Efetividade da Política de Posicionamento de Blocos no Balanceamento de Réplicas do HDFS

Rhauani Weber Aita Fazul[1], **Patrícia Pitthan Barcelos**[1]

[1]Laboratório de Sistemas de Computação (LSC)
Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brasil

{rwfazul, pitthan}@inf.ufsm.br

Abstract. *The Hadoop Distributed File System (HDFS) is designed to store and transfer data in large scale. To ensure availability and reliability, it uses data replication as a fault tolerance mechanism. However, this strategy can significantly affect replication balancing in the cluster. This paper provides an analysis of the default data replication policy used by HDFS and measures its impacts on the system behavior, while presenting different strategies for cluster balancing and rebalancing. In order to highlight the required requirements for efficient replica placement, a comparative study of the HDFS performance has been conduct considering a variety of factors that may result in cluster imbalance.*

Resumo. *O HDFS, sistema de arquivos distribuído do Apache Hadoop, foi projetado para armazenar e transferir volumes massivos de dados. Para garantir confiabilidade e disponibilidade, o HDFS utiliza a replicação de dados como técnica de tolerância a falhas. No entanto, esta estratégia pode afetar significativamente o balanceamento de réplicas no cluster. Este trabalho analisa o modelo de replicação adotado pelo HDFS e mede seu impacto no comportamento do sistema, enquanto pontua possíveis abordagens de balanceamento. De forma a evidenciar os requisitos necessários para um posicionamento de réplicas eficiente, um estudo comparativo do desempenho do HDFS foi conduzido levando em consideração diferentes fatores que podem resultar no desbalanceamento.*

1. Introdução

A evolução computacional das últimas décadas proporcionou inúmeros avanços tecnológicos. Aprimoramentos em metodologias e processos organizacionais permitiram um grande aumento na produção e na coleta de dados e informações potencialmente úteis. Para lidar com esse novo cenário, surgem as plataformas dedicadas à gerência de *big data*, como o *framework* Apache Hadoop[1]. Já consolidado, o Hadoop oferece um ecossistema composto de soluções eficientes baseadas em computação paralela e distribuída.

No sistema de arquivos distribuído do Hadoop, o HDFS, são implementados diversos mecanismos voltados a tolerância a falhas (TF), dentre eles a replicação de dados. Base para o funcionamento do sistema, a replicação visa proporcionar confiabilidade por meio de redundância, enquanto estimula possíveis melhorias de desempenho ao permitir que as aplicações explorem uma maior disponibilidade dos dados. Neste sentido, a forma

[1]https://hadoop.apache.org/

como as cópias desses dados são posicionadas dentro do HDFS acaba se tornando essencial para garantir o balanceamento de réplicas entre os nodos. Quando não realizada de forma otimizada, a replicação pode desbalancear a utilização dos recursos computacionais e degradar o desempenho geral do sistema. Todavia, mensurar o impacto real da distribuição de réplicas no funcionamento do HDFS torna-se uma tarefa complexa, tendo em vista que diversos fatores podem influenciar o equilíbrio do sistema.

Este trabalho investiga sistematicamente diferentes situações que podem ocasionar o desbalanceamento de réplicas no HDFS. Para tal, experimentos foram conduzidos de forma a avaliar a efetividade da atual política de posicionamento do sistema durante a distribuição dos dados, levando em consideração cenários com potencial impacto no equilíbrio de carga do *cluster*. Diferenças no comportamento do sistema, em especial mudanças de desempenho na execução de aplicações voltadas a entrada e saída (E/S), foram analisadas ao empregar soluções dedicadas ao balanceamento de réplicas no HDFS.

O artigo está organizado em oito seções. O *framework* Apache Hadoop é apresentado na Seção 2. A Seção 3 é dedicada aos principais mecanismos de TF implementados pelo HDFS, enquanto a Seção 4 aprofunda-se na replicação de dados. Estratégias para o balanceamento de réplicas no HDFS são apresentadas na Seção 5. A Seção 6 detalha a metodologia empregada nos experimentos. A Seção 7 exibe e discute os resultados obtidos. Por último, a Seção 8 realiza as considerações finais e direciona os trabalhos futuros.

2. Apache Hadoop

Já reconhecido pela indústria, o *framework* Apache Hadoop é uma plataforma *open-source* de propósito geral para análise e processamento de *big data*. Sua arquitetura, voltada à computação distribuída e ao processamento paralelo em larga escala, baseia-se em três componentes principais [Achari 2015], sendo eles: o *Hadoop Distributed File System* (HDFS), o modelo MapReduce e o *Yet Another Resource Negotiator* (YARN).

O HDFS é um sistema de arquivos distribuído altamente escalável e flexível, projetado para ser tolerante a falhas mesmo quando executa em *clusters* com *hardware* comum. Dentro do HDFS segue-se uma arquitetura mestre-escravo formada por dois tipos de nós: `NameNode` (NN) e `DataNode` (DN). O NN é o servidor mestre, responsável por manter todo *namespace* e a árvore de diretórios do sistema. Além disso, cabe ao NN controlar a distribuição e o acesso dos dados, que são armazenados efetivamente nos DNs.

Capaz de operar sobre um volume massivo de dados, o HDFS possui uma estrutura de armazenamento própria. Quando um arquivo é inserido no sistema, ao invés de armazená-lo em seu formato original, gera-se uma sequência de blocos. Os blocos consistem em segmentos de dados criados automaticamente a partir da divisão do arquivo inicial e, com exceção do último, possuem tamanho fixo (128MB na versão 2 do Hadoop). Sendo um sistema confiável, que preza pela disponibilidade e integridade dos dados, o HDFS necessita de meios que garantam a tolerância e recuperação mediante a falhas. Os principais mecanismos de TF implementados no sistema são apresentados na Seção 3.

Outro aspecto importante no funcionamento do HDFS é sua capacidade de levar a computação (processamento) para onde os dados estão armazenados, possibilitando explorar de forma otimizada ferramentas como o MapReduce [White 2015]. O modelo de programação MapReduce foi projetado como um *framework* de processamento paralelo,

capaz de manipular grandes volumes de dados em ambientes distribuídos. O Hadoop consegue executar programas MapReduce, nativamente paralelos, escritos em diferentes linguagens de programação. A execução baseia-se na divisão das unidades de trabalho (*jobs*) em um conjunto de tarefas independentes (*map tasks* e *reduce tasks*). As tarefas rodam em nós do *cluster* HDFS e são resilientes a falhas, assim caso um nodo que esteja executando uma tarefa falhe, a tarefa em questão será designada a um nó diferente do *cluster*. O escalonamento das tarefas MapReduce (MRv2) é realizado pelo YARN.

Sendo o centro de arquitetura, o YARN surgiu como uma plataforma de uso geral incorporada após a segunda versão do Hadoop. Até então, apenas um paradigma de computação distribuído era suportado (MRv1), fazendo com que o Hadoop operasse exclusivamente em *batch* [Achari 2015]. Com o YARN, tornou-se possível que aplicações de propósitos diferentes, tais como análise em tempo real e *streaming*, compartilhassem um gerenciador de recursos comum [White 2015]. Para tal, capacidades envolvendo a gerência de *jobs* e recursos foram separadas dos processos do MapReduce e delegadas ao YARN, aprimorando o desempenho e a tolerância a falhas da plataforma como um todo.

3. Tolerância a Falhas no HDFS

Por executar sobre equipamentos de baixo custo, o HDFS necessita de meios eficientes e seguros para lidar com falhas. Dentre seus principais mecanismos de tolerância a falhas estão o estabelecimento de *checkpoints*, o modo de alta disponibilidade do NN, as mensagens *heartbeat* e a replicação de blocos [Foundation 2018].

Como uma técnica de TF reativa, o estabelecimento de *checkpoints* opera por meio de um nodo específico dedicado ao registro dos metadados do sistema, que são originalmente armazenados no NN. Para tal, periodicamente o arquivo com o estado corrente do *namespace* (*FSImage*) e o *log* das edições realizadas desde o último *checkpoint* (*EditLog*) são mesclados, criando uma imagem atualizada com o contexto do sistema e a salvando em um dispositivo fisicamente distinto do NN em execução. Quando ocorre uma falha, o sistema recupera o último registro armazenado e o utiliza para recuperação de seu estado.

Embora esta estratégia ofereça certa proteção contra a perda de dados, ela não fornece recuperação automática em caso de falhas no NN [Turkington 2013], exigindo que um administrador manualmente inicie um NN com o último *checkpoint* e configure os DNs para reconhecer este novo nó mestre. Como neste período de transição todo sistema fica fora de serviço, o NN acaba sendo um ponto único de falha (SPOF). Pensando em remover esta condição, criou-se a funcionalidade do HDFS *High Availability* (HA) [White 2015]. O modo de operação HA foi um grande avanço introduzido no Hadoop versão 2, consistindo em um par de NNs (um ativo e outro em *stand by*) sincronizados, o que permite que o nodo mestre em espera assuma a gerência do *cluster* em caso de indisponibilidade do NN primário, sem interrupção significativa no funcionamento do sistema.

Além dos mecanismos para manter o NN operando corretamente, a confiabilidade no HDFS depende da segurança dos dados armazenados nos DNs. De forma a promover o monitoramento ativo dos nodos, os processos do NN e dos DNs comunicam-se constantemente através de mensagens *heartbeat*. As mensagens *heartbeat* são enviadas pelos DNs em intervalos periódicos, informando seu estado e a situação dos blocos de dados neles armazenados, sendo que o NN pode responder estas mensagens com instruções específicas para alguma operação necessária no sistema de arquivos. Falhas de nodo são

identificadas caso nenhuma mensagem *heartbeat* seja recebida pelo NN dentro de um limite de tempo previamente definido, fazendo com que o DN seja marcado como inativo.

Mesmo que os DNs inativos sejam identificados pelas mensagens *heartbeat*, o HDFS deve evitar a perda dos dados em caso de falhas. Assim, surge a principal estratégia para o funcionamento do sistema: a replicação de blocos. Sendo o objeto de análise deste trabalho, o processo de replicação é apresentado e explicado com detalhes na Seção 4.

4. Replicação de Blocos

Conforme já citado, o HDFS armazena arquivos como uma sequência de blocos. Para garantir a disponibilidade dos dados e aumentar a tolerância a falhas do sistema, estes blocos são replicados e distribuídos em múltiplos DNs do *cluster*. Assim, caso um nodo falhe, ainda é possível acessar a cópia dos dados através de um DN que armazene a réplica.

A quantidade de réplicas geradas para cada bloco é definida a partir de um parâmetro configurável, chamado Fator de Replicação (FR). O FR pode ser determinado pela aplicação durante a criação de cada arquivo, sendo possível alterá-lo posteriormente. Atuando como nó mestre, o NN é encarregado de todas as decisões acerca da replicação dos blocos no HDFS e, cabe a ele, tomar ações para que o FR definido seja respeitado.

Desta forma, além da replicação inicial realizada durante o armazenamento, a re-replicação de blocos específicos pode se tornar necessária. Dentre suas causas estão o aumento do FR de um arquivo, a corrupção de alguma réplica ou falhas no funcionamento de DNs [Foundation 2018]. Ao incrementar o FR, novas réplicas deverão ser criadas para atingir a conformidade com o novo fator. Já a corrupção de um bloco afeta a integridade dos dados e necessita de um mecanismo de *checksum* para ser descoberta [Turkington 2013]. Para tal, quando um bloco é armazenado no HDFS, associa-se a ele um *checksum*. Ao recuperar um bloco, seu respectivo *checksum* também é retornado, assim, após realizar um novo cálculo, é possível compará-los. Caso as somas não sejam compatíveis, o bloco é marcado como corrompido. O estado de todos os blocos de um DN é enviado periodicamente para o NN na forma de um *Blockreport* e, ao identificar um bloco corrompido, o NN pode, se necessário, disparar a criação de uma nova réplica.

A re-replicação também pode ser ocasionada quando um DN do *cluster* deixar de operar corretamente. Problemas na comunicação entre o NN e o DN ou falhas que levem o DN a um estado indisponível, envolvendo tanto *hardware* quanto *software*, podem impedir que as mensagens *hearbeat* sejam recebidas. Caso o limite de tempo definido, referente a tolerância máxima na ausência das mensagens de um DN, seja extrapolado, o DN em questão será marcado como inativo pelo NN. Além de não receber novas requisições de E/S, um DN inativo ocasiona o decremento do FR dos blocos nele mantidos. Como o NN mantém o controle do estado de todos os blocos no sistema de arquivos, a re-replicação será realizada a partir de um dos DNs ativos que possua a cópia dos dados.

Ambos os processos de replicação e re-replicação demandam uma decisão do NN: a escolha do DN para o armazenamento efetivo do bloco replicado. Esta decisão é realizada respeitando um modelo de referência para a distribuição das réplicas no *cluster*.

4.1. Política de Posicionamento de Réplicas

Para cada bloco a ser armazenado no HDFS, o NN deve selecionar os DNs para o recebimento das réplicas. Esta escolha deve ser realizada de modo que a disponibilidade dos

dados seja preservada em caso de falhas e o desempenho do sistema possa ser otimizado.

Instâncias do HDFS são capazes de executar em múltiplos nodos, dispostos em diferentes *racks*. Sendo um sistema tolerante a falhas, o HDFS deve evitar a perda de dados mesmo se um *rack* inteiro falhe. Assim, o NN segue um modelo inteligente para o posicionamento das réplicas baseado na arquitetura do *cluster* (*rack-aware*), que parte de duas premissas [Achari 2015]: (i) um DN pode armazenar apenas uma única réplica de um mesmo bloco; e (ii) um *rack* pode conter, no máximo, duas das réplicas de um bloco.

Seguindo estas duas premissas, a política de posicionamento de réplicas é aplicada. A estratégia corrente no HDFS, considerando um FR de 3, armazena a primeira réplica no nodo local ou, caso o cliente não esteja em um DN, em um DN aleatório; a segunda réplica em um *rack* remoto distinto (*off-rack*) e; a réplica final, em um nodo distinto, escolhido aleatoriamente no mesmo *rack* da segunda réplica [Foundation 2018]. Em caso de um FR maior que 3, os nodos seguintes são escolhidos arbitrariamente, embora o sistema tente acordar com as duas premissas anteriores para distribuir dos blocos.

Esta estratégia permite aprimorar o desempenho geral do HDFS durante operações de E/S no sistema. Na operação de escrita, após o NN criar uma lista com os DNs selecionados, é construído um *pipeline* de replicação entre os nodos [White 2015], possibilitando que, quando aplicável, um DN simultaneamente receba dados do DN anterior no *pipeline* e encaminhe para o próximo DN da lista. Já durante a operação de leitura, como o HDFS segue uma estratégia *rack-aware*, torna-se possível identificar o nodo mais próximo do cliente que possui a réplica necessária, diminuindo o tempo gasto com tráfego de dados.

Embora permita aumentar a confiabilidade do sistema em caso de falhas (redundância de dados em diferentes *racks*) e contribua com uma melhor utilização dos recursos computacionais (proveito da largura de banda dos múltiplos *racks* durante E/S), a política de posicionamento atual não distribui os blocos de maneira igualitária dentro do HDFS, seja em nível de nodo ou de *rack*. Os problemas inerentes ao desbalanceamento e soluções voltadas ao balanceamento das réplicas são apresentados na Seção 5.

5. Balanceamento do *Cluster*

O HDFS foi projetado para suportar um grande volume de dados. Naturalmente, o armazenamento de arquivos maiores resulta em uma quantidade maior de blocos a ser distribuída dentro do *cluster*. Quando não realizada de forma equilibrada entre os nodos, esta distribuição pode contribuir com o desbalanceamento de réplicas. A política padrão, embora proporcione um balanceamento mínimo (réplicas de um mesmo bloco não são mantidas pelo mesmo DN), não fornece garantias de uma distribuição homogênea.

Ao seguir a política, um *rack* é selecionado para manter dois terços das réplicas de um determinado bloco, o que favorece o desbalanceamento inter-*rack*. A utilização de disco dos DNs também não é levada em consideração para a seleção, contribuindo com o desbalanceamento de carga entre os DNs. Outros aspectos que podem influenciar o estado de balanceamento do HDFS são: (i) a ocorrência de falhas de DN, onde blocos necessitam ser re-replicados; (ii) o comportamento da aplicação do cliente, que pode executar diretamente em um DN, fazendo com que este, de acordo com a política atual, armazene uma cópia local para preservar a localidade dos dados; e (iii) a adição de um novo nó ao *cluster*, já que este irá competir igualmente com outros DNs para o recebimento dos blocos replicados, resultando em um período de subutilização significativo [Turkington 2013].

Conforme o desbalanceamento se intensifica, problemas de desempenho tendem a surgir. Como citado na Seção 2, o HDFS se esforça para levar o processamento para onde os dados estão armazenados, o que reduz o consumo da largura de banda do *cluster*. Considerando o MapReduce, isso é realizado através da otimização da localidade dos dados [White 2015], que consiste na execução das tarefas da etapa de *map* dentro dos nodos que possuem os blocos necessários para a realização do *job*. Quando o *cluster* está desbalanceado, DNs que armazenam um maior número de blocos são propensos à sobrecarga pela execução de mais operações de E/S. Se todos os DNs que possuem uma cópia do bloco requisitado já executarem uma tarefa *map*, o escalonador irá buscar por um nodo livre no mesmo *rack* de uma das réplicas. Se não houver disponibilidade, um nodo *off-rack* será selecionado, o que geralmente resulta em um maior custo de transferência de dados.

Assim, por ser um sistema baseado no modelo de acesso WORM (*write once*, *read y many*) e ter como uma prioridade maximizar a vazão durante operações de leitura dos dados [Achari 2015], espera-se que o balanceamento de réplicas aprimore a utilização do HDFS. Embora o ecossistema do Hadoop não implemente um monitor nativo, que dispare ações de balanceamento automáticas em seu sistema de arquivos, a arquitetura do HDFS é compatível com esquemas de re-balanceamento [Foundation 2018]. Com isso, o balanceamento no HDFS pode ser endereçado tanto de um modo preventivo quanto reativo. As Seções 5.1 e 5.2 apresentam abordagens que exemplificam ambas as estratégias.

5.1. Soluções Preventivas

Uma forma de tratar o desbalanceamento de réplicas no HDFS é agir no momento da distribuição inicial dos blocos. Com isso, é possível impedir - ou reduzir as chances - que o *cluster* fique desequilibrado. Muitas das soluções complementares existentes na literatura envolvem a criação de novas políticas de posicionamento réplicas para o HDFS que, proativamente, de forma direta ou indireta, promovem o balanceamento de réplicas.

Um aspecto a ser considerado para o balanceamento é o volume de dados armazenado em cada DN. Neste sentido, [Ibrahim et al. 2016] propõem um novo algoritmo de posicionamento com objetivo de distribuir as réplicas de uma forma totalmente equilibrada entre os nodos. Para tal, baseado em suas cargas, os DNs são marcados como livres ou ocupados. Durante a seleção do *rack* para manter o bloco, priorizam-se os *racks* com um maior número de nodos livres. Após, escolhe-se o DN que, estando de acordo com a política padrão, possuir o menor espaço em uso entre todos os demais do *rack*. Já [Patole et al. 2015] partem da criação de partições virtuais no *cluster* de modo a permitir que o posicionamento das réplicas seja *load-aware*. Os resultados demostram que, basear-se no estado de utilização dos DNs para a seleção permite uma distribuição uniforme da carga entre os *racks* e uma redução no tempo de leitura dos dados armazenados no HDFS.

Para otimizar a localidade dos blocos em ambientes heterogêneos que executam o Hadoop, características específicas de cada nodo podem ser consideradas durante a distribuição das réplicas. Neste sentido, a estratégia de posicionamento proposta por [VishnuVardhan and Baruah 2016] é baseada na capacidade de computação dos DNs. O algoritmo implementa uma política que atribui dinamicamente uma maior quantidade de blocos aos DNs com maior desempenho de processamento. Um balanceamento mínimo é garantido durante a distribuição das réplicas, de modo a evitar que DNs com *hardware* inferior fiquem subutilizados. Em [Dharanipragada et al. 2017], por sua vez, é proposta

uma política de posicionamento que baseia-se na latência de disco de cada nodo, a qual os autores introduziram para ser enviada ao NN juntamente com o *Blockreport* dos DNs.

5.2. Soluções Reativas

O balanceamento reativo no HDFS atua como uma estratégia corretiva que permite aumentar o nível de equilíbrio de carga entre os nodos do *cluster*. Para tal, a redistribuição dos blocos pode ser realizada visando tanto o balanceamento inter-*rack* quanto inter-DN.

Exemplos dessa abordagem incluem [Liu et al. 2013], que propõem um algoritmo de balanceamento que atua primariamente em equilibrar *racks* sobrecarregados, contribuindo com uma distribuição mais uniforme dos dados armazenados no HDFS. Em [Dharanipragada et al. 2017] é introduzido um algoritmo modificado (*LatencyBalancer*) que leva em consideração tanto a latência dos discos dos nodos quanto o espaço em uso nos DNs para realocar os blocos. Sabendo que o processo de balanceamento pode ser trabalhoso em função do estado em que o *cluster* se encontra, [Shah and Padole 2018] focam em otimizar o processo de redistribuição das réplicas aproveitando-se da capacidade de processamento dos nodos. A partir de uma classificação inicial pela heterogeneidade e desempenho de cada nó, os blocos são movimentados apenas para DNs que possibilitem minimizar o tempo e o tráfego necessário para a transferência dos dados.

Outra possibilidade para o balanceamento reativo está presente na distribuição do Hadoop. Sendo a solução empregada durante os experimentos realizados neste trabalho, a Seção 5.2.1 apresenta detalhadamente o funcionamento do `HDFS Balancer`.

5.2.1. HDFS *Balancer*

O `HDFS Balancer` [Shvachko et al. 2010] (referenciado posteriormente como Balanceador) é uma ferramenta integrada do Hadoop destinada ao balanceamento de réplicas entre dispositivos de armazenamento do HDFS. Sua operação é disparada sob demanda pelo administrador do *cluster*. A partir de sua política padrão (`DataNode`), o Balanceador iterativamente move blocos de DNs superutilizados para DNs subutilizados com base em um valor de *threshold*, que é passado como parâmetro para a execução. Representado como uma porcentagem no intervalo de (0, 1), o *threshold* determina o valor máximo que a utilização dos DNs (proporção do espaço em uso no nodo para a capacidade total do nodo) pode diferir da utilização geral do *cluster* (proporção do espaço em uso no *cluster* para a capacidade total do *cluster*). Quando a utilização de cada DN estiver dentro desse limite, que tem o valor padrão de 10%, o *cluster* é tido como balanceado.

Em um sistema como o HDFS, que lida com um constante fluxo de dados, a operação de balanceamento de réplicas pode apresentar alta complexidade. O Balanceador precisa considerar diversos aspectos durante sua execução, como por exemplo, os múltiplos dispositivos de armazenamento que um mesmo DN é capaz de suportar. O algoritmo padrão empregado pelo Balanceador é composto por quatro fases principais, que são executadas iterativamente, sendo elas [Hortonworks 2018]:

1. Classificação dos grupos de dispositivos: para cada DN são obtidas informações como a sua capacidade de armazenamento total, espaço disponível (e utilizado) e a lista de seus dispositivos de armazenamento (um nodo pode possuir tanto um

único dispositivo quanto múltiplos). Os dispositivos são divididos em grupos de acordo com seus tipos. Em seguida, cada grupo é classificado em:

- Superutilizado (*Over-utilized*): quando a porcentagem de utilização do grupo de um determinado DN for maior que a média de utilização de todos os dispositivos do *cluster* acrescida do valor de *threshold*;
- Acima da média (*Above-average*): quando a porcentagem de utilização do grupo de um determinado DN estiver entre a média de utilização de todos os dispositivos do *cluster* e a média acrescida do valor de *threshold*;
- Abaixo da média (*Over-average*): quando a porcentagem de utilização do grupo de um determinado DN estiver entre a média de utilização de todos os dispositivos do *cluster* e a média subtraída do valor de *threshold*;
- Subutilizado (*Under-utilized*): quando a porcentagem de utilização do grupo de um determinado DN for menor que a média de utilização de todos os dispositivos do *cluster* subtraída do valor de *threshold*.

Se o *cluster* HDFS não possui nenhum grupo de dispositivos de armazenamento classificado como subutilizado ou superutilizado ele é considerado balanceado. Caso contrário, a execução do Balanceador continua na fase 2.

2. Pareamento dos grupos: nesta fase definem-se pares origem-destino entre os grupos gerados na fase anterior. A estratégia para criação dos pares utiliza inicialmente grupos de um mesmo *rack* e, caso não seja possível, procura formar pares em qualquer *rack*. Independentemente, mantém-se a seguinte ordem de prioridade:
   - Origem: grupo de dispositivos de armazenamento do DN classificado como superutilizado. Destino: grupo classificado como subutilizado;
   - Origem: grupo de dispositivos de armazenamento do DN classificado como superutilizado. Destino: grupo classificado como abaixo da média;
   - Origem: grupo de dispositivos de armazenamento do DN classificado como acima da média. Destino: grupo classificado como subutilizado.
3. Agendamento de movimentação dos blocos: para cada par definido na fase 2, selecionam-se os blocos para serem redistribuídos. Um bloco do grupo de origem é eletivo ao movimento se: (i) estiver armazenado em um dispositivo do mesmo tipo que o destino; (ii) não possuir uma réplica já existente no destino; (iii) estiver de acordo com a política de posicionamento padrão do HDFS; e (iv) não houver movimentações já agendadas no seu grupo. Após definir o bloco a ser realocado, o Balanceador se esforça em otimizar o processo de balanceamento selecionando o DN mais próximo do destino que possuir uma réplica do bloco a ser movimentado como *proxy*, diminuindo assim o tráfego necessário para a transferência de dados.
4. Transferência do bloco: o DN destino faz a cópia do bloco mantido no DN *proxy* para o armazenamento local. Ao final, ele envia um alerta para o NN, que então dispara a operação de deleção da réplica armazenada no DN origem. Se, após todas as movimentações da iteração serem completadas, o *cluster* ainda não estiver equilibrado (em função do *threshold* definido), uma nova iteração será iniciada. Caso cinco iterações consecutivas não concluírem corretamente ou, se o contato com NN for perdido, a execução do Balanceador é interrompida.

A *daemon* do Balanceador foi originalmente projetada para operar sem afetar as atividades normais do *cluster* ou interferir com os clientes e suas aplicações [White 2015]. De todo modo, é possível definir a largura de banda máxima que o Balanceador pode consumir durante sua execução (por padrão 1MB/s). Quanto maior a largura permitida, mais

rápido o balanceamento será realizado, porém aumentando as chances de sobrecarga por gerar maior concorrência com os demais processos correntes no sistema. Outras funcionalidades configuráveis do Balanceador incluem fixar DNs específicos para serem balanceados e estabelecer listas de exclusões com réplicas que não devem ser redistribuídas.

6. Metodologia

De forma a medir o impacto da política de posicionamento atual no balanceamento de réplicas do *cluster* e, também, avaliar o comportamento do sistema após a execução de uma solução reativa ao desbalanceamento, conduziu-se uma análise empírica por meio da execução de testes de desempenho no HDFS. A aplicação utilizada durante os experimentos foi um *benchmark* integrado na distribuição do Hadoop, o `TestDFSIO` [White 2015].

Por possuir um comportamento *I/O bound*, o `TestDFSIO` possibilita medir o desempenho do HDFS através da execução paralela de tarefas MapReduce focadas em E/S. Alinhando-se aos interesses deste trabalho, o *benchmark* permite, através da operação de escrita, avaliar o equilíbrio da distribuição dos blocos realizada durante o processo de replicação. Já, através da operação de leitura, torna-se possível investigar a existência de possíveis diferenças de desempenho motivadas pelo balanceamento de réplicas no *cluster*.

O ambiente de experimentação baseou-se em uma distribuição do Hadoop na versão 2.9.2 operando em modo totalmente distribuído. Para tal, utilizou-se a plataforma Grid'5000[2], onde, em um *cluster* com rede Gigabit Ethernet, foram configurados 1 NN e 10 DNs para execução em 10 nodos distintos (NN executando no mesmo nó do primeiro DN), cada um com as seguintes configurações: 2 CPUs Intel Xeon E5520 com 4 cores por CPU e frequência de 2.27GHz, 32GB de memória RAM e capacidade na unidade de disco rígido de 557GB, executando uma distribuição Debian GNU/Linux 9.7 (*stretch*).

6.1. Cenários de Teste

Para possibilitar um entendimento mais amplo do balanceamento de réplicas no HDFS, diferentes cenários de teste foram considerados. Os aspectos elencados que, direta ou indiretamente, relacionam-se com o desbalanceamento do *cluster*, foram: ocorrência de falhas de DNs, fator de replicação dos blocos e volume de dados no sistema.

A análise da ocorrência de falhas de DN mostra-se interessante, pois possibilita investigar o efeito do processo de re-replicação no balanceamento do HDFS. Estudos passados demostram que mesmo uma única falha de nodo é capaz de causar forte impacto na usabilidade do sistema, resultando em um comportamento inesperado na execução de aplicações MapReduce e, assim, podendo aumentar seu tempo de execução [Dinu and Ng 2012]. Já o fator de replicação possui um papel importante em como os dados são distribuídos entre os *racks* e seus nodos. Aumentar o FR de um arquivo otimiza tanto a disponibilidade dos blocos como a localidade dos dados, possibilitando aprimoramentos em operações de E/S no HDFS [Ciritoglu et al. 2018]. O acréscimo do volume de dados manipulado pelo sistema, por sua vez, permite acentuar os resultados de ambos os aspectos anteriores. Dessa forma, os cenários de teste foram idealizados considerando a combinação das seguintes variações dos parâmetros de interesse em observação:

[2]Grid'5000 é uma plataforma para experimentos apoiada por um grupo de interesses científicos hospedado pelo Inria e incluindo CNRS, RENATER e diversas Universidades, bem como outras organizações (mais detalhes em `https://www.grid5000.fr`).

- Fator de Replicação: variação pelo incremento em relação ao valor padrão, assim considerando-se FRs de 3 e 4 réplicas por bloco;
- Carga no sistema: escrita de 10 arquivos pelo *benchmark*. As execuções variaram o tamanho destes arquivos em 10GB, 15GB e 20GB.
- Quantidade de falhas de DN: variação entre execuções da aplicação sem falhas até ocorrência de 50% de falhas nos DNs, assim considerando 0, 1, 2, 3, 4 e 5 falhas de DNs durante a operação de escrita realizada pelo `TestDFSIO` (versão 1.8);

O FR (e o FR mínimo) de cada arquivo foi fixado para seu valor correspondente antes da execução do *benchmark*, de modo que todos os arquivos posteriormente escritos pelo `TestDFSIO` fossem replicados com base no mesmo fator. A variação dos tamanhos ocorreu entre testes diferentes e, em um mesmo teste, se manteve o mesmo para todos os arquivos. Sendo assim, a quantidade de blocos escrita em cada teste pode ser calculada como: Blocos $= 10 * (\text{Tam}_a/128) * \text{FR}_a$, onde 10 e 128 são valores fixos referentes a quantidade de arquivos escritos e ao tamanho de cada bloco e FR_a e Tam_a são variáveis referentes ao FR e ao tamanho total (em MB) de cada arquivo, respectivamente.

Quando aplicável, as falhas de DN foram induzidas durante a escrita dos arquivos. A introdução das falhas foi realizada pelo comando *kill* do Linux aos processos dos DNs, os quais foram selecionados aleatoriamente. As falhas são identificadas pelo NN em decorrência da ausência de mensagens *heartbeat* em um intervalo pré-definido. Como o tempo para o NN marcar um DN como inativo é conservadoramente longo, este período foi reduzido (através de um conjunto de parâmetros de configuração do HDFS) para aproximadamente 20 segundos. O tempo para ocorrência da primeira falha de DN e o intervalo entre as falhas foi fixado em 60 segundos. Desta forma, o processo de re-replicação é disparado pelo NN antes da escrita dos arquivos ser finalizada pelo *benchmark*.

O algoritmo de testes considerou, em respectiva ordem: (i) a execução do *benchmark* `TestDFSIO` em modo de escrita com o FR, a carga e o número de falhas correspondente do cenário; (ii) a execução do `TestDFSIO` para leitura de todos arquivos armazenados no sistema (neste ponto o *cluster* está sujeito ao desbalanceamento de réplicas); (iii) a execução do Balanceador; e, por fim, (iv) uma nova execução do `TestDFSIO` em modo leitura . Dada a proporção da carga utilizada nos testes para a capacidade de disco total dos nodos, o *threshold* do Balanceador foi reduzido para 3%, de modo a possibilitar um balanceamento mais significativo. Os resultados exibidos na Seção 7 consideram os valores médios obtidos em 20 execuções do *benchmark* em cada um dos cenários de teste.

7. Experimentação e Resultados

Com base na metodologia empregada, para cada cenário (antes e após o balanceamento) foram coletadas informações e métricas do sistema de arquivos, sendo estas: (i) tempo de balanceamento; (ii) carga total armazenada no HDFS; (iii) desvio padrão na utilização dos DNs; (iv) tempo de execução, sendo este o tempo médio para a leitura dos arquivos gasto pela aplicação MapReduce; (v) *throughput* alcançado durante a execução do *benchmark* em modo leitura; e (vi) taxa média na transferência dos dados durante a leitura.

7.1. Três Réplicas por Bloco

Na Tabela 1 são apresentados os resultados considerando o FR padrão do HDFS. O desbalanceamento de réplicas, decorrente da distribuição dos blocos baseada na política de

posicionamento padrão do sistema, pode ser observado através do desvio padrão (DP) da utilização dos DNs antes do balanceamento. Quanto maior o DP, maior a variação em torno do valor médio ideal para o balanceamento, que é calculado considerando a porcentagem de dados que cada DN mantém em relação ao volume total armazenado no *cluster*. Como exemplo, em um cenário com 10 DNs e duas falhas, o valor médio seria de 12,5%, o que representa a parcela dos dados que cada um dos 8 DNs (ativos) restantes deveria armazenar em seu nodo para alcançar um balanceamento ótimo no *cluster*.

Tabela 1. Comportamento do HDFS antes e após o balanceamento com FR = 3.

Quantidade de falhas de DN	**zero**		**uma**		**duas**		**três**		**quatro**		**cinco**	
Balanceamento de réplicas	**antes**	**após**	**antes**	**após**	**antes**	**após**	**antes**	**após**	**antes**	**após**	**antes**	**após**
Tamanho dos arquivos: 10GB												
Tempo de balanceamento (s)	4866		980		648		674		1032		606	
Carga total no HDFS (GB)	303,72	304,10	302,36	303,37	302,34	305,19	303,43	306,37	303,95	304,16	301,03	302,38
DP da utilização (%) dos DNs	8,23	1,50	3,71	2,85	4,27	3,00	3,91	2,69	4,26	2,93	6,61	3,02
Tempo de execução (s)	459,25	343,44	436,59	354,63	391,80	356,43	330,75	300,52	340,48	312,67	369,56	343,89
Throughput médio (MB/s)	31,69	41,49	51,38	58,31	42,48	46,97	55,75	59,34	43,47	45,98	39,29	42,34
Taxa média de E/S (MB/s)	33,27	45,09	78,81	86,53	56,78	60,55	71,79	77,39	47,54	51,81	43,58	46,80
Tamanho dos arquivos: 15GB												
Tempo de balanceamento (s)	8235		2904		1679		1032		2074		649	
Carga total no HDFS (GB)	455,27	455,71	455,51	455,66	455,88	456,59	455,57	454,58	455,47	456,31	455,17	457,77
DP da utilização (%) dos DNs	8,21	0,96	4,21	1,51	3,45	1,81	3,11	2,30	2,29	0,51	2,15	1,09
Tempo de execução (s)	603,07	459,11	660,08	594,04	526,55	444,51	533,63	472,44	511,66	462,61	483,33	446,24
Throughput médio (MB/s)	35,20	48,34	36,82	40,87	42,13	44,81	38,40	47,11	40,85	44,34	39,03	41,70
Taxa média de E/S (MB/s)	37,37	52,54	44,84	49,02	46,99	52,07	43,47	54,45	45,81	49,47	40,25	42,48
Tamanho dos arquivos: 20GB												
Tempo de balanceamento (s)	4611		1920		751		3610		2245		3591	
Carga total no HDFS (GB)	606,21	606,42	606,33	609,07	606,67	607,19	605,89	608,47	605,49	609,74	605,69	606,21
DP da utilização (%) dos DNs	3,90	1,39	2,94	1,47	2,11	1,44	4,34	1,63	3,40	1,52	4,75	1,01
Tempo de execução (s)	628,84	564,10	669,09	613,83	682,86	621,55	652,21	589,17	639,41	561,94	776,25	620,37
Throughput médio (MB/s)	47,00	51,01	49,22	52,70	48,12	51,10	40,14	44,83	42,03	45,91	32,27	39,99
Taxa média de E/S (MB/s)	54,95	59,49	60,16	64,41	56,22	60,23	43,66	48,71	47,71	51,68	33,55	41,16

Após a execução do Balanceador, nota-se a redução do DP em torno do volume de dados mantido nos DNs, indicando um maior equilíbrio na distribuição das réplicas. O tempo necessário para o balanceamento mostrou-se substancialmente elevado em alguns casos (vide cenário com arquivos de 15GB e sem falhas). Isso se dá pela ocorrência de um desequilíbrio acentuado no posicionamento inicial dos blocos no *cluster* e, também, pela forma como o Balanceador verifica a conformidade com o *threshold*. De modo a calcular as diferenças de desempenho no HDFS em termos de tempo de execução, *throughput* e taxa de transferência, utiliza-se a variação percentual dada por $((T_b - T_a)/T_a * 100)$, onde T_a e T_b equivalem, respectivamente, aos resultados da métrica em análise antes e após o balanceamento (quando negativa, a porcentagem obtida representa uma redução).

Em todos os testes idealizados com este fator de replicação, observou-se ganhos de desempenho decorrentes do balanceamento. As melhorias na leitura dos dados, exibidas na Tabela 1 pela redução da média dos tempos de execução após o uso do Balanceador, são possibilitadas por uma exploração otimizada da localidade espacial dos dados durante a fase *map* das tarefas executadas pelo `TestDFSIO`. Considerando os cenários de zero a cinco falhas com os arquivos de 10GB, as reduções nos tempos de execução do *benchmark* foram de, respectivamente, 25,22%, 18,77%, 9,03%, 9,14%, 8,17% e 6,94%. Já, para os arquivos com tamanho de 15GB, os valores observados foram de 23,87%, 10%, 15,58%, 11,47%, 9,59% e 7,67%. Para a configuração de 20GB os ganhos de desempenho obtidos foram na escala de 10,29%, 8,26%, 8,98%, 9,66%, 12,12% e 20,08%, respectivamente.

Aumentos no *throughput* e na taxa média de E/S também foram alcançados. O

throughput é dado pela razão do volume total de dados processados (em MB) pela soma dos tempos (em segundos) gastos por cada tarefa (devido ao paralelismo, este valor é superior ao tempo de execução total da aplicação). A taxa de E/S, por sua vez, relaciona a velocidade de transferência média obtida por cada tarefa pela quantidade total de tarefas *map* executadas (para o `TestDFSIO` isso equivale ao número de aquivos manipulados em cada operação). Utilizando como exemplo as diferentes cargas nos cenários sem falhas e com uma única falha de DN, obteve-se aumentos no *throughput* de 30,91, 37,34% e 8,54% no cenário sem falhas e de 13,48%, 10,99% e 7,08% no cenário com uma falha. Já, na taxa de E/S, as melhorias após o balanceamento foram de 35,54%, 40,59% e 8,27% no cenário sem falhas e de 9,8%, 9,33% e 7,06% no cenário com uma falha.

7.2. Quatro Réplicas por Bloco

Os resultados exibidos na Tabela 2 consideram as operações de escrita e leitura de arquivos com um FR fixo em quatro. O primeiro ponto a ser ressaltado envolve os cenários com cinco falhas de DN, onde o Balanceador não precisou mover nenhum bloco (o tempo de balanceamento de 3 segundos exibido na Tabela refere-se à verificação de conformidade com o *threshold*). A razão disso reside na proximidade do FR e do número de DNs ativos. Como, de acordo com a política de posicionamento, um DN pode conter uma única réplica de um mesmo bloco, a medida que o FR (quatro) aproxima-se do número de DNs ativos após a introdução das falhas (cinco), o HDFS tende naturalmente ao balanceamento.

Tabela 2. Comportamento do HDFS antes e após o balanceamento com FR = 4.

Quantidade de falhas de DN	**zero**		**uma**		**duas**		**três**		**quatro**		**cinco**	
Balanceamento de réplicas	**antes**	**após**	**antes**	**após**	**antes**	**após**	**antes**	**após**	**antes**	**após**	**antes**	**após**
Tamanho dos arquivos: 10GB												
Tempo de balanceamento (s)	648		624		853		776		765		3	
Carga total no HDFS (GB)	403,37	405,82	403,56	405,08	405,07	408,53	405,43	404,20	401,67	403,16	403,16	403,16
DP da utilização (%) dos DNs	2,77	1,83	2,04	1,41	2,24	1,55	2,75	1,93	2,96	1,20	1,07	1,07
Tempo de execução (s)	381,71	354,95	401,03	315,39	316,58	284,01	338,80	290,14	316,49	283,02	395,45	387,36
Throughput médio (MB/s)	42,42	45,77	39,16	46,77	49,78	53,65	40,89	47,95	43,01	47,40	34,63	35,33
Taxa média de E/S (MB/s)	48,70	54,09	42,58	52,69	58,02	64,13	47,16	53,23	48,75	53,62	39,86	40,13
Tamanho dos arquivos: 15GB												
Tempo de balanceamento (s)	3916		1526		648		929		1651		3	
Carga total no HDFS (GB)	606,98	604,71	607,21	610,02	607,63	605,89	606,27	609,53	606,71	609,18	604,72	604,72
DP da utilização (%) dos DNs	3,24	1,41	2,10	1,25	1,75	1,10	2,25	1,73	1,40	0,31	0,23	0,23
Tempo de execução (s)	546,30	502,37	477,46	425,41	487,33	428,56	498,07	445,78	463,95	420,01	659,37	652,33
Throughput médio (MB/s)	40,76	45,66	48,19	51,26	42,74	45,74	38,88	44,50	41,84	45,96	30,35	31,57
Taxa média de E/S (MB/s)	47,44	51,16	55,55	58,87	49,64	52,91	41,15	48,06	44,73	48,52	33,18	34,27
Tamanho dos arquivos: 20GB												
Tempo de balanceamento (s)	3680		2706		1678		1294		1307		3	
Carga total no HDFS (GB)	807,48	807,13	807,79	806,38	807,94	809,20	808,70	806,93	808,53	810,48	808,63	808,63
DP da utilização (%) dos DNs	2,82	1,09	2,12	1,00	1,88	0,89	2,00	1,11	1,42	0,24	0,48	0,48
Tempo de execução (s)	703,71	603,11	635,26	570,80	720,25	579,68	626,69	556,71	717,63	629,38	1004,16	991,27
Throughput médio (MB/s)	42,70	46,93	46,31	50,61	38,40	48,01	41,85	46,43	35,58	40,03	28,11	29,31
Taxa média de E/S (MB/s)	47,83	52,07	53,87	62,39	41,97	53,96	45,25	48,86	37,76	42,24	33,13	34,33

Sendo assim, com esse fator de replicação, a análise dos resultados alcançados pelo balanceamento de réplicas no HDFS é necessária apenas sobre os cenários de zero a quatro falhas com as três configurações de tamanho dos arquivos. Em relação ao tempo de execução médio do `TestDFSIO` em modo leitura, as reduções obtidas (ganhos de desempenho proporcionados pelo balanceamento) considerando os arquivos de 10GB foram de, respectivamente, 7,01%, 21,35%, 10,29%, 14,36% e 10,57%. Para os arquivos de 15GB, os valores foram de 8,04%, 10,9%, 12,06%, 10,5% e 9,47%. Já, para a configuração de 20GB, os ganhos foram de 14,3%, 10,15%, 19,52%, 11,17% e 12,3%, respectivamente.

Para as métricas de *throughput* e taxa média de entrada e saída também notou-se aprimoramentos no HDFS com o balanceamento de réplicas. Por exemplo, considerando os cenários sem ocorrência de falhas e com uma única falha de DN com os diferentes tamanhos de arquivos, o *throughput* teve aumento na ordem de 7,91%, 12,02% e 9,92% no cenário sem falhas e de 19,43%, 6,37% e 9,27% no cenário com uma falha. Para a taxa de transferência, as melhorias foram de 11,07%, 7,84% e 8,86% no cenário sem falhas e de 23,75%, 5,97% e 15,82% no cenário com uma falha após a execução do Balanceador.

Embora os experimentos utilizem cargas que podem ser consideradas baixas para cenários de *big data*, enfatiza-se que, considerando a metodologia de análise, o incremento de tamanho dos arquivos se mostra relevante. Dessa forma, o desbalanceamento de réplicas evidenciado nos experimentos com 10GB, 15GB e 20GB propende-se para cenários com cargas maiores. Também nota-se que o comportamento do sistema não seguiu nenhuma tendência aparente entre os diferentes cenários analisados. Uma das possíveis causas para tal, reside em características próprias do HDFS que, projetado para ser capaz de executar sobre *clusters* com *hardware* comum, está sujeito a comportamentos inesperados mediante a ocorrência de falhas [Dinu and Ng 2012].

Adicionalmente, ao relacionar os resultados obtidos nos cenários de testes realizados com o fator de replicação 4 (Tabela 2), com os experimentos idealizados com o fator 3 (Tabela 1), percebe-se que, quando comparado ao desempenho alcançado em um mesmo cenário pelo *benchmark* ao utilizar o FR padrão do sistema, o incremento do fator não aprimorou a usabilidade geral do HDFS. Embora exista o aumento na disponibilidade dos dados (uma réplica a mais para cada bloco armazenado), a aplicação em uso não conseguiu, de maneira generalizada em todos cenários, se beneficiar da réplica sobressalente.

8. Considerações Finais

No HDFS os blocos armazenados são replicados entre diferentes nós do *cluster*. A escolha dos DNs para manter estas réplicas é essencial para otimizar a confiabilidade e o desempenho geral do sistema. Para um posicionamento eficiente dos blocos entre os nodos, diferentes parâmetros, tais como o fator de replicação, o número de DNs inativos e a carga armazenada no sistema, devem ser considerados. Visando analisar a influência destes fatores no comportamento do HDFS, este trabalho investigou de forma sistemática a capacidade da atual política de posicionamento em sustentar o balanceamento de réplicas.

De acordo com os resultados obtidos, pode-se observar que o desbalanceamento é inerente à política padrão do HDFS e que, ao utilizar uma ferramenta dedicada como o Balanceador, permitiu-se aprimorar significativamente o desempenho geral do sistema em todos os cenários de teste idealizados neste trabalho, atestando a importância do balanceamento de réplicas no HDFS. Sendo assim, espera-se que os experimentos conduzidos neste trabalho possam estimular melhorias nas estratégias de replicação e re-replicação de dados implementadas no ecossistema do Hadoop.

Trabalhos futuros envolvem o desenvolvimento de uma solução reativa voltada ao balanceamento de réplicas, baseada na criação de uma nova política de balanceamento para o `HDFS Balancer`. A abordagem proposta irá endereçar aspectos que, até então, não são considerados pelo Balanceador, como otimizar a localidade espacial dos dados e promover o balanceamento em nível de *rack*. Para isso, a nova política irá estender e flexibilizar a operação de balanceamento no HDFS, permitindo que diferentes métricas

de usabilidade do sistema e da disposição física dos nodos no *cluster* sejam consideradas durante a execução do Balanceador, possibilitando que a ferramenta seja configurada para atuar de forma customizada sobre o problema do desbalanceamento de réplicas no HDFS.

Referências

Achari, S. (2015). *Hadoop Essentials*. Packt Publishing Ltd, 1st edition.

Ciritoglu, H. E., Batista de Almeida, L., Cunha de Almeida, E., Buda, T. S., Murphy, J., and Thorpe, C. (2018). Investigation of replication factor for performance enhancement in the hadoop distributed file system. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 135–140. ACM.

Dharanipragada, J., Padala, S., Kammili, B., and Kumar, V. (2017). Tula: A disk latency aware balancing and block placement strategy for hadoop. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2853–2858. IEEE.

Dinu, F. and Ng, T. (2012). Understanding the effects and implications of compute node related failures in hadoop. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, pages 187–198. ACM.

Foundation, A. S. (2018). "HDFS Architecture". `hadoop.apache.org/docs/r2.9.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign`. Janeiro.

Hortonworks (2018). "HDFS Administration". `https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.6.5/bk_hdfs-administration/content/ch_balancing-in-hdfs.html`. Janeiro.

Ibrahim, I. A., Dai, W., and Bassiouni, M. (2016). Intelligent data placement mechanism for replicas distribution in cloud storage systems. In *IEEE International Conference on Smart Cloud (SmartCloud)*, pages 134–139. IEEE.

Liu, K., Xu, G., and Yuan, J. (2013). An improved hadoop data load balancing algorithm. *Journal of Networks*, 8(12):2816.

Patole, A., Kumar, S. M., Chandran, P., and Shabeera, T. (2015). Load-aware replica placement in multiuser hadoop environment using mst. In *2015 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, pages 376–381. IEEE.

Shah, A. and Padole, M. (2018). Load balancing through block rearrangement policy for hadoop heterogeneous cluster. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 230–236. IEEE.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The hadoop distributed file system. In *Symposium on Mass Storage Systems and Technologies*, pages 1–10. IEEE.

Turkington, G. (2013). *Hadoop Beginner's Guide*. Packt Publishing Ltd, 1st edition.

VishnuVardhan, C. B. and Baruah, P. K. (2016). Improving the performance of heterogeneous hadoop cluster. In *2016 4th International Conference on Parallel, Distributed and Grid Computing (PDGC)*, pages 225–230. IEEE.

White, T. (2015). *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 4th edition.