Política Customizada de Balanceamento de Réplicas para o `HDFS Balancer` do Apache Hadoop

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. *Data replication is a fundamental mechanism of the Hadoop Distributed File System (HDFS). However, the way data is spread across the cluster directly affects the replication balancing. The HDFS Balancer is a Hadoop integrated tool which can balance the storage load on each machine by moving data between nodes, although its operation does not address the specific needs of applications while performing block rearrangement. This paper proposes a customized balancing policy for HDFS Balancer based on a system of priorities, which can be adapted and configured according to usage demands. The priorities define whether HDFS parameters, or whether cluster topology should be considered during the operation, thus making the balancing more flexible.*

Resumo. *A replicação de dados é a base para o funcionamento do sistema de arquivos distribuído do Apache Hadoop (HDFS). Todavia, a forma como os dados são posicionados no cluster impacta diretamente o balanceamento de réplicas. O HDFS Balancer é uma solução integrada ao Hadoop que visa o equilíbrio do sistema, porém sua operação não permite explorar as necessidades específicas das aplicações. Este trabalho propõe uma política de balanceamento customizada para o HDFS Balancer baseada em um sistema de prioridades, que pode ser adaptado e configurado de acordo com as demandas de uso. As prioridades permitem definir quais parâmetros do HDFS ou da topologia do cluster devem ser considerados, assim flexibilizando o balanceamento.*

1. Introdução

O framework Apache Hadoop[1] desempenha um importante papel no armazenamento e no processamento de dados em larga escala. Um de seus principais componentes é o HDFS (*Hadoop Distributed File System*), um sistema de arquivos confiável e tolerante a falhas voltado a ambientes distribuídos. Para atender requisitos de desempenho e disponibilidade, o HDFS implementa o mecanismo de replicação de dados. Embora indispensável para a tolerância a falhas, a replicação afeta diretamente o balanceamento do *cluster*.

Quando os nodos armazenam quantidades desproporcionais de dados, o HDFS pode deixar de explorar os recursos computacionais de forma otimizada, prejudicando o desempenho e demais aspectos de funcionamento do sistema. O `HDFS Balancer` é uma das abordagens criadas para endereçar este problema, todavia seu modo de operação não

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

permite que características e propriedades específicas dos componentes do *cluster* sejam levadas em consideração durante o balanceamento, limitando a flexibilidade do processo.

Este trabalho apresenta uma política customizada de balanceamento de réplicas para o HDFS que baseia-se em diversas propriedades do sistema e da topologia do *cluster*. De modo a atender às necessidades de diferentes cenários de uso, definiu-se um sistema de prioridades configurável para a escolha dos nodos envolvidos no balanceamento. Os parâmetros utilizados permitem flexibilizar a operação do `HDFS Balancer`, além de possibilitar otimizações voltadas à execução do Hadoop em ambientes heterogêneos.

O artigo está organizado em seis seções. A Seção 2 apresenta os principais componentes do Apache Hadoop. A Seção 3 é dedicada ao sistema de arquivos HDFS, além de detalhar o processo de replicação de dados e as causas do desbalanceamento de réplicas. Na Seção 4 são apresentados os principais trabalhos relacionados. A Seção 5 descreve a política de balanceamento proposta. A Seção 6 aponta as considerações finais.

2. Apache Hadoop

O Apache Hadoop é uma plataforma *open-source* voltada à manipulação de *big data*. Por oferecer um eficiente sistema de gerência de dados e aplicações em arquiteturas de alto desempenho, tais como *clusters* e *grids*, o Hadoop é amplamente utilizado por empresas e pesquisadores. Sendo um *framework* flexível, capaz de oferecer alta disponibilidade em ambientes distribuídos, o Hadoop possui um ecossistema variado, composto por diversos componentes e ferramentas. Dentre seus pilares estão os utilitários presentes no `Hadoop Common`, o gerenciador de recursos YARN, o modelo de programação paralelo MapReduce e o sistema de arquivos distribuído HDFS [White 2015].

O módulo `Common` define bibliotecas para diversas funcionalidades da ferramenta, podendo ser utilizado em conjunto com outros projetos e *frameworks* da Apache como o Spark, HBase, Zookeeper e Ambari. O YARN (*Yet Another Resource Negotiator*) é uma ferramenta focada no gerenciamento de recursos e na alocação de tarefas. Implementado a partir da segunda versão do Hadoop, a ideia central do YARN é baseada em um *Resource Manager* global e em um *Application Master* por aplicação, conforme observado na Figura 1 [Foundation 2018]. O *Resource Manager* é responsável por escalonar os recursos do ambiente às aplicações e se comunicar com os *Node Managers*, que são executados em cada nó do *cluster*. Os *Node Managers* monitoram os recursos, tais como memória e CPU, e reportam a situação de seu nodo ao *Resource Manager* através do gerenciamento de *containers*. O *Application Master*, por sua vez, negocia os recursos requisitados ao *Resource Manager* pelas aplicações e trabalha em conjunto com os *Node Managers* e seus *containers* durante a execução de tarefas no sistema.

Com o YARN, o Hadoop oferece uma plataforma de execução de propósito geral eficiente e flexível, que permite a execução de aplicações com base em diferentes modelos de programação. Dentre os modelos de computação suportados pelo YARN, destaca-se o MapReduce, um paradigma dedicado à construção de aplicações paralelas que operam sobre volumes massivos de dados. Uma aplicação MapReduce é composta por - além de outros métodos opcionais - funções de mapeamento *map* e funções de redução *reduce*, que são baseadas em princípios de linguagens de programação funcionais [Achari 2015]. Durante a fase de *map*, a aplicação é preparada com uma divisão de tarefas através do ambiente utilizado, dividindo a entrada e definindo o trabalho a ser realizado por cada

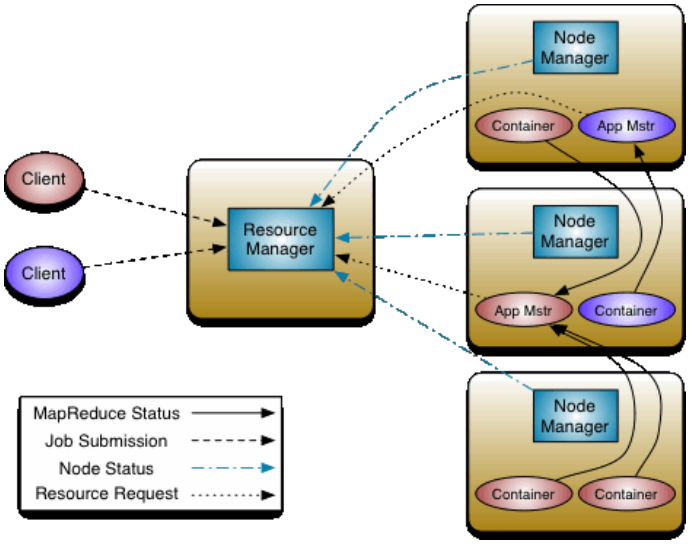

Figura 1. Arquitetura do gerenciador de recursos YARN [Foundation 2018].

nodo. Os resultados das funções de mapeamento são reunidos na fase de redução onde, a partir de um novo processamento, gera-se a saída final da aplicação. Sendo gerenciadas pelo YARN, as tarefas MapReduce são nativamente tolerantes a falhas.

Um dos princípios do Hadoop é mover a aplicação até os dados, evitando mover os dados em si. Isso possibilita aumentar a eficiência de processamento geral da plataforma [Jain and Goyal 2017]. O MapReduce foi projetado para possibilitar tal comportamento enquanto executa sobre o HDFS [Achari 2015]. Sendo o componente alvo da solução proposta neste trabalho, o funcionamento do HDFS é detalhado na Seção 3.

3. Sistema de Arquivos HDFS

Otimizado para armazenar um volume massivo de dados, o HDFS é um sistema de arquivos distribuído e tolerante a falhas capaz de executar sobre *hardware* de baixo custo. Em um *cluster* HDFS, duas classes de nodos compõem uma arquitetura mestre-escravo, o `NameNode` e o `DataNode`. O `NameNode` (mestre) gerencia o *namespace* e os metadados do sistema, além de controlar o acesso e a distribuição dos arquivos. Enquanto isso, os `DataNodes` (escravos) são os nodos que mantém e recuperam os dados. A arquitetura geral do HDFS é representada na Figura 2 [Foundation 2018].

O armazenamento no HDFS é baseado no conceito de blocos onde, quando um arquivo é inserido no sistema, este é quebrado em uma lista de segmentos de dados independentes de tamanho fixo (blocos), que são distribuídos entre os nodos do *cluster*[2]. É responsabilidade do HDFS garantir confiabilidade e disponibilidade, de modo que nenhum bloco seja perdido e que dados corrompidos possam ser restaurados. Neste contexto, meios que garantam a integridade e disponibilidade dos dados através da tolerância e recuperação mediante falhas são imprescindíveis. Os mecanismos de tolerância a falhas mais expressivos do HDFS são o estabelecimento de *checkpoints*, a comunicação constante por meio das mensagens *heartbeat* e a replicação de blocos [Cowsalya and Mugunthan 2015].

O estabelecimento de *checkpoints* corresponde a um mecanismo de tolerância

[2]A partir da versão 2 do Hadoop utiliza-se 128MB como tamanho de bloco padrão.

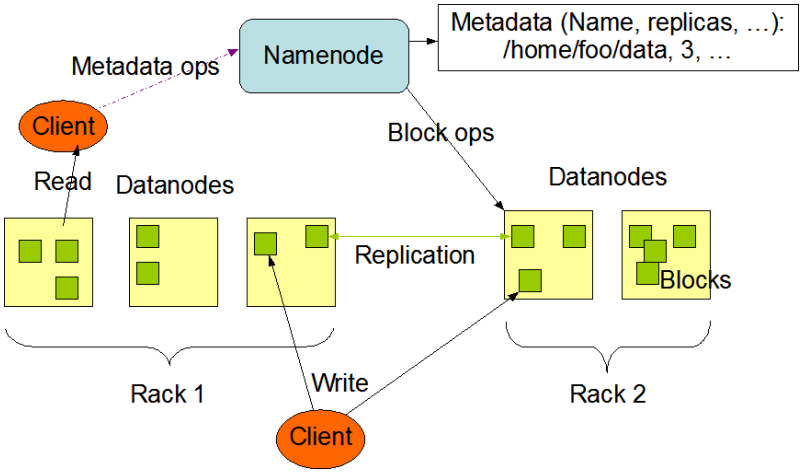

Figura 2. Arquitetura geral do sistema de arquivos distribuído HDFS [Foundation 2018].

a falhas reativo que possibilita recuperar o estado do HDFS através de um contexto estável previamente salvo [White 2015]. O salvamento de *checkpoints* é essencial para a manutenção do `NameNode` e de seus metadados. O *namespace* do HDFS é replicado em um arquivo chamado *FSImage*, armazenado no sistema de arquivos local do `NameNode`. Nesse arquivo, são mantidas as informações sobre o mapeamento de blocos e as configurações e propriedades do sistema. Para evitar a criação de um novo *FSImage* a cada operação realizada armazena-se, também em disco local, um *log* de edições (*EditLog*), que mantém as transações mais recentes realizadas após a criação do *FSImage*. O *merge* entre esses arquivos caracteriza o estabelecimento de um *checkpoint*. Para que o `NameNode` não seja interrompido durante este procedimento, o *merge* é realizado pelo *SecondaryNameNode*. O *SecondaryNameNode* mantém uma cópia do FSImage e a atualiza a cada novo *checkpoint* estabelecido. Para efetuar o *merge*, o *EditLog* é transferido do `NameNode` ao *SecondaryNameNode* sempre que um novo *checkpoint* for requisitado.

Outro mecanismo de tolerância a falhas no HDFS consiste nas mensagens *heartbeat*, as quais são utilizadas como uma forma de detecção de falhas operacionais em `DataNodes` [Foundation 2018]. Periodicamente, os `DataNodes` ativos enviam avisos ao `NameNode` a fim de atualizá-lo sobre seu estado. Quando um `DataNode` deixa de funcionar, seja por falhas de *hardware*, de *software* ou por problemas na comunicação, o `NameNode` o define como inativo e deixa de enviar instruções ao mesmo. A detecção de um `DataNode` falho acontece quando um *timeout* previamente configurado for atingido sem que o `NameNode` receba pelo menos uma mensagem *heartbeat* informando o estado do `DataNode`. Além de compor um mecanismo de tolerância a falhas, as mensagens *heartbeat* atuam como uma ferramenta de suporte gerencial ao reportar informações sobre o espaço de armazenamento utilizado e disponível dos nodos.

Em complemento às mensagens *heartbeat*, o HDFS utiliza a replicação de blocos para garantir tanto a confiabilidade como a disponibilidade dos dados. Assim, caso um `DataNode` falhe, os blocos comprometidos podem ser recuperados a partir de outros nodos, evitando que as aplicações deixem de operar corretamente. Além disso, conforme observado na Figura 2, a partir dos blocos armazenados nos `DataNodes`, as operações de leitura e escrita são realizadas fazendo uso das réplicas mais próximas do cliente, assim

aprimorando o desempenho das aplicações. Sendo fundamental para o balanceamento do sistema, a Seção 3.1 apresenta o mecanismo de replicação de dados com detalhes.

3.1. Replicação no HDFS

Para garantir a disponibilidade dos dados em caso de falhas, o HDFS implementa a estratégia de replicação de blocos. Atuando como um mecanismo de tolerância a falhas baseado em redundância, a replicação faz com que cópias dos blocos sejam armazenadas em diferentes nodos do *cluster*. O número de réplicas a ser gerado para cada bloco de um arquivo é definido a partir de um parâmetro configurável, o fator de replicação.

A replicação garante que, com um fator de replicação de n, os dados armazenados no HDFS fiquem seguros mesmo com $n - 1$ falhas simultâneas. A localização das réplicas tem uma função importante para a confiabilidade e o desempenho da replicação. Neste contexto, o `NameNode` é o responsável pela forma de distribuição das réplicas dentro do sistema, devendo identificar uma solução otimizada para garantir uma melhor comunicação entre as réplicas. Atualmente, a distribuição das réplicas segue uma política de posicionamento *rack-aware* [Foundation 2018].

A Figura 3 [Fazul et al. 2019] ilustra, para cada um dos blocos ($b1$ a $b5$), o funcionamento da política de posicionamento considerando o fator de replicação padrão fixo de três réplicas. Ao seguir a política, a primeira réplica é armazenada no *nodo* do cliente ou, caso este esteja executando fora do *cluster*, em um `DataNode` definido arbitrariamente. As duas réplicas seguintes são armazenadas em um *rack* remoto diferente do local da primeira réplica, porém em dois `DataNodes` distintos [White 2015]. No caso de réplicas subsequentes, o sistema garante que um `DataNode` não mantenha mais de uma réplica de um mesmo bloco, evitando também armazenar diversas cópias em um único *rack*.

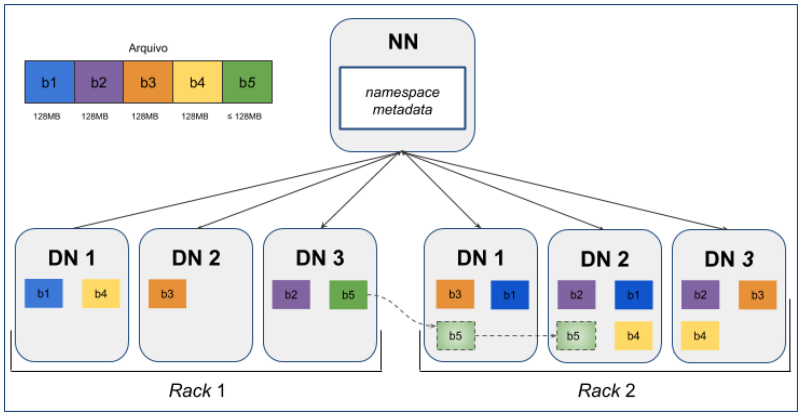

Figura 3. Processo de replicação de dados ao armazenar um arquivo no HDFS [Fazul et al. 2019].

Todo o processo de replicação é transparente ao usuário do HDFS, que precisa interagir apenas com um `DataNode`. Isto é possibilitado através de um *pipeline* de replicação, onde os blocos são automaticamente repassados pelos `DataNodes` até que o fator de replicação seja satisfeito. O bloco *b5* da Figura 3 representa o *pipeline*, onde um `DataNode` pode, simultaneamente, receber blocos e encaminhá-los para o próximo `DataNode` da lista. Com isso, a política de replicação torna o acesso do cliente aos da-

dos mais rápido (devido a maior disponibilidade dos dados) e garante a confiabilidade dos dados mesmo em caso de falhas de *racks* inteiros (réplicas dispostas em múltipllos *racks*).

Para garantir a tolerância a falhas, além do posicionamento inicial dos blocos realizado durante o armazenamento dos arquivos, é necessário que o `NameNode` monitore ativamente o estado das réplicas, disparando, quando necessário, o processo de re-replicação. As causas que podem resultar na re-replicação de blocos são o aumento do fator de replicação de um arquivo, a corrupção de uma réplica, falhas de *hardware* em determinado nodo ou a indisponibilidade de um `DataNode` [Foundation 2018].

Para identificar falhas em `DataNodes` utilizam-se as mensagens *heartbeat*. Quando operando corretamente, um `DataNode` envia mensagens periódicas para o `NameNode`, informando seu estado de funcionamento e reportando a lista de todos seus blocos. Se o `NameNode` não receber mensagens de um `DataNode` no intervalo de tempo pré-definido, o `DataNode` em questão é marcado como inativo e deixa de receber requisições de acesso aos dados. Neste caso, o fator de replicação dos blocos armazenados em um `DataNode` inativo é decrementado. Isso faz com que o `NameNode` dispare o processo de re-replicação a partir de cópias dos dados mantidas em algum dos `DataNodes` ativos. Os `DataNodes` escolhidos para receber as novas réplicas são definidos de forma arbitrária, porém respeitando as restrições impostas pela política de posicionamento.

3.2. Desbalanceamento de Réplicas

Uma das características principais do modelo de programação primário utilizado pelas aplicações que executam no HDFS, o MapReduce, é levar o processamento para onde os dados estão armazenados. Essa característica explora a localidade dos blocos durante a fase de *map*, promovendo melhorias de desempenho no sistema [White 2015]. Todavia, a política padrão de posicionamento de réplicas não define nenhum critério para distinguir a situação de utilização dos `DataNodes`, o que pode favorecer uma distribuição desproporcional dos dados entre os nodos durante as fases de replicação e a re-replicação.

De forma a investigar essa condição, definiu-se um cenário de teste envolvendo a escrita de múltiplos arquivos no HDFS. A Figura 4 ilustra o experimento realizado, que foi executado na plataforma Grid'5000[3] e levou em consideração a escrita de cinco arquivos de 25GB e um fator de replicação de três réplicas por bloco, totalizando uma carga de total de 375GB armazenada no HDFS. O ambiente foi configurado com cinco `DataNodes`, cada um com capacidade de armazenamento de 300GB. A aplicação utilizada para a escrita dos arquivos foi o TestDFSIO [White 2015], um *benchmark* distribuído, integrado ao Hadoop, que testa o desempenho do HDFS com aplicações MapReduce voltadas a operações de entrada e saída.

As porcentagens representam a utilização dos `DataNodes` (proporção do espaço utilizado para o espaço de armazenamento total de seus nodos) após a escrita dos arquivos. Já que, para uma distribuição totalmente equilibrada, cada um dos cinco `DataNodes` deveria armazenar 20% do total dos dados (para o cenário, isso representa 75GB, que equivale a uma utilização de 25% em cada `DataNode`), percebe-se como a política de posicionamento padrão do HDFS favorece o desbalanceamento de réplicas.

[3]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`)

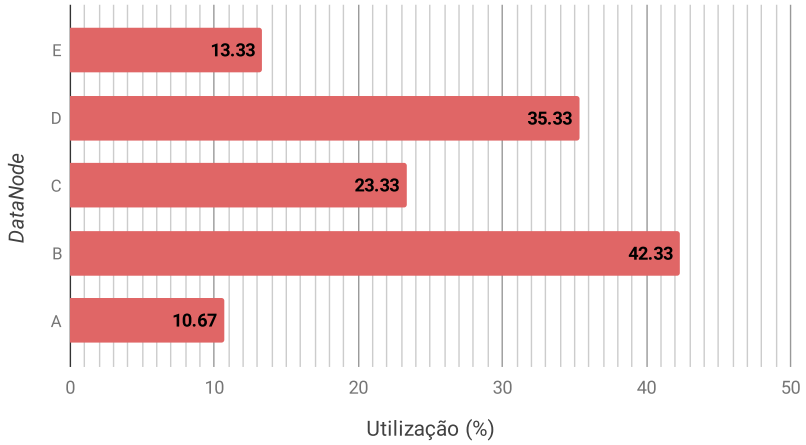

Figura 4. Desbalanceamento inerente da política de posicionamento. Fonte: Elaborada pelos autores.

Com este desequilíbrio, `DataNodes` com um maior número de blocos armazenados tendem a ficar sobrecarregados por executarem um maior número de operações de entrada e saída, enquanto `DataNodes` que armazenam menos dados ficam propensos à ociosidade. Assim, o desbalanceamento de réplicas torna-se uma condição prejudicial à usabilidade geral do HDFS, degradando o desempenho de aplicações MapReduce voltadas a entrada e saída e fazendo com que os recursos computacionais não sejam explorados de forma otimizada [Fazul et al. 2019].

Diversos aspectos podem influenciar o balanceamento do sistema, como por exemplo o comportamento da aplicação do cliente, mudanças no fator de replicação, a ocorrência de falhas de `DataNode` e o acréscimo de novos nodos ao *cluster*. Dessa forma, estratégias e soluções voltadas ao equilíbrio de réplicas tornam-se indispensáveis para a manutenção do HDFS. Na Seção 4 são descritas as principais abordagens existentes na literatura que alinham-se aos interesses deste trabalho.

4. Trabalhos Relacionados

Estudos passados investigaram problemas inerentes do desbalanceamento de réplicas no HDFS e, através de diferentes experimentos, atestaram que benefícios e aprimoramentos de desempenho podem ser alcançados mediante o equilíbrio do *cluster* [Fazul et al. 2019]. Motivado por isso, este trabalho define uma nova política que visa otimizar a forma como a operação de balanceamento é conduzida dentro do sistema de arquivos do Hadoop. A seguir são apresentadas as abordagens e as estratégias relacionadas mais expressivas.

O balanceamento de réplicas no HDFS pode ser endereçado tanto de uma forma proativa quanto de uma forma reativa. Abordagens proativas, como a de [Ibrahim et al. 2016], possibilitam, através de modificações na política de posicionamento padrão do sistema, impedir (ou minimizar as chances) que o *cluster* fique desbalanceado. Com as alterações no algoritmo, a replicação é conduzida de modo a considerar o volume de dados armazenado em cada `DataNode` para a distribuição dos blocos no HDFS. Outro exemplo, visto em [Lin and Lin 2015], define a função de um nodo auxiliar ao `NameNode` – o *BalanceNode* – que contempla situações que podem influenciar o equilíbrio do *cluster*.

O balanceamento reativo, por sua vez, visa o equilíbrio através da redistribuição dos blocos já armazenados no sistema. Exemplos incluem [Liu et al. 2013], que propõem um algoritmo para o balanceamento de *racks* com base em prioridade. A estratégia é focada no equilíbrio de carga entre *racks* superutilizados e subutilizados, reduzindo as chances de falha total de *rack* devido à sobrecarga. Em [Dharanipragada et al. 2017] é apresentado o balanceador `Tula` que, além da utilização dos `DataNodes`, considera variações na latência de escrita e leitura nos discos de armazenamento dos nodos para a realocação dos blocos. Com isso, os `DataNodes` que apresentarem menor latência de disco recebem um número maior de blocos. Os resultados com esta estratégia demonstraram reduções de cerca de 20% no tempo de execução de aplicações MapReduce.

Em contraste com o balanceador `Tula`, que não considera variações nos recursos de processamento e de memória dos nodos no *cluster*, em [Shwe and Aritsugi 2018] é proposto um algoritmo de balanceamento baseado na capacidade de computação dos `DataNodes`. Sendo voltado a instâncias do HDFS executando em ambientes heterogêneos, os blocos são redistribuídos apenas para `DataNodes` pré-determinados que permitam minimizar o custo de transferência de dados inter-*rack* e entre os nodos.

Outra solução para o balanceamento reativo, integrada na distribuição do Hadoop sob a forma de um utilitário, é o `HDFS Balancer` [Shvachko et al. 2010]. Sendo a base para a criação de novos balanceadores, tais como o algoritmo modificado proposto por [Lin and Lin 2015], o `HDFS Balancer` foi implementado de modo a ser extensível a modificações e à integração de novas funcionalidades. Para promover o equilíbrio do *cluster*, a ferramenta opera iterativamente movimentando blocos de `DataNodes` que apresentarem uma alta utilização (origem) para `DataNodes` com um menor volume de dados armazenado (destino). A Seção 4.1 exemplifica um cenário de uso do `HDFS Balancer` dentro do sistema de arquivos do Hadoop.

4.1. Operação do `HDFS Balancer`

O `HDFS Balancer` é guiado por um valor de *threshold*, o qual limita a diferença máxima que a utilização dos `DataNodes` (relação do espaço em uso no nodo para a capacidade de armazenamento total do nodo) e a utilização total do *cluster* (relação do espaço em uso no *cluster* para a capacidade de armazenamento total do *cluster*) pode assumir.

Para exemplificar, consideramos o cenário apresentado na Seção 3.2, onde temos 375GB armazenados no HDFS em um ambiente com 5 `DataNodes`, cada um com capacidade de armazenamento de 300GB. Sendo assim, o *cluster* possui uma capacidade total de 1500GB (300×5) com utilização em 25% ($375/1500$). Dessa forma, considerando o valor de *threshold* padrão de 10%, o `HDFS Balancer` irá executar até que os discos de todos os `DataNodes` estejam com utilização entre 15% ($25\% - $ *threshold*) e 35% ($25\% + $ *threshold*) da capacidade total, o que representa um volume de dados armazenado entre 45GB ($300 \times 15\%$) e 105GB ($300 \times 35\%$) em cada `DataNode`.

De forma a demonstrar experimentalmente esse comportamento, foi conduzido o balanceamento de réplicas no HDFS. A Figura 5 exibe o resultado da execução do `HDFS Balancer` com base no cenário apresentado anteriormente. A utilização de todos os `DataNodes` após o balanceamento ficou em conformidade com o valor de *threshold* estabelecido, atestando dessa forma o sucesso da operação.

Ao conduzir o balanceamento, otimiza-se a localidade espacial dos dados e, por-

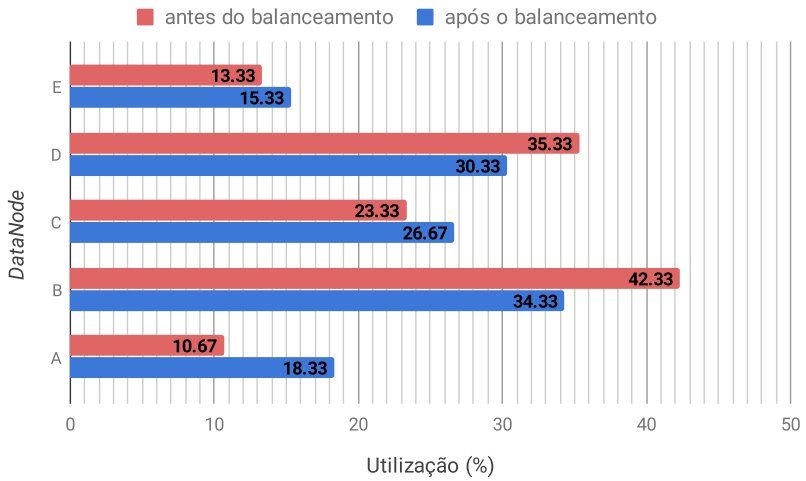

Figura 5. Equilíbrio de réplicas promovido pelo `HDFS Balancer`. Fonte: elaborada pelos autores.

tanto, aplicações de entrada e saída tendem a ser beneficiadas. Neste sentido, a Figura 6 ilustra 20 execuções distintas do *benchmark* TestDFSIO, voltadas para a leitura dos dados armazenados no HDFS com aplicações MapReduce. Percebe-se que, na grande maioria das iterações após o uso do `HDFS Balancer`, o tempo necessário para a operação de leitura foi reduzido, indicando melhorias de desempenho decorrentes do balanceamento.

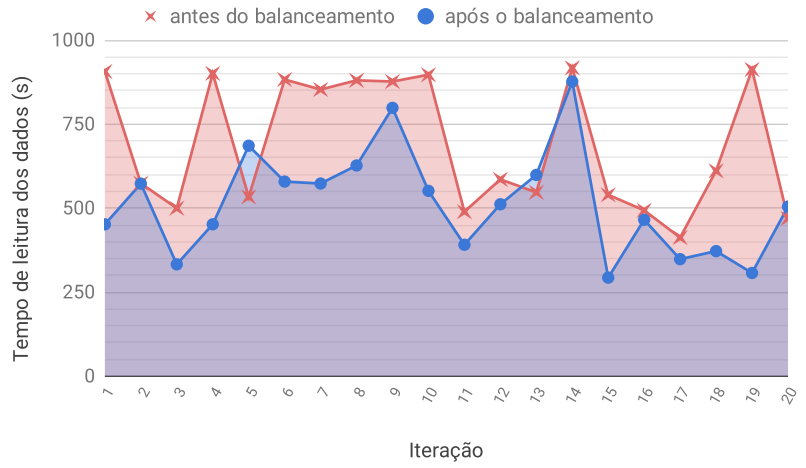

Figura 6. Aprimoramento de desempenho alcançado pelo balanceamento. Fonte: elaborada pelos autores.

O algoritmo base para a operação do `HDFS Balancer` pode ser visto com detalhes em [Hortonworks 2018]. De forma geral, os dados são transferidos a partir da formação de pares origem-destino entre `DataNodes` primariamente classificados como superutilizados (% de uso maior que a média de utilização do *cluster* acrescida do *threshold*) e `DataNodes` classificados como subutilizados (% de uso menor que a média de utilização do *cluster* decrescida do *threshold*). A quantidade de dados a ser movida em uma iteração depende, dentre outros fatores, da existência de blocos que, após realocados em outros nodos, possibilitem manter a conformidade com a política de posicionamento de réplicas padrão do sistema.

5. Política `Custom` para o `HDFS Balancer`

Atualmente, o `HDFS Balancer`, em cada iteração, percorre sequencialmente listas criadas a partir da situação de utilização dos nodos. Caso um `DataNode` superutilizado seja identificado, este é marcado como a origem para a realização de uma futura transferência de blocos. Em seguida, procura-se formar um par com um `DataNode` destino, que é escolhido entre os `DataNodes` classificados como subutilizados ou, caso não existam nodos nessa condição, dentre os `DataNodes` com uma utilização abaixo da média geral do *cluster*. Quando não existirem `DataNodes` superutilizados, procura-se por `DataNodes` subutilizados, que são definidos como o destino para o recebimento de blocos. Após, o par é formado com um `DataNode` origem escolhido dentre os nodos restantes que apresentarem uma utilização acima da média. Alguns problemas operacionais podem ser destacados na abordagem padrão do `HDFS Balancer`:

- A formação do par origem-destino sempre prioriza nodos de um mesmo *rack*, sem permitir qualquer modificação nesse princípio;
- Dentre a lista dos nodos eletivos à formação do par, a escolha efetiva do `DataNode` baseia-se em aleatoriedade, não considerando nenhum potencial aspecto de otimização durante este pareamento;
- Caso não seja possível formar um par com `DataNodes` de um mesmo *rack*, não existem critérios ou métricas para a eleição de um dos *racks* restantes do *cluster* na procura do nodo mais adequado para o balanceamento.

Neste sentido, este trabalho apresenta a política `Custom`, que visa aprimorar o balanceamento de réplicas no HDFS ao realizar otimizações e adicionar novas funcionalidades para o `HDFS Balancer`. A política implementa um sistema de prioridades dos nodos baseado tanto na topologia física do *cluster* quanto em diferentes características de usabilidade do sistema. Com isso, a formação dos pares deixa de depender de aleatoriedade e torna-se totalmente configurável via parâmetros passados para a execução da ferramenta. As prioridades definidas na nova política são apresentadas na Seção 5.1.

5.1. Prioridades para o Balanceamento

A política `Custom` é caracterizada por permitir funcionalidades de priorização na formação de pares de nodos origem-destino durante a fase de balanceamento. Cabe ressaltar que, atualmente, o `HDFS Balancer` não implementa nenhuma das prioridades definidas nesta seção, sendo parte do escopo deste trabalho a codificação e incorporação dessas funcionalidades como uma extensão do código da ferramenta.

Em todos os parâmetros elencados, a variação máxima do volume de dados em cada `DataNode` após o balanceamento continua sendo controlada pelo valor de *threshold*. Além disso, a disposição dos blocos após o balanceamento permanece respeitando a política de posicionamento padrão do HDFS. A partir destas diretrizes, a priorização pode ser realizada com base em um ou mais dos parâmetros descritos a seguir:

Capacidade de processamento dos nodos

Esta prioridade é dedicada a ambientes heterogêneos que executam o Hadoop. O parâmetro determina se diferenças na capacidade de computação dos `DataNodes` devem ser consideradas durante a fase de balanceamento. Com esta priorização, `DataNodes` com maior potencial de processamento, determinado pela quantidade

de memória RAM e por métricas do processador de cada nodo, são suscetíveis ao recebimento de um maior número de blocos.

Capacidade de armazenamento dos nodos

Da mesma forma que a anterior, esta prioridade dedica-se a ambientes heterogêneos que executam o Hadoop. O parâmetro determina se a capacidade total e a capacidade disponível dos dispositivos de armazenamento dos `DataNodes` devem ser relevantes no processo de balanceamento. Através desta priorização, `DataNodes` com um maior espaço de armazenamento disponível em seus discos tornam-se responsáveis por manter uma maior quantidade de réplicas.

Utilização do rack

Tendo em vista a possibilidade de que múltiplos *racks* de um *cluster* possuam diferentes quantidades de nodos, o equilíbrio em nível de `DataNode` pode se apresentar problemático, conduzindo determinados *racks* à sobrecarga. Esse parâmetro possibilita que o balanceamento inter-*rack* seja idealizado no HDFS. O nível de equilíbrio depende, dentre outros fatores, da existência de *racks* que permitam a manutenção da conformidade com as restrições impostas pela política de posicionamento padrão do sistema.

Disponibilidade dos dados

Durante a formação dos pares, priorizam-se `DataNodes` que, após o balanceamento, permitam que as réplicas estejam dispostas na maior quantidade de *racks* possível. Embora a disponibilidade final dos blocos de um arquivo continue sendo dependente de seu respectivo fator de replicação, este parâmetro permite aumentar a confiabilidade do sistema em preservar os dados mesmo no caso de falhas simultâneas de *racks*. Tomando como exemplo um fator de replicação padrão de três réplicas por bloco, cria-se um esforço para que a segunda e a terceira réplicas sejam dispostas em dois *racks* remotos distintos do local da primeira réplica.

Suscetibilidade a falhas

Para a seleção de um *rack* durante a redistribuição dos blocos, realiza-se uma verificação acerca da quantidade de `DataNodes` inativos. Embora `DataNodes` inativos sejam resultado tanto de falhas de *hardware* quanto de *software*, *racks* que possuírem uma proporção elevada entre `DataNodes` inativos em relação aos ativos, podem estar passando por um período de sobrecarga. Desta forma, permite-se priorizar *racks* que possuam um menor índice de falhas de `DataNode`.

Disponibilidade de banda

A política padrão do `HDFS Balancer` considera que a largura de banda para movimentação de blocos intra-*rack* é maior que a largura de banda para transferências entre diferentes *racks*. Todavia, se existir uma grande quantidade de movimentações de bloco já agendadas para um determinado *rack*, sua banda livre pode ser comprometida. Sendo assim, cria-se um parâmetro de controle que permite priorizar a formação de pares *off-rack* quando o *rack* local estiver envolvido

em múltiplas transferências durante a etapa de balanceamento.

Utilização dos nodos

Esta prioridade possibilita a criação de pares entre `DataNodes` origem-destino seja baseada na diferença do volume de dados armazenado em cada nodo. Dessa forma, o `DataNode` que mantiver a maior quantidade de dados entre todos os demais `DataNodes` do *cluster* será pareado com o `DataNode` que possuir a menor quantidade de dados e assim sucessivamente, até todos os pares da iteração serem criados. De modo prático, este parâmetro implica na ordenação (de acordo com a utilização) das listas dos nodos durante a formação dos pares, sendo a lista dos `DataNodes` superutilizados e acima da média ordenada decrescentemente e a lista dos `DataNodes` subutilizados e abaixo da média ordenada crescentemente.

Estado do nodo

Prioridade que permite verificar se o nodo em questão já possui alguma aplicação em execução, evitando assim prejuízo de desempenho das tarefas em operação no *cluster* devido ao balanceamento. Caso o nodo esteja ocupado, duas ações podem ser tomadas: escolher outro nodo disponível ou aguardar o término da aplicação corrente, sendo a espera máxima controlada por um valor de *timeout* configurável.

5.2. Requisitos de Funcionamento

Para possibilitar que a política `Custom` seja otimizada, é importante manter uma relação custo-benefício. Dessa forma, estratégias que permitam acelerar a operação do `HDFS Balancer` tornam-se necessárias. Neste sentido, após definir o bloco a ser realocado, utiliza-se como *proxy* um `DataNode` que possuir uma das réplicas do bloco a ser movimentado e que estiver mais próximo do destino. Assim, ao invés de recuperar o bloco diretamente do `DataNode` origem para o armazenamento local, o `DataNode` destino faz a cópia do bloco mantido no `DataNode` *proxy* e, então, informa o `NameNode` que a réplica armazenada na origem já pode ser removida. Com isso, diminui-se o tráfego de dados inter-*rack*, permitindo reduzir o tempo de balanceamento.

Em relação à execução do `HDFS Balancer`, sua *deamon* é disparada sob demanda sempre que o administrador do sistema julgar necessário. A política customizada é selecionada utilizando o parâmetro *policy* com valor definido em *custom*. Tendo em vista o sistema de prioridades apresentado na Seção 5.1, cabe ao administrador adequar a ordem de relevância dos parâmetros - e, se esses devem ser considerados durante o balanceamento - de acordo com as necessidades e características específicas das aplicações a serem executadas no HDFS. A prioridade elencada para cada parâmetro é configurável, variando de acordo com a instância do `HDFS Balancer` inicializada.

Considerando como exemplo o cenário apresentado na Seção 3.2 (Figura 4) e uma execução do `HDFS Balancer` com a política `Custom`, onde a prioridade de *utilização dos nodos* foi definida como critério para a redistribuição dos blocos, a operação de balanceamento se dará da seguinte forma: o `Datanode` com a maior utilização (B) será pareado com o `Datanode` de menor utilização (A). Após, outro par origem-destino é formado entre os `Datanodes` D e E. Caso o *cluster* consiga ser levado a um estado de equilíbrio com estes pares, o que implica a existência de um número suficiente

de blocos no `Datanode` origem que não possuam réplicas no destino (caso contrário a transferência é proibida pela política de posicionamento), o `Datanode` C não precisará ser pareado, pois já está em conformidade com o *threshold* estipulado.

Ao observar o comportamento do `HDFS Balancer` com sua política padrão (Seção 4.1, Figura 5), percebe-se uma possível otimização ao utilizar a política customizada. Em decorrência do aumento na utilização do `DataNode` C após o balanceamento, sabe-se que este nodo precisou ser pareado em, no mínimo, uma iteração com outro `Datanode` do *cluster*. Ao evitar esta operação, a política *Custom* tende a reduzir o tempo necessário para executar o balanceamento de réplicas no HDFS.

6. Considerações Finais

O `HDFS Balancer` é uma solução disponibilizada pelo Hadoop para promover o equilíbrio de réplicas. Uma etapa essencial para o balanceamento é a definição dos pares origem-destino para a redistribuição de blocos entre os `DataNodes`. Porém, em sua política padrão, não existe nenhum critério que possibilite o HDFS determinar como nodos subutilizados e superutilizados devem ser pareados para a transferência dos blocos.

Este trabalho apresentou a política `Custom`, que visa promover otimizações durante o balanceamento de réplicas realizado pelo `HDFS Balancer` através da exploração de diferentes aspectos e métricas de uso dos componentes do *cluster*. Ao possibilitar que uma ou mais prioridades sejam configuradas, a política proposta permite flexibilizar a operação de balanceamento de acordo com as demandas, sejam estas voltadas a ambientes heterogêneos que executam o Hadoop ou relacionadas a requisitos de desempenho, disponibilidade dos dados, eficiência no balanceamento, dentre outros.

A política customizada se encontra em fase de implementação e validação. Pretende-se submeter seu código-fonte como um *patch* de atualização com funcionalidades adicionais para o `HDFS Balancer`, o que implica na execução de toda rotina de testes automatizados através do servidor *Jenkins*[4], para cumprimento dos requisitos do Apache Hadoop. Trabalhos futuros envolvem a utilização da política apresentada neste trabalho como base para a implementação de um sistema de monitoramento visando uma abordagem dinâmica de balanceamento de réplicas para o sistema de arquivos HDFS.

Referências

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

Cowsalya, T. and Mugunthan, S. (2015). Hadoop architecture and fault tolerance based hadoop clusters in geographically distributed data center. *ARPN Journal of Engineering and Applied Sciences*, 10(7):2818–2821.

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

Fazul, R. W. A., Cardoso, P. V., and Barcelos, P. P. (2019). Análise do impacto da replicação de dados implementada pelo apache hadoop no balanceamento de carga. *Anais do X Computer on the Beach*. No prelo.

[4] `https://jenkins.io/`

Foundation, A. S. (2018). "HDFS Architecture". https://hadoop.apache.org/docs/r2.9.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. Fevereiro.

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.

Jain, H. and Goyal, A. (2017). An improved approach for analysis of hadoop data for all files. *International Journal of Computer Applications*, 157(4).

Lin, C.-Y. and Lin, Y.-C. (2015). A load-balancing algorithm for hadoop distributed file system. In *International Conference on Network-Based Information Systems*, pages 173–179. IEEE.

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

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.

Shwe, T. and Aritsugi, M. (2018). A data re-replication scheme and its improvement toward proactive approach. *ASEAN Engineering Journal*, 8(1):36–52.

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