A Confiabilidade dos Racks e a Disponibilidade dos Dados como Métricas para o Balanceamento de Réplicas de um Cluster HDFS

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

[1]Pós-Graduação em Ciência da Computação (PGCC)
Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brasil

{rwfazul, pitthan}@inf.ufsm.br

Abstract. *Data replication is essential to achieve reliability on the HDFS, but it can intensify cluster imbalance. The HDFS Balancer is an Apache Hadoop daemon designed to perform replica balancing. The balancer, however, is not optimized to meet the potential demands of fault tolerance and data availability during data redistribution. This work presents a customization for the HDFS Balancer that evaluates the system's racks based on the failure rate of its DNs to determine which nodes should receive more or less data. In association, we used a priority that customizes the selection and redistribution of blocks during the balancing aiming at increasing the final availability of the data in the cluster.*

Resumo. *A replicação, embora vital para o bom funcionamento do HDFS, favorece o desbalanceamento do cluster. O HDFS Balancer é uma solução integrada ao Apache Hadoop voltada ao balanceamento de réplicas. O balanceador, entretanto, não é otimizado para atender possíveis demandas de tolerância a falhas e disponibilidade dos dados ao realizar a redistribuição das réplicas. Este trabalho apresenta uma customização para o HDFS Balancer que considera a propensão a falhas dos racks do sistema para definir quais nodos devem receber uma maior ou menor quantidade de dados. Em conjunto, utilizou-se uma prioridade que esforça-se em aumentar a disponibilidade geral dos dados armazenados no HDFS durante o reposicionamento das réplicas no cluster.*

1. Introdução

O *framework* Apache Hadoop[1] provê um sistema de arquivos para o armazenamento distribuído e confiável em *clusters* com *hardware* de baixo custo, o *Hadoop Distributed File System* (HDFS). Para garantir alta confiabilidade e disponibilidade no armazenamento de volumes massivos de dados, o HDFS implementa diversas técnicas de tolerância a falhas. A replicação de dados, amplamente utilizada em ambientes distribuídos, possui um papel fundamental para o bom funcionamento do sistema de arquivos do Hadoop.

O posicionamento das réplicas no *cluster* é essencial para a confiabilidade e disponibilidade no HDFS. Uma distribuição desequilibrada afeta a localidade dos dados e pode fazer com que os recursos computacionais deixem de ser explorados de forma otimizada [White 2015]. Uma das soluções existentes para alcançar o balanceamento de réplicas no

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

Hadoop é através de seu balanceador nativo, o `HDFS Balancer`: ferramenta que movimenta dados entre nodos até que a utilização de seus dispositivos de armazenamento fique dentro de um intervalo controlado. Entretanto, por atuar de forma generalizada, o `HDFS Balancer` não é otimizado para atender potenciais demandas de confiabilidade e disponibilidade dos dados durante a redistribuição das réplicas armazenadas no sistema.

Este trabalho apresenta uma estratégia de balanceamento customizada para o `HDFS Balancer` que considera métricas de confiabilidade dos *racks* e disponibilidade dos dados para realizar o equilíbrio das réplicas no *cluster*. Com isso, *racks* com baixa proporção de nodos falhos são priorizados para o recebimento de um maior volume de dados durante o balanceamento. Além disso, os blocos a serem redistribuídos pelo `HDFS Balancer` passam a ser selecionados com base em potenciais ganhos de disponibilidade após a transferência. Uma investigação experimental em um ambiente heterogêneo foi conduzida de forma a avaliar a efetividade da solução.

O trabalho está organizado como segue. A Seção 2 apresenta o *framework* Apache Hadoop e o HDFS, aprofundando-se no mecanismo de replicação de dados. A Seção 3 explora as causas e os problemas inerentes ao desbalanceamento de réplicas no HDFS. A Seção 4 é dedicada aos principais trabalhos relacionados. A Seção 5 detalha a solução de balanceamento de réplicas desenvolvida. A Seção 6 exibe e discute os resultados obtidos. Por fim, a Seção 7 aponta as considerações finais e norteia os trabalhos futuros.

2. Apache Hadoop

Já reconhecido pela indústria, o Apache Hadoop [Foundation 2019] é um *framework open source* voltado à análise e gerência de *big data*. O Hadoop foi projetado de modo a fornecer alta confiabilidade, disponibilidade e escalabilidade no processamento e armazenamento de dados em ambientes distribuídos. Dentre os principais componentes de seu ecossistema estão o MapReduce, o YARN e o HDFS [White 2015].

O modelo de programação paralelo MapReduce compõe a camada de processamento do Hadoop. Uma aplicação MapReduce é guiada por funções de mapeamento (*map*) e de redução (*reduce*), que baseiam-se em princípios de linguagens de programação funcional [Achari 2015]. As tarefas MapReduce são resilientes a falhas e rodam de forma isolada e simultânea em diferentes nodos de um *cluster* HDFS.

Já o YARN (*Yet Another Resource Negotiator*) é uma tecnologia adicionada na segunda versão do Hadoop como o elemento central da arquitetura. O YARN destina-se ao gerenciamento de recursos e ao escalonamento de tarefas, oferecendo uma plataforma de execução de propósito geral eficiente e flexível. Isto possibilita que diferentes motores de processamento, como o MapReduce, compartilhem uma camada comum de gerenciamento de recursos. Completando o ecossistema, o HDFS apresenta-se como a camada de armazenamento do Hadoop, além de ser incorporado por *frameworks* como Apache Spark, Storm e Flink e ser uma camada de armazenamento compatível com tecnologias como Apache Flume e HBase. O funcionamento do HDFS é detalhado na Seção 2.1.

2.1. Hadoop Distributed File System (HDFS)

O HDFS é um sistema de arquivos distribuído e flexível, projetado para armazenar volumes de dados massivos de forma confiável em ambientes compostos por *hardware*

de baixo custo (*commodity hardware*) [Shvachko et al. 2010]. Um *cluster* HDFS implementa uma arquitetura mestre-escravo a partir de um `NameNode` (NN) e múltiplos `DataNodes` (DN). O NN é o servidor mestre, responsável pela manutenção do *namespace* do sistema e pelo controle da distribuição e acesso dos arquivos. Já os DNs são os *workers* voltados ao armazenamento efetivo dos dados.

Ao inserir um arquivo no HDFS, este é fragmentado em blocos de dados de tamanho fixo (128MB, por padrão). O NN então encarrega-se de distribuir todos os blocos entre os DNs do *cluster*. Para atender altas demandas de confiabilidade e disponibilidade, evitando que dados sejam perdidos no advento de falhas de nodo, o HDFS necessita de mecanismos de detecção e recuperação mediante a falhas.

Com uma alta resiliência, permite-se que as aplicações, coordenadas pelo YARN, manipulem os dados armazenados no HDFS de forma segura e confiável. Os principais mecanismos de tolerância a falhas (TF) implementados pelo HDFS são: o estabelecimento de *checkpoints*, o modo de alta disponibilidade do NN (HDFS-HA), as mensagens *Heartbeat* e a replicação de dados [Foundation 2019]. Este trabalho, por sua vez, tem seu foco na replicação de dados, que é detalhada a seguir.

2.1.1. Replicação e Re-Replicação de Blocos

A replicação é o principal mecanismo de TF do HDFS, sendo utilizada para garantir uma alta confiabilidade e disponibilidade dos dados, além de auxiliar a suprir altas demandas de acesso. Após a segmentação do arquivo original a ser inserido no sistema, são geradas cópias dos blocos de dados, que são armazenadas em diferentes DNs do *cluster*. Com a redundância mantida no HDFS, o acesso aos dados pode ser feito a partir de múltiplos DNs, de forma que, no advento de falha de um nodo, nenhum dado seja perdido.

Uma aplicação pode especificar (por arquivo) a quantidade de réplicas a ser gerada para cada bloco através do Fator de Replicação (FR). O FR tem um valor padrão de três réplicas por bloco, o que permite manter a integridade dos dados mesmo na ocorrência de duas falhas de DNs em simultâneo. Para assegurar a disponibilidade dos dados em cenários onde ocorram falhas consecutivas, o NN deve monitorar ativamente o estado das réplicas de cada um dos blocos e, quando necessário, realizar a re-replicação.

As causas primárias que resultam na re-replicação são [Foundation 2019]: a corrupção de uma réplica armazenada no sistema, o incremento do FR de um arquivo ou a ocorrência de uma falha, que conduza um determinado DN do *cluster* a um estado inativo. Neste último caso, o HDFS faz uso das mensagens *Hearbeat* para identificar falhas operacionais em DNs. Periodicamente, os DNs enviam *Hearbeats* para o NN informando seu estado de funcionamento. Se o servidor mestre não receber mensagens de um nodo dentro de um período pré-definido, o NN marca este DN como inativo e dispara a re-replicação, restaurando a conformidade com o FR. Assim como a replicação, todo o processo de re-replicação é transparente ao cliente.

Além de controlar o número de réplicas ativas de cada bloco, o NN é responsável por definir o destino das réplicas ao distribuir os dados. Esta escolha deve assegurar alta TF e permitir que a disponibilidade proporcionada pela replicação seja explorada de forma otimizada. Para tal, o NN segue uma Política de Posicionamento de Réplicas

(PPR) [White 2015]. Considerando o FR padrão, a PPR armazena a primeira réplica no mesmo nodo do cliente (para clientes executando fora do *cluster*, um DN é escolhido de forma arbitrária). As duas réplicas seguintes são posicionadas em dois DNs distintos de um *rack* remoto diferente do *rack* da primeira réplica. Em caso de um FR elevado, as demais réplicas são armazenadas em DNs selecionados de forma aleatória, porém evitando armazenar muitas réplicas de um mesmo bloco em um único *rack* [Foundation 2019].

Para otimizar o processo de escrita das múltiplas cópias de um bloco no sistema de arquivos, o HDFS implementa uma técnica de *pipeline* de replicação. Com isso, os DNs selecionados para o recebimento das réplicas de um determinado bloco podem, ao mesmo tempo, receber e encaminhar dados para o próximo DN do *pipeline*. Adicionalmente, ao armazenar mais de uma réplica no mesmo *rack*, tende-se a reduzir o número de *switches* de rede pelos quais os dados precisam trafegar.

A redundância dos dados também é utilizada como forma de melhorar o desempenho. Durante operações de leitura, a largura de banda de múltiplos *racks* pode ser explorada para suprir altas demandas de acesso. Além disso, a PPR – seguindo uma estratégia de distribuição *rack-aware* – possibilita atender requisições a partir de réplicas locais, armazenadas nos DNs do *rack* mais próximo da origem da solicitação, o que tende a reduzir o tempo gasto com o tráfego de dados [Foundation 2019].

Embora a replicação baseada na PPR aumente a confiabilidade e a disponibilidade dos dados (blocos não são perdidos mesmo se um *rack* inteiro falhar) e contribua com uma melhor utilização dos recursos computacionais (operações de I/O podem tirar proveito da largura de banda de múltiplos *racks*), ela influencia o desequilíbrio do *cluster*. As principais causas que levam ao desbalanceamento de réplicas e os problemas de uma distribuição de dados desequilibrada são apresentados a seguir, na Seção 3.

3. Localidade dos Dados no HDFS

O HDFS foi projetado para fornecer alto *throughput* no atendimento a requisições baseadas no modelo de acesso *write once, read many* [Achari 2015]. Para tal, seguindo a premissa de que mover a computação é menos custoso do que mover os dados, o Hadoop tenta, ao máximo, executar as tarefas computacionais no local onde os blocos residem [Foundation 2019]. Esta funcionalidade, denominada localidade dos dados (*data locality*) [White 2015], permite aumentar o *throughput* médio do sistema e reduzir o tráfego de rede entre *switches*, já que o acesso aos dados passa a ser realizado localmente.

Motores de processamento distribuídos podem utilizar a localidade dos dados como estratégia para otimizar a execução de aplicações no *cluster*. Considerando o MapReduce, o escalonador tenta designar as tarefas de mapeamento para o nodo onde os dados de entrada estão mantidos [Turkington 2013]. Se todos os nodos que armazenem as réplicas requisitadas estiverem ocupados com outras tarefas, então um nodo livre, mais próximo de onde os dados residem, será buscado. Preferencialmente são selecionados nodos de um mesmo *rack*, porém se, devido a restrições de recursos, não existirem nodos livres em um mesmo *rack*, um nodo *off-rack* será selecionado, o que resultará na transferência de dados inter-*rack* e, possivelmente, entre *switches*.

Com a replicação, a possibilidade de existir nodos livres que permitam aproximar a computação dos dados é alta. Todavia, uma distribuição desequilibrada dos dados

através do *cluster* resulta no desbalanceamento de réplicas do sistema e tende a afetar a localidade dos dados no HDFS [White 2015], conforme aborda a Seção 3.1.

3.1. Desbalanceamento de Réplicas

No HDFS, as réplicas são posicionadas de forma a otimizar a TF e o desempenho do sistema. Porém, nem sempre é possível garantir que os dados sejam distribuídos de forma equilibrada entre os DNs. Um dos fatores que resulta no desbalanceamento de réplicas no HDFS é a adição de novos DNs ao *cluster*. Os DNs novos permanecem subutilizados por um período significativo, já que estes competem igualmente com os demais DNs do *cluster* para o recebimento dos dados [Turkington 2013].

A replicação por si só também contribui com o desbalanceamento do HDFS [Hortonworks 2019]. Seguindo a PPR, durante o armazenamento das réplicas de um mesmo bloco, a seleção dos DNs é realizada arbitrariamente pelo NN, sendo uma cópia mantida no nodo local, o que favorece o desbalanceamento inter-DN. Critérios que poderiam auxiliar a manter o *cluster* equilibrado, como considerar a proporção de dados armazenada nos DNs, não são empregados pela política padrão do HDFS [Shvachko et al. 2010]. Adicionalmente, de acordo com a PPR, um *rack* é selecionado para manter dois terços das réplicas de um bloco, o que tende ao desequilíbrio inter-*rack*.

A medida que o desbalanceamento se agrava, a localidade dos dados é afetada, resultando em uma maior quantidade de transferências intra-*rack* e *off-rack* durante a execução das tarefas de computação no sistema. Com isso, além de aumentar o consumo da largura de banda do *cluster*, há uma tendência a sobrecarregar DNs com alta utilização (nodos que armazenam uma maior quantidade de dados) e reduzir significativamente o desempenho de aplicações *I/O bound*. As principais abordagens para mitigar os problemas do desbalanceamento de réplicas no HDFS são apresentadas na Seção 4.

4. Trabalhos Relacionados

O balanceamento de réplicas no HDFS pode ser tratado de forma proativa ou reativa. Soluções proativas, tais como a de [Ibrahim et al. 2016], atuam no momento da distribuição inicial dos blocos e envolvem a criação de novas políticas de posicionamento de réplicas para o HDFS que, preventivamente, de forma direta ou indireta, contribuem com uma distribuição de dados equilibrada. Embora estas soluções esforcem-se em manter o HDFS em um estado balanceado, nem sempre é possível impedir o desequilíbrio na distribuição de réplicas. Em determinadas situações, como a adição de novos DNs ao *cluster*, o uso de soluções de rebalanceamento torna-se essencial para evitar períodos de subutilização de recursos computacionais. Este trabalho é voltado a este tipo de solução.

O balanceamento reativo no HDFS atua como uma estratégia corretiva que permite reposicionar os dados já armazenados, visando uma distribuição mais homogênea, seja entre *racks* ou entre DNs. Exemplos de soluções nesta categoria incluem [Liu et al. 2013], que apresentam um algoritmo aprimorado para o balanceamento de *racks* sobrecarregados com base em prioridade que esforça-se em reduzir as chances de falhas de nodo devido à sobrecarga. Para tal, é construída uma lista contendo os *racks* sobrecarregados no HDFS (prioritários) e outras duas listas, ordenadas de acordo com fatores estipulados pelos autores, contendo os *racks* que estão habilitados a receber os dados dos *racks* com sobrecarga durante o balanceamento. A execução ocorre de forma iterativa até que as listas fiquem vazias, o que implica em uma distribuição mais uniforme dos dados no *cluster*.

Em [Dharanipragada et al. 2017], por sua vez, é proposto um algoritmo modificado que, além da utilização dos DNs, considera variações na latência de escrita e leitura dos dispositivos de armazenamento dos nodos. Com isso, os DNs que apresentarem menor latência de disco recebem um número maior de blocos. Já [Shah and Padole 2018] focam em otimizar o processo de realocação das réplicas aproveitando-se da capacidade de processamento dos nodos em instâncias do Hadoop executando em ambientes heterogêneos. Neste sentido, os blocos são redistribuídos apenas para DNs específicos, que permitam reduzir o tempo gasto com a transferência dos dados.

O HDFS também disponibiliza uma solução destinada ao balanceamento reativo, o `HDFS Balancer` [Foundation 2019]. A partir de sua política de balanceamento, a ferramenta redistribui réplicas de nodos com alta utilização ($origem$) para nodos com baixa utilização ($destino$). Por ser a base para a solução apresentada neste trabalho, a Seção 4.1 detalha o funcionamento do balanceador nativo do HDFS.

4.1. HDFS Balancer

O `HDFS Balancer` é uma ferramenta, integrada na distribuição do Hadoop, responsável pela análise do posicionamento dos blocos e reposicionamento dos dados entre os dispositivos de armazenamento do HDFS [Shvachko et al. 2010]. A execução do balanceador é disparada sob demanda pelo administrador do *cluster*.

A ferramenta recebe como parâmetro um *threshold* (porcentagem de 0% a 100%, com valor padrão de 10%), que é utilizado para definir quando o *cluster* está balanceado. Um único DN pode conter múltiplos dispositivos de diferentes tipos (e.g. disco ou SSD). Considerando que $G_{i,t}$ representa o grupo de dispositivos do tipo t de um mesmo DN, o *threshold* limita a diferença máxima entre a utilização de um dado grupo ($U_{i,t}$) e a utilização média de todos os dispositivos do tipo t do *cluster* ($U_{\mu,t}$) [White 2015].

Uma operação de balanceamento pode ser separada em diferentes etapas, que são executadas iterativamente pela ferramenta [Hortonworks 2019]. Inicialmente, cada $G_{i,t}$ é classificado em uma de quatro categorias, sendo elas: (i) superutilizado, quando $U_{i,t} > U_{\mu,t} + threshold$; (ii) acima da média, quando $U_{\mu,t} + threshold \geq U_{i,t} > U_{\mu,t}$; (iii) abaixo da média, quando $U_{\mu,t} \geq U_{i,t} \geq U_{\mu,t} - threshold$; e (iv) subutilizado, quando $U_{\mu,t} - threshold > U_{i,t}$. A execução do balanceador encerra quando o *cluster* não possuir nenhum grupo classificado como superutilizado ou subutilizado.

Após a classificação, define-se, para cada grupo, o volume de dados a ser movimentado na iteração. Este valor é determinado por uma variável denominada *maxSize2Move* que, no caso de grupos $origem$ (superutilizados ou acima da média), equivale a quantidade de dados a ser transferida, enquanto nos grupos $destino$ (abaixo da média ou subutilizados), equivale a quantidade a ser recebida. Em seguida, inicia-se a etapa de pareamento dos grupos, onde formam-se pares $origem - destino$ entre grupos de dispositivos de um mesmo tipo t. O balanceador sempre inicia tentando formar pares entre nodos que residem no mesmo *rack*.

A seguir, tem início a etapa de agendamento das movimentações. Para tal, o balanceador define quais réplicas movimentar do grupo $origem$ para o $destino$, garantindo que o número de *racks* onde as réplicas do bloco em questão residem não seja reduzido, i.e., réplicas movimentadas devem continuar a respeitar a PPR. Por fim, as movimentações pré-definidas para a iteração são efetivadas. Quando aplicável, o balanceador emprega

uma estratégia para otimizar a transferência dos dados [Shvachko et al. 2010], de forma que o DN que possua uma das réplicas do bloco a ser movimentado e esteja mais próximo do $destino$ seja utilizado como *proxy*, reduzindo assim o tráfego inter-*rack*.

Se, após as movimentações serem concluídas, ainda existirem nodos superutilizados ou subutilizados, uma nova iteração de balanceamento é iniciada. Caso contrário, o *cluster* atingiu um estado balanceado, onde a utilização de todos os grupos está em conformidade com o *threshold*. Todavia, por atuar de forma generalizada, o balanceador nem sempre consegue endereçar potenciais demandas de uso durante a redistribuição dos dados, tais como requisitos elevados de confiabilidade e disponibilidade.

5. Política de Balanceamento de Réplicas Customizada

Para permitir que o `HDFS Balancer` considere as diferentes características topológicas do *cluster* e métricas do sistema de arquivos durante sua operação, elencou-se um conjunto de customizações para a política de balanceamento padrão da ferramenta. Estas modificações apresentam-se como otimizações e novas funcionalidades para o balanceador nativo do HDFS, formando uma política de balanceamento de réplicas customizada.

De modo a flexibilizar o uso do balanceador, a política customizada foi estruturadas na forma de um sistema de prioridades, que pode ser adaptado e configurado de acordo com as necessidades. As prioridades[2] foram agrupadas em categorias de acordo com seus objetivos de uso, conforme ilustra a Tabela 1.

Tabela 1. Sistema de prioridades definido pela política customizada.

Categoria	Prioridade	Objetivo de uso
Capacidade dos nodos	Capacidade de armazenamento Capacidade de processamento	Customizar o balanceamento em ambientes heterogêneos a partir de diferenças de *hardware* dos DNs.
Estados dos nodos	Utilização dos nodos Classificação dos nodos Carga dos nodos	Reduzir o custo de processamento e transferência de dados do balanceamento através de priorizações com base em métricas recuperadas em tempo de execução.
Estados dos *racks*	**Confiabilidade dos *racks*** Utilização dos *racks*	Permitir que o balanceamento seja conduzido considerando características dos *racks* que agrupam os DNs do *cluster*.
Distribuição dos dados	**Disponibilidade dos dados**	Priorizar movimentações de réplicas que permitam aumentar a disponibilidade final dos blocos armazenados.

Este trabalho destina-se à investigação do comportamento da política customizada considerando o uso simultâneo de duas prioridades: "confiabilidade dos *racks*" e "disponibilidade dos dados". Em trabalhos anteriores, as prioridades de confiabilidade e disponibilidade foram avaliadas de forma totalmente independente [Fazul et al. 2019a, Fazul et al. 2019b]. A associação destas prioridades, por sua vez, é algo inédito na política customizada e abre um conjunto de novas otimizações para o balanceamento de réplicas, apresentando-se assim como a contribuição principal do presente artigo.

5.1. Confiabilidade dos Racks

Esta prioridade avalia a confiança dos *racks* com base na suscetibilidade a falhas de seus DNs. Desta forma, antes de selecionar um *rack* para o recebimento dos blocos, realiza-se

[2]Em todas as prioridades há a garantia de que a variação máxima do volume de dados em cada DN após o balanceamento de réplicas continue sendo controlada pelo valor do *threshold*. Além disso, a disposição dos blocos após o balanceamento permanece respeitando a PPR padrão do HDFS.

uma verificação acerca da quantidade de DNs inativos no *cluster*, criando um esforço para armazenar uma maior quantidade de dados em *racks* com menor índice de falhas de DNs.

Para a implementação desta prioridade foi preciso tornar o balanceador consciente do estado dos DNs do *cluster*. Por padrão, o balanceador apenas considera DNs ativos, portanto foi necessário adicionar novos métodos responsáveis pela descoberta dos DNs inativos no sistema. Após, foi realizado o mapeamento do número de DNs ativos e inativos em cada *rack* do HDFS, calculando a taxa de falhas de seus DNs, i.e., proporção de DNs inativos para a quantidade total de DNs do *rack*.

Em seguida, foi calculado um fator de confiança para cada *rack* a partir do valor inverso de sua taxa de falhas. A normalização *min-max*[3] foi aplicada para manter os valores dentro de um intervalo controlado. Assim, torna-se o valor mínimo do conjunto em 0, o máximo em 1 e, qualquer outro valor, em um decimal proporcional entre 0 e 1.

Por fim, foi implementado um método destinado a atualizar a variável *maxSize2Move*, já utilizada pelo balanceador original, de acordo com o fator de confiança dos *racks*. Por padrão, esta variável equivale ao volume de dados (em *bytes*) necessário para levar a utilização de um determinado grupo de dispositivos ($U_{i,t}$) até a utilização média dos dispositivos de armazenamento tipo t do *cluster* ($U_{\mu,t}$). Com a alteração, o valor de *maxSize2Move* de grupos pertencentes a um *rack* com alto fator de confiança (próximo de 1) é incrementado, enquanto grupos de um *rack* com baixo fator de confiança (próximo de 0) têm o valor de *maxSize2Move* decrementado.

5.2. Disponibilidade dos Dados

Esta prioridade customiza a forma de seleção e redistribuição dos blocos durante o balanceamento, visando aumento na disponibilidade final dos dados. Embora a disponibilidade dos blocos de um arquivo continue dependendo de seu respectivo FR, esta prioridade faz com que os blocos transferidos fiquem dispostos na maior quantidade de *racks* possível, aumentando assim a confiabilidade do sistema em preservar os dados mesmo na ocorrência de falhas simultâneas de *rack*.

Para que o `HDFS Balancer` considere a disponibilidade dos dados durante a redistribuição dos blocos, foi necessário modificar o algoritmo responsável por selecionar os blocos a serem movimentados do grupo $origem$ para o grupo $destino$. Por padrão, o balanceador garante que o movimento do bloco não irá reduzir o número de *racks* em que suas réplicas estão armazenadas, ou seja, para que a transferência do grupo $origem$ para o $destino$ seja considerada válida, é necessário estar de acordo com a PPR.

Desta forma, antes de fazer o agendamento da movimentação, determina-se em quais *racks* as réplicas do bloco em análise estão atualmente mantidas e, verifica-se se a transferência da réplica do grupo $origem$ para o $destino$ irá proporcionar aumento de disponibilidade. Para isso, o *rack* do grupo $origem$ deve conter, no mínimo, duas réplicas do bloco (já que uma delas será redistribuída) e o *rack* do grupo $destino$ não pode conter nenhuma das réplicas (caso contrário, não haverá aumento de disponibilidade).

Se as condições forem satisfeitas, o bloco será considerado como um bom candidato para a movimentação, permitindo o aumento da disponibilidade dos dados no HDFS.

[3] A equação $T'_{Ri} = (T_{Ri} - min)/(max - min)$ realiza a normalização *min-max* sobre a taxa de falhas de um *rack* Ri (T_{Ri}) a fim de obter seu respectivo fator de confiança (T'_{Ri}).

Tomando como exemplo o FR padrão de três réplicas por bloco, a segunda e a terceira réplicas devem ser dispostas em dois *racks* remotos distintos do local da primeira réplica.

Além das condições anteriores, ambas as situações a seguir também fazem com que a movimentação da réplica seja considerada válida: (i) blocos que já estejam armazenados no maior número de *racks* possível em relação à quantidade de *racks* do *cluster*; (ii) blocos que já estejam com a disponibilidade máxima, onde cada uma de suas réplicas está disposta em um *rack* distinto.. Esta flexibilização é necessária pois, por mais que a prioridade de disponibilidade dos dados se esforce em selecionar réplicas que, ao serem movimentadas, fiquem dispostas em um maior número de *racks*, o `HDFS Balancer` deve continuar a operar em benefício do equilíbrio do *cluster*.

5.3. Associação entre as Prioridades

As prioridades apresentadas atuam em diferentes momentos do fluxo de execução do `HDFS Balancer`. A prioridade "confiabilidade dos *racks*" aplica as customizações durante a etapa de classificação dos grupos de dispositivos. Deste modo, a definição da categoria dos grupos é realizada em sequência ao mapeamento dos DNs ativos e inativos do *cluster*, bem como o cálculo do fator de confiança de cada *rack*. Isso permite que a definição do volume de dados a ser movimentado entre os grupos seja feita de acordo com a métrica de confiabilidade adotada pela prioridade.

Já a prioridade "disponibilidade dos dados" atua durante a etapa de agendamento das movimentações dos blocos, que inicia depois do pareamento dos grupos de dispositivos de armazenamento dos DNs. Após a priorização das transferências que posicionam as réplicas no maior número de *racks* possível (de acordo com o FR de seus blocos), as movimentações são efetivadas, dando fim à iteração de balanceamento.

A disposição das réplicas através do *cluster* afeta a confiabilidade e a disponibilidade dos dados no HDFS. O uso em conjunto das prioridades descritas neste trabalho mostra-se interessante pois permite endereçar demandas de uso e necessidades de diferentes classes de aplicações durante o balanceamento. Dentre as aplicações que podem se beneficiar do balanceamento customizado com as prioridades associadas estão aquelas voltadas à classificação e ao agrupamento de dados, as que possuem foco em indexação e rankeamento e as que fazem uso intensivo de dados e entrada/saída (E/S).

Com a prioridade "confiabilidade dos *racks*", permite-se que o `HDFS Balancer` diferencie e priorize os nodos para recebimento de uma maior ou menor quantidade de dados de acordo com o fator de confiança estimado para cada *rack* do *cluster* com base na taxa de falhas de seus DNs. Embora DNs inativos possam ser resultado tanto de falhas de *hardware* quanto de *software*, *racks* que possuírem uma proporção elevada entre DNs inativos em relação aos ativos podem estar passando por um período de sobrecarga.

A prioridade "disponibilidade dos dados", por sua vez, customiza a estratégia de seleção dos blocos a serem redistribuídos de modo a atender possíveis requisitos de confiabilidade ao aprimorar a TF do sistema em caso de falhas simultâneas de *rack*. Além disso, a disponibilidade extra pode ser usada como um meio de explorar a largura de banda de múltiplos *racks* de modo a promover otimização de desempenho. Os benefícios impulsionados pela associação entre as duas prioridades são investigados na Seção 6.

6. Experimentação

Para validar a utilização associada das prioridades e avaliar a efetividade da solução, investigou-se o comportamento do HDFS antes e após o balanceamento de réplicas com as customizações propostas. O experimento foi realizado na plataforma GRID'5000[4], com o Hadoop (versão 2.9.2) em modo de operação totalmente distribuído, executando sobre uma distribuição Debian 10 (*buster*). De forma a tornar o cenário de teste mais próximo de ambientes reais que executam o HDFS, o experimento foi executado sobre um ambiente heterogêneo, com diferenças de *hardware* entre os nodos computacionais.

Assim, foram configurados 18 nodos nos *clusters suno* e *uvb* – ambos pertencentes ao *site Sophia-Antipolis* – dispostos em 4 *racks* distintos (R_1 a R_4). Os nodos dos *racks* R_1 e R_2 (Dell PowerEdge R410) possuíam dois processadores Intel Xeon E5520 (Nehalem, 2.27GHz, 2 CPUs/node, 4 cores/CPU), 32GB de memória RAM, 598GB de capacidade de armazenamento HDD SAS Dell (Raid) e uma conexão *Ethernet* de 1 Gbps. Já os nodos dos *racks* R_3 e R_4 (Dell PowerEdge C6100) possuíam dois processadores Intel Xeon X5670 (Westmere, 2.93GHz, 2 CPUs/node, 6 cores/CPU), 96GB de memória RAM, 250GB de armazenamento HDD SATA Western Digital, uma conexão de rede *Ethernet* de 1 Gbps e uma conexão InfiniBand de 40 Gbps.

A carga dos dados foi realizada com o `TestDFSIO` [White 2015], um *benchmark* distribuído que testa o desempenho do HDFS através da execução de operações paralelas de E/S intensivas. Foram escritos 17 arquivos de 20GB, cada um com o FR padrão de 3 réplicas por bloco. Ao total, o volume de dados armazenado no sistema foi de 1018,09GB, que equivale a aproximadamente 8144 blocos de dados de 128MB cada.

Inicialmente os *racks* R_1, R_2, R_3 e R_4 possuíam, respectivamente, 3, 4, 5 e 6 DNs. Para que diferentes fatores de confiança fossem quantificados pela prioridade de confiabilidade dos *racks*, falhas de DN foram induzidas durante a escrita dos arquivos pelo *benchmark*. A introdução das falhas foi realizada pelo comando *kill* do Linux aos processos dos DNs selecionados de forma arbitrária. Ao total, foram induzidas 3 falhas de DN no *rack* R_4, 2 falhas no *rack* R_3 e 1 falha de DN no *rack* R_2. Desta forma, após as falhas, cada *rack* manteve exatamente 3 DNs ativos.

As falhas são identificadas pelo NN pela ausência de mensagens *Heartbeat* em um período pré-definido. Como o tempo para o NN marcar um DN como inativo é relativamente longo (por padrão, mais de 10 minutos), de forma a evitar re-replicações desnecessárias devido a instabilidades momentâneas de DNs, este período foi reduzido para aproximadamente 20 segundos. O tempo para a 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*.

Para estimar a confiabilidade dos *racks*, utiliza-se o cálculo detalhado na Seção 5.1, onde é aplicada a normalização *min-max* sobre a taxa de falhas de cada um dos *racks* do *cluster* HDFS (razão do número de DNs inativos pelo total de DNs do *rack*). Com isso, obteve-se, respectivamente para cada *rack*, os seguinte fatores de confiança, representados por T'_{Ri}: (i) T'_{R1} = 1,0; (ii) T'_{R2} = 0,5; (iii) T'_{R3} = 0,2; (iv) T'_{R4} = 0,0.

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

6.1. Resultados e Discussão

Após a escrita dos arquivos, o *cluster* ficou com uma utilização média ($U_{\mu,t}$) de 26,06% (1018,06GB ocupados de um total de 2,82TB). A Tabela 2 exibe o estado de utilização dos DNs antes e após a execução do balanceador com as prioridades de confiabilidade dos *racks* e disponibilidade dos dados, supondo um *threshold* de balanceamento de 5%.

Tabela 2. Estado do HDFS antes e após o balanceamento de réplicas.

Rack	DataNode	antes do balanceamento		após o balanceamento	
		$O_{i,DISK}$ (GB)	$U_{i,DISK}$ (%)	$O_{i,DISK}$ (GB)	$U_{i,DISK}$ (%)
R_1	DN_{01}	66,28	13,15	124,69	24,74
	DN_{02}	116,89	23,19	125,46	24,90
	DN_{03}	70,25	13,94	125,45	24,89
R_2	DN_{04}	212,08	42,08	139,29	27,64
	DN_{05}	30,74	6,10	111,81	22,19
	DN_{06}	30,99	6,15	108,30	21,49
R_3	DN_{07}	86,00	46,71	48,41	26,29
	DN_{08}	85,25	46,30	48,91	26,56
	DN_{09}	83,67	45,44	49,34	26,80
R_4	DN_{10}	72,58	39,42	38,94	21,15
	DN_{11}	83,59	45,40	48,29	26,23
	DN_{12}	79,76	43,32	49,23	26,74

Inicialmente, percebe-se uma discrepância no volume de dados armazenados em cada DN, o que é esperado devido à distribuição não homogênea feita pela PPR. Após a execução do balanceador, estando de acordo com o esperado, cada DN ficou com utilização dentro dos limites de balanceamento dados em função do *threshold*, i.e., 21,06% ($U_{\mu,t} + threshold$) e 31,06% ($U_{\mu,t} - threshold$). Ressalta-se que, devido à heterogeneidade do ambiente utilizado para os testes, os *racks* R_1 e R_2 possuem DNs com maior capacidade de armazenamento, logo, finalizam como uma maior ocupação para atingir uma utilização proporcional a dos DNs dos *racks* R_3 e R_4.

A ocupação do sistema pode ser analisada considerando a utilização total de cada um dos *racks* do *cluster* (proporção entre o volume de dados total acumulado pelos DNs do *rack* e a soma total das capacidades de armazenamento dos DNs). A Tabela 3 exibe, para cada *rack*, a ocupação em GB ($O_{Ri,DISK}$) e a utilização ($U_{Ri,DISK}$) com base na distribuição dos arquivos baseadas na PPR (antes do balanceamento) e depois da redistribuição das réplicas (após o balanceamento).

Tabela 3. Utilização dos *racks* antes e após o balanceamento de réplicas.

Rack	T'_{Ri}	antes do balanceamento		após o balanceamento	
		$O_{Ri,DISK}$ (GB)	$U_{Ri,DISK}$ (%)	$O_{Ri,DISK}$ (GB)	$U_{Ri,DISK}$ (%)
R_1	1,0	253,42	16,76	375,60	24,84
R_2	0,5	273,81	18,11	359,40	23,77
R_3	0,2	254,92	46,15	146,66	26,55
R_4	0,0	235,93	42,71	136,46	24,70

No momento anterior ao balanceamento, o fator de confiança (T'_{Ri}) não é determinante para os valores de ocupação ($O_{Ri,DISK}$) e utilização ($U_{Ri,DISK}$) dos *racks*. Desta

forma, *racks* com baixa confiabilidade podem vir a armazenar um alto volume de dados. O *rack* R_3, por exemplo, está com a maior utilização do *cluster*. Após a execução do balanceador com a prioridade de confiabilidade dos *racks*, percebe-se que o volume total armazenado em cada *rack* passou a respeitar os respectivos fatores de confiança. Assim, os *racks* com maior T'_{Ri} (R_1 e R_2), ou seja, menor incidência de falhas de DNs, ficaram responsáveis por manter uma maior quantidade de dados. Analogamente, os *racks* com menor T'_{Ri} (R_3 e R_4) tiveram uma redução no volume de dados armazenado.

Ao total, 2036 blocos foram movimentados pelo balanceador, dos quais 1527 atingiram total disponibilidade após a realocação, ou seja, ficaram dispostos no maior número de *racks* distintos dado o FR. A representatividade destas movimentações foi de 75%, correspondendo a porcentagem de movimentações que permitiram aumentar o número de *racks* onde as réplicas estavam dispostas, otimizando assim a disponibilidade dos blocos.

De forma a analisar a otimização de desempenho proporcionada pelo aumento da disponibilidade, foram realizadas 15 execuções distintas do *benchmark* `TestDFSIO` voltadas à leitura dos dados armazenados no HDFS. A Figura 1 apresenta os tempos de leitura dos dados em cada uma das 15 execuções. Com o posicionamento inicial dos blocos seguindo a PPR (antes do balanceamento), a média dos tempos foi de $1187{,}57$ segundos. Após o balanceamento, este valor foi reduzido para $988{,}51$ segundos. Considerando a variação percentual dada por $((T_b - T_a) \,/\, T_a \times 100)$, onde T_a e T_b equivalem, respectivamente, às médias aritméticas dos tempos das 15 execuções do *benchmark* antes e após o balanceamento de réplicas, a variação alcançada foi de $-16{,}76\%$, indicando a redução obtida no tempo de leitura dos dados.

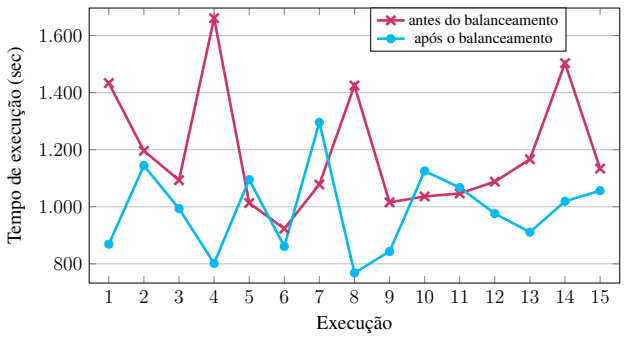

Figura 1. Tempos para leitura dos dados antes e após o balanceamento.

Com as melhorias na localidade dos dados impulsionadas pelo balanceamento de réplicas, outras métricas de desempenho do HDFS podem ser otimizadas. A Figura 2 exibe o *throughput* obtido durante a leitura dos dados antes e após o balanceamento. Para o `TestDFSIO`, 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 total de execução) [Shvachko et al. 2010]. O *throughput* médio foi de $28{,}77$MB/s antes do balanceamento e de $36{,}67$MB/s após o uso do balanceador. A variação percentual alcançada foi de $27{,}44\%$, indicando aumento no *throughput* da aplicação após o balanceamento com a associação das prioridades.

A Figura 3 exibe as taxas de E/S alcançadas pelo *benchmark*. Para o `TestDFSIO`, esta taxa relaciona a velocidade de transferência média obtida por cada tarefa pela quantidade total de tarefas *map* executadas [Shvachko et al. 2010]. A quantidade de tarefas

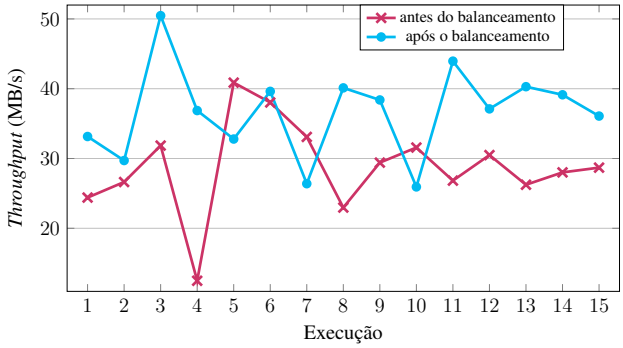

Figura 2. Throughput médio antes e após o balanceamento.

de mapeamento geradas equivale, por padrão, à quantidade de arquivos manipulados (17 arquivos, neste experimento). A taxa média de E/S foi de 37,13MB/s antes do balanceamento e de 45,34MB/s após o balanceamento com as prioridades de confiabilidade e disponibilidade. A variação percentual, considerando os valores médios das execuções, foi de 22,11%, indicando aumento na taxa de transferência.

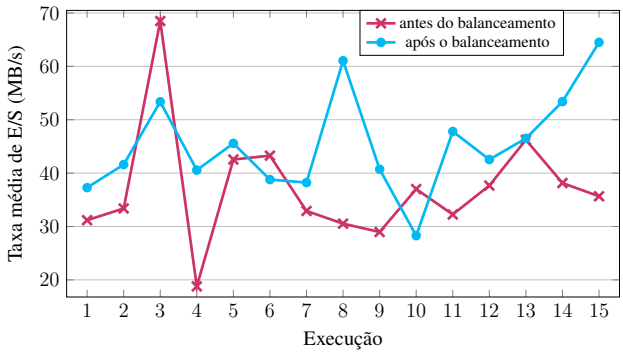

Figura 3. Taxa média de E/S antes e após o balanceamento.

As otimizações providas pelas prioridades de confiabilidade dos *racks* e disponibilidade dos dados, entretanto, tendem a aumentar o custo (tempo e consumo de banda) necessário para o balanceamento. Ao total 279,59GB de dados foram movimentados em 31 iterações de balanceamento que demandaram aproximadamente 3 horas e 27 minutos para serem completadas. Cabe ao administrador do *cluster* avaliar o *trade-off* entre o ganho de disponibilidade e confiabilidade e o *overhead* adicional na operação de balanceamento dado o alto número de transferências de dados inter-*rack*.

7. Considerações Finais

Este trabalho apresentou uma estratégia de balanceamento de réplicas customizada para o `HDFS Balancer` baseada em métricas de confiabilidade dos *racks* e de disponibilidade dos dados. A solução proposta visa flexibilizar a operação do balanceador nativo do HDFS, de forma que a taxa de falha dos *racks* seja utilizada para definição do volume de dados mantido por cada nodo após o balanceamento e que as réplicas para redistribuição sejam selecionadas visando aumentar a disponibilidade final dos dados armazenados.

Após detalhar o funcionamento das prioridades, foram conduzidos experimentos voltados a validar e avaliar a efetividade de implementação em um ambiente heterogêneo.

Os resultados demostram que a priorização do balanceamento do HDFS com as prioridades propostas otimiza a localidade dos dados, aprimorando o desempenho geral do sistema de arquivos do Hadoop no atendimento de operações de entrada e saída.

Trabalhos futuros envolvem a execução de outros *benchmarks* considerando cenários de falhas mais próximos de ambientes reais, de forma a avaliar métricas de desempenho que podem ser otimizadas ao utilizar o `HDFS Balancer` com a associação das prioridades apresentadas. Pretende-se também estender o cálculo do fator de confiança dos *racks* para permitir a configuração do período no qual o histórico de falhas de nodos deve ser considerado. Planeja-se implementar este comportamento através do monitoramento dos *racks* do *cluster* com o Apache ZooKeeper, um serviço para coordenação e configuração de aplicações distribuídas. Com isso, o fator de confiança será definido por cálculos sobre o histórico de falhas mantido em uma fila circular de *zNodes*.

Referências

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

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

Fazul, R. W. A., Cardoso, P. V., and Barcelos, P. P. (2019a). Improving data availability in hdfs through replica balancing. In *2019 9th Latin-American Symposium on Dependable Computing (LADC)*, pages 1–6. IEEE.

Fazul, R. W. A., Cardoso, P. V., and Barcelos, P. P. (2019b). O balanceamento de réplicas em um cluster hdfs com base na confiabilidade dos racks. In *Anais do Simpósio Brasileiro de Engenharia de Sistemas Computacionais (SBESC 2019)*, pages 31–38. SBC.

Foundation, A. S. (2019). "HDFS Architecture". `hadoop.apache.org/docs/r2.9.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign`. Novembro.

Hortonworks (2019). "Balancing data across an HDFS cluster". `https://docs.cloudera.com/HDPDocuments/HDP3/HDP-3.1.4/data-storage/content/balancing_data_across_hdfs_cluster.html`. Dezembro.

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, New York. IEEE.

Liu, K., Xu, G., and Yuan, J. (2013). An improved hadoop data load balancing algorithm. *Journal of Networks*, 8(12):2816–2822.

Shah, A. and Padole, M. (2018). Load balancing through block rearrangement policy for hadoop heterogeneous cluster. In *2018 Int. Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 230–236, Bangalore. 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, 1 edition.

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