Otimização da Redistribuição de Réplicas no HDFS com base na Carga dos Nodos

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

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

{rwfazul, pitthan}@inf.ufsm.br

Abstract. *The HDFS Balancer is the native solution for balancing the distribution of the block replicas in the Hadoop Distributed File System. However, the balancer is dependent on manual triggering and its operating policy does not consider the overload of the computational environment during the balancing. This work presents a strategy that automates the decision-making process for the configuration and execution of the HDFS Balancer based on active system monitoring. Besides, the tool starts to operate aiming at a minimum balance while attempting to reduce the overhead in other applications running in the cluster by prioritizing underloaded nodes for the redistribution of the replicas.*

Resumo. *O HDFS Balancer é a solução nativa para o balanceamento de réplicas no sistema de arquivos distribuído do Hadoop. Entretanto, o balanceador possui limitações de uso, uma vez que depende do disparo manual e sua política de balanceamento não considera o estado de sobrecarga do ambiente computacional. Este trabalho apresenta uma estratégia que automatiza a tomada de decisão para a configuração e execução do HDFS Balancer com base no monitoramento ativo do sistema. Em complemento, implementou-se uma customização para que a ferramenta passe a operar visando um balanceamento mínimo, enquanto esforça-se em reduzir a sobrecarga nas demais aplicações no cluster ao priorizar nodos com baixa carga para a redistribuição das réplicas.*

1. Introdução

Conforme a capacidade de produção de dados aumenta, a demanda por sistemas computacionais capazes de manter e manipular volumes massivos de dados de forma confiável e eficiente torna-se igualmente maior. Nesse contexto, sistemas distribuídos, como *clusters* e *grids*, são amplamente utilizados, uma vez que permitem distribuir os dados e as tarefas computacionais em vários nodos. Diversas ferramentas e *frameworks* tiram proveito das arquiteturas desses sistemas a fim de fornecer plataformas dedicadas ao processamento de alto desempenho e ao armazenamento eficiente de *big data*.

O *Hadoop Distributed File System* (HDFS) é um sistema de arquivos distribuído e escalável voltado ao armazenamento confiável de grandes volumes de dados. Além de ser a camada de armazenamento do Apache Hadoop [Foundation 2020a], o HDFS é incorporado por diferentes *frameworks* de processamento paralelo, tais como Apache Spark, Samza e Storm, também sendo compatível com tecnologias como Apache HBase e Phoenix [White 2015]. O HDFS segue uma arquitetura mestre-escravo, na qual o servidor

mestre, conhecido como `NameNode` (NN), é responsável por manter o *namespace* do sistema, além de gerenciar o acesso e a distribuição dos arquivos. Já os *workers*, denominados `DataNodes` (DNs), são os nodos dedicados ao armazenamento dos dados.

Para assegurar alta confiabilidade, disponibilidade e integridade dos dados, mesmo quando executa em *clusters* com *hardware* de baixo custo (i.e., não confiável) [Achari 2015], o HDFS dispõe de diferentes mecanismos de tolerância a falhas (TF). O principal mecanismo de TF – e a base do modelo de armazenamento do HDFS – é a replicação de dados. Quando um arquivo é inserido no sistema, ele é segmentado em blocos de dados de tamanho fixo (por padrão 128MB), que são replicados e distribuídos para o armazenamento em DNs distintos, evitando a perda de dados no advento de falhas.

A replicação, embora essencial para o bom funcionamento e desempenho do HDFS, pode contribuir com o desbalanceamento na distribuição dos dados entre os nodos. A medida que o desequilíbrio de réplicas se intensifica no *cluster*, a localidade dos dados é afetada e os recursos computacionais podem deixar de ser explorados de forma otimizada [Turkington 2013]. O `HDFS Balancer` [Shvachko et al. 2010] é a solução nativa para o balanceamento de réplicas no HDFS. Mesmo sendo otimizado para a redistribuição de dados, a ferramenta possui limitações em sua operação, uma vez que sua política de balanceamento não considera o estado de sobrecarga do ambiente computacional para decisões acerca da transferência de dados entre os DNs. Além disso, a configuração dos atributos de balanceamento e a escolha do melhor momento para execução do `HDFS Balancer` ficam a critério do administrador do sistema.

Este trabalho apresenta uma solução focada em automatizar a configuração e o disparo do balanceamento de réplicas no HDFS com base na carga computacional dos nodos do *cluster*. Em complemento, implementou-se uma customização para o `HDFS Balancer` que altera o comportamento padrão da ferramenta, fazendo com que ela priorize nodos com baixo tráfego de comunicação durante a redistribuição dos blocos de dados. Uma investigação experimental foi conduzida de forma a avaliar a efetividade da estratégia desenvolvida em reduzir o *overhead* causado pela operação de balanceamento.

O trabalho está organizado em 6 seções. A Seção 2 é dedicada à fundamentação teórica envolvendo a replicação e o balanceamento de dados no HDFS. A Seção 3 elenca os principais trabalhos relacionados, com foco no `HDFS Balancer`. A Seção 4 detalha a solução de balanceamento proposta. A Seção 5 exibe e discute os resultados obtidos na experimentação. Ao final, a Seção 6 conclui o artigo e direciona os trabalhos futuros.

2. Replicação de dados no HDFS

Uma forma comum de garantir alta confiabilidade e disponibilidade de dados em sistemas distribuídos é através da replicação, em que a redundância dos dados é mantida no sistema, de modo que, na ocorrência de falhas, ao menos uma das cópias dos dados seja preservada. Para implementar este mecanismo, o HDFS baseia-se em um Fator de Replicação (FR), que determina o número de réplicas a serem geradas para cada bloco. Um fator de n garante que nenhum dado será perdido mesmo se $n-1$ DNs falhem simultaneamente. O FR é definido por arquivo e possui um valor padrão de três [White 2015].

É responsabilidade do NN monitorar o número de réplicas disponíveis e não corrompidas de cada bloco armazenado no sistema de arquivos, assegurando que o FR especificado seja respeitado. Sendo assim, quando o NN detectar, através da ausência de

mensagens *heartbeat* [Foundation 2020a], uma falha operacional em algum dos DNs do *cluster*, ele deve disparar a operação de re-replicação dos blocos necessários. Com isso, mesmo em cenários com ocorrência de múltiplas falhas em um curto período, é possível manter a confiabilidade e a disponibilidade dos dados no HDFS.

A escolha dos DNs para o armazenamento das réplicas, originadas tanto da replicação inicial quanto da re-replicação, é essencial para a confiabilidade e desempenho do HDFS. A otimização do posicionamento das réplicas no *cluster* é, inclusive, uma característica que distingue o HDFS de outros sistemas de arquivos [Foundation 2020a]. Para possibilitar tal otimização, segue-se uma Política de Posicionamento de Réplicas (PPR), que visa, de acordo com a arquitetura do *cluster*, aprimorar a confiabilidade do sistema e a disponibilidade dos dados, bem como a utilização da largura de banda da rede. Considerando o FR padrão, a PPR armazena a primeira réplica no nodo do cliente[1] e as duas réplicas seguintes em outros dois DNs em um *rack* remoto distinto do *rack* da primeira réplica. As demais réplicas são armazenadas em DNs escolhidos de forma aleatória, evitando manter muitas réplicas no mesmo *rack* [White 2015].

A estratégia implementada pela PPR assegura alta confiabilidade, uma vez que, mesmo se um *rack* inteiro falhar, nenhum bloco de dados será perdido. Além disso, com seu modelo de distribuição *rack-aware*, a PPR torna possível identificar o *rack* mais próximo do cliente que possui a réplica requisitada. Com isso, as réplicas locais – ou mais próximas da origem da solicitação – podem ter preferência em relação às réplicas remotas [Foundation 2020a], o que tende a diminuir o tempo gasto com transferências de dados. A Seção 2.1 detalha como a localidade espacial dos blocos é explorada pelo sistema de arquivos para suprir altas demandas de acesso e, nesse contexto, como uma distribuição de dados desbalanceada através do *cluster* pode afetar o funcionamento do HDFS.

2.1. Localidade dos Dados

Seguindo o modelo de acesso WORM (*write once*, *read many*) [Achari 2015], o HDFS visa fornecer um alto *throughput* de leitura durante operações sobre os dados armazenados. Uma forma de minimizar o congestionamento de rede e maximar o *throughput* médio do sistema é explorar a *localidade dos dados* [White 2015], que consiste no processo de mover a computação para o mais perto possível de onde os dados estão mantidos, ao invés de mover os dados para onde a aplicação está executando.

Além de fornecer as interfaces para enviar a aplicação até os nodos onde os dados residem, o Hadoop esforça-se em executar as tarefas computacionais no mesmo nodo que armazena as réplicas necessárias para a operação [Foundation 2020a]. Isso é conhecido como *otimização da localidade dos dados* [White 2015] e é a razão do bom desempenho do HDFS. Tendo em vista que cada bloco é replicado, por padrão, em três DNs distintos, as chances de que uma tarefa computacional consiga processar a maior parte dos dados localmente[2] é elevada [Achari 2015]. Contudo, para assegurar que o sistema de arquivos tire proveito da localidade espacial dos dados do melhor modo possível, as réplicas devem ser posicionadas de forma otimizada através do *cluster*.

[1]Para clientes executando fora do *cluster*, o DN da primeira réplica é escolhido de forma arbitrária, embora o sistema evite escolher nodos que estejam muito ocupados [White 2015].

[2]Caso não seja possível executar a tarefa junto com os dados de entrada, buscam-se os nodos com o caminho de rede mais rápido para o local onde os dados estão armazenados (usualmente no mesmo *rack*).

Uma distribuição de réplicas desequilibrada afeta a localidade dos dados, fazendo com que os recursos computacionais deixem de ser explorados de forma eficiente. Considerando DNs subutilizados, ou seja, com um baixo volume de dados em seus dispositivos de armazenamento, as tarefas computacionais movidas para esses nodos possivelmente terão que acessar réplicas remotas não locais, aumentando o consumo da largura de banda do *cluster*. Por outro lado, quando um DN passa a armazenar um volume de dados muito elevado, ele tende a deixar de receber novas réplicas, o que reduz o paralelismo de leitura [Foundation 2020a] e a capacidade em explorar a localidade temporal das réplicas armazenadas em seu nodo (conjunto de trabalho potencialmente desatualizado). Em ambos os casos, um maior número de transferências intra-*rack* ou, até mesmo, *off-rack* tornam-se necessárias durante a execução das aplicações no *cluster*.

O desbalanceamento no HDFS pode ocorrer por diferentes motivos, dentre eles o comportamento da PPR padrão, uma vez que: (i) para possibilitar um alto *throughput* de escrita caso execute diretamente em um DN do *cluster*, a PPR faz com que esse armazene sempre uma das réplicas localmente, o que pode torná-lo superutilizado; (ii) considerando o FR padrão, a PPR seleciona um *rack* para manter dois terços das réplicas de um determinado bloco, tendendo ao desequilíbrio inter-*rack*; e (iii) a PPR escolhe DNs de forma arbitrária para o recebimento das réplicas, tanto no processo de replicação inicial quanto na re-replicação, favorecendo o desbalanceamento inter-DN. Outra situação que causa o desbalanceamento é a adição de novos DNs ao sistema, já que esses irão competir igualmente com os outros DNs ativos para o recebimento das réplicas, resultando em um período de subutilização significativo dos recursos computacionais [Turkington 2013].

Para mitigar os problemas inerentes ao desequilíbrio na distribuição dos blocos de dados entre os nodos no *cluster*, o uso de soluções de balanceamento torna-se necessário. As principais soluções voltadas ao equilíbrio de réplicas no HDFS que alinham-se ao contexto deste trabalho, são apresentadas na Seção 3.

3. Trabalhos Relacionados

O balanceamento de réplicas no HDFS pode ser conduzido de forma proativa ou reativa. Abordagens proativas contribuem para que o sistema seja mantido em um estado balanceado, agindo no momento da distribuição inicial dos blocos. Todavia, em certas situações, como a ocorrência de falhas e a adição de novos nodos no sistema, nem sempre é possível impedir o desequilíbrio na distribuição dos dados. Neste sentido, as soluções reativas para rebalanceamento de réplicas permitem redistribuir os blocos já armazenados no sistema de arquivos através de ações corretivas. Este trabalho é voltado a abordagens reativas.

Exemplos de abordagens reativas incluem [Liu et al. 2013], que propõem um algoritmo aprimorado para o balanceamento entre *racks* com base em prioridade. A estratégia contribui com uma distribuição mais uniforme dos dados entre os *racks* visando reduzir as chances de falha devido à sobrecarga. Em [Dharanipragada et al. 2017] é introduzido um módulo de balanceamento (*LatencyBalancer*) que considera variações na latência de escrita e leitura dos discos de armazenamento dos nodos para a realocação dos dados no HDFS. Assim, os DNs que apresentarem menor latência de disco no *cluster* ficam propensos ao recebimento de um maior número de blocos

A estratégia apresentada em [Shah and Padole 2018], por sua vez, otimiza o processo de redistribuição das réplicas aproveitando-se da capacidade de processamento dos

nodos para reduzir o tempo gasto com a transferência dos dados. Baseando-se na capacidade de computação dos DNs, os blocos são redistribuídos apenas para DNs específicos, determinados a partir da heterogeneidade e desempenho dos nodos. Outra solução possível para o balanceamento reativo de réplicas, integrada na distribuição do Hadoop, é o `HDFS Balancer`. Por ser a base para o desenvolvimento deste trabalho, a Seção 3.1 detalha o funcionamento do `HDFS Balancer`.

3.1. HDFS Balancer

A solução oficial para balanceamento de réplicas entre os dispositivos de armazenamento no HDFS é o `HDFS Balancer` [Shvachko et al. 2010]: uma *daemon* que opera redistribuindo réplicas de nodos que possuírem um alto volume de dados para nodos com baixa utilização. A execução da ferramenta deve ser disparada pelo administrador do sistema[3] que, além de configurações adicionais, deve definir o *threshold* de balanceamento.

Para o *cluster* ser considerado balanceado, a utilização de todos os DNs deve estar em concordância com o *threshold* (porcentagem entre 0% e 100%). Tomando $G_{i,t}$ como o grupo de dispositivos do tipo t (e.g., disco ou SSD) do DN i, o *threshold* limita a diferença máxima entre a utilização de um $G_{i,t}$ ($U_{i,t}$) e a utilização média do *cluster* ($U_{\mu,t}$), considerando os dispositivos do tipo t. Reduzir o valor de *threshold* garante um maior equilíbrio na distribuição das réplicas entre os DNs, porém demanda mais transferências de dados, consumindo a largura de banda do *cluster* e aumentando o tempo de execução necessário para efetivar o processo de balanceamento no sistema de arquivos. O *threshold* padrão adotado pelo `HDFS Balancer` é de 10% [Foundation 2020a].

A Figura 1 [Foundation 2020a] ilustra uma visão alto nível do processo de balanceamento no HDFS. Todas as decisões referentes ao balanceamento são realizadas pelo *rebalancing server* (balanceador), que executa iterativamente. Após solicitar ao NN as informações acerca da utilização dos DNs do *cluster* (passo 1), o balanceador classifica cada $G_{i,t}$ em [Hortonworks 2019]: (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 $U_{\mu,t} \geq U_{i,t} \geq U_{\mu,t} - threshold$; (iv) ***subutilizado***, quando $U_{\mu,t} - threshold > U_{i,t}$. Para cada grupo, calcula-se a variável ***maxSize2Move***, que equivale ao volume de dados (em *bytes*) necessário para levar sua $U_{i,t}$ até a $U_{\mu,t}$. Ressalta-se que uma iteração de balanceamento tem o objetivo de fazer com que a distribuição das réplicas nos DNs superutilizados e subutilizados fique menos desequilibrada (aproxime-se de $U_{\mu,t}$), ao invés de reduzir o número de nodos desbalanceados propriamente dito.

Cada $G_{i,t}$ superutilizado (origem) é pareado com um ou mais $G_{i,t}$ subutilizados (destino). Se algum grupo superutilizado possuir ***maxSize2Move*** satisfeito, ele não será pareado na iteração corrente. Para os grupos superutilizados remanescentes, são selecionados candidatos nos $G_{i,t}$ classificados como abaixo da média. Se ainda houver algum $G_{i,t}$ subutilizado, procura-se candidatos entre os $G_{i,t}$ acima da média restantes. Em seguida, o balanceador solicita ao NN o mapeamento dos blocos existentes na origem (passo 2) e seleciona as réplicas a serem movimentadas para o grupo destino. Há a garantia de que as réplicas redistribuídas continuem em concordância com a PPR.

Como estratégia para diminuir o tráfego de dados necessário para a transferência

[3] Ainda que a arquitetura do HDFS seja compatível com rebalanceamento automático de acordo com um *threshold*, tal funcionalidade não é implementada de forma nativa [Foundation 2020a].

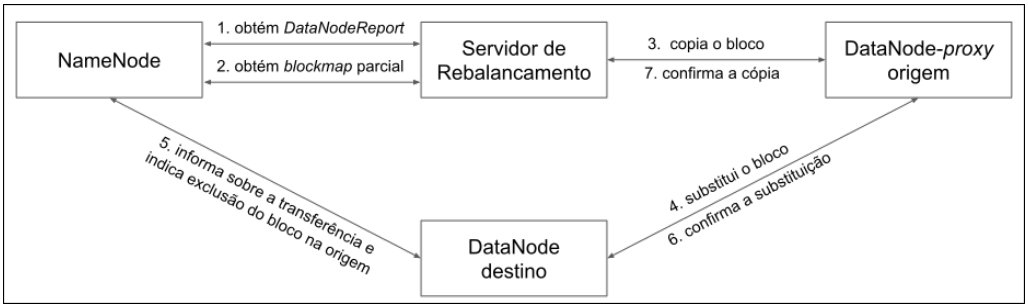

Figura 1. Balanceamento de réplicas no HDFS (adaptado de [Foundation 2020a]).

dos blocos entre nodos de diferentes *racks*, o DN mais próximo do destino e que possuir uma réplica do bloco a ser movimentado é utilizado como um *proxy*. Assim, o balanceador solicita que o *proxy* copie o bloco para o destino (passo 3), que então solicita ao DN destinto que substitua o bloco mantido na origem para seu armazenamento local (passo 4). Após a cópia do bloco, o destino envia um alerta para o NN (passo 5), que então, por padrão, dispara a operação de exclusão da réplica armazenada na origem. A seguir, o destino confirma a substituição do bloco para o *proxy* (passo 6), que a repassa para o balanceador (passo 7). Se, ao final da redistribuição de todas as réplicas dos blocos agendados para movimentação na iteração, o *cluster* ainda possuir grupos superutilizados ou subutilizados, uma nova iteração de balanceamento é iniciada.

4. Solução de Balanceamento Proposta

Mesmo com suas estratégias para otimizar a redistribuição das réplicas, o balanceador nativo do HDFS apresenta limitações de uso. Considerando o momento pré-operacional, o `HDFS Balancer` está sujeito a uma condição de configuração manual e disparo conforme a demanda, exigindo que, para uma tomada de decisão eficiente, o administrador do *cluster* possua um conhecimento apurado do comportamento do sistema e suas aplicações. Já durante sua execução, o balanceador opera de uma forma generalizada, desconsiderando parâmetros como o tráfego de comunicação dos DNs e o volume de dados a ser transferido. Assim, embora projetada para operar em segundo plano sem afetar os demais clientes e aplicações [White 2015], a operação de balanceamento pode apresentar um alto custo em termos de processamento de dados e de consumo de largura de banda.

De modo a endereçar tais problemas, implementou-se uma solução de balanceamento de réplicas que otimiza o uso do `HDFS Balancer`. A Figura 2 apresenta a estrutura da solução proposta neste trabalho, ilustrando a interação entre os módulos de monitoramento – $agentes$, $gerente$ e $controlador$ – e o `HDFS Balancer`. A seguir, a Seção 4.1 descreve a estratégia adotada para automatizar a configuração e a execução do balanceador. A Seção 4.2, por sua vez, detalha as customizações empregadas ao `HDFS Balancer` de modo a tornar a ferramenta consciente do estado dos DNs e evitar que *overheads* inapropriados sejam causados pela operação de balanceamento no *cluster*.

4.1. Automatização do uso do HDFS Balancer

Visando identificar cenários favoráveis ao balanceamento de réplicas e, assim, automatizar o processo de tomada de decisão na configuração e disparo do `HDFS Balancer`, empregou-se uma estratégia baseada no monitoramento ativo do ambiente computacional. Seguindo o modelo agente-servidor, cada nodo do *cluster*, que também executa um

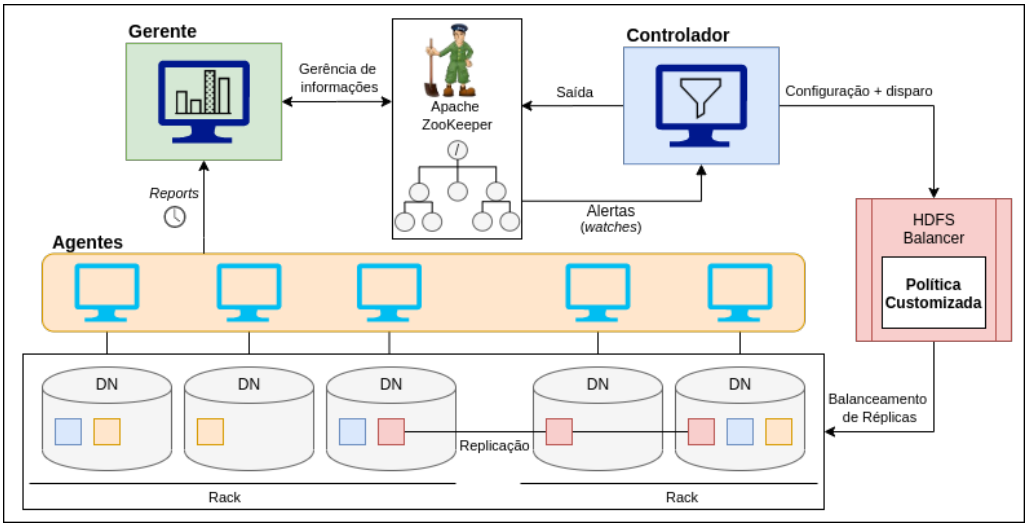

Figura 2. Estrutura da solução de balanceamento de réplicas proposta.

processo DN do HDFS, passa a ser monitorado por um $agente$, que é responsável por coletar, em tempo real, informações dos dispositivos de armazenamento e da instância HDFS em sua máquina. Periodicamente, os $agentes$ reportam as observações para um servidor $gerente$, que, tendo uma visão global do estado do sistema, é responsável por analisar os *reports* e tomar as decisões envolvendo o balanceamento de réplicas.

Para o controle interno dos módulos de monitoramento e para a manutenção de configurações utilizou-se o *framework* Apache ZooKeeper [Foundation 2020b]. O ZooKeeper – projeto *open source* que fornece serviços de coordenação confiável para ambientes distribuídos – possui um *namespace* compartilhado, organizado hierarquicamente em uma árvore de registradores de dados, denominados $znodes$ [Junqueira and Reed 2013]. Os $znodes$ mantêm os *reports* dos $agentes$, as definições realizadas pelo $gerente$ e o resultado da operação do `HDFS Balancer` no *cluster* recebido pelo módulo $controlador$.

De forma a determinar se uma ação corretiva se faz necessária no sistema de arquivos, o $gerente$ analisa as informações de ambiente mantidas na árvore de $znodes$ e calcula a divergência entre a utilização dos dispositivos de armazenamento ($U_{i,t}$) e a média de utilização do *cluster* ($U_{\mu,t}$). Se, considerando o *threshold* base de 10%, existirem DNs superutilizados ou subutilizados, inicia-se um procedimento para a definição dos atributos de balanceamento (posteriormente mapeados como parâmetros de execução do balanceador). Essa definição é inspirada nas configurações recomendadas em [Hortonworks 2019] para os modos de balanceamento *background*, *default* e *fast*, conforme exibe a Tabela 1.

Com a solução proposta neste trabalho, a escolha do modo e do *threshold* de balanceamento a serem utilizados pelo `HDFS Balancer` são feitas pelo $gerente$ com base na carga de processamento dos DNs. Para definir a carga de um DN, recupera-se a estimativa da quantidade de *threads* em execução em seu nodo e calcula-se a média de carga do *cluster*. Tal estimativa é dada pelo atributo *xCeiverCount* e é utilizada pelo Hadoop como uma forma de avaliar o tráfego de comunicação dos nodos para fins de balanceamento de computação. Na sequência, seguindo a estratégia empregada pela PPR para rejeitar a escrita em nodos que excedam o limite de conexões abertas, verifica-se se um DN i qualquer está em um estado ocupado através da equação $xCeiverCount_i >$

Tabela 1. Configurações de balanceamento (adaptado de [Hortonworks 2019]).

Configuração	*background*	*default*	*fast*
Largura de banda máxima disponível para consumo por DataNode durante a operação de balanceamento	1MB	1MB	10GB
Máximo de movimentações de blocos simultâneas por DataNode durante o balanceamento	4 × quantidade de discos	5	4 × quantidade de discos
Número total de *threads* (i.e., movimentações simultâneas) no balanceamento de todo *cluster*	1000	1000	20000
Volume de dados máximo para movimentação entre dois grupos em uma iteração de balanceamento	1GB	10GB	100GB
Tamanho de bloco mínimo permitido para movimentação do grupo origem para o grupo destino	10MB	10MB	100MB

$xCeiverCount_{\mu} \times$ *fator de carga*. O fator de carga é uma propriedade configurável com valor padrão de 2 [Foundation 2020a]. Sendo assim, os DNs com tráfego de comunicação maior que duas vezes a média do *cluster* são considerados ocupados.

Após verificar a quantidade de DNs ocupados no sistema, o $gerente$ adota as seguintes definições: (i) seleciona-se o modo *fast* se até um terço dos DNs do *cluster* estiverem em um estado ocupado e, aproveitando-se da maior velocidade desse modo para proporcionar um maior equilíbrio na distribuição das réplicas, utiliza-se o valor reduzido de 5% para o *threshold*; (ii) seleciona-se o modo *default* e o *threshold* de balanceamento padrão de 10%, se até dois terços dos DNs do *cluster* estiverem ocupados; e (iii) caso mais de dois terços dos DNs estejam ocupados, seleciona-se o modo *background*, também com o *threshold* padrão. Em sequência, conforme ilustrado na Figura 2, o $gerente$ notifica o $controlador$, que é o módulo responsável pela interação com o `HDFS Balancer`.

A comunicação entre os módulos $gerente$ e $controlador$ é possibilitada pelo mecanismo de $watches$ do ZooKeeper [Haloi 2015]. O $controlador$ possui um $watch$ no $znode$ que mantém as informações do $gerente$ e, com isso, é notificado sempre que o conteúdo ou a sub-árvore desse $znode$ forem alterados. Ao receber a notificação, o $controlador$ dispara a *daemon* do `HDFS Balancer` passando os parâmetros de execução definidos previamente pelo $gerente$. Quando o processo de balanceamento finalizar, o $controlador$ coleta a saída da operação, que também é armazenada pelo ZooKeeper e, posteriormente, pode ser utilizada pelo $gerente$ para a definição de atributos de balanceamento otimizados. Com a estratégia de coordenação ativa relatada nesta seção, remove-se a condição de configuração manual e disparo sob demanda do `HDFS Balancer`.

4.2. Customização com base no Estado dos Nodos

De forma a reduzir o *overhead* causado pela operação de balanceamento foi implementada uma customização para o `HDFS Balancer`, representada por "Política Customizada" na Figura 2. Essa customização complementa a estratégia para a configuração automática relatada na Seção 4.1 e faz com que o balanceador passe a considerar o estado dos DNs durante a redistribuição das réplicas. Para tal, métricas obtidas em tempo de execução são utilizadas para a priorização dos nodos, fazendo com que a movimentação de dados ocorra principalmente entre DNs com baixo tráfego de comunicação. Além disso, o `HDFS Balancer` começa a operar visando um equilíbrio mínimo, reduzindo o número de transferências de dados necessárias para balancear os dados armazenados no sistema.

A Figura 3 exibe, em alto nível de abstração, o algoritmo customizado para o cálculo da variável *maxSize2Move* em função da carga estimada dos nodos. Primeira-

mente, calcula-se *bytes2Avg*, que representa a quantidade de *bytes* necessária para levar a utilização de um grupo ($U_{i,t}$) até a utilização média do *cluster* ($U_{\mu,t}$) (L. 3). Em seguida, calcula-se *minSize2Move*, que equivale ao volume de dados mínimo necessário para que a utilização de um grupo fique de acordo com o *threshold* de balanceamento (L. 4). Para os $G_{i,t}$ classificados como acima ou abaixo da média, esse valor será zero, pois os grupos já estão em concordância com o *threshold*. Já para os $G_{i,t}$ superutilizados e subutilizados, *minSize2Move* equivale, respectivamente, ao volume de dados (em *bytes*) necessário para levar a $U_{i,t}$ até os limites superior ($U_{\mu,t}$ + *threshold*) e inferior ($U_{\mu,t}$ − *threshold*).

```
 1: procedure CALCMAXSIZE2MOVEBASEDONNODELOAD
 2:   utilizationDiff ← U_{i,t} − U_{μ,t}
 3:   bytes2Avg ← |utilizationDiff| × capacity / 100
 4:   minSize2Move ← calcMinSize2Move(capacity, utilization, average)
 5:   key ← r.getDatanodeInfo().getDatanodeUuid()
 6:   loadBasedBytes ← (bytes2Avg − minSize2Move) × (1 − loadMap.get(key))
 7:   maxSize2Move ← minSize2Move + loadBasedBytes
 8:   if thresholdDiff < 0 then                          ▷ verifica se é o G_{i,t} destino
 9:     maxSize2Move ← min(remaining, maxSize2Move)      ▷ valida o espaço disponível
10:   end if
11:   return min(maxSize2Move, max)                     ▷ max por padrão = 10GB
12: end procedure
```

Figura 3. Definição da variável *maxSize2Move* com base na carga dos nodos.

A diferença entre *bytes2Avg* e *minSize2Move* é então ponderada com base no valor mantido em um *loadMap*, resultando na variável *loadBasedBytes* (L. 6). Esse *loadMap* registra, para cada DN, um valor que representa sua carga (C'_i), estabelecido pela normalização *min-max*[4] sobre o atributo *xCeiverCount* do DN (C_i). A equação $C'_i = (C_{i,t} - min) \;/\; (max - min)$ demonstra a normalização. Ao fim, *maxSize2Move* é calculada a partir de *loadBasedBytes*. Note que é necessário acrescentar o valor de *minSize2Move* à *loadBasedBytes* (L. 7), garantindo que a $U_{i,t}$ de grupos superutilizados e subutilizados seja levada, ao mínimo, até os limites para a conformidade com o *threshold*.

A partir destas modificações, o `HDFS Balancer` opera visando o balanceamento mínimo, diminuindo o tempo e o número de transferências necessárias para o equilíbrio de réplicas no sistema de arquivos. O balanceador também passa considerar a carga dos nodos, fazendo com que a redistribuição de dados ocorra prioritariamente entre nodos com baixo tráfego de comunicação. Assim, cria-se o esforço em impedir que os DNs que já estejam enfrentando períodos de sobrecarga elevada devido ao alto número de conexões abertas (i.e., nodos ocupados) sejam impactados ainda mais, evitando prejudicar a execução de outras tarefas no *cluster* durante o processo de balanceamento de réplicas.

5. Experimentos

Visando avaliar a efetividade da solução de balanceamento proposta neste trabalho, realizou-se um experimento na plataforma GRID'5000[5]. O ambiente de testes foi configurado com 10 nodos no *cluster grisou* do *site Nancy*, onde cada nodo (Dell PowerEdge

[4]A normalização *min-max* torna o valor mínimo de um conjunto em 0, o valor máximo em 1 e qualquer outro valor em um decimal proporcional entre 0 e 1, mantendo os valores dentro de um intervalo controlado.

[5]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`)

R630) executou uma distribuição Debian GNU/Linux 10 (*buster*) e possuía as seguintes configurações: 2 CPUs Intel Xeon E5-2630 v3 (Haswell, 2.40GHz, 8 cores/CPU), 128GB de memória RAM, 558GB de capacidade de armazenamento HDD (SCSI Seagate) e uma conexão Ethernet de 1Gbps mais quatro conexões de 10Gbps.

O Apache Hadoop (versão 2.9.2) foi configurado para operação em modo distribuído com 1 NN e 10 DNs (um por nodo). Seguindo a estrutura apresentada na Figura 2, cada nodo executou um módulo $agente$ para o monitoramento e a coleta de informações. Uma instância replicada do Apache ZooKeeper (versão 3.6.1) foi utilizada para a comunicação entre os módulos $gerente$ e $controlador$. Para a carga de dados no sistema, utilizou-se o `TestDFSIO` [White 2015] (versão 1.8): um *benchmark I/O bound* distribuído que mede o desempenho do HDFS através de tarefas voltadas para E/S intensiva. Foram escritos 30 arquivos de 30GB cada, com um FR padrão de 3 réplicas por bloco, o que totalizou um volume de dados de aproximadamente $2{,}64$TB (21.600 réplicas de 128MB). Após a escrita, a utilização média do *cluster* ($U_{\mu,DISK}$) ficou em $57{,}01\%$.

A Tabela 2 exibe o estado do *cluster* em relação à porcentagem de utilização dos DNs ($U_{i,DISK}$) em três cenários: (i) PPR inicial, com a distribuição dos dados baseada na PPR do HDFS (sem balanceamento); (ii) Balanceador padrão, considerando o momento anterior e posterior ao disparo do `HDFS Balancer` com as configurações padrão (modo $default$ da Tabela 1); e (iii) Solução proposta, considerando o momento anterior e posterior ao emprego da estratégia apresentada na Seção 4 para a automatização e customização do balanceamento de réplicas no HDFS com base na carga dos nodos (C'_i)[6].

Tabela 2. Estado de ocupação e utilização do HDFS nos três cenários.

Cenário		**PPR inicial**	**Balanceador padrão**		**Solução proposta**	
DN	$\mathbf{C'_i}$	s/ bal. $U_{i,DISK}$ (%)	s/ bal. $U_{i,DISK}$ (%)	c/ bal. $U_{i,DISK}$ (%)	s/ bal. $U_{i,DISK}$ (%)	c/ bal. $U_{i,DISK}$ (%)
DN_{01}	1,00	41,81	38,97	57,20	42,90	47,01
DN_{02}	0,60	40,27	39,51	53,59	41,48	48,24
DN_{03}	0,20	76,06	59,70	48,87	75,52	59,95
DN_{04}	0,00	40,37	42,83	48,53	71,28	62,28
DN_{05}	0,00	49,54	66,24	48,32	49,20	47,41
DN_{06}	0,00	68,46	40,27	47,01	72,47	63,27
DN_{07}	0,20	40,60	41,11	47,28	40,76	52,03
DN_{08}	1,00	78,84	71,13	62,99	63,59	63,59
DN_{09}	0,60	41,08	72,57	63,14	43,72	47,45
DN_{10}	0,40	63,28	68,05	64,04	39,63	49,81
	σ	15,94	14,66	7,13	14,85	7,24

Com a PPR inicial, há uma grande discrepância no volume de dados mantido entre os DNs, que é evidenciada pelo desvio padrão (σ) elevado da utilização dos DNs (em %). Conforme descrito na Seção 2.1, a razão disso reside na própria política de posicionamento, que não garante uma distribuição realmente equilibrada das réplicas. Já no cenário com o uso balanceador padrão, a utilização de cada DN se manteve dentro dos limites inferior ($U_{\mu,DISK} - threshold$) e superior ($U_{\mu,DISK} + threshold$) de balanceamento,

[6]A coluna C'_i na Tabela 2 exibe os valores obtidos pela normalização *min-max* sobre o fluxo de comunicação dos nodos do *cluster*, que é determinado pela variável *xCeiverCount*.

ou seja, $47{,}01\%$ ($57{,}01\% - 10\%$) e $67{,}01\%$ ($57{,}01\% + 10\%$), respectivamente. Todavia, percebe-se como a carga dos nodos não foi considerada como critério para a redistribuição das réplicas. Com isso, DNs com alto tráfego de comunicação em seus nodos (i.e., com C_i' igual ou próxima a $1{,}0$), tais como DN_{01} e DN_{08}, tiveram de despender tempo com processamento e transferência de dados durante o balanceamento. Ao total, $210{,}12$GB de dados foram movimentados em 12 iterações de balanceamento, que demandaram 2 horas e 40 minutos para serem concluídas. Vale ressaltar que, com a abordagem padrão, a execução do `HDFS Balancer` ficou dependente do disparo manual pelo administrador.

No cenário com o emprego da solução proposta, por sua vez, todas as decisões acerca do equilíbrio de réplicas no *cluster* são feitas pela estrutura de monitoramento apresentada na Seção 4. Além de automatizar a configuração e o disparo do balanceador, a customização implementada faz com que o `HDFS Balancer` priorize a movimentação dos dados de acordo o estado dos nodos, minimizando a sobrecarga em DNs que já estejam com um alto tráfego de comunicação em seus nodos. Esse comportamento pode ser observado na Tabela 2, onde o DN_{01} (subutilizado) recebeu apenas a quantidade de blocos necessária para levar sua utilização até o limite inferior para ser considerado como abaixo da média. Já o DN_{08}, por estar dentro do *threshold* e ter C_i' em $1{,}0$, não foi pareado em nenhuma iteração de balanceamento (i.e., não enviou nem recebeu dados).

Assim, por operar visando o balanceamento mínimo do sistema e evitar consumo desnecessário da largura de banda do *cluster*, a solução proposta permitiu reduzir para 2 horas e 9 minutos o tempo de execução do `HDFS Balancer`. Ao total, 170GB de dados foram movimentados em 9 iterações. Sendo assim, conforme evidenciado pelos desvios padrões da utilização dos DNs obtidos nos três cenários da Tabela 2, a estratégia para a redistribuição de réplicas detalhada ao decorrer deste trabalho é capaz de manter o estado de equilíbrio do *cluster* em um nível similar ao do `HDFS Balancer` padrão ao mesmo tempo que evita sobrecarga adicional causada pelo processo de balanceamento.

Visando investigar possíveis melhorias de desempenho em aplicações de E/S possibilitadas pelo equilíbrio das réplicas em um momento pós-balanceamento, foram consideradas 20 execuções do `TestDFSIO` em modo leitura nos três cenários analisados. A média aritmética dos tempos de leitura foi de $1158{,}77$s no cenário com a PPR inicial, $1097{,}59$s com o balanceador padrão e de $1029{,}11$s com o emprego da solução proposta. Considerando o cenário com a PPR inicial em relação ao cenário com a solução proposta, a variação percentual obtida foi de $-11{,}19\%$, que representa a redução no tempo necessário para a leitura dos dados com a estratégia apresentada neste trabalho. Comparando ao cenário baseado no `HDFS Balancer` padrão, por sua vez, a variação percentual foi de $-6{,}24\%$. Sendo assim, além de reduzir o tempo e a largura de banda necessários para o balanceamento, o emprego da solução proposta neste experimento permitiu uma exploração otimizada da localidade espacial dos dados armazenados no HDFS.

6. Considerações Finais

Este trabalho apresentou uma estratégia de balanceamento de réplicas para o HDFS baseada no monitoramento ativo do ambiente computacional. A solução proposta foi desenvolvida visando à automatização da configuração e do disparo do balanceador nativo do sistema de arquivos do Hadoop, o `HDFS Balancer`. Incorporou-se também uma customização na política de operação da ferramenta que, além de redistribuir a quantidade

de blocos mínima necessária para o equilíbrio dos dados, esforça-se em evitar sobrecarga nas demais tarefas em execução no *cluster* devido ao processo de balanceamento.

Uma avaliação experimental foi conduzida para validar e avaliar a efetividade da implementação frente à política de posicionamento de réplicas base do HDFS e ao balanceador com suas configurações padrão. Os resultados obtidos demostram que a estratégia apresentada é capaz de manter o equilíbrio na distribuição dos blocos enquanto considera o estado dos nodos. Para tal, o volume de dados mínimo necessário para o balanceamento do *cluster* passa a ser movimentado, prioritariamente, entre nodos com baixo tráfego de comunicação. Desse modo, foi possível remover a condição de disparo manual do `HDFS Balancer` e explorar, de forma otimizada, a localidade dos dados no sistema.

Trabalhos futuros envolvem uma validação aprofundada da estratégia em cenários com ocorrência de falhas de nodos e com aplicações *CPU bound* em execução no *cluster* durante o balanceamento, bem como considerando variações no fator de replicação e no volume de dados armazenado. Além disso, pretende-se incorporar à solução proposta um módulo voltado para a gerência e análise das informações históricas atualmente mantidas no Apache ZooKeeper, assim permitindo a definição de atributos de balanceamento otimizados com base em estimativas sobre o estado do sistema de arquivos.

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 *2017 IEEE International Conference on Big Data (Big Data)*, pages 2853–2858. IEEE.

Foundation, A. S. (2020a). "HDFS Architecture". `hadoop.apache.org/docs/r3.3.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html`. Maio.

Foundation, A. S. (2020b). "ZooKeeper". `https://zookeeper.apache.org/doc/r3.6.1/zookeeperOver.html`. Maio.

Haloi, S. (2015). *Apache Zookeeper Essentials*. Packt Publishing Ltd, 1st edition.

Hortonworks (2019). "Scaling Namespaces and Optimizing Data Storage". `docs.cloudera.com/HDPDocuments/HDP3/HDP-3.1.5/data-storage/content/balancing_data_across_hdfs_cluster.html`. Junho.

Junqueira, F. and Reed, B. (2013). *ZooKeeper: Distributed Process Coordination*. O'Reilly Media, Inc., 1st edition.

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, 1st edition.

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