th international conference on Parallel Processing and Applied Mathematics*, pages 228–239, Berlin, Heidelberg, 2006. Springer-Verlag.

[31] W. Gropp and E. Lusk. The mpi communication library: its design and a portable implementation. In *Proceedings of the Scalable Parallel Libraries Conference, 1993*, pages 160 –165, 1993.

[32] R. Grover and M. J. Carey. Extending map-reduce for efficient predicate-based sampling. In *Proceedings of the 28th IEEE International Conference on Data Engineering*, 2012.

[33] Joseph F. Hair, William C. Black, Barry J. Babin, Rolph E. Anderson, and Ronald L. Tatham. *Multivariate data analysis*. Person, 6th edition, 2005.

[34] Zhen He, Byung Suk Lee, and Robert Snapp. Self-tuning cost modeling of user-defined functions in an object-relational dbms. *ACM Trans. Database Syst.*, 30:812–853, September 2005.

[35] Herodotos Herodotou and Shivnath Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. *PVLDB*, 4(11):1111–1122, 2011.

[36] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. Starfish: A self-tuning system for big data analytics. *Proceedings of the 5th Conference on Innovative Data Systems Research*, 2011.

[37] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41:59–72, March 2007.

[38] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 261–276, New York, NY, USA, 2009. ACM.

[39] Raj Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley, 1991.

[40] Dawei Jiang, Beng Chin Ooi, Lei Shi, and Sai Wu. The performance of mapreduce: an in-depth study. *Proc. VLDB Endow.*, 3:472–483, 2010.

[41] Yohan Jin, Minqing Hu, H. Singh, D. Rule, M. Berlyant, and Zhuli Xie. Myspace video recommendation with map-reduce on qizmt. In *Proceedings of the IEEE Fourth International Conference on Semantic Computing (ICSC)*, pages 126 –133, sept. 2010.

[42] Rini T. Kaushik and Milind Bhandarkar. Greenhdfs: towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster. In *Proceedings of the 2010 international conference on Power aware computing and systems*, HotPower'10, pages 1–9, Berkeley, CA, USA, 2010. USENIX Association.

[43] S. Kikuchi and S. Tsuchiya. Configuration procedure synthesis for complex systems using model finder. In *Proceedings of the 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 95 –104. IEEE Computer Society Press, march 2010.

[44] F. Larroca and J. L. Rougier. Robust regression for minimum-delay load-balancing. In *Proceedings of the 21st International Teletraffic Congress*, pages 1–8, sept. 2009.

[45] Amazon Elastic MapReduce. Disponível em: <http://aws.amazon.com/pt/elasticmapreduce/>, Último acesso em: 10 abr. 2012.

[46] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. Spam filtering with naive bayes - which naive bayes? In *Proceedings of the 3rd Conference on Email and Anti-Spam*, 2006.

[47] B. K. Mishra, S. K. Singh, and C. V. Joshi. Modeling arrival rate and service rate for next generation network as an open queue using m/m/1 queue model. In *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*, ICWET '10, pages 401–405, New York, NY, USA, 2010. ACM.

[48] Raymond H. Myers, Douglas C. Montgomery, and Christine M. Anderson-Cook. *Response Surface Methodology Process and Product Optimization Using Designed Experiments*. Wiley, 3rd edition, 2009.

[49] The Comprehensive R Archive Network. Disponível em: <http://www.r-project.org/>, Último acesso em: 10 mar. 2012.

[50] R. Tutorials. Chi Sqaure Test of Independence. Disponível em: <http://ww2.coastal.edu/kingw/statistics/r-tutorials/independ.html>, Último acesso em: 08 abr. 2012.

[51] MapReduce-Brasil. Processo para configuração eficiente do Hadoop. Disponível em: <http://redmine.lsd.ufcg.edu.br/projects/confighadoop/wiki>, Último acesso em: 03 abr. 2012.

[52] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 165–178, New York, NY, USA, 2009. ACM.

[53] N.B. Rizvandi, J. Taheri, and A.Y. Zomaya. On using pattern matching algorithms in mapreduce applications. In *Proceedings of the IEEE 9th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 75 –80, 2011.

[54] Li Sa. Proceedings of the collaborative filtering recommendation algorithm based on cloud model clustering of multi-indicators item evaluation. In *Business Computing and Global Informatization (BCGIN), 2011 International Conference on*, pages 645 –648, july 2011.

[55] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. Wiley, 6th edition, 2002.

[56] Iam Sommerville. *Software Engineering*. Addison Wesley, 9th edition, 2010.

[57] Qinbao Song, Guangtao Wang, and Chao Wang. Automatic recommendation of classification algorithms based on data set characteristics. *Pattern Recogn.*, 45(7):2672–2689, July 2012.

[58] G. Tabunshchik, O. Kirsanova, and Z. Zaporizhzhya. Nonparametric methods analysis for medico-ecological research. *Proceedings of the International Conference on Modern Problems of Radio Engineering, Telecommunications, and Computer Science*, pages 657–658, 2006.

[59] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 3st edition, 2007.

[60] Apache. Welcome to Apache Hadoop. Disponível em: <http://hadoop.apache.org>, Último acesso em: 07 mar. 2012.

[61] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 231–242, New York, NY, USA, 2010. ACM.

[62] Impetus. Hadoop Performance Tuning. Disponível em: <http://pt.scribd.com/doc/23046928/hadoop-performance-tuning>, Último acesso em: 07 mar. 2012.

[63] Starfish: Self tuning Analytics System. Disponível em: <http://www.cs.duke.edu/starfish/>, Último acesso em: 10 mar. 2012.

[64] Apache. MapReduce Tutorial. Disponível em: <http://hadoop.apache.org/common/docs/r0.20.2/mapred_tutorial.html>, Último acesso em: 31 mar. 2012.

[65] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, 1st edition, 2009.

[66] T. Wirtz and Rong Ge. Improving mapreduce energy efficiency for computation intensive workloads. In *Proceedings of the International Green Computing Conference and Workshops (IGCC)*, pages 1–8, 2011.

[67] Claes Wohlin, Per Runeson, Martin Höst, Magnus Ohlsson, Björn Regnell, and Andres Wesslén. *Experimentation in Software Engineering: An Introducion*. Kluwer Academic Publishers, 1st edition, 2000.

[68] Changrong Yan and Dixin Zhang. The impact of dividend policy preferences on stock returns in china a-shape market. In *Proceedings of the 4th International Joint Conference on Computational Sciences and Optimization (CSO)*, pages 718 –722, april 2011.

[69] Ustun Yildiz, Adnene Guabtni, and Anne H. H. Ngu. Towards scientific workflow patterns. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, WORKS '09, pages 13:1–13:10, New York, NY, USA, 2009. ACM.

[70] Matei Zaharia, Andy Konwinski, Anthony D Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. *Symposium A Quarterly Journal In Modern Foreign Literatures*, 57(4):29–42, 2008.