Dynamic Architecture for Data Replica Balancing in HDFS: Stability, Efficiency, and Data Locality Evaluations

Rhauani Weber Aita Fazul^1, **Odorico Machado Mendizabal**^{1},
Patrícia Pitthan Barcelos^2

^{1}Post Graduate Program in Computer Science
Federal University of Santa Catarina (UFSC) – Florianópolis – SC – Brazil

^{2}Department of Languages and Computer Systems
Federal University of Santa Maria (UFSM) – Santa Maria – RS – Brazil

rhauani.fazul@posgrad.ufsc.br, odorico.mendizabal@ufsc.br,
patricia.pitthan@ufsm.br

Abstract. *Hadoop Distributed File System (HDFS) is known for its specialized strategies and policies tailored to enhance replica placement. This capability is critical for ensuring efficient and reliable access to data replicas, particularly as HDFS operates best when data are evenly distributed within the cluster. In this study, we conduct a thorough analysis of the replica balancing process in HDFS, focusing on two critical performance metrics: stability and efficiency. We evaluated these balancing aspects by contrasting them with conventional HDFS solutions and employing a novel dynamic architecture for data replica balancing. On top of that, we delve into the optimizations in data locality brought about by effective replica balancing and their benefits for data-intensive applications.*

1. Introduction

In an era marked by an increasing capacity to generate valuable data, there is a high demand for computing systems capable of handling and processing vast volumes of data. Distributed file systems, which manage data storage and access across multiple networked servers, are essential to contemporary computing environments, particularly in cloud computing, big data analytics, and businesses that require scalable, reliable, and accessible data storage solutions. A notable example in this area is the Hadoop Distributed File System (HDFS) [Foundation 2023]. The HDFS plays a crucial role in supporting applications dealing with large datasets and seamlessly integrates with several parallel processing technologies [White 2015]. These include Apache Spark, Storm, and Kafka.

HDFS ensures high reliability and availability primarily through data replication [Foundation 2023]. Replication not only is HDFS's main fault tolerance mechanism providing data resilience, but it also enhances the file system's performance by leveraging data locality [White 2015]. In that sense, the distribution of replicas across the cluster's nodes is a key factor for maintaining a healthy and balanced data environment in the cluster [Fazul and Barcelos 2022b], ensuring an optimized use of computing resources.

Over time, replicated data distribution may become unbalanced [White 2015]. This imbalance arises from various factors such as uneven data growth, differential node capacities, or varied workload distributions across the cluster. As data accumulates, some nodes may store disproportionately large amounts of data, while others remain underutilized [Foundation 2023]. This situation can lead to inefficiencies, such as increased load on certain nodes, slower data retrieval times, and a higher risk of data loss in case of node failures. Thus, maintaining an HDFS cluster balanced, given the potential trade-offs, is a preferred choice for handling large-scale data processing tasks.

The HDFS *Balancer* [Shvachko et al. 2010] is a native tool that analyzes data placement within the cluster and facilitates data balancing through replica rearrangement. However, this process relies on manual intervention [Cloudera, Inc. 2021] and requires knowledge of the current system usage to make proper decisions about when and how to balance the data. By recognizing these limitations, previous research has focused on improving the replica balancing process in distributed file systems. Prior work includes the development of a customized balancing policy – the *Prioritized Replica Balancing Policy* (PRBP) [Fazul and Barcelos 2023] – and an automated structure for configuring and triggering the HDFS *Balancer* [Fazul and Barcelos 2021]. These solutions laid the groundwork for the creation of the *Dynamic Architecture for Data Replica Balancing* (DARB) [Fazul and Barcelos 2022a], a sophisticated solution for data balance on HDFS.

Although DARB improves replica balancing in HDFS, further investigation of its performance is still necessary. In particular, we want to understand how stable and efficient the balanced state obtained by rebalancing events is. This paper assesses the effectiveness of replica balancing by evaluating the benefits and drawbacks of the operation, considering the frequency of subsequent rebalancing actions needed to maintain cluster balance. We devised scenarios to compare the impacts of multiple balancing operations performed by DARB against the standard HDFS behavior, with and without using the HDFS *Balancer*. Our analysis encompasses multi-phase experiments, considering variables such as data loads, background applications, node failures, and the addition of new nodes. This comprehensive approach provides deeper insights into the trade-offs between the benefits and the cost of redistributing replicas and the gains in reliability, availability, and performance achieved through balancing. The experimental analysis was conducted in a real, distributed, and multi-rack environment running HDFS. To the best of our knowledge, no work in the literature has compared and evaluated the facets of replica balancing and data locality in HDFS regarding balance stability and efficiency.

The rest of the paper is organized as follows. Section 2 highlights the importance of data locality and balancing in HDFS. Section 3 presents related work involving possible approaches for data balancing. Section 4 formalizes DARB as a proactive and reactive architecture for replica balancing. Section 5 describes the experimental evaluation, and Section 6 presents the concluding remarks and outlines directions for future work.

2. Data Replication and Replica Placement in HDFS

HDFS is a distributed file system designed for high reliability and availability. It provides fault tolerance even in clusters running over commodity hardware. Within an HDFS cluster, the architecture follows a server-worker model, composed of two types of nodes [Foundation 2023]: the *NameNode* (NN) and the *DataNode* (DN). The NN is the master server, managing the system's namespace and metadata, maintaining the directory tree, and controlling file access and distribution. The DNs are responsible for data storage and retrieval, allowing the data files to be distributed across network-connected machines.

HDFS was designed for storing data on the petabyte scale [White 2015]. And, to help with that, it adopts a block-based storage strategy, where files are automatically segmented into fixed-size (128MB by default) data blocks and stored independently. Hadoop clusters may contain thousands of nodes responsible for both computing and data storage. As HDFS may operate on low-cost equipment, the chances of DN failures are high [Foundation 2023]. So, the system must maintain its consistency and be able to restore compromised blocks, ensuring high availability and reliability.

To increase data reliability and availability in distributed environments, HDFS replicates the file blocks, maintaining data redundancy in the system through replication [Shvachko et al. 2010]. With replication, multiple copies of blocks are maintained on different DNs of the cluster, ensuring that, in case of failures, at least one copy of the data remains accessible. The Replication Factor (RF) determines the number of replicas of each block, and it is configurable per file, with a standard value of three replicas per block [White 2015]. The NN continuously monitors the number of replicas of each block to ensure compliance with the established RF. To this end, the DNs periodically communicate with the NN through heartbeats messages. If the NN does not receive messages from a DN, it marks the DN as inactive and initiates the re-replication of under-replicated blocks, selecting a source DN that contains an existing replica of the block and a destination DN for the new copy. Re-replication may occur due to multiple reasons [Foundation 2023], such as replica corruption, failures in one or more storage disks of DNs, an increase in the RF of a file, or DN unavailability, whether due to network partition causing some subset of DNs to lose connectivity with the NN or due to crash failures in their nodes.

Replication in HDFS, therefore, is a dynamic and continuous process. To distribute the data in an HDFS cluster, the replication and re-replication strategies follow a Replica Placement Policy (RPP) [White 2015]. This approach is fundamental in optimizing data availability and reliability, reducing bandwidth consumption in write operations, and improving read performance. The standard RPP is an initial effort in this regard, and part of Hadoop's objectives [Foundation 2023] is to validate and improve this policy in real systems, encouraging research and testing of more advanced placement policies.

The current RPP uses rack awareness to enhance fault tolerance and performance. With a standard RF of three, it ensures that replicas are distributed across three distinct DNs, keeping one of the replicas on the local DN if the client is running on it [Shvachko et al. 2010]. Additionally, it ensures that, at most, two-thirds of the replicated blocks are on the same rack. If the RF is greater than three, additional DNs are randomly selected, although efforts are made to avoid concentrating too many block replicas on the same rack. It addresses fault tolerance by preventing data loss even if a complete rack fails. Performance is optimized by maximizing aggregate read bandwidth as DNs from different racks are available for I/O requests. In this sense, HDFS prioritizes reading replicas close to the client, preferring local data over remote [Foundation 2023].

2.1. Data Locality and Replica Balancing

HDFS was conceived with the idea that the most efficient data processing strategy for files follows the write-once-read-many (WORM) model [Foundation 2023]. Generally, applications in an HDFS cluster deal with large volumes of data and require continuous (streaming) access to their datasets. Therefore, the time required to process a complete dataset is more critical than the latency to read the first record.

One of the fundamental principles of Hadoop is to maximize throughput by co-locating data with the compute nodes. This principle follows the premise that *moving computation is cheaper than moving data* [Foundation 2023] and it is known as *data locality optimization* [White 2015]. It improves performance in processing large datasets by reducing the overall bandwidth consumption, network congestion, and read latency. Replication increases the likelihood that a computational task can process most data blocks locally.Yet, cluster imbalances, marked by notable differences in the volume of data stored across the cluster's nodes, are expected over time.

The RPP prevents replicas of the same block from being stored on the same DN. Nonetheless, this approach does not ensure a fully balanced distribution. By allocating two-thirds of a block's replicas to a single rack, the RPP inadvertently contributes to inter-rack imbalances [Fazul and Barcelos 2022b]. The current policy lacks native mechanisms to address this problem, such as using node utilization to determine where new replicas should be placed within the file system [Shvachko et al. 2010].

Replica imbalance can lead to increased intra-rack and off-rack transfers, as tasks assigned to a node with few local replicas need to access data from other nodes [Fazul and Barcelos 2022b]. This increases bandwidth consumption across the cluster and places additional strain on DNs already under heavy data utilization [White 2015]. As some nodes reach their maximum storage capacity and may no longer accept new data blocks, the write and read parallelism of the cluster is reduced, which impacts the performance of applications that heavily rely on data, I/O, and, in a reduced extent, even CPU-bound applications [Fazul and Barcelos 2023].

Replicated data stored in HDFS can become unbalanced for several reasons [Cloudera, Inc. 2021]: (i) the constraint satisfaction algorithm implemented by the RPP for file block allocation, which, by default, does not consider the node's usage, can lead to uneven data distribution among the DNs and their racks because of randomness (the problem becomes serious when the cluster is nearly full); (ii) the re-replication process, which follows the initial policy and can also contribute to imbalance during data distribution; (iii) the occurrence of DN failures, as inactive DNs induce the re-replication of data blocks; (iv) the skewed behavior of the client application behavior, especially those running directly on a DN, as one of the replicas will be stored in its local storage to preserve data locality, usually resulting in higher utilization on its storage devices; and (v) the addition of new DNs to the system since existing blocks are not automatically reallocated, and HDFS will consider these nodes as candidates for new blocks along with all other DNs in the cluster, thus some DNs will remain lightly utilized for a significant period.

3. Related Work

Data replication and replica placement are pivotal in enhancing fault tolerance, reliability, and availability within distributed file systems. These areas have garnered attention from researchers, especially in optimizing data locality in data-intensive systems. The study of [Shwe and Aritsugi 2018] introduced a re-replication scheme to improve performance and reliability. By dividing data blocks into priority groups and selecting the DNs for storage based on utilization, this method balances workloads across nodes while minimizing the impact and duration of the re-replication process. The work of [Zhang et al. 2015] explored the potential benefits of increasing the number of replicas of in-demand data in

HDFS. They used an adaptive replication system that elevates the RF of highly accessed data, thereby optimizing data availability and reducing job execution times. Similarly, [Joshi et al. 2022] conducted an experimental analysis to assess the impact of adjusting the RF and block size settings on the performance of Apache Spark applications.

In [Yin and Deng 2022], the authors investigated methods for placing replicas in edge-cloud environments and proposed a strategy to place them on edge nodes based on their load and the cost-effective value. In previous research [Fazul and Barcelos 2022b], we lead a practical investigation of the RPP variations in HDFS. This research filled a gap in the existing literature by exploring beyond the default RPP. The evaluation considered various stages, including file writing, re-replication post-failures, and replica rebalancing. Variations of the RPPs and their impact on replication and balancing were compared, providing valuable insights for developing future balancing solutions.

Focusing on replica balancing in HDFS, several studies have examined and sought to enhance this feature. Broadly, two approaches emerge: proactive and reactive. Proactive methods aim to maintain data balance preemptively, while reactive approaches offer corrective strategies for reestablishing balance after imbalances occur. As an example of proactive approaches, the work of [Dai et al. 2017] presented an improved RPP tailored for heterogeneous clusters, adhering to standard requirements while promoting a balanced data distribution. A block placement policy that considers disk latencies and other constraints to boost job performance appeared in [Dharanipragada et al. 2017]. The authors in [Liu et al. 2021] introduced a group-based RPP for large-scale and batch-processing geospatial 3D raster data to optimize replica placement and minimize network overhead caused by randomly storing files from adjacent regions on multiple nodes.

Since it is not always possible to prevent an unbalanced replica distribution, reactive approaches are necessary to rebalance the data already stored in the file system. In [Dharanipragada et al. 2017], the authors also presented a modified algorithm that considers node utilization and disk latencies for relocating data. The strategy in [Shah and Padole 2018] focused on replication optimization by leveraging node processing capacities and redistributing blocks based on node heterogeneity and performance.

Hadoop also provides an integrated reactive solution for replica balancing, known as HDFS *Balancer* [Shvachko et al. 2010]. This tool works iteratively, guided by a balancing *threshold* ranging from 0% to 100% with a default setting of 10%. Let $G_{i,t}$ be a group of storage devices of type t belonging to DN i. The HDFS *Balancer* operates by assessing the utilization of each of the groups ($U_{i,t}$) against the average utilization of all storage devices of type t in the cluster ($U_{\mu,t}$), redistributing the replicas until the utilization of each DN remains within a range defined by $U_{\mu,t} \pm \textit{threshold}$ [Cloudera, Inc. 2021]. The existing operational mode of the HDFS *Balancer*, while functional, is hampered by certain limitations. It relies on manual configuration and on-demand execution, which requires the HDFS administrator to meticulously decide on configurations that enhance reliability and performance without significantly disrupting the cluster's overall health. Such decisions often require a thorough understanding of the file system intricacies, including the dynamics of client applications and their interactions with the system. As a result, relying solely on the HDFS *Balancer* can lead to issues like delays or omissions, sub-optimal configuration choices, and poorly timed activations.

4. Dynamic Architecture for Data Replica Balancing

Although HDFS architecture supports rebalancing schemes to automatically move data from one DN to another if free space on a node falls below a threshold, such functionality is not natively implemented [Foundation 2023]. To address this limitation, we developed a *Dynamic Architecture for Data Replica Balancing* (DARB). The evolution of the architecture unfolded across different stages and validations [Fazul and Barcelos 2021, Fazul and Barcelos 2022a], and now it embodies both proactive and reactive approaches for data replica balancing tailored for distributed file systems based on replication. DARB leverages the HDFS *Balancer* structure for effectively managing the distributed file system while avoiding the pitfalls of periodic, time-based scheduling models, often leading to unnecessary intrusiveness and suboptimal balancing operations. Instead, DARB introduces context-awareness and event-triggered mechanisms, continuously monitoring HDFS components' behavior and adjusting definitions accordingly to the system context.

Figure 1 outlines our solution's architecture and key monitoring modules. Proactively, DARB monitors the environment to automate the balancing process, identifying favorable moments for balancing. To this end, DARB uses an event observation model based on Hadoop *Metrics* [Foundation 2023]: statistical information about events and measurements exposed by Hadoop daemons belonging to several layers, such as HDFS, MapReduce, and YARN. *Metrics* standardizes the flow to obtain parameters, and we built three monitoring modules around it.

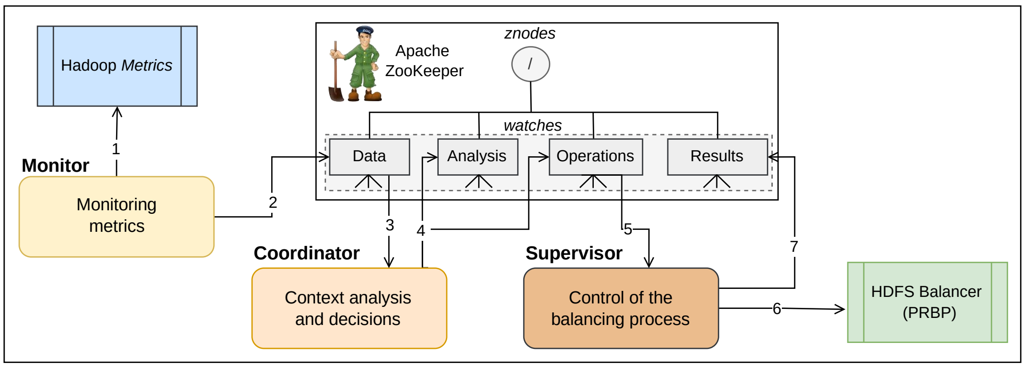


Figure 1. Overview of DARB and its component interactions.

The *monitor* module is responsible for continuously collecting and consuming the predefined metrics available through Hadoop *Metrics* (step 1 of Figure 1). It performs the initial data preprocessing and stores the processed metrics in the namespace of the Apache ZooKeeper [Haloi 2015], which is incorporated by DARB to maintain the configuration information and coordinate communication between the modules. Saving the data into a subtree of the *Data znode* (step 2) triggers a notification – through the ZooKeeper's *watches* mechanism [Haloi 2015] – to the *coordinator* module (step 3).

The proactive strategy of DARB is based on this event-driven strategy to automatically define *when* the balancing process should be performed according to the state of the file system. With a global view of the system's element states and the observed metrics, the *coordinator* analyzes the environment, evaluating the applicability of replica balancing in the file system. To decide if the HDFS is suitable for balancing, the *coordinator* relies on a set of trigger events [Fazul and Barcelos 2022a]. A trigger event comprises one or more observed metrics and determines whether the balancing should be performed at that moment or postponed. Currently, the supported trigger events are: (i) new read operation and data volume increase in the system; (ii) completion of write operation and low cluster load; and (iii) change in cluster configuration/topology. After identifying a trigger event and based on the current level of cluster imbalance, DARB triggers the HDFS *Balancer* configured accordingly to the current context.

The results of the *coordinator* analysis are stored in a *znode* of the *Analyses* subtree (first stage of step 4). Moreover, suppose any corrective action proves necessary given the conditions set by the trigger events. In that case, the *coordinator* defines the balancing attributes (such as the *threshold* and operation mode for the HDFS *Balancer*) to be applied and stores them in a *znode* in the *Operations* subtree (second stage of step 4), resulting in a notification to the *supervisor* module (step 5).

The reactive strategy of DARB consists of a personalized policy for HDFS *Balancer* called *Prioritized Replica Balancing Policy* (PRBP) [Fazul and Barcelos 2023]. The PRBP exposes balancing *guidelines* to support the balancing process considering the cluster architecture and context of the system and its applications. Upon receiving the notification and based on the definitions of the *coordinator*, the *supervisor* triggers replica balancing by executing the HDFS *Balancer* daemon with the appropriate guidelines of the PRBP and waits for its completion (step 6). The seamless choice of guidelines is essentially the definition of *how* the balancing should be executed. Once the replica redistribution is completed, the *supervisor* captures statistics about the balancing operation and saves them in the *Results* subtree (step 7). These data, alongside the historical analyses of the *coordinator*, can be used to guide future balancing operations. In this way, the architecture decisions relate to both the current cluster context and historical information.

Based on this architecture, the replica balancing on HDFS transitions from a generic procedure to one that considers the parameters of the applications running on the cluster. This context-sensitive approach is deemed the essence of DARB as it offers an evolutionary approach for replica balancing in distributed file systems like HDFS. Decisions about the moment and the best strategy become transparent to the system's administrator, eliminating the need to manually monitor the file system's state and properly configure/execute the HDFS *Balancer*.

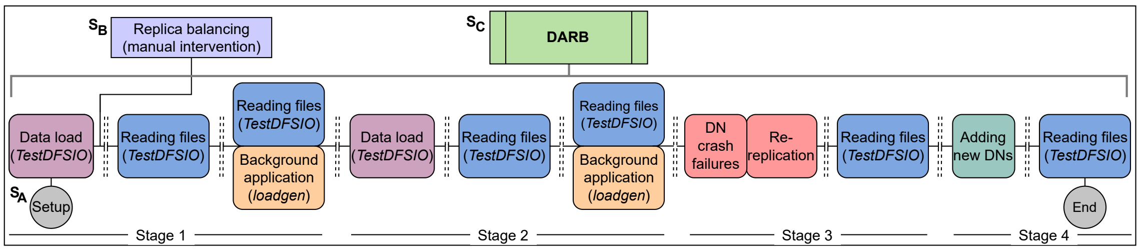
5. Experiments and Results

In this study, we evaluate the behavior of HDFS across three distinct scenarios to assess its performance in handling replica balancing under various operational conditions. The experiments were carried out on the Grid'5000 platform.[1] We used an environment comprising 10 nodes from the *gros* cluster at the *Nancy site*. Each node (Dell PowerEdge R640), running a Debian GNU/Linux 11 (*bullseye*), featured an Intel Xeon Gold CPU (2.20 GHz, 18 cores/CPU), 96GB of RAM, 447GB of storage (SSD), and dual Ethernet connections with 25Gbps each (SR-IOV enabled).

The experimental flow, depicted in Figure 2, consisted of four stages. The first focuses on data loading and reading operations – conducted with and without concurrent

[1]Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several Universities as well as other organizations (see `https://www.grid5000.fr`).

background applications in the cluster – using *TestDFSIO* [White 2015]: a distributed benchmark that measures HDFS performance through intensive parallel I/O operations. The second stage involves new I/O operations, including loading additional data and executing fresh read operations. The third stage introduces crash failures in the DNs, intentionally induced to highlight the re-replication process, followed by subsequent data reading. The final stage examines read operations after adding new nodes into the system. We included idle windows between each step, as indicated by the dashed lines. This period of inactivity is strategically placed to ensure file system dteadiness, effectively mitigating inadvertent disruptions from previous operations.



S_B
Replica balancing (manual intervention)
S_C
DARB
Data load (TestDFSIO)
Reading files (TestDFSIO)
Reading files (TestDFSIO)
Background application (loadgen)
Data load (TestDFSIO)
Reading files (TestDFSIO)
Reading files (TestDFSIO)
Background application (loadgen)
DN crash failures
Re-replication
Reading files (TestDFSIO)
Adding new DNs
Reading files (TestDFSIO)
S_A
Setup
End
Stage 1
Stage 2
Stage 3
Stage 4

Figure 2. Overview of the execution flow in the test scenarios.

We combined three test scenarios with the four stages. Scenario S_A, serving as the baseline, considers HDFS's default behavior without any modifications. It reflects most users' typical setup in a regular HDFS deployment, providing a fundamental understanding of the system's default data allocation and balancing strategies. In scenario S_B, we introduced manual intervention by executing the HDFS *Balancer* (with default configuration) after the first data load, aiming to understand the impact of such interventions on data distribution and system efficiency. Scenario S_C incorporates DARB, hypothesized to provide a more dynamic and responsive balancing mechanism. For our analysis, we borrowed two performance metrics most known from the load balancing field [Xu and Lau 1996]: *stability* and *efficiency*. The former refers to the ability of an algorithm to coerce any initial workload distribution into a balanced state. The latter measures the time required to reduce the variance or reach the equilibrium. In addition to these metrics, we considered optimizations in data locality provided by replica balancing.

5.1. Balancing Stability Analysis

In this work, we examined balancing stability through a windowed analysis, evaluating the cluster's balance state at specific stages. This allows observing how the system's stability fluctuates over time. Stability, in this case, is quantified based on the standard deviation (σ) of the volume of data stored across the DNs, considering both occupation and utilization percentage. By analyzing the variations in data volume within the tested stages, we gain insights into the uniformity of data distribution across the cluster, with a lower standard deviation indicating a more balanced and stable system.

In *Stage 1*, the data was loaded with *TestDFSIO*. For each test scenario, 25 files of 25GB each were written to the system with a default RF of 3 replicas per block, totaling approximately 1.46TB of data ($4{,}000$ blocks of 128MB each and $12{,}000$ replicas). Figure 3 presents the cluster's state considering the utilization of the DNs ($U_{i,SSD}$, where i

refers to the i^{th} DN). The average cluster utilization ($U_{\mu,SSD}$) after the data load, for all scenarios, was 39.47%, depicted by the dashed blue line.

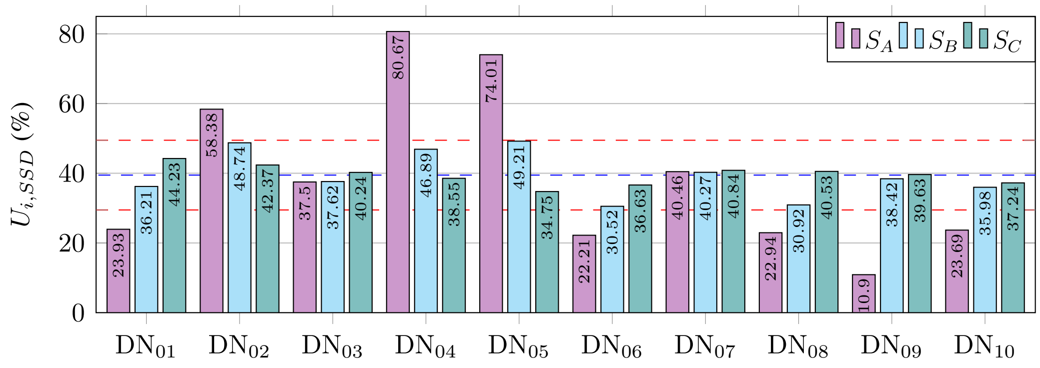


Figure 3. HDFS state considering DNs utilization after data load in the first stage.

In scenario S_A, using the standard RPP, there was a discrepancy in the volume of data stored on the DNs. This imbalance was evident from the high σ in occupation and utilization: 90.72GB and 23.87%, respectively. In scenario S_B, after running the HDFS *Balancer* on-demand with default settings, the utilization of the DNs was maintained within the lower and upper balancing limits considering a standard *threshold* of 10% (*i.e.*, $U_{\mu,SSD} \pm$ *threshold*), represented by the dashed red lines. The rebalancing resulted in a σ of 25.87GB and 6.81%. In contrast, scenario S_C used DARB, which automatically performed replica balancing upon detecting trigger events related to new data being loaded in the file system. In this sense, when DARB detects a trigger event, it looks at the cluster's balance level at that time taking the necessary actions to keep/achieve balancing stability. This approach effectively reduced the disparities in data volume across the DNs, with σ values for occupation and utilization dropping to 10.66GB and 2.81%.

In *Stage 2*, additional data was loaded into HDFS: 10 files of 25GB each, increasing the $U_{\mu,SSD}$ to 59.34% (totaling approximately 2.2TB). In scenario S_A, the σ values for occupation and utilization were 92.71GB and 24.4%, reinforcing that the RPP does not guarantee balanced data distribution. In scenario S_B, the imbalance was less pronounced but remained accentuated, with deviations of 50.86GB and 13.38%. This demonstrated that a single balancing operation was insufficient for maintaining the cluster in a balanced state after new files were written. Conversely, in scenario S_C, DARB promptly responded to workload variation caused by new files, keeping the cluster balanced. It achieved σ values for the occupation and utilization of the DNs of 8.63GB and 2.27%, respectively.

Stage 3 represents an environment with failures. To emulate the faulty behavior, we introduced crash failures, spaced 60s apart, in two DNs (DN_{09} and DN_{10}) within the same logical rack. The reduction in active DNs (down to 8) raised the average cluster utilization ($U_{\mu,SSD}$) to 74.13%. Upon failure, the data held in the decommissioned DNs are automatically redistributed by HDFS to the remaining active DNs. In scenario S_A, this led to a σ in occupation and utilization of 57.57GB and 15.15%, respectively. Although there was a reduction in the discrepancy, it is noteworthy that the re-replication process in the cluster indirectly benefited replica balancing in the cluster, now with fewer candidates to receive the re-replicated blocks. Similarly, in scenario S_B, the σ was 47.51GB and 12.50%. As for scenario S_C, DARB automatically detected that the cluster topology changed due to failures, assessing the need for rebalancing. Given that the DNs still respected the adaptable thresholds defined by DARB after the re-replication process, further replica rearrangement was not deemed necessary. The σ in the occupation and utilization of the DNs were 11.35GB and 2.99%.

Lastly, in *Stage 4*, the cluster configuration was modified by adding new DNs (DN$_{09^*}$ and DN$_{10^*}$). Figure 4 illustrates the final utilization state of the HDFS. In both scenarios S_A and S_B, the data already stored were not redistributed to the new DNs. The cluster administrator must manually configure and execute the HDFS *Balancer* to force rebalancing. Apart from the unbalanced distribution inherited in the RPP, the addition of the two DNs aggravated the imbalance, as noticed by the elevated σ of occupation and utilization: 57.57GB and 33.99% in S_A and 47.51GB and 33.12% in S_B. In contrast, the replica balancing was automatically handled by DARB in scenario S_C, considering the new and existing DNs, resulting in much lower σ values of 11.54GB and 3.04%.

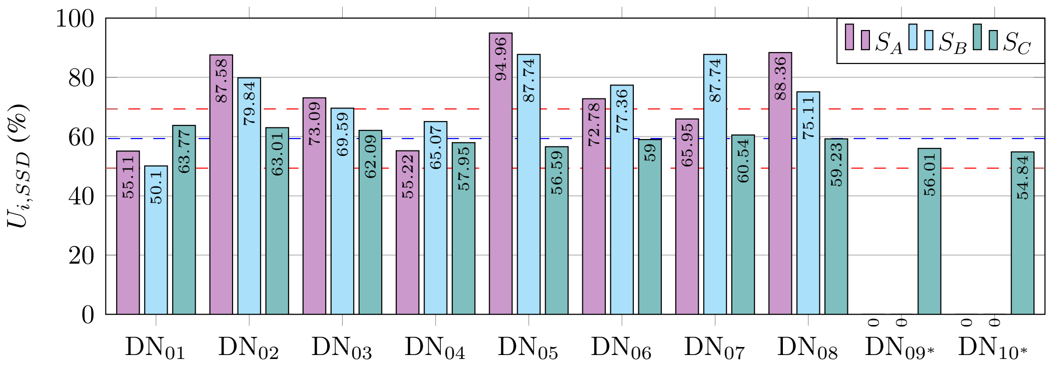


Figure 4. Final HDFS state considering DNs utilization in the fourth stage.

These results demonstrate the stability of replica placement algorithms within HDFS, highlighting the essential role of rebalancing solutions. The default RPP falls short of achieving a balanced distribution, notably after incorporating new data into the system. Executing HDFS *Balancer* on-demand at specific times is helpful but does not maintain adequate balance levels after new data loads. Regular execution of the balancer is recommended, but it requires the administrator to meticulously monitor the cluster's status and determine the optimal timing for its activation. Considering that different situations can influence the balancing state of the system, this routine task is prone to inappropriate choices. All of this is aggravated if new nodes become part of the HDFS instance. In response to that, among the tested scenarios, DARB showed superior capability regarding balancing stability, autonomously guiding the cluster toward a balanced state.

5.2. Balancing Efficiency and Data Locality Optimization Analysis

Figure 5 displays the execution times for each read operation conducted in the experiment. As Figure 2 indicates, each stage includes a reading phase, consisting of 10 execution runs of *TestDFSIO* in read mode. For the first and second stages, we executed the additional 10 benchmark runs with a background application running in the cluster (a generic map/reduce load generator called *loadgen* [White 2015]).

In executions $[1, 10]$ of the benchmark in *Stage 1* (*i.e.*, without concurrent applications in the cluster), it is noticeable that the read times in scenario S_B – after manual

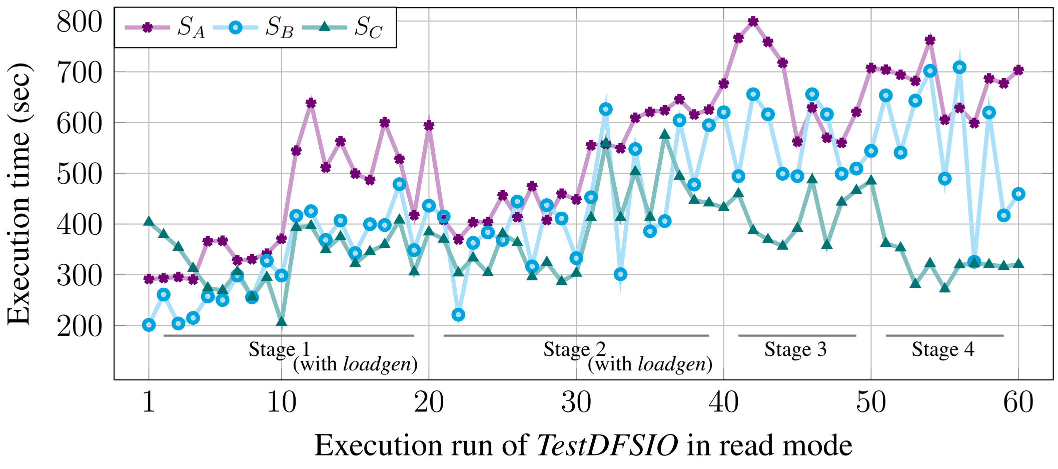


Figure 5. HDFS performance across all read operations.

execution of the HDFS *Balancer* – are lower than in scenario S_A. It means that a balanced cluster can take better advantage of the spatial locality of the stored data. In the first 4 read operations in scenario S_C, the execution time of *TestDFSIO* was higher compared to the other scenarios. This occurred because DARB automatically triggered balancing operations that coincided with the initial read operations. Such behavior is expected due to trigger events acting upon write operations followed by read operations, leading to data rebalancing. After this period, the performance in scenario S_C aligns closely with S_B, as both scenarios benefit from replica rearrangement.

Table 1 presents the average read times for each read phase. For scenarios S_B and S_C, the percentage change relative to the baseline (S_A) is shown. Focusing on interactions $[1, 10]$, scenario S_A, with data distribution based on the initial RPP, had the poorest read performance. The lower execution times were observed in scenario S_B, where the balancing operation concluded before the read operations started. In scenario S_C, the balancing operations were conducted dynamically across the experiment flow and did not exhibit such pronounced gains in this phase.

Scenario	**Metric**	**Stage 1**		**Stage 2**		**Stage 3**	**Stage 4**
	It.	$[1, 10]$	$[11, 20]$	$[21, 30]$	$[31, 40]$	$[41, 50]$	$[51, 60]$
$\mathbf{S_A}$	Read time	322.67s	538.30s	421.79s	607.92s	664.86s	674.22s
$\mathbf{S_B}$	Read time	252.24s	402.04s	373.33s	501.66s	559.98s	555.94s
	% change	-21.83%	-25.31%	-11.49%	-17.48%	-15.77%	-17.54%
$\mathbf{S_C}$	Read time	316.57s	363.87s	329.02s	468.98s	413.07s	318.73s
	% change	-1.89%	-32.4%	-21.99%	-22.85%	-37.87%	-52.73%

Table 1. Average read performance in HDFS across all four tested stages.

In terms of the efficiency of the balancing process, in scenario S_A, with the HDFS default behavior, there were no balancing operations. In contrast, in scenario S_B, the HDFS *Balancer* was executed right after the first data load in the first stage. This operation moved a total of 328.57GB of data, taking 2574s to complete (average throughput for balancing 148.63MB/s). In scenario S_C, a total of seven rebalancing operations were conducted by DARB. In the first stage, the HDFS *Balancer* relocated 539.81GB of data in four balancing executions, totaling 3324s to complete (average throughput of 162.4MB/s). The durations for each balancing operation were 1962s, 572s, 431s, and 359s. Considering this, a trade-off becomes evident between the effort put into balancing and the balancing efficiency. Scenario S_c demonstrated the highest level of overall balance, as indicated by the reduced standard deviation in data distribution. However, this was achieved at the cost of increased duration and resource investment for balancing as it demanded more time to transfer larger volumes of data, thereby consuming more bandwidth.

In iterations $[11, 20]$, the background applications significantly impacted read performance in scenario S_A. The other scenarios maintained similar performances, although also affected by higher resource competition and concurrency. Scenario S_C achieved the best performance, as DARB already addressed the necessary balancing in the initial iterations. Considering the average read times throughout the first stage ($[1, 20]$), the results were: 432.89s for S_A, 329.46s for S_B, and 334.69s for S_C. Thus, in the overall read performance for *Stage 1*, scenario S_C performed very closely to S_B.

In *Stage 2*, with new data loaded into the system, scenario S_A experienced significant performance degradation, both in iterations $[21, 30]$ and $[31, 40]$ with the background applications in the cluster. Scenario S_B was also affected due to the increased imbalance in the cluster after adding new files. However, DARB in scenario S_C effectively performed corrective actions, maintaining cluster balance. Even though an increase in time from iterations $[21, 30]$ to $[31, 40]$ was expected in all scenarios as the cluster was serving other background applications, scenario S_C showed consistent performance improvements compared to the baseline. The average read times for the entire stage ($[21, 40]$) were 516.17s for S_A, 435.46s for S_B, and 397.71s for S_C. These results support the benefits of maintaining replica balancing after new data loads to improve the performance of subsequent read operations. In this stage, two balancing operations were activated by DARB after the second data load, moving 168.01GB of data. The total time for data rearrangement was 1024s (average throughput of 164.07MB/s). The first operation took 469s to complete, while the second required 555s. In this stage, the trade-off between balancing effort and efficiency proved highly beneficial. The optimizations in data locality within a well-balanced cluster significantly reduced read times.

In *Stage 3*, failures caused an increase in read times due to reduced active DNs (*i.e.*, less read parallelism). The average read times for scenarios S_A and S_B in iterations $[41, 50]$, as shown in Table 1, were higher than those in iterations $[31, 40]$, which included background applications. In scenario S_C, the average read time was lower even without DARB handling new balancing operations since the cluster balance was not impacted by the re-replication process, as mentioned in Section 5.1. This indicates that the previous balancing operations conducted by DARB whitin *Stage 2* provided high stability and were enough to keep the cluster at balanced levels even after failures.

In *Stage 4*, new DNs were added to the cluster, leading to a substantial difference in behavior among the scenarios. In S_A and S_B there was no redistribution of the stored data replicas to the new DNs. In contrast, in scenario S_C, the balancing process was automatically performed by DARB upon detecting changes in the cluster's topology. To this end, a single balancing operation was performed by DARB, redistributing 182.82GB of data in 511s (an average throughput of 357.77MB/s). The efficiency of this operation is attributed to DARB configuring the HDFS *Balancer* in *fast* mode upon detecting new nodes, aiming for rapid system rebalancing to include the new nodes in subsequent read operations (*i.e.*, avoid underutilization of resources).

In scenarios S_A and S_B, the execution times for iterations $[51, 60]$ remained similar to those in $[41, 50]$. Conversely, DARB in scenario S_C allowed leveraging the new DNs in the following operations, enhancing the overall cluster bandwidth utilization. This capability resulted in the most significant performance gain among the scenarios: a reduction of 52.73% in the time required to read the files in scenario S_C compared to the baseline. Even with reduced balancing efficiency, as more data replicas were redistributed across the cluster, DARB showed significant improvements in relation to the other scenarios, ensuring optimal data distribution and enhanced read performance as the cluster evolves.

6. Conclusions and Future Work

Data replication is the primary mechanism for fault tolerance in HDFS, but it can lead to unbalanced data distribution, affecting performance. In this study, we conducted an in-depth analysis of the replica balancing process in HDFS, emphasizing the critical role of the HDFS *Balancer*. Our experimental investigation considered variations in data volume, background loads, node availability, and cluster topology, offering a new perspective on replica balancing and demonstrating the importance of stability and efficiency in the data replica placement algorithms used in HDFS.

We found that the default replica placement policy struggles to maintain data balance and that the on-demand use of the HDFS *Balancer* provides a temporary solution but fails to sustain equilibrium amid continuous data loads and changes in cluster topology. Here, DARB, an autonomous balancing solution, proves more effective, dynamically adapting to context changes. While it may impact balancing efficiency by reallocating more data for improved stability, DARB brings substantial benefits for long-duration applications dependent on replicated data. The evaluation results also show that data imbalance in HDFS directly affects data locality, underscoring the importance of regular and strategic balancing interventions. The performance enhancements observed in scenarios where balancing is frequently conducted, specially after data loads and node additions in environments characterized by frequent changes and growth, highlight the importance of dynamic and responsive balancing mechanisms like DARB.

In conclusion, our study reaffirms the significance of effective replica balancing in HDFS, not only for maintaining data equilibrium but also for ensuring efficient system performance. As the demand for data-intensive applications continues to increase, we hope the insights from this research will help shape the evolution of data balancing strategies in HDFS. Looking forward, future work could focus on adapting the developed balancing architecture for new environments and file systems. Additionally, we aim to integrate predictive models to foresee imbalance scenarios and address them preemptively. Such advancements could further enhance the efficiency and stability of replica balancing, contributing significantly to the field of distributed file systems.

References

Cloudera, Inc. (2021). Managing data storage. `https://docs.cloudera.com/runtime/7.2.12/scaling-namespaces/topics/hdfs-balancing-data-across-hdfs-cluster.html`. November.

Dai, W., Ibrahim, I., and Bassiouni, M. (2017). An improved replica placement policy for Hadoop Distributed File System running on cloud platforms. In *4th Int. Conf. on Cyber Security and Cloud Computing (CSCloud)*, pages 270–275, New York. IEEE.

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*, pages 2853–2858, Boston. IEEE.

Fazul, R. W. A. and Barcelos, P. P. (2021). Automation and Prioritization of Replica Balancing in HDFS. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, SAC '21, page 235–238, New York, NY, USA. ACM.

Fazul, R. W. A. and Barcelos, P. P. (2022a). An Event-Driven Strategy for Reactive Replica Balancing on Apache Hadoop Distributed File System. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, SAC '22.

Fazul, R. W. A. and Barcelos, P. P. (2022b). The HDFS Replica Placement Policies: A Comparative Experimental Investigation. In *Distributed Applications and Interoperable Systems*, pages 151–166, Cham. Springer International Publishing.

Fazul, R. W. A. and Barcelos, P. P. (2023). PRBP: A prioritized replica balancing policy for HDFS balancer. *Software: Practice and Experience*, 53(3):600–630.

Foundation, A. S. (2023). Apache Hadoop – HDFS Architecture. `https://hadoop.apache.org/docs/r3.3.6`. October.

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

Joshi, B. Y., Sawai, D., et al. (2022). Performance Tuning Of Apache Spark Framework In Big Data Processing with Respect To Block Size And Replication Factor. *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology*, 14(02):152–158.

Liu, Z., Hua, W., Liu, X., Liang, D., Zhao, Y., and Shi, M. (2021). An Efficient Group-Based Replica Placement Policy for Large-Scale Geospatial 3D Raster Data on Hadoop. *Sensors*, 21(23):8132.

Shah, A. and Padole, M. (2018). Load Balancing through Block Rearrangement Policy for Hadoop Heterogeneous Cluster. In *International Conference on Advances in Computing, Communications and Informatics*, 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*.

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., 4 edition.

Xu, C. and Lau, F. C. (1996). *Load balancing in parallel computers: theory and practice*, volume 381. Springer Science & Business Media.

Yin, Y. and Deng, L. (2022). A dynamic decentralized strategy of replica placement on edge computing. *International Journal of Distributed Sensor Networks*, 18(8):9.

Zhang, Q., Zhang, S. Q., Leon-Garcia, A., and Boutaba, R. (2015). Aurora: Adaptive block replication in distributed file systems. In *2015 IEEE 35th International Conference on Distributed Computing Systems*, pages 442–451, New York. IEEE, IEEE.