HAL
archives-ouvertes.fr

How fast can one resize a distributed file system?

Nathanael Cheriere, Matthieu Dorier, Gabriel Antoniu

To cite this version:

Nathanael Cheriere, Matthieu Dorier, Gabriel Antoniu. How fast can one resize a distributed file system?. Journal of Parallel and Distributed Computing, Elsevier, 2020, 140, pp.80-98. 10.1016/j.jpdc.2020.02.001 . hal-02961875

HAL Id: hal-02961875
https://hal.archives-ouvertes.fr/hal-02961875

Submitted on 8 Oct 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

How Fast Can One Resize a Distributed File System?

Nathanaël Cheriere[a,*], Matthieu Dorier[b,*], Gabriel Antoniu[a,*]

[a] *Univ Rennes, Inria, CNRS, IRISA, Rennes, France*
[b] *Argonne National Laboratory, Lemont, IL, USA*

Abstract

Efficient resource utilization becomes a major concern as large-scale distributed computing infrastructures keep growing in size. Malleability, the possibility for resource managers to *dynamically* increase or decrease the amount of resources allocated to a job, is a promising way to save energy and costs.

However, state-of-the-art parallel and distributed storage systems have not been designed with malleability in mind. The reason is mainly the supposedly high cost of data transfers required by resizing operations. Nevertheless, as network and storage technologies evolve, old assumptions about potential bottlenecks can be revisited.

In this study, we evaluate the viability of malleability as a design principle for a distributed storage system. We specifically model the minimal duration of the commission and decommission operations. To show how our models can be used in practice, we evaluate the performance of these operations in HDFS, a relevant state-of-the-art distributed file system. We show that the existing decommission mechanism of HDFS is good when the network is the bottleneck, but can be accelerated by up to a factor 3 when storage is the limiting factor. We also show that the commission in HDFS can be substantially accelerated. With the highlights provided by our model, we suggest improvements to speed both operations in HDFS. We discuss how the proposed models can be generalized for distributed file systems with different assumptions and what perspectives are open for the design of efficient malleable distributed file systems.

Keywords: Elastic Storage, Distributed File System, Malleable File System, Model, Decommission, Commission

*Corresponding authors
Email addresses: nathanael.cheriere@irisa.fr (Nathanaël Cheriere), mdorier@anl.gov (Matthieu Dorier), gabriel.antoniu@inria.fr (Gabriel Antoniu)

1. Introduction

Reducing idle resources is a major concern for large-scale infrastructures such as clouds and supercomputers, as it directly leads to lower energy consumption and lower costs for the end user. Job malleability is an effective means to reduce idle resources. It consists of having the resource manager dynamically increase or decrease the amount of resources allocated to a job.

Resource managers for malleable jobs [1, 2] and malleable frameworks [3, 4, 5] have been proposed in earlier work. However, with the advent of BigData, many applications require *co-located data storage* (each computing node also stores data locally). But such co-located distributed storage systems[1] limit job malleability since they are typically not malleable. Since the storage system is co-deployed with the application, the application cannot be shrunk below the size of the storage system, and the distributed storage system becomes a bottleneck once the application expands above a certain size.

Most distributed storage systems already provide basic *commission* and *decommission* operations (consisting in adding and removing nodes, respectively), usually for maintenance purposes. They are, however, rarely used in practice for optimizing resource usage since they are assumed to have high performance overhead.

Since networks and storage devices are regularly improved (e.g., SSDs, NVRAM, or even in-memory file systems [6]), it is time to revisit this assumption and evaluate whether malleability can be leveraged for resource usage optimization in future distributed storage systems. Fast storage rescaling can indeed favor a quick response to new requests for resources and to sudden variations in workload.

In this paper, we focus on the cost of commission and decommission operations in the context of distributed storage systems that leverage data replication. We devise theoretical, implementation-independent models of the minimal duration required to undertake these operations. These models provide a baseline to evaluate implementations of these operations. As a case study, we evaluate how HDFS, a representative

[1] This work can easily be applied not only to distributed *file* systems but also to various kinds of distributed storage systems (e.g, object-based). We thus use the generic denomination *distributed storage systems*.

state-of-the-art distributed file system with both the commission and decommission mechanisms already implemented, behaves against these theoretical models. We show that the decommission mechanism of HDFS is efficient when data is stored in memory but can be improved by up to a factor of 3 when data is stored in secondary storage. Moreover, we show that these models can be used to predict decommission times when they are instantiated based on real measurements. We also show that the commission algorithms used by HDFS are not optimized to be fast. We suggest modifications that would further improve commission and decommission times in HDFS. Finally, we discuss the applicability of our models to other storage systems with different design assumptions.

This paper extends our previous work [7] focused on decommission only. In this paper we complete our previous work by providing a model for the commission, together with a study of the commission mechanism implemented in HDFS.

We discuss the relevance of malleability in Section 2 and present the related work in Section 3. We specify the assumptions used for the models in Section 4 and establish the models of the minimal duration for the commission and the decommission in Sections 5 and 6. We evaluate the performance of the these mechanisms in HDFS (Sections 7 and 8). The generality and usefulness of the proposed models are discussed in detail in Section 9.

2. Context and motivation

In this section, we discuss the relevance of malleability in general and for storage systems specifically.

2.1. Relevance of malleability

Malleable jobs are jobs for which the amount of computing resources can be increased or decreased in reaction to an external order, without needing to be restarted. Malleability is an effective means to reduce the amount of idle resources on a platform. It helps users save money on cloud resource rental or make better usage of the core-hours allocated to them on supercomputers. It also gives the platform operator more resources to rent, while cutting down the energy wasted by idle resources.

Some frameworks [3, 4, 5] provide support for malleability. However, few applications are malleable in practice. Many workflow execution engines such as the ones

presented by Wilde et al. [8] make workflows malleable: each job can be executed on any resource; and once the jobs running on a node are finished, the node can be given back to the resource manager. However, efficient support for malleable storage is missing.

Note that the notion of *malleability*, coming from the scheduling field, differs from *horizontal scalability*. Malleability refers to the possibility for a job to be dynamically resized, whereas scalability refers to the performance of a system at various scales.

2.2. Relevance of distributed storage system malleability

Having an efficient malleable distributed storage system, that is, a file system in which storage resources can be dynamically and efficiently added and removed, could benefit both cloud computing and HPC systems.

Cloud platforms. One of the key selling points of cloud platforms is their elasticity: one can get almost as many resources as needed. In practice, however, most applications must be stopped and restarted to use newly allocated resources. This lack of dynamism is in part explained by the inefficiency of existing mechanisms for resource commission and decommission and in part by the fact that some services deployed along with applications, in particular data storage services, are not themselves dynamic.

A malleable distributed storage system, able to add and release nodes *dynamically and efficiently without having to be restarted*, would enable truly dynamic elasticity on the cloud through on-the-fly resource reallocation. The gain in efficiency would be even larger when compute and storage resources are located on the same nodes; efficient dynamic commission and decommission would then operate for computation and storage at the same time.

HPC systems. Similar benefits can be expected in HPC. Recent works [9, 10] have proposed to push at least parts of storage systems to the compute nodes of HPC infrastructures. DeltaFS [9] pushes metadata management to compute nodes to offload storage clusters, while the work of Dorier et al. [10] enables the quick design and implementation of storage systems (deployed on compute nodes) tailored to applications. Implementing malleability in these works would enable their use in conjunction with malleable jobs run on HPC platforms [4, 11].

2.3. *Focus on the minimal duration of rescaling operations*

This paper aims to provide a *theoretical minimal duration* of commission and decommission operations. The models for these operations are useful in several situations. First, they help us understand where the bottlenecks of these operations are. Second, resources managers can use these models anticipate the impact of commission and decommission in order to decide when to add or remove nodes to/from a cluster. Third, they can be used as a baseline to compare against when implementing or optimizing these operations.

3. Related Work

Malleability has been explored in past work in various ways. Some work focused on implementing malleable applications [3, 4, 5]; some focused on resource managers able to exploit optimization opportunities available with malleable jobs [1, 2]; many efforts were dedicated to scheduling malleable jobs [12, 13]. Rare, however are papers focusing on the malleability of file systems.

Among them, the SCADS Director [14] is a resource manager that aims to ensure some service-level objective by managing data: it chooses when and where to move data, when and if some nodes can be added or removed, and the number of replicas needed for each file, thanks to the interface proposed by the SCADS file system. The authors of this work propose an algorithm for both node commission and decommission but do not study its efficiency. Their algorithm includes a decision mechanism for determining which nodes to add or remove. In contrast, we assume that the file system must follow commands from an independent resource manager, and we propose a model for the time of both operations that can be integrated in external scheduling strategies.

Lim, Babu, and Chase [15] propose a resource manager based on HDFS. It chooses when to add or remove nodes and the parameters of the rebalancing operations. However, it simply uses HDFS without considering the time of the commission or decommission operations. Both [14] and [15] focus on ways to leverage malleability rather than on improving it. They are complementary to our work.

Another class of file systems implements malleability to some extent: for example, Rabbit [16], Sierra [17], or SpringFS [18] shut down nodes to save energy. Be-

cause of the fault tolerance mechanism, these nodes must be ready to rejoin the cluster quickly, and the data cannot be erased from them. This difference has many important consequences. Instead of commissioning nodes, they update the information already present on the node, which is mainly a problem of data consistency, whereas we focus our attention on the time needed to distribute data. In the case of the decommission, their approach restricts the reallocation of these machines to other jobs. In our work we consider a more general case of node decommission, where the decommissioned nodes can be shut down but also allocated to new jobs without restrictions. This gives more freedom to the resource manager to reach its objective, whether it is saving energy, improving resource utilization, or maximizing gains.

The commission mechanism is a particular case of rebalancing the data on a cluster: some nodes are empty (the newly added ones), and the other ones host data, and after the commission we want all of them to host the same amount of data. Efficient data rebalancing has been extensively studied in different contexts: for RAID systems [19], for HDFS [20], or for hash tables [21]. To the best of our knowledge, however, no studies focus on the minimum duration of data rebalancing.

4. Assumptions

To be able to compute a theoretical minimal duration, we make several assumptions regarding the initial state of the storage system.

4.1. Scope of our study: which type of storage system?

The decommission mechanism is similar to the one often used for fault tolerance. When a node crashes, its data needs to be recreated on the remaining nodes of the cluster. Similarly, when a node is decommissioned, its data needs to be moved onto the remaining nodes of the cluster.

With this in mind, we reduce the scope of our study to storage systems using *data replication* as their fault tolerance mechanism. This crash recovery mechanism is highly parallel and is fast: most of the nodes share some replicas of the data with the crashed nodes and thus can send its data to restore the replication level to its original level. Moreover, this technique does not require much CPU power. It is used by HDFS [20] and RAMCloud [6].

We do not consider *full node replication*, used in systems where sets of nodes host exactly the same data, since the recovery mechanism is fundamentally different from the one used with data replication. Thus the models are not built for systems such as Ceph [22] and Lustre [23]. We discuss the case of node replication in Section 9.3.

We exclude from our scope systems using erasure coding for fault tolerance, such as Pelican [24], since such mechanisms require CPU power to regenerate missing data, and a theoretical minimal duration would therefore have to take into account the usage of the CPU to be as realistic as possible.

Another major fault tolerance mechanism for storage systems is lineage, used in Tachyon [25] for Spark [26]. We do not consider lineage in this paper because the base principles differ greatly from the ones needed for efficient decommission. With lineage, the sequence of operations used to generate the data is saved safely; and, in case of a crash, the missing data is regenerated. Consequently, a file system using lineage must be tightly coupled with a framework, and the CPU power needed to recover data depends on the application generating the data. The case of systems using erasure coding or lineage is discussed in Section 9.4.

4.2. Assumptions about the cluster infrastructure

We make three assumptions about the cluster in order to build a comprehensible model.

Assumption 1: Homogeneous cluster

All nodes have the same characteristics, in particular they have the same network throughput (S_{Net}) and the same throughput for reading from / writing to storage devices (S_{Read}, S_{Write}).

Moreover, we assume that either the network or the storage is the bottleneck for the commission or the decommission. Both operations heavily rely on data transfers, so those aspects are likely to be the bottlenecks.

Assumption 2: Ideal network

The network is full duplex, data can be sent and received with a throughput of S_{Net} at any time, and there is no interference.

This assumption determines the maximum throughput each node can individually reach. Thus, building upon this assumption ensures we model the fastest possible decommission. In order to build generic models, we do not consider the topology of the network and focus only on the I/O of each individual node.

The latency of the network is ignored in this assumption because of the large amount of data transferred during rescaling operations. Thus, transfer costs should be dominated by the bandwidth and the latency should be a negligible part of the transfer times.

Assumption 3: Ideal storage devices

Storage devices can either read or write, but cannot do both simultaneously. Also, the writing speed is not higher than the reading speed ($S_{Write} \leq S_{Read}$)

Assumption 3 holds for most modern storage devices.

Moreover, we assume that all resources are available for both the commission and the decommission operations without restrictions.

4.3. Assumptions on the initial distribution of the data

We denote as a *data object* the unit of information stored by users on the storage system (which can be files, objects, blobs, or even chunks of larger files). We distinguish data objects from the space they occupy on the storage devices. We denote the occupied storage space as simply *data*. The size of the data objects is not the same as the size of the data because of the replication. With a replication factor of r, the data is r times larger than the size of the data objects. Finally, we denote as a *replica* each copy of a data object and do not consider any hierarchy between replicas.

The initial *data placement* (the placement of all replicas of all data objects on the nodes of the storage cluster) is important for the performance of commissions and decommissions. Thus, we make assumptions in this respect.

Assumption 4: Data balance

Each node initially hosts the same amount of data D.

Assumption 4 matches the load-balancing target of policies implemented in existing file systems, such as HDFS [20] or RAMCloud [6].

Assumption 5: Uniform data replication

Each object stored in the storage system is replicated on $r \geq 1$ *distinct nodes.*

Assumption 5 simply states that data replication is the fault tolerance strategy used by the considered distributed storage systems. We include the case $r = 1$ (without replication) for reference.

We denote as *exclusive data* of a subset of nodes the data objects that have all their replicas on nodes included in the specified subset.

Assumption 6: Uniform data placement

All sets of r distinct nodes host the same amount of exclusive data, independently of the choice of the r nodes.

This assumption reflects the way data is placed on storage systems using data replication. In case of failures of up to $r-1$ nodes, each of the remaining nodes can participate equally to the system's recovery since they all host exactly the same amount of data that needs to be replicated. This situation is approached by some existing storage systems such as HDFS by randomly choosing the hosts of each replica.

4.4. Formalizing the problem

At the end of both rescaling operations (commissions and decommissions), the data placement should satisfy the following objectives.

Objective 1: No data loss

No data can be lost during either operation.

Objective 2: Maintenance of the replication factor

Each object stored on the storage system is replicated on r distinct nodes. Moreover, the replication factor of the objects should not drop below r during the operations.

Objective 3: Maintenance of data balance

All nodes host the same amount of data D' at the end of the operation.

Objective 4: Maintenance of a uniform data placement

All sets of r distinct nodes host the same amount of exclusive data, independently of the choice of the r nodes.

Objective 1 is obvious for a storage system. Objectives 2, 3, and 4 are the counterparts of Assumptions 5, 4, and 6 and are here to ensure a data placement that is the same as if the cluster always had its new size. Moreover, reaching the objectives also prepares the data placement for a future rescaling operation.

Both the assumptions and the objectives reflect the goal of the load-balancing policies implemented in many current state-of-the-art distributed file systems such as HDFS [20] or RAMCloud [6].

5. Modeling the Commission

In this section, we study the time needed to commission nodes in a cluster.

5.1. Problem definition

Commissioning (adding) nodes to a storage system involves two steps. First, the cluster receives a notification about nodes ready to be used. Second, the data stored in the cluster is balanced among all nodes to homogenize the load on the servers.

Ideally, at the end of the operation, the system should not have any traces of the commission; it should appear as if it always had the larger size. It is important in order to ensure a normal operating state, as well as to prepare for any operation of commission or decommission that could happen afterwards.

In this work, we look for the minimal duration t_{com} of the commission of x empty nodes (new nodes) to a cluster of N nodes (the old nodes). At the end of the commission, all objectives defined earlier (Objectives 1, 2, 3, and 4) must be satisfied.

5.2. Data to move

The commission is mainly a matter of transferring data from old to new nodes. In the following parts, the amount of data to transfer from sets to sets is quantified.

5.2.1. Data needed per new node

With the objectives of not losing data, of maintaining data replication, and of load-balancing (Objectives 1, 2, and 3), the fact that each of the N nodes initially host D data (Assumption 4) and that x new nodes are added, we deduce the following.

$$\textit{Each node must host } D' = \frac{ND}{N+x} \textit{ of data at the end of the commission.} \quad \textbf{(Prop. 1)}$$

5.2.2. Data needed by the new nodes

With the amount of data needed per node, we obtain the amount of data that must be written onto the new nodes (Prop. 2).

$$D_{\rightarrow new} = xD' = \frac{xND}{N+x} \quad \textbf{(Prop. 2)}$$

5.2.3. Required data movements from old to new nodes

Without considering data replication, the old nodes should transfer to the new nodes as much data as they need.

Because of data replication, however, some objects must have multiple replicas to be written on the new nodes. This requirement is particularly important because those objects could be sent once to new nodes and then forwarded from new nodes to new nodes to reduce the amount of data to send from old nodes to new ones.

Let us denote as p_i the probability that an object has exactly i replica(s) on the new nodes. Since we want a specific final distribution of data, those probabilities are known (Def. 1).

$$p_i = \begin{cases} \frac{\binom{x}{i}\binom{N}{r-i}}{\binom{N+x}{r}} & \forall 0 \leq i \leq r \\ 0 & \forall i > r \end{cases}. \quad \textbf{(Definition 1)}$$

The problem is modeled as an urn problem [27]: x white balls, N black ones, we extract r of them (Assumption of uniformity 6) and compute the probability that exactly i white balls are selected.

Minimum amount of data to read and send from old nodes. Of all unique data, only the part that has at least a replica to place on the new nodes must be moved. This amount is expressed as $D_{old \rightarrow new}$.

$$D_{old \to new} = \frac{ND}{r}(1 - p_0). \qquad \textbf{(Prop. 3)}$$

Data that can be moved from either new or old nodes to new nodes. Of course, reading and sending the minimum amount of data are not enough to complete the commission. The remaining data can be read either from old nodes or from new nodes after they receive the first replicas (from the old nodes). The total amount of this data is $D_{old/new \to new}$.

$$D_{old/new->new} = \frac{ND}{rx}(\frac{rx}{N+x} + p_0 - 1). \qquad \textbf{(Prop. 4)}$$

5.2.4. Data placement that does not involve data transfers between old nodes

The preceding sections focused on the data transfers to new nodes; however, data transfers between old nodes could compete with the essential ones.

To avoid data transfers between old nodes, we need to design a data redistribution scheme for the old and new nodes that has the following property: the data that was present initially on an old node is either staying on it or being transferred to new nodes.

1. Group objects according to the placement of their replica; *i.e.*, two objects whose replicas are on the same set of servers will be considered in the same group.
2. Divide {groups} according to the proportions in the new placement; *i.e.*, from a given group C of objects, select a proportion p_i (for all i in $[0,r]$) of objects that will be replicated i times in the new servers.
3. For each subdivision, assign the corresponding number of replicas to the new nodes uniformly and remove the same number of replicas from the old nodes uniformly.

Algorithm 1: Algorithm designed to rebalance data without transferring data between old nodes.

Assuming that objects can always be divided in multiple objects of any smaller size, Algorithm 1 avoids all data transfers between old nodes and satisfies all the objectives. **(Prop. 5)**

With this result, since no data transfers are required between old nodes, it will not have any impact on the minimal duration of the commission.

Of course, in practice objects cannot be indefinitely divided. So when relaxing the goals of load-balancing and uniform data distribution, as it is in practice, the data transfers between old nodes can be ignored.

5.3. A model when the network is the bottleneck

We can determine the time needed to transfer data from the amount of data. However, two cases must be considered, depending on the relative speed of the network with respect to that of the storage. In the first, a slow network is the bottleneck, and the nodes do not receive data fast enough to saturate the storage's bandwidth. In the second case, storage is slow and becomes a bottleneck (i.e., storage cannot write at the speed at which the data is received from the network).

In this section, we consider the case where the network is the bottleneck of the system.

5.3.1. Many possible bottlenecks

The operation of commission is composed of multiple concurrent actions such as sending and receiving data. Moreover, two strategies are possible when sending data: send the minimum amount of data from old nodes and forward it between new nodes, or balance the amount of data sent by the old and the new nodes.

Thus, we design a model of the minimal duration needed by each action; and then, because all actions must finish to complete the commission, we extract the maximum of the times required for each action to obtain the minimal time needed to commission nodes.

5.3.2. Receiving data

Each new node must receive D' data, with a network throughput of S_{Net}. Thus the time needed to do so is at least T_{recv}.

$$T_{recv} = \frac{ND}{(N+x)S_{Net}} \qquad \textbf{(Prop. 6)}$$

5.3.3. Sending the minimum from old nodes

If one chooses the strategy of sending as little data as possible from old nodes, the minimal time needed is $T_{old \to new}$.

$$T_{old \to new} = \frac{D}{rS_{Net}}(1 - p_0) \qquad \textbf{(Prop. 7)}$$

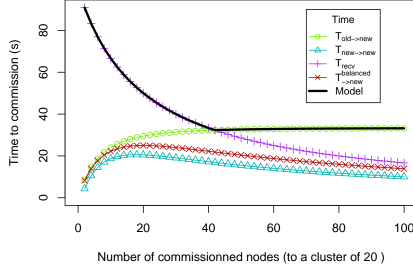

Figure 1: Important times when adding nodes to a cluster of 20 nodes each hosting 100 GB of data, $S_{Net} = 1\ GBps$, $r = 3$.

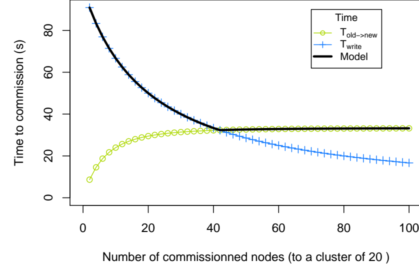

Figure 2: Important times when adding nodes to a cluster of 20 nodes each hosting 100 GB of data. $S_{Read} = S_{Write} = 1\ GBps$, $r = 3$.

Of course, sending the minimum from old nodes means that the new nodes will spend some time $T_{new\to new}$ to forward the data.

$$T_{new\to new} = \frac{ND}{rxS_{Net}} \sum_{i=2}^{r}(i-1)p_i \qquad \textbf{(Prop. 8)}$$

5.3.4. Balancing the sending operations between old and new nodes

The previous strategy can be easily improved when the new nodes spend more time forwarding data than the old nodes spend sending it (*i.e.* $T_{new\to new} > T_{old\to new}$). In this situation, the old nodes should send more data, thus reducing the amount that must later be forwarded by new nodes.

In that case, the minimum time required to send all the needed data to their destination is $T^{balanced}_{\to new}$.

$$T^{balanced}_{\to new} = \frac{xND}{(N+x)^2 S_{Net}} \qquad \textbf{(Prop. 9)}$$

$$T_{recv} \geq T^{balanced}_{\to new} \qquad \textbf{(Prop. 10)}$$

$T^{balanced}_{\to new}$ is always smaller than the time needed for the new nodes to receive data (Property 10). Thus, equally distributing the task of sending data between new and old node does not have any impact on the duration of the operation.

5.3.5. Commission time with a network bottleneck

The commission, in the case of a network bottleneck, cannot be faster than t_{com}.

$$t_{com} = max(T_{old\to new}, T_{recv}) \qquad \textbf{(Prop. 11)}$$

Indeed, the minimum commission time is at least as long as the time needed to receive the data and at least as long as the time needed to send it (balancing the sending operations between old and new nodes if needed).

In Figure 1, we can observe the different minimum times that have been used in constructing the model in the context of a 20-node cluster initially hosting 100 GB of data per node. When less than 40 nodes are added at once, the bottleneck is the reception of the data by the new nodes. When more than 40 nodes are added, however, the old nodes do not manage to send the data they have to send as fast as the new nodes can receive it, and thus the emission is the bottleneck.

5.4. A model when the storage is the bottleneck

In the case of a storage bottleneck, similar actions to the network bottleneck case are required (reading and writing data), but the time needed to finish each action depends on the characteristics of the storage devices.

In the following, the duration of each action are evaluated in the context of a storage bottleneck.

5.4.1. Writing data

Each new node must write D' data on its storage, it takes at least T_{write}.

$$T_{write} = \frac{ND}{(N+x)S_{Write}} \qquad \textbf{(Prop. 12)}$$

5.4.2. Reading the minimum from old nodes

The part of the data that must be read from old nodes can be read in at least $T_{old \to new}$.

$$T_{old \to new} = \frac{D(1-p_0)}{rS_{Read}} \qquad \textbf{(Prop. 13)}$$

5.4.3. When buffering is possible

If the data can be put in a faster buffer than the storage (typically from drive to memory), reading from memory is a lot faster than from disk and thus can be ignored.

In this case, the relevant objects are read once from the storage of the old nodes, sent to new nodes, stored in the storage and onto a buffer, and then forwarded from the buffer to other new nodes if needed.

In that case, the minimal duration of the commission t_{com} is defined as follows.

$$t_{com} = max(T_{write}, T_{old \to new}) \quad \textbf{(Prop. 14)}$$

In Figure 2, we can observe the different times that are important in constructing the model in the context of a 20-node cluster initially hosting 100 GB of data per node.

As in the case of a network bottleneck, when less than 40 nodes are added at once, the writing is the bottleneck. In contrast, when more than 40 nodes are added simultaneously, the old nodes are not numerous enough to read the unique data they must read as fast as the new nodes can write it.

5.4.4. When buffering is not possible

Buffering may not be usable, in particular in the case of in-memory storage (in this case, the buffer would have the same throughput as the main storage).

Because the storage cannot read and write at the same time (Assumption 3), the new nodes should prioritize their writing. When the number of commissioned nodes increases, however, the old nodes will spend more time reading all the data ($T^{alldata}_{old \to new}$) than the new nodes will spend writing it (T_{Write}).

$$T^{alldata}_{old \to new} = \frac{xD}{(N+x)S_{Read}} \quad \textbf{(Prop. 15)}$$

In this situation, the new nodes can spend some time to forward data to other new nodes and reduce the time ($T^{balanced}_{old \to new}$) needed to read data from the old nodes.

$$\textit{If } x \geq \frac{NS_{Read}}{S_{Write}}, \textit{ new nodes can forward data,}$$
$$\textit{and } T^{balanced}_{old \to new} = \frac{xND}{(N+x)^2} \frac{S_{Read} + S_{Write}}{S_{Read} S_{Write}}. \quad \textbf{(Prop. 16)}$$

Thus, we deduce the minimal duration of the commission when no buffering is possible.

$$t_{com} = \begin{cases} T_{write} & \text{if } x \leq \dfrac{NS_{Read}}{S_{Write}}, \\ max(T_{old \to new}, T^{balanced}_{old \to new}) & \text{else.} \end{cases} \quad \textbf{(Prop. 17)}$$

In Figure 3, we show the different times that are important in constructing the model, in our usual example of a 20-node cluster. When less than 20 nodes are added, the bottleneck is the new nodes not writing fast enough. When between 20 and 80 nodes are added at once, the old nodes cannot read all the data fast enough by themselves;

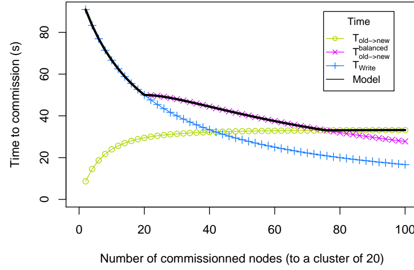

Figure 3: Important times when adding nodes to a cluster of 20 nodes each hosting 100 GB of data. $S_{Read} = S_{Write} = 1\ GBps$, $r = 3$.

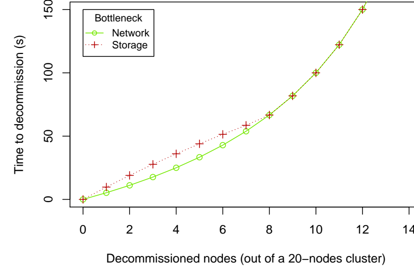

Figure 4: Time needed to transfer 100 GB from each of the leaving nodes on two different settings: either the network is the bottleneck and has a bandwidth of 1 GB/s, or the storage is the bottleneck and has read and write speeds of 1 GB/s. The cluster is composed of 20 nodes.

thus the new nodes start to forward part of the data they receive to other new nodes. But when more than 80 nodes are added at once, the new nodes do not manage to read the unique data they must read fast enough and are the bottlenecks.

5.5. Observation

In both cases of a storage bottleneck and a network bottleneck, the more nodes that are commissioned at once, the faster the operation finishes.

Thus, it is faster to add many nodes at once to match the workload than to add few nodes after few nodes until the workload is matched.

6. Modeling the Decommission

In this section, we study the time needed for the decommission and, in particular, provide a model of the minimal possible duration.

6.1. Problem definition

The decommission (removing) of nodes from a storage cluster is composed mainly of two steps. First, in order to ensure that no data is lost (Objective 1), the data is transferred out of the leaving nodes and sent to the remaining nodes. Second, the storage system on the leaving nodes is shut down, and these nodes are returned to the resource manager.

In this section, we design a model t_{decom} of the minimal time needed to decommission x nodes from a cluster of N nodes. More precisely, t_{decom} is the minimum time

needed between the reception of the order to decommission some nodes from the resource manager (thus the choice of the leaving nodes does not depend on the cluster) and the moment when the leaving nodes can be safely removed. At the end of the decommission, the remaining nodes must satisfy the objectives.

For the major part of this section, we assume a replication factor $r > 1$ since it is fundamentaly different from the model in the absence of replication. We build the model without replication in section 6.6.

6.2. A model for the decommission time

To establish a model for the minimal decommission time, we note that data writing is the bottleneck of this operation, for three reasons.

First, any remaining node shares some data with all the leaving nodes (Assumption 6); thus, any remaining node can read and send data at the same rate as leaving nodes. Second, only the remaining nodes can receive and write some data in their storage: since leaving nodes will be removed form the cluster, having them store more data is pointless. Third, storage devices have a lower writing speed than their reading speed (Assumption 3).

Thus, the theoretical minimal duration of the decommission is equal to the amount of data to write (D_{write}) divided by the writing speed of the whole cluster $S_{write}^{cluster}$.

$$t_{decom} = \frac{D_{write}}{S_{write}^{cluster}} \qquad \textbf{(Definition 2)}$$

6.3. Data to write

Since the replication factor must be left unchanged after the decommission (Objective 2), all data present on the leaving nodes must be written on remaining nodes. Thus the data to write D_{write} is known.

$$D_{write} = xD \qquad \textbf{(Prop. 18)}$$

6.4. Writing speeds

Determining the writing speed of the cluster is more complex. Two cases must be considered, as was done for the commission: either the network is the bottleneck, or the storage is.

6.4.1. First case: the network is the bottleneck

We assume the network is full duplex, and without interference (Assumption 2). In this case, the remaining nodes can receive data at the network speed S_{Net}, even if they send data at the same time. Each of the $N-x$ remaining nodes can receive and write data at the network speed S_{Net}. We denote as $S_{write}^{cluster}$ the aggregated speed at which the cluster can receive data during decommissions.

$$S_{write}^{cluster} = S_{Net}(N-x) \qquad \textbf{(Prop. 19)}$$

From this, we deduce the minimal duration of the decommission t_{decom}.

$$t_{decom} = \frac{xD}{S_{Net}(N-x)}. \qquad \textbf{(Prop. 20)}$$

6.4.2. Second case: the storage is the bottleneck

If the storage is the bottleneck ($S_{Read} \leq S_{Net}$), the situation is slightly different: most storage devices (disk, RAM, or NVRAM) cannot read and write at the same time (Assumption 3).

However, by using some buffering (reading once from the storage) and keeping a copy in memory, what is read can be written more than once. From this, we denote as $R(N,x)$ the ratio data written divided by data read.

$$p_i = \begin{cases} 0 & \text{if } i > r, \\ \frac{\binom{r}{i}\binom{N-r}{x-i}}{\binom{N}{x}} & \text{for } i \leq r. \end{cases} \qquad \textbf{(Definition 3)}$$

$$R(N,x) = \begin{cases} 1 & \text{without buffering,} \\ \frac{\sum_{i=1}^{r} i p_i}{\sum_{i=1}^{r} p_i} & \text{if other cases.} \end{cases} \qquad \textbf{(Prop. 21)}$$

The ratio $R(N,x)$ (Property 21) is expressed with the probability p_k of an object to have k replicas on the leaving nodes (Definition 3), which is a classical urn problem [27]. In that case, the data is read once but is written k times to ensure the replication factor.

There are two cases for the writing speed of the cluster because leaving nodes can only read (they will leave the cluster at the end of the decommission): either the leaving nodes can read enough data to saturate the writing on the remaining nodes, or they cannot.

In the first case, the writing speed of the cluster is defined as $S^{cluster}_{write}$.

$$S^{cluster}_{write} = S_{Write}(N-x). \tag{Prop. 22}$$

In the second case, the remaining nodes are not saturated by the amount of data received from the leaving nodes and thus can also read and write more data to accelerate the decommission. In this case, the writing speed of the cluster is defined as follows.

$$S^{cluster}_{write} = \frac{NR(N,x)S_{Read}S_{Write}}{S_{Write} + R(N,x)S_{Read}}. \tag{Prop. 23}$$

The leaving nodes are able to saturate the remaining nodes when more than $T(N,x)$ nodes are decommissioned at once. The threshold $T(N,x)$ can be expressed as follows.

$$T(N,x) = \frac{NS_{Write}}{R(N,x)S_{Read} + S_{Write}} \tag{Prop. 24}$$

With Properties 2, 18, 22, 23, and 24, the model for the time to decommission d_{decom} in the case of storage as a bottleneck is deduced.

$$t_{decom} = \begin{cases} \dfrac{xD}{S_{Write}(N-x)} + t_0 & \text{if } x \geq T(N,x), \\ \dfrac{x \cdot D \cdot (S_{Write} + R(N,x)S_{Read})}{N \cdot R(N,x) \cdot S_{Read} \cdot S_{Write}} + t_0 & \text{in other cases.} \end{cases} \tag{Prop. 25}$$

6.5. *Observations*

Three interesting observations can be made regarding the minimal duration.

6.5.1. *Impact of the data hosted per node*

The decommission time is proportional to the amount of data hosted per node. Thus, the decommission scales linearly with the amount of data hosted for a given platform.

Figure 4 summarizes the minimal decommission times for both kinds of bottlenecks, on an artificial platform that exposes the differences in behavior between both bottlenecks. The minimal decommission times expected for existing hardware can be found in Section 9.6.

6.5.2. *Impact of the proportion of decommissioned nodes*

In the case of a network bottleneck, the decommission time depends only on the proportion of leaving nodes in the cluster. In this situation, decommissioning 20 nodes in a cluster of 100 or 4 in a cluster of 20 will take the same time if each node hosts the same amount of data.

6.5.3. Decommission one by one or in batch

We have the following property.

In the case of a network bottleneck, decommissioning a set of nodes in k consecutive steps takes as much time as decommissioning the same nodes all at once. **(Prop. 26)**

The reason behind this unexpected result is that even if more data is transferred (data is moved to a node that is decommissioned later), the transfer speed is also higher. The bottleneck in this case comes from the amount of nodes that can write, which is higher in the first step than in the last steps. Note that this result cannot be found in the case where the bottleneck is at the storage level: the storage compensates for the nodes that cannot write and have a constant speed.

6.6. Decommission without replication

The proposed model only works when the data objects are replicated. Without replication, the model is simpler: only the nodes leaving can read and send data since they are the only ones hosting the data that needs to be transferred, and thus reading and sending the data can also be a bottleneck.

Thus, the model of the minimal duration of the decommission without replication is the maximum of the time needed to read and send the data to move and of the time needed to receive and write it. From this, we deduce the model for the decommission without replication (Property 27).

$$t_{decom} = \max\left(\frac{xD}{(N-x)min(S_{Write}, S_{Net})}, \frac{D}{min(S_{Read}, S_{Net})}\right) \qquad \textbf{(Prop. 27)}$$

7. Decommission in HDFS

In this section we use the previously defined models to evaluate commission and decommision in a practical setting: we focus on the case of HDFS, a relevant state-of-the-art distributed file system, in which these operations are implemented. We consider two cases of bottlenecks: HDFS with its storage in RAM (bottleneck at the network level); HDFS with storage on disk drives (bottleneck at storage level). For all configurations, we compare experimental measurements with the theoretical minimal duration

and propose improvements for the transfer scheduler of HDFS that would decrease the duration of the operation.

This section presents a study on decommission, where HDFS exhibits good practical results for this operation. A similar study for commission is presented in Section 8.

7.1. Experimental setup

7.1.1. Testbed

The experiments presented in this section have been performed on the Grid'5000 [28] experimental testbed. The *paravance* cluster from Rennes was used for the decommission measurements. Each node has 16 cores, 128 GB of RAM, a 10 Gbps network interface, and two hard drives. The file system's cache has been reduced to 64 MB in order to limit its effects as much as possible. It has not been completly disabled since HDFS relies on it to improve disk read and write performance. Unless stated otherwise, 20 nodes from this cluster were used for each experiment.

7.1.2. HDFS

We deployed HDFS and Hadoop 2.7.3. One node acted as both DataNode (slave) and NameNode (master) while the others were used only as DataNodes. One drive was reserved for HDFS to store its data. Most of the configuration was left to its default values, including the replication factor, which was left unchanged at 3.

The data on the nodes was generated using the *RandomWriter* job of Hadoop, which yields a typical data distribution for HDFS.

7.1.3. HDFS in memory

To experiment RAM-based storage with HDFS, we used the same setup as in the paper introducing Tachyon [25]: a tmpfs partition of 96 GB was mounted, and HDFS used it to store data. A tmpfs partition is a space in RAM that is used exactly (and natively by Linux systems) as a file system. It is seen as a drive by HDFS, but the speeds are a lot higher (6 GB/s reading and 3 GB/s writing), moving the bottleneck from the drives to the network.

7.2. Experiment protocol

In order to measure the decommission time of HDFS, a random subset of nodes was selected among the DataNodes except the one hosting the NameNode, and the

Table 1: Parameters used for the experiments.

Parameter	RAM setup	Disk setup	Default
dfs.namenode.decommission.interval	1	1	3
dfs.namenode.replication.work.multiplier.per.iteration	25	2	2
dfs.namenode.replication.max-streams-hard-limit	30	4	4

command to decommission those nodes was given to the NameNode. The recorded time is the time elapsed between the moment the NameNode receives the command and the moment the NameNode indicates that the data has been transferred and the decommission process is finished.

For all experiments, measurements were repeated 10 times. All shown boxplots represent, from top to bottom, the maximum observed value, the third quartile, the median, the first quartile, and the minimum.

Some parameters in the configuration of HDFS were optimized for the experiments: HDFS checked the decommission status every second (instead of 30 s by default) with the parameter *dfs.namenode.decommission.interval*. This gives us the decommission times with a precision of 1 s. Moreover, since HDFS schedules data transfers every 3 s, we used the parameters presented in Table 1 to schedule enough transfers to maximize the bandwidth utilization while avoiding unbalanced work distribution. The value of these parameters has been determined by doing a parameter sweep.

7.3. Decommission in HDFS: when the bottleneck is at the network level

To create a setup with a bottleneck at the network level, we configured HDFS with RAM-based storage (we could measure writing at 3 GB/s in memory, including an overhead induced by the file system, while transfers on the network are done at 1.1 GB/s).

7.3.1. Closeness of HDFS to the theoretical minimal duration

Figure 5 shows the decommission times observed for various amounts of data hosted per node and various numbers of nodes to decommission. In addition, the figure shows the theoretical minimum decommission time for this platform computed with the model presented in Section 6. The number of nodes that can be decommissioned is limited by the capacity of the cluster after decommission. Thus the maximum number of nodes that can be decommissioned is different depending on the amount of data stored per node.

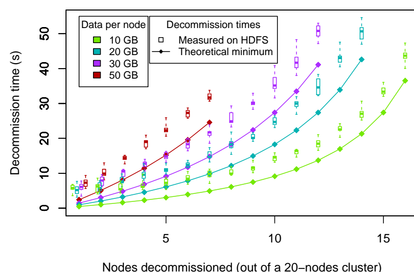

Figure 5: Decommission time measured on the platform presented in Sec. 7.3. The minimum theoretical time obtained with the model on this platform is added.

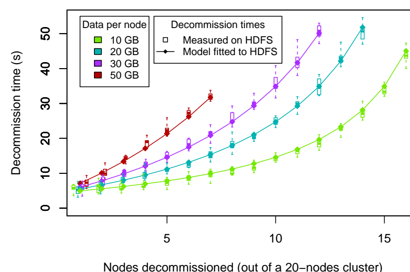

Figure 6: Model fitted to HDFS on the platform presented in Sec. 7.3. The model fits the data with a coefficient of determination r^2 of 0.98.

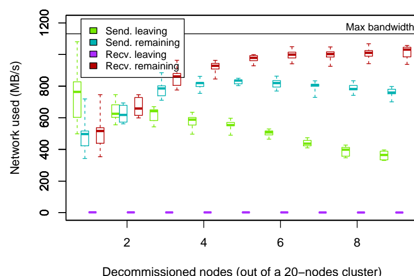

Figure 7: Average bandwidth measured for leaving and remaining nodes for both reception and emission on the platform presented in Sec. 7.3. Each node hosts 40 GB of data, and the maximum bandwidth was measured with a benchmark.

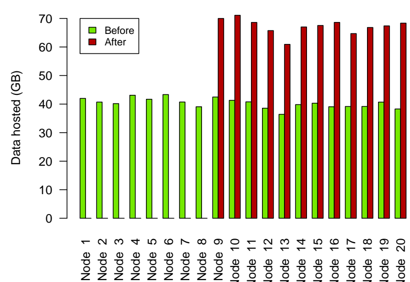

Figure 8: Amount of data before and after the decommission on the nodes of the cluster on the platform presented in Sec. 7.3. Data was generated for 40 GB per node, and 8 nodes were decommissioned.

We observe that the decommission times are short, especially for small numbers of decommissioned nodes. In particular, no decommission lasts more than 55 s. If we consider the scenario of KOALA-F [1] in which one or two nodes are commissioned or decommissioned every 5 minutes, the decommission would take less than 13 s each time, which is a cost of at most 5% of the time to save 5 to 10% of the energy and/or renting cost of the hardware. Moreover, we observe that the measured values are close to the theoretical minimal duration: the decommission mechanism of HDFS is fast, but can be improved.

7.3.2. Fitting the model to HDFS

Since the decommission in this case is close to the model, we show the model can be fitted to HDFS to estimate the duration of the decommission.

In Figure 6, we use linear regression to determine the values of S_{net} and t_0 that would fit the model and explain the decommission time of HDFS. The values obtained are $t_0 = 4.4$ s and $S_{net} = 0.98$ GB/s with a coefficient of determination of 0.983, which means that the variance in the measures is explained at 98.3% by the model with these parameters. These values indicate mainly that the decommission process uses 90% of the network bandwidth to receive data on the remaining nodes that is the main bottleneck and that there is a flat cost of 4.6 s.

The network bandwidth determined by the regression matches the observations, as we can see in Figure 7, when the transfer durations are long enough to have a steady transfer speed. The value of t_0 includes many delays due to the implementation of HDFS, such as the scheduling of the transfers done only every 3 s (on average 1.5 s delay) or the verification of the status of the decommission every second (on average 0.5 s delay). It also includes the imbalance in the scheduling that appears at the end of the transfers: because of the scheduler, some nodes have the maximum amount of transfers to do, while other have none.

Note that the model explains the decommission times of HDFS well even if some of the assumptions needed by the model are not fulfilled by HDFS: the data is not evenly distributed (see Figure 8), and the transfer speeds are not constant (see Figure 7, especially the reception speed that should not change). That explains why the value of t_0 is higher than expected: it compensates for the lower transfer speeds for small amounts of data transferred.

7.3.3. Improvement to the decommission time in HDFS

Although the duration of the decommission is close to the theoretical minimum, it can still be improved. Parameter tuning by reducing the heartbeat rate, increasing the transfer scheduling rate, and checking more often the status of the decommission could decrease the value of t_0. However, the scheduler should be redesigned to improve the bandwidth utilization that becomes important for large amounts of data transferred. Indeed, the current transfer scheduler of HDFS tries to balance the transfers on the sender side, ignoring the receivers; but, as the model shows, the bottleneck is the receiving side. Thus, load-balancing should be done considering primarily the receivers. All the above can serve for the design of future optimized transfer schedulers in HDFS (this is beyond the scope of this paper).

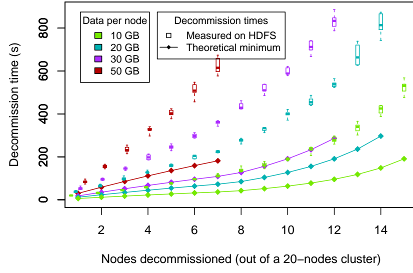

Figure 9: Decommission time measured on the platform presented in Sec. 7.4. The minimum theoretical time obtained with the model is added.

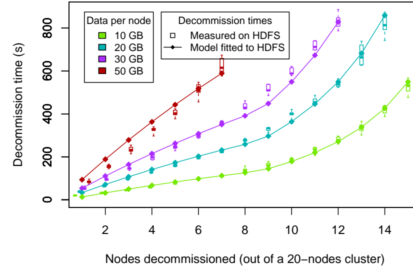

Figure 10: Model fitted to HDFS on the platform presented in Sec. 7.4. The model is enough to explain the performance of HDFS and has a coefficient of determination r^2 of 0.983.

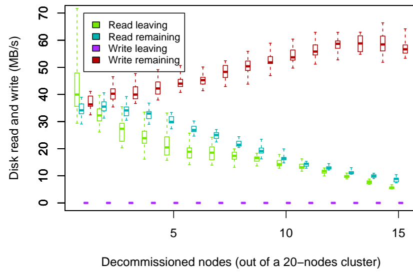

Figure 11: Average disk usage measured for leaving and remaining nodes on the platform presented in Sec. 7.4. Each node hosts 40 GB of data.

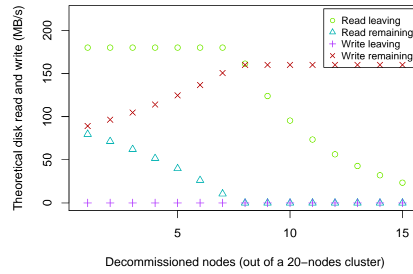

Figure 12: Theoretical disk utilization for leaving and remaining nodes obtained with the model on the platform presented in Sec. 7.4.

Overall, the decommission mechanism of HDFS is efficient when the network is the bottleneck: its performance is close to the theoretical limits.

7.4. Decommission in HDFS: when the bottleneck is at storage level

To create a setup where the bottleneck is at storage level, we configured HDFS to store data on the drive (read speed: 180 MB/s, write speed: 160 MB/s), a lot slower than the network (1.1 GB/s).

7.4.1. Closeness of HDFS to the theoretical minimal duration

Figure 9 shows the decommission times observed and the minimal theoretical time to do so. As we can see, even if the measures follow the same trends as the theoretical minimal duration, HDFS is about 3 times slower than what could be achieved on the platform.

In Figure 9 we present the measurements with the same configuration (number of nodes and data hosts per node) as the ones presented in Section 7.3 for better comparison, even if the technical constraint would allow larger experiments. In particular, when comparing with Figure 5, we observe that the decommission times are up to 20 times slower when using the drive. However, the drive should be only 13 times slower than the network in the worst case (reading and writing at the same time). This confirms that the decommission in this configuration is a lot less efficient than the one presented in Section 7.3.

7.4.2. Fitting the model to HDFS

Since the pattern of the measures follows that of the theoretical minimal duration, we use regression to fit the model for the results. The parameters obtained match a platform with a reading speed of 50.7 MB/s, a writing speed of 55.1 MB/s, and an initialization time t_0 of -3.55 s, and a coefficient of determination of 0.983 as shown in Figure 10. The negative initialization time is due to the fact that the transfer scheduler of HDFS balances reads instead of writes: this does not match the scheduling strategy expected by the model.

7.4.3. Possible Improvements to decommission in HDFS

HDFS schedules data transfers by balancing the reads and send operations, but the bottlenecks are the receive and write operations. Figure 11 shows that all nodes read data at approximately the same speed, resulting in high competition for the drive accesses on remaining nodes that must read and write data. In contrast, the disk of leaving nodes are underloaded since they do not write.

A scheduling strategy leveraging the model is simple: the scheduler should balance the writing operations, prioritize them, then maximize reading from leaving nodes. This would lead to read and write patterns like those presented in Figure 12. If remaining nodes can write all that is read by leaving nodes, then they also read to accelerate the decommission. If they cannot, the leaving nodes have their reading speed reduced while remaining nodes simply stop reading.

Since all these modifications can be made exclusively for the rescaling operations, they should not impact the performance of HDFS outside of rescaling operations.

When the bottleneck is at storage level, the decommission mechanism in HDFS

suffers from inappropriate scheduling: the scheduler balances the read load instead of the write load, and disk accesses are inefficient due to the resource contention. Some simple changes could make decommission in HDFS up to $3\times$ *faster.*

8. Commission in HDFS

In this section we evaluate the performance of the commission operation in HDFS against the model established in Section 5.

8.1. Experimental setup

The setup used to evaluate the commission of HDFS is the same as presented in Section 7.1 except that the *grisou* cluster of Grid'5000 located in Nancy has been used. This cluster has the same hardware as the one described in Section 7.1.

8.2. Experiment protocol

To measure the commission time of HDFS, we first deployed it on 10 nodes, and then a subset of the unused nodes in the cluster (with 2 to 30 nodes) was randomly selected and added to HDFS. HDFS does not rebalance the data by itself, however, thus we used the internal rebalancer to balance the data between new and old nodes. The recorded time is the time taken by the rebalancer to balance the data between old and new nodes, since adding nodes takes hardly any time compared with the time needed to balance the data among the nodes.

For all experiments with in-memory storage, measurements were repeated 10 times (these experiments lasted for 39 h). Because of the duration of the experiments, however, measurements for disk drive storage were repeated 5 times (the experiements lasted for 84 h).

8.3. Rebalancing algorithm used in HDFS

Algorithm 2 is used by HDFS to rebalance the data in a cluster. As done for the decommission, some parameters of this algorithm were adjusted to improve the commission time. The delay between two iterations was reduced from 9 s to 1 s. Moreover, HDFS checks whether the wave of transfers is finished only every 30 s; this delay has also been reduced to 1 s. The rebalancer limits both the throughput of each node used for rebalancing data and the number of concurrent data transfers. Both limits have been

Input: *threshold*: maximum difference between the ideal storage utilization on each node and the final one, provided by the user.

```
1  repeat
2  |  Compute the average storage utilization per available node on the cluster.
3  |  Cluster nodes according to their storage utilization (u):
4  |     if u > average + threshold then the node is Over-Utilized
5  |     else if u > average then the node is Above-Average
6  |     else if u > average − threshold then the node is Below-Average
7  |     else if u ≤ average − threshold then the node is Under-Utilized
8  |  Pair the nodes (source and target) with the following priority:
9  |     • Over-Utilized and Under-Utilized,
10 |     • Over-Utilized and Below-Average,
11 |     • Under-Utilized and Above-Average.
12 |  Select data to move from the source to the target:
13 |     • Target must not already host the same object.
14 |     • Data must not already be scheduled to move.
15 |  Execute the data transfers
16 |     • no more than threshold ∗ cluster_capacity amount of data is moved
   |       during each iteration.
17 |     • Replicas can be sent from the source or from another node hosting
   |       the replica.
18 |  Wait for all transfers to finish.
19 until All nodes are Above-Average or Below-Average
```

Algorithm 2: Algorithm used by HDFS to rebalance the data among the nodes, in the case of a single-rack configuration.

removed. The threshold of the rebalancing done by HDFS is set to 2%, which means that the rebalancing will stop if all nodes are within a 2% margin of the total node capacity of their ideal amount of data.

8.4. Commission in HDFS

Figures 13 and 14 show the time needed by HDFS to commission nodes to a cluster of 10 nodes with various amounts of data initially on the nodes.

Figure 13 presents the duration of the commission operation when the network is the bottleneck, while Figure 14 shows the duration of the commission operation when the storage (drives) is the bottleneck. Both figures show the same pattern: the theoretical minimal duration and the observed results are opposite of each other. The model suggests that the time needed to commission nodes should decrease as the number of added nodes increases, but the commission times of HDFS increase greatly as the number of new nodes grows.

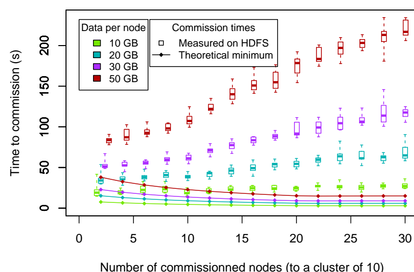

Figure 13: Commission time measured when there is a network bottleneck. The minimum theoretical time obtained with the model is added.

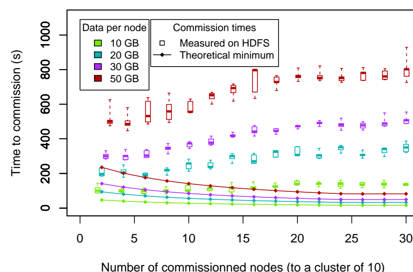

Figure 14: Commission time measured when there is a storage bottleneck. The minimum theoretical time obtained with the model is added.

These are not surprising results: the rebalancing algorithm is different from an optimal commission algorithm. The rebalancing algorithm was designed to balance the load across storage nodes, without the constraint that some of these nodes just arrived.

The results highlight the fact that this rebalancing algorithm was not designed to be fast but, on the contrary, to limit the impact of a rebalancing operation on the performance of HDFS. *Our model, on the other hand, has been designed with commission speed as the primary objective.*

8.5. Hints to improve the commission mechanism

The model highlighted two important bottlenecks: the reception (or writing) of the data and the old nodes sending (reading) data. Hence, in order to improve the commission in HDFS (or any distributed storage system using replication), the old nodes should send as little data as possible, while the reception of data on the new nodes should be balanced.

9. Discussion

In this section, we discuss various points about the models as well as the assumptions.

9.1. Is the theoretical minimal duration for commission too optimistic?

The observed times to commission nodes in HDFS greatly differ from the theoretical minimal duration, thus raising a natural question: Isn't this model too optimistic?

To answer this question, we developed a benchmark, Pufferbench [29], for the commission and decommission processes. Results show that one can design a commission algorithm that exhibits performance close to the theoretical minimal duration: on average, the commission operation can be 16% slower than the theoretical minimal duration, whereas in HDFS it is 440% slower.

9.2. How dependent are these models on HDFS?

The models are generic and do not rely on HDFS. They model the minimal duration of rescaling operations of distributed storage systems that follow the system assumptions: they balance their data across the nodes (Assumption 4), they replicate their data (Assumption 5), and they place their data uniformly across the nodes (Assumption 6). Moreover, the hints given to improve the transfer scheduler of HDFS can also be used to improve the duration of both operations in any distributed storage system using data replication.

The main storage systems matching these assumptions are GFS [30] and HDFS. However, the model can also be applied outside of distributed storage systems. For instance, Kafka [31] stores streams of records reliably using data replication as a fault tolerance mechanism. Thus, the duration of rescaling operations in Kafka can be modeled using this work.

9.3. What about systems using node replication?

The minimal duration of rescaling operations in distributed storage systems relying on node replication can be obtained by extending the presented work. To build these models, we follow the same assumptions on the platform as the other models (Assumptions 1, 2, and 3), we also assume that data can be buffered.

In a typical configuration, the nodes of a distributed storage system using node replication are grouped in *placement groups*. All nodes in a single placement group are mirrors from one another and host exactly the same data.

Commissioning nodes to a placement group is limited only by the duration of the reception and writing of data on the new nodes. With D as the amount of data on a single node of the placement group, the duration of the commission is

$$t_{com} = \frac{D}{\min(S_{Net}, S_{Write})}.$$

The decommission of nodes in a placement group is simple, since the other nodes of the group already host the same data as the ones leaving, no data movement are required. Thus, $t_{decom} = 0$.[2]

The models of the time required to commission or decommission placement groups can be deduced from the ones presented in this work. Assuming all placement groups are composed of k nodes, each placement group can be abstracted as a single node with a storage reading speed of $min(k * S_{Read}, k * S_{Net})$ (data can be read and sent from any of the k nodes of a placement group), and a storage writing speed of $min(S_{Net}, S_{Write})$ (data must be received and written on each node). From this abstraction, the duration of rescaling operations is obtained using the models with a storage bottleneck and $r = 1$. The fact that there is no difference between a network and storage bottleneck when $r = 1$ allows to include the difference between the outgoing and ingoing network bandwidth of the placement groups.

We leave the case of placement groups with varying sizes and the practical study of the commission and decommission of systems using node replication for future work.

9.4. What about systems using erasure coding or lineage?

In the case of systems using erasure coding or lineage, recreating data using the fault tolerance recovery mechanism is likely to be slower than moving data out of the nodes to decommission or to the newly commissioned nodes. Thus, if the specificity of their fault tolerance mechanism is not leveraged during rescaling operations, the models without replication ($r = 1$) can be used.

Note, however, that faster rescaling operations may be possible if the fault tolerance mechanisms are leveraged. For example, with lineage, one could recompute large amount of data and thus avoid many data transfers.

9.5. How useful are the models?

The models helps particularly in understanding the bottlenecks of the operations and thus gives hints to optimize it in distributed storage systems such as HDFS.

A more interesting utilization we foresee would consist of using the estimated time for the commission or decommission to efficiently and dynamically schedule resource

[2] If nodes are reallocated between placement groups (*e.g.* to balance the number of nodes per group), the duration of the operation can be modeled as a commission since the decommission is instantaneous.

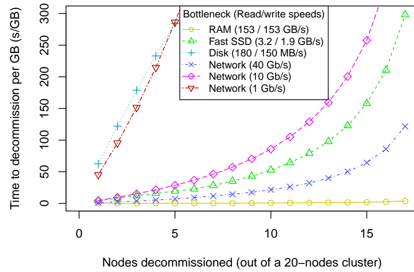

Figure 15: Minimal decommission time (for 100 GB per node) for different existing technologies on a 20-node cluster.

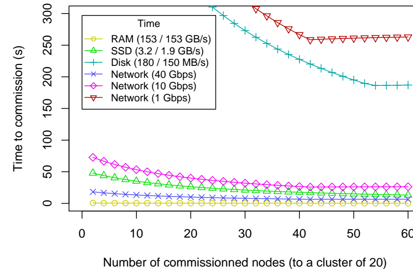

Figure 16: Minimal commission time (for 100 GB per node) for different existing technologies on a 20-node cluster.

allocations for malleable applications. With these models, resource schedulers can more easily anticipate the commission or decommission times of nodes, or even estimate whether it is interesting to add nodes that will soon be taken back.

Since these are realistic models of the commission and decommission times, they can be used as a baseline to evaluate the implementation of such mechanisms in distributed storage systems in general.

These models were essential in the development of Pufferscale [32, 33], a rescaling manager that organize data transfers during rescaling operations while balancing the load across the cluster. Knowing the duration of the operation enables to schedule additional data transfers to improve load balancing, without increasing the duration of the operation.

9.6. Can we predict operation times for various technologies?

Since the models are generic, one can use them to predict the (de)commission times that could be reached when other storage technologies, existing or emerging, are used. As an example, Figure 15 illustrates expected decommission times for various settings: storage bottleneck with RAM (from the Cray XC series [34]), drive (see Section 7.1), and one of the fastest SSDs [35] and network bottleneck with different bandwidths. In Figure 16, the minimum commission time for the same hardware is presented.

From these figures we can see that the commission and decommission times decrease with newer technologies, strengthening the idea that malleable file systems can currently be useful as the cost of the malleability is decreasing.

9.7. *Why is distributed storage system malleability important?*

A malleable distributed storage system that stores its data on HDDs can be considered too slow to be useful, especially with the network bandwidth available that negates the need for local storage. However, if we consider faster storage (that would make the network the bottleneck) such as some of the fast SSDs or even the RAM, the local storage improves the performance of I/O-intensive applications. Moreover, having malleability allows applications to easily dig into unused yet available resources or to avoid having idle resources. Thus a malleable distributed storage system on fast storage in such a setting would make an application able to exploit all available resources to their maximum.

The Cybershake workflow [36] is a good example that would greatly benefit from a malleable execution engine and file system. This workflow is malleable since each of its 815,000 short jobs can be launched on any resources. Moreover, it writes only 920 GB of data but reads 217 TB of it [37]. This amount of data could easily be stored on a few nodes with large amounts of RAM, and the read operations would be greatly accelerated using by local storage.

To conclude, malleability is getting increasingly interesting as a means to better use resources on shared platforms. Thus, even if the malleability of distributed storage systems is currently limited, we confirm in this paper that its cost is low (except for HDDs), and we provide information that can be integrated in scheduling strategies for malleable jobs.

9.8. *Would the models be relevant if a workload is present?*

The models presented in Sections 5 and 6 assume that all resources are available for the resizing operation, but this is often not the case. Some implementations might limit the bandwidth used for the operations or give a lower priority to the commission or decommission to favor the execution of applications. In both cases, this choice is made when implementing the distributed file system. Thus, one can add trade-offs such as limiting the bandwidth available for the data transfers of the commission or decommission, and hence reduce the S_{Net} accordingly.

9.9. *Can any of the assumptions be relaxed?*

The models are based on many assumptions and objectives, many of which are common among file systems, but one in particular could be relaxed for the decommis-

sion. If the user prioritizes the decommission time over fault tolerance, one need not to always maintain the replication factor (Objective 2).

In [38], we studied the possibility to release nodes faster by allowing the system to lower the number of replicas during decommission. This method is promising in the case of a network bottleneck. However, in the case of a storage bottleneck, it comes with drawbacks such as a longer time needed by the storage system to recreate all replicas, and an uncertain reduction in cost (energetical and financial), on top of the temporary reduction of fault tolerance.

9.10. *Can the uniformity assumption be ignored?*

The models are built on the assumption that the data is uniformly distributed among the nodes. This is almost impossible to do in practice. Systems such as HDFS place data using randomness in order to have a distribution of the data close to uniformity. We have seen in Section 7 that HDFS has performance close to the theoretical minimal duration for the decommission (Fig. 5), even if the data distribution (Fig. 8) is not uniform among the nodes.

If one does not want to use this assumption, the amount of data D can be set to the minimum amount of data hosted by a single node, which guarantees that it is a theoretical minimal duration.

9.11. *How can one determine where the bottleneck is?*

Determining which of the storage and the network is the bottleneck is necessary in order to know which model to use. However, estimating which one is the bottleneck can be hard. A rough estimation would be the following. The network is the bottleneck if it limits at any point the reading or writing of data from storage: $S_{Net} < S_{Read}$. Conversely, the bottleneck is located at the storage level if the read/write speeds cannot keep up with the speed at which data is sent and received through the network: $\frac{S_{Read} \cdot S_{Write}}{S_{Read} + S_{Write}} < S_{Net}$. Note that it does not included the possibility of buffering data in memory.

There is also a possibility of having bottlenecks both at the storage and the network level at the same time. We leave this less-intuitive situation outside the scope of this study.

10. Conclusion

Efficient commission and decommission of nodes are essential to enable the design of malleable distributed storage systems. To the best of our knowledge, this is the first study that provides a model of the minimal duration of these operations, regardless of their implementation. Using these generic models, we study the commission and decommission of HDFS and highlight potential improvements thanks to the better understanding of the various possible bottlenecks of the operations. For the decommission, we show that the only resources needed are network and drive bandwidth and that the model can be used to predict decommission times. For the commission, we show that the implementation of the mechanism in HDFS is not optimized for speed and could be greatly accelerated. Finally, we discuss the generality of these models to systems with different assumptions and we discuss how they can provide a base for putting malleability in practice in future distributed storage system.

A further challenge left for future work is the implementation of an efficient malleable distributed storage system (where commision and decommision perform closely to the identified theoretical minimal duration), and to subsequently evaluate its benefits on real-world applications.

Acknowledgment

The work resented in this paper is the result of a collaboration between the KerData project team at Inria, and Argonne National Laboratory, in the framework of the Data@Exascale Associate team, within the Joint Laboratory for Extreme-Scale Computing (JLESC, `https://jlesc.github.io`).

Experiments presented in this paper were carried out on 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`).

This material is based upon work supported by the U.S. Department of Energy, Office of Science under contract DE-AC02-06CH11357.

[1] A. Kuzmanovska, R. H. Mak, D. Epema, KOALA-F: A Resource Manager for

Scheduling Frameworks in Clusters, IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (2016) 592–595.

[2] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, Í. Goiri, S. Krishnan, J. Kulkarni, S. Roa, Morpheus: Towards Automated SLOs for Enterprise Clusters, USENIX Symposium on Operating Systems Design and Implementation (2016) 117–134.

[3] S. S. Vadhiyar, J. J. Dongarra, SRS: A Framework for Developing Malleable and Migratable Parallel Applications For Distributed Systems, Parallel Processing Letters 13 (2) (2003) 291–312.

[4] L. V. Kale, S. Kumar, J. Desouza, A Malleable-Job System for Timeshared Parallel Machines, IEEE/ACM International Symposium on Cluster Computing and the Grid.

[5] J. Buisson, F. André, J. Pazat, A Framework for Dynamic Adaptation of Parallel Components, International Conference Parallel Computing (2005) 1–8.

[6] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, R. Stutsman, The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM, ACM SIGOPS Operating Systems Review 43 (4) (2010) 92–105.

[7] N. Cheriere, G. Antoniu, How Fast Can One Scale Down a Distributed File System?, in: BigData 2017, 2017.

[8] M. Wilde, I. Foster, K. Iskra, P. Beckman, Z. Zhang, A. Espinosa, M. Hategan, B. Clifford, I. Raicu, Parallel Scripting for Applications at the PetaScale and Beyond, Computer 10 (2009) 50–60.

[9] Q. Zheng, K. Ren, G. Gibson, B. W. Settlemyer, G. Grider, DeltaFS: Exascale File Systems Scale Better Without Dedicated Servers, Parallel Data Storage Workshop (2015) 1–6.

[10] M. Dorier, P. Carns, K. Harms, R. Latham, R. Ross, S. Snyder, J. Wozniak, S. Gutierrez, B. Robey, B. Settlemyer, G. Shipman, J. Soumagne, J. Kowalkowski, M. Paterno, S. Sehrish, Methodology for the Rapid Development of Scalable HPC Data Services, in: 3rd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-

DISCS), 2018.

[11] S. Prabhakaran, M. Neumann, S. Rinke, F. Wolf, A. Gupta, L. V. Kale, A Batch System with Efficient Adaptive Scheduling for Malleable and Evolving Applications, International Parallel and Distributed Processing Symposium (2015) 429–438.

[12] K. Jansen, L. Porkolab, Linear-Time Approximation Schemes for Scheduling Malleable Parallel Tasks, Algorithmica 32 (2002) 507–520.

[13] G. Mounie, C. Rapine, D. Trystram, Efficient Approximation Algorithms for Scheduling Malleable Tasks, ACM Symposium on Parallel Algorithms and Architectures 3 (1999) 23–32.

[14] B. Trushkowsky, P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, D. A. Patterson, The SCADS Director: Scaling a Distributed Storage System under Stringent Performance Requirements, USENIX Conference on File and Storage Technologies (2011) 163–176.

[15] H. C. Lim, S. Babu, J. S. Chase, Automated Control for Elastic Storage, International Conference on Autonomic Computing (2010) 1–10.

[16] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, K. Schwan, Robust and Flexible Power-Proportional Storage, ACM Symposium on Cloud Computing (2010) 217–228.

[17] E. Thereska, A. Donnelly, D. Narayanan, Sierra: Practical Power-Proportionality for Data center Storage, Conference on Computer Systems (2011) 169.

[18] L. Xu, J. Cipar, E. Krevat, A. Tumanov, N. Gupta, C. Mellon, M. a. Kozuch, I. Labs, G. R. Ganger, S. Clara, SpringFS : Bridging Agility and Performance in Elastic Distributed Storage, USENIX Conference on File and Storage Technologies (2014) 243–255.

[19] A. Miranda, T. Cortes, CRAID: Online RAID Upgrades Using Dynamic Hot Data Reorganization., in: FAST, Vol. 14, 2014, pp. 133–146.

[20] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The Hadoop Distributed File System, IEEE Symposium on Mass Storage Systems and Technologies (2010) 1–10.

[21] P. B. Godfrey, I. Stoica, Heterogeneity and Load Balance in Distributed Hash Tables, in: INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer

and Communications Societies. Proceedings IEEE, 2005, pp. 596–606.

[22] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, C. Maltzahn, Ceph: A scalable, high-performance distributed file system, in: Proceedings of the 7th symposium on Operating systems design and implementation, USENIX Association, 2006, pp. 307–320.

[23] P. Schwan, et al., Lustre: Building a file system for 1000-node clusters, in: Proceedings of the 2003 Linux symposium, Vol. 2003, 2003, pp. 380–386.

[24] S. Balakrishnan, R. Black, A. Donnelly, P. England, A. Glass, D. Harper, S. Legtchenko, A. Ogus, E. Peterson, A. Rowstron, P. England, R. Black, A. Donnelly, A. Glass, D. Harper, A. Ogus, E. Peterson, A. Rowstron, Pelican: A Building Block for Exascale Cold Data Storage, in: Operating Systems Design and Implementation, 2014, pp. 351–365.

[25] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, I. Stoica, Reliable, Memory Speed Storage for Cluster Computing Frameworks, in: ACM Symposium on Cloud Computing, 2014, pp. 1 – 15.

[26] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, Spark: Cluster Computing with Working Sets, HotCloud 10 (10) (2010) 95.

[27] Y. Dodge, D. Commenges, The Oxford dictionary of statistical terms, Oxford University Press on Demand, 2006.

[28] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, L. Sarzyniec, Adding Virtualization Capabilities to the Grid'5000 Testbed, in: Cloud Computing and Services Science, Vol. 367, 2013, pp. 3–20.

[29] N. Cheriere, M. Dorier, G. Antoniu, Pufferbench: Evaluating and Optimizing Malleability of Distributed Storage, in: 3rd IEEE/ACM International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS), 2018, pp. 35–44.

[30] S. Ghemawat, H. Gobioff, S.-T. Leung, The Google File System, ACM SIGOPS Operating Systems Review 37 (5) (2003) 29.

[31] J. Kreps, N. Narkhede, J. Rao, et al., Kafka: A distributed messaging system for log processing, in: Proceedings of the NetDB, 2011, pp. 1–7.

[32] Pufferscale, gitlab.inria.fr/Puffertools/Pufferscale, Accessed 8/10/19.

[33] N. Cheriere, Towards Malleable Distributed Storage Systems: from Models to Practice, Ph.D. thesis, ENS Rennes (2019).

[34] Cray XC Series, www.cray.com/sites/default/files/Cray-XC-Series-Brochure.pdf, Accessed 19/06/17.

[35] NVMe SSD 960 PRO/EVO, www.samsung.com/semiconductor/minisite/ssd-/downloads/document/NVMe_SSD_960_PRO_EVO_Brochure_Rev_1_1.pdf, Accessed 19/06/17.

[36] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman, S. Callaghan, D. Okaya, Others, SCEC CyberShake Workflow - Automating Probabilistic Seismic Hazard Analysis Calculations, in: Workflows for e-Science, Springer, 2007, pp. 143–163.

[37] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and Profiling Scientific Workflows, Future Generation Computer Systems 29 (3) (2013) 682–692.

[38] N. Cheriere, M. Dorier, G. Antoniu, Is it Worth Relaxing Fault Tolerance to Speed Up Decommission in Distributed Storage Systems?, in: 19th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid), 2019.

Appendix

Lemma 1:

The probability of finding a specific object on a node is $\frac{r}{N}$ *with N the number of nodes in the cluster.*

Proof:

Let us compute the probability of finding an object O on a node A.

The number of sets of r nodes that contain node A is $\binom{N-1}{r-1}$. The probability of one of these sets to host O is $\frac{1}{\binom{N}{r}}$.

Thus, the probability of finding O on A is $\frac{r}{N}$.

QED

Property 1:

Each node must host $D' = \frac{ND}{N+x}$ *of data at the end of the commission.*

Proof:

Objectives 1 and 2 ensure that there is as much data on the cluster at the end of the commission as there was in the initial situation.

Objective 3 ensures that each node hosts the same amount of data.

Thus, the amount of data on a node at the end of the commission D' is the total amount of data on the cluster divided by the number of nodes:

$$D' = \frac{ND}{N+x}.$$

QED

Property 2:

$$D_{\to new} = xD' = \frac{xND}{N+x}$$

Proof:

There are x new nodes, and each hosts D' of data. Thus

$$D_{\to new} = xD' = \frac{xND}{N+x}.$$

QED

Definition 1:

$$p_i = \begin{cases} \frac{\binom{x}{i}\binom{N}{r-i}}{\binom{N+x}{r}} & \forall 0 \le i \le r \\ 0 & \forall i > r \end{cases}.$$

Detail:

The problem is modeled as an urn problem [27]: x white balls, N black ones. We extract r of them (Assumption 6) and compute the probability that exactly i white balls are selected.

QED

Lemma 2:

$$\sum_{i=0}^{r} p_i = 1$$

Proof:

All files have between 0 and r replicas on the new nodes.

QED

Lemma 3:

$\sum_{i=0}^{r} ip_i = \frac{xr}{N+x}$.

Proof:

The data stored on the new nodes at the end of the commission D_{new} can be expressed in two different manners:

- With the amount of data per node:

$$D_{new} = x\frac{ND}{N+x}$$

- With the probability of finding a replica on them:

$$D_{new} = \frac{ND}{r}\sum_{i=0}^{r} ip_i$$

Thus,

$$\sum_{i=0}^{r} ip_i = \frac{xr}{N+x}.$$

QED

Property 3:

$D_{old\to new} = \frac{ND}{r}(1-p_0)$.

Proof:

All the unique data that must be transferred to new nodes must be read from the old nodes.

$$D_{old\to new} = \frac{ND}{r}\sum_{i=1}^{r} p_i$$

$$D_{old\to new} = \frac{ND}{r}(1-p_0)$$

QED

Property 4:

$$D_{old/new->new} = \frac{ND}{rx}\left(\frac{rx}{N+x} + p_0 - 1\right).$$

Proof:

$D_{old/new \to new}$ is the amount of data that must be stored on new nodes $D_{\to new}$ minus the replicas that can be read only from old nodes $D_{old \to new}$.

$$\begin{aligned} D_{old/new \to new} &= D_{\to new} - D_{old \to new} \\ &= \frac{xND}{N+x} - \frac{ND}{r}(1 - p_0) \\ &= \frac{ND}{rx}\left(\frac{rx}{N+x} + p_0 - 1\right) \end{aligned}$$

QED

Property 5:

Assuming that objects can always be divided in multiple objects of any smaller size, Algorithm 1 avoids all data transfers between old nodes and satisfies all the objectives.

Proof:

Objectives 1 and 2 are satisfied by design since data is transferred from node to node.

No data transfers occur between old nodes by design.

Quantifying data transfers

Let S^r_{old} be the set of sets of r distinct old nodes.

S^r_{old} contains exactly $\binom{N}{r}$ elements.

Let A be a set of r distinct old nodes ($A \in S^r_{old}$).

Let D_A be the amount of data exclusive to A.

$$D_A = \frac{ND}{r\binom{N}{r}}$$

The second step of Algorithm 1 divides the exclusive data of A into $r+1$ distinct subsets of sizes D^0_A, ..., D^r_A.

$$\forall 0 \leq i \leq r, D_A^i = p_i D_A$$

Then, during the third step of Algorithm 1, for all $i \in [0, r]$, each new node receives a part d_A^i of the exclusive data stored on D.

$$\forall 0 \leq i \leq r, d_A^i = \frac{iD_A^i}{x}$$

During the same phase, each node in A loses dr_A^i of the exclusive data initially storage on A.

$$\forall 0 \leq i \leq r, dr_A^i = \frac{iD_A^i}{r}$$

Load-balancing (Objective 3)

Algorithm 1 assigns D'_{new} data to each new node. D'_{new} is the sum of all d_A^i for all possible i and A.

$$\begin{aligned}
D'_{new} &= \sum_{A \in S_{old}^r} \sum_{i=0}^{r} d_A^i \\
&= \sum_{A \in S_{old}^r} \sum_{i=0}^{r} \frac{iD_A^i}{x} \\
&= \binom{N}{r} \sum_{i=0}^{r} \frac{NDip_i}{rx\binom{N}{r}} \\
&= \frac{ND}{xr} \sum_{i=0}^{r} ip_i \\
&= \frac{xD}{N+x} (with\ prop.\ 3) \\
&= D'
\end{aligned}$$

Algorithm 1 leaves D'_{old} data to an old node n. D'_{old} is D minus the sum of all dr_A^i for all possible i and all A that include n.

Let $S_{old}^r(n)$ be the set of sets of r distinct nodes that include n. $S_{old}^r(n)$ contains $\binom{N-1}{r-1}$ sets.

$$
\begin{aligned}
D'_{old} &= D - \sum_{A \in S^r_{old}(n)} \sum_{i=0}^{r} dr^i_A \\
&= D - \sum_{A \in S^r_{old}(n)} \sum_{i=0}^{r} \frac{iD^i_A}{r} \\
&= D - \sum_{A \in S^r_{old}(n)} \sum_{i=0}^{r} \frac{NDip_i}{r^2\binom{N}{r}} \\
&= D - \binom{N-1}{r-1} \frac{ND}{r^2\binom{N}{r}} \sum_{i=0}^{r} ip_i \\
&= D - \binom{N}{r} \frac{r}{N} \frac{ND}{r^2\binom{N}{r}} \frac{xr}{N+x} \textit{(with prop. 3)} \\
&= D - \frac{xD}{N+x} \\
&= D'
\end{aligned}
$$

With this, the objective of load-balancing is satisfied.

Exclusive data (Objective 4)

Let A be a set of r distinct nodes.

Let k be the number of new nodes in A.

Let D_{ex} be the amount of exclusive data on A.

In order to show that the distribution satisfies objective 4, D_{ex} should be equal to $\frac{ND}{r\binom{N+x}{r}}$.

The amount of exclusive data on A is composed of objects that have $r-k$ replicas on old nodes and k replicas on new nodes. Before being assigned to the new nodes, the k replicas could have been on any other two old nodes. Sicne the algorithm does not move data between old nodes, however, the data present on the old nodes after the redistribution was initially on the same old nodes.

From this, we deduce that D_{ex} is the product of the following.

1. nb, the number of sets of r distinct nodes containing the $r-k$ old nodes of A;
2. D^k_A, the amount of data from a set of r distinct nodes that was assigned to exactly k new nodes by Algorithm 1;
3. p_{remain}, the proportion of that data that remains on the $r-k$ old nodes of A;
4. p_{distr}, the proportion of that data that is assigned to the k new nodes of A;

Using the urn problem, we have

$$p_{remain} = \frac{\binom{r-k}{r-k}\binom{k}{0}}{\binom{r}{r-k}} = \frac{1}{\binom{r}{r-k}}$$

$$p_{distr} = \frac{\binom{k}{k}\binom{x-k}{0}}{\binom{x}{k}} = \frac{1}{\binom{x}{k}}$$

$$nb = \binom{N-r+k}{k}.$$

Thus,

$$D_{ex} = nb \times D_A^k \times p_{remain} \times p_{distr}.$$

After simplification,

$$D_{ex} = \frac{ND}{r\binom{N+x}{r}}.$$

Thus, the objective of uniformity is satisfied.

QED

Property 6:

$$T_{recv} = \frac{ND}{(N+x)S_{Net}}$$

Proof:

$$T_{recv} = \frac{D'}{S_{Net}}.$$

Since $D' = \frac{ND}{N+x}$,

$$T_{recv} = \frac{ND}{(N+x)S_{Net}}.$$

QED

Property 7:

$$T_{old \to new} = \frac{D}{rS_{Net}}(1 - p_0)$$

Proof:

$$T_{old \to new} = D_{old \to new} / S_{Net}^{old},$$

where $S_{Net}^{old} = NS_{Net}$, the aggregated network speed of the old nodes.

Thus,

$$T_{old \to new} = \frac{D}{rS_{Net}}(1 - p_0).$$

QED

Property 8:

$$T_{new \to new} = \frac{ND}{rxS_{Net}} \sum_{i=2}^{r} (i-1) p_i$$

Proof:

$$T_{new \to new} = D_{old/new \to new} / S_{Net}^{new},$$

where $S_{Net}^{new} = xS_{Net}$, the aggregate network speed of the new nodes. Thus,

$$T_{new \to new} = \frac{ND}{rxS_{Net}} \sum_{i=2}^{r} (i-1) p_i$$

QED

Property 9:

$$T_{\to new}^{balanced} = \frac{xND}{(N+x)^2 S_{Net}}$$

Proof:

Let us denote as Y the amount of data that must be transferred between new nodes to balance send times for transfers from old to new nodes and between new nodes.

$$T^{balanced}_{old \to new} = \frac{D_{\to new} - Y}{S^{old}_{Net}} = \frac{1}{NS_{Net}}(x\frac{ND}{N+x} - Y)$$

$$T^{balanced}_{new \to new} = \frac{Y}{S^{new}_{Net}} = \frac{1}{xS_{Net}}Y$$

Then $T^{balanced}_{old \to new} = T^{balanced}_{new \to new}$, and thus $Y = \dfrac{x^2 ND}{(N+x)^2}$

QED

Lemma 4:

If $T_{old \to new} \leq T_{new \to new}$, *then*
$T_{old \to new} \leq T^{balanced}_{\to new} \leq T_{new \to new}$.

Proof:

Assuming $T_{old \to new} \leq T_{new \to new}$, we have the following.

$$T_{old \to new} \leq T_{new \to new}$$

$$1 - p_0 \leq \frac{N}{x} \sum_{i=2}^{r} (i-1) p_i$$

$$\sum_{i=1}^{r} p_i \leq \frac{N}{x} \sum_{i=1}^{r} (i-1) p_i$$

$$\sum_{i=1}^{r} p_i \leq \frac{N}{x} \sum_{i=1}^{r} i p_i - \frac{N}{x} \sum_{i=1}^{r} p_i$$

$$\frac{x}{N}(1 + \frac{N}{x}) \sum_{i=1}^{r} p_i \leq \sum_{i=1}^{r} i p_i$$

$$(\frac{x}{N} + 1) \sum_{i=1}^{r} p_i \leq \sum_{i=1}^{r} i p_i$$

$$
\begin{aligned}
&T^{balanced}_{\to new} - T_{old\to new} \\
&= \frac{D}{S_{Net}}\left(\frac{xN}{(N+x)^2} - \frac{1-p_0}{r}\right) \\
&= \frac{D}{rS_{Net}}\left(\frac{N}{N+x}\sum_{i=1}^{r} ip_i - \sum_{i=1}^{r} p_i\right) \text{ using the properties on } p_i \\
&\geq \frac{D}{rS_{Net}}\left(\frac{N}{N+x}\frac{N+x}{x}\sum_{i=1}^{r} p_i - \sum_{i=1}^{r} p_i\right) \text{ using the assumption} \\
&\geq 0
\end{aligned}
$$

$$
\begin{aligned}
&T_{new\to new} - T^{balanced}_{\to new} \\
&= \frac{D}{S_{Net}}\left(\frac{N}{rx}\sum_{i=2}^{r}(i-1)p_i - \frac{xN}{(N+x)^2}\right) \\
&= \frac{D}{S_{Net}}\left(\frac{N}{rx}\sum_{i=2}^{r}(i-1)p_i - \frac{N}{r(N+x)}\sum_{i=0}^{r} ip_i\right) \text{ with prop. 3} \\
&= \frac{ND}{xrS_{Net}}\left(\sum_{i=1}^{r} ip_i - \sum_{i=1}^{r} p_i - \frac{x}{N+x}\sum_{i=1}^{r} ip_i\right) \\
&= \frac{ND}{xrS_{Net}}\left(\frac{N}{N+x}\sum_{i=1}^{r} ip_i - \sum_{i=1}^{r} p_i\right) \\
&\geq \frac{ND}{xrS_{Net}}\left(\frac{N}{N+x}\frac{N+x}{N}\sum_{i=1}^{r} p_i - \sum_{i=1}^{r} p_i\right) \text{ using the assumption} \\
&\geq 0
\end{aligned}
$$

QED

Lemma 5:

If $T_{old\to new} \geq T_{new\to new}$, then
$T_{old\to new} \geq T^{balanced}_{\to new} \geq T_{new\to new}$.

Proof:

Basically the same as Lemma 4.

QED

Property 10:

$T_{recv} \geq T^{balanced}_{\rightarrow new}$

Proof:

From lemmas 4 and 5,

$$T_{recv} - T^{balanced}_{\rightarrow new} = \frac{N^2 D}{(N+x)^2 S_{Net}} \geq 0.$$

QED

Property 11:

$t_{com} = max(T_{old\rightarrow new}, T_{recv})$

Proof:

The commission time is the maximum between T_{recv} (time to receive data) and the time to send the data: $T_{old\rightarrow new}$ (if $T_{new\rightarrow new} \leq T_{old\rightarrow new}$) or $T^{balanced}_{old\rightarrow new}$ (if $T_{new\rightarrow new} \geq T_{old\rightarrow new}$).

After applying Properties 4, 5, and 10, we have

$$t_{com} = max(T_{old\rightarrow new}, T_{recv}).$$

QED

Property 12:

$$T_{write} = \frac{ND}{(N+x)S_{Write}}$$

Proof:

$$T_{write} = \frac{D'}{S_{Write}}.$$
$$T_{write} = \frac{ND}{(N+x)S_{Write}}.$$

QED

Property 13:

$$T_{old\to new} = \frac{D(1-p_0)}{rS_{Read}}$$

Proof:

$$T_{old\to new} = \frac{D_{old\to new}}{S^{old}_{Read}}.$$

where $S^{old}_{Read} = NS_{Read}$ is the aggregated reading speed of the old nodes.

Thus,

$$T_{old\to new} = \frac{D(1-p_0)}{rS_{Read}}.$$

QED

Property 14:

$$t_{com} = max(T_{write}, T_{old\to new})$$

Proof:

The commission cannot be faster than reading all unique data and writing it.

QED

Property 15:

$$T_{old \to new}^{alldata} = \frac{xD}{(N+x)S_{Read}}$$

Proof:

$$\begin{aligned} T_{old \to new}^{alldata} &= \frac{D'}{S_{Read}} \\ &= \frac{xD}{(N+x)S_{Read}}. \end{aligned}$$

QED

Property 16:

If $x \geq \frac{NS_{Read}}{S_{Write}}$*, new nodes can forward data,*
and $T_{old \to new}^{balanced} = \frac{xND}{(N+x)^2} \frac{S_{Read} + S_{Write}}{S_{Read} S_{Write}}$.

Proof:

Let y be the proportion of time used to write data on new nodes.

During the time t,

$D_o^R = tNS_{Read}$ data is read from the old nodes,

$D_n^R = t(1-y)xS_{Read}$ data is read from the new nodes,

$D_o^W = 0$ data is written on old nodes,

$D_n^W = tyxS_{Write}$ data is written on new nodes.

$$D_n^W + D_o^W = D_o^R + D_n^R$$

Thus, $y = \frac{(N+x)}{x} \frac{S_{Read}}{S_{Read}+S_{Write}}$ and $T_{old \to new}^{balanced} = \frac{D_{\to new}}{xyS_{Write}}$

$$T_{old \to new}^{balanced} = \frac{xND}{(N+x)^2} \frac{S_{Read} + S_{Write}}{S_{Read} S_{Write}}.$$

This is possible if and only if $y \leq 1$, thus $x \geq \frac{NS_{Read}}{S_{Write}}$.

QED

Property 17:

$$t_{com} = \begin{cases} T_{write} & \text{if } x \leq \frac{NS_{Read}}{S_{Write}}, \\ max(T_{old\to new}, T_{old\to new}^{balanced}) & \text{else.} \end{cases}$$

Proof:

If $T_{Write} \leq T_{old\to new}^{alldata}$, then the balanced strategy is used. Similarly to Property 4, we have

$$T_{Write} \leq T_{old\to new} \leq T_{old\to new}^{alldata}.$$

However, the old nodes still have to send the minimum amount of data for a duration of $T_{old\to new}$.

In the other case, writing is the bottleneck, and T_{Write} is the time needed for the commission.

$$t_{com} = max(T_{Write}, T_{old\to new}, T_{old\to new}^{balanced})$$

QED

Definition 2:

$$t_{decom} = \frac{D_{write}}{S_{write}^{cluster}}$$

Property 18:

$$D_{write} = xD$$

Proof:

Here x nodes are leaving the cluster, and each host D data (Assumption 4). Thus,

$$D_{write} = xD.$$

QED

Property 19:

$$S_{write}^{cluster} = S_{Net}(N-x)$$

Property 20:

$$t_{decom} = \frac{xD}{S_{Net}(N-x)}.$$

Proof:

Using Definition 2 as well as Properties 19 and 18, we obtain the result.

QED

Definition 3:

$$p_i = \begin{cases} 0 & \text{if } i > r, \\ \frac{\binom{r}{i}\binom{N-r}{x-i}}{\binom{N}{x}} & \text{for } i \leq r. \end{cases}$$

Detail:

This is the probability for each object to have exactly i replicas on the x leaving nodes. This is modeled as a classical urn problem [27].

QED

Property 21:

$$R(N,x) = \begin{cases} 1 & \text{without buffering,} \\ \frac{\sum_{i=1}^{r} i p_i}{\sum_{i=1}^{r} p_i} & \text{if other cases.} \end{cases}$$

Proof:

If an object has $k > 0$ replicas on leaving nodes, it needs to be read once using the buffering and written k times on remaining nodes.

The data to write $D_{towrite}$ is, for each possible number of replicas on leaving nodes, the probability for an object to have that number of replicas on leaving nodes multiplied by the total amount of data to move and the number of times this object has to be written.

$$D_{towrite} = xD \sum_{i=1}^{r} i p_i$$

Similarly, D_{toread} is the data to read: for each possible number of replicas on leaving nodes, the probability of the data having that number of replicas on leaving nodes multiplied by the total amount of data and the number of times the data has to be read, which is one.

$$D_{toread} = xD \sum_{i=1}^{r} p_i$$

With $D_{towrite}$ and D_{toread}, we can deduce the ratio of data to write with respect to the data to read as $R(N,x)$. The ratio is 1 in the case of storage in RAM, since data must be read as many times as it is written.

$$R(N,x) = \begin{cases} 1 & \text{for a storage in RAM,} \\ \frac{\sum_{i=1}^{r} i * p_i}{\sum_{i=1}^{r} p_i} & \text{if other cases.} \end{cases}$$

QED

Property 22:

$$S_{write}^{cluster} = S_{Write}(N-x).$$

Proof:

Each of the remaining nodes can receive data with a throughput of S_{Write} $S_{write}^{cluster} = S_{Write}(N-x)$.

QED

Property 23:

$$S_{write}^{cluster} = \frac{NR(N,x)S_{Read}S_{Write}}{S_{Write}+R(N,x)S_{Read}}.$$

Proof:

To obtain the writing speed of the cluster in case of storage bottleneck, we first consider the amount of data read and written during duration t. Leaving nodes can read at full speed, and remaining nodes can read and write. We denote as d the proportion of time that remaining nodes spend writing.

$$\begin{cases} data_written = & t\cdot(N-x)\cdot d\cdot S_{Write} \\ data_read = & t\cdot(N-x)\cdot(1-d)\cdot S_{Read} \\ & +t\cdot x\cdot S_{Read} \end{cases}$$

$$R(N,x) = \frac{(N-x)\cdot d\cdot S_{Write}}{(N-x)\cdot(1-d)\cdot S_{Read}+x\cdot S_{Read}}$$

Thus,

$$d = \frac{R(N,x)\cdot S_{Read}\cdot N}{(N-x)(S_{Write}+R(N,x)\cdot S_{Read})}$$

We find the following.

$$\begin{aligned} S_{write}^{cluster} &= (N-x)\cdot d\cdot S_{Write} \\ &= \frac{NR(N,x)S_{Read}S_{Write}}{S_{Write}+R(N,x)S_{Read}} \end{aligned}$$

QED

Property 24:

$$T(N,x) = \frac{NS_{Write}}{R(N,x)S_{Read} + S_{Write}}$$

Proof:

We consider d from the demonstration of Property 23.

Here d has two constraints: $0 \leq d \leq 1$. $d \geq 0$ means $S_{Read} \geq 0$ which is implicit, and $d \leq 1$ results in $T(N,x)$.

QED

Property 25:

$$t_{decom} = \begin{cases} \dfrac{xD}{S_{Write}(N-x)} + t_0 & \text{if } x \geq T(N,x), \\ \dfrac{x \cdot D \cdot (S_{Write} + R(N,x)S_{Read})}{N \cdot R(N,x) \cdot S_{Read} \cdot S_{Write}} + t_0 & \text{in other cases.} \end{cases}$$

Proof:

Using Properties 2, 18, 22, 23, and 24, and after simplification, we have the result.

QED

Property 26:

In the case of a network bottleneck, decommissioning a set of nodes in k consecutive steps takes as much time as decommissioning the same nodes all at once.

Proof:

We demonstrate that the time to decommission in k steps is $t_{decom}(k) = \dfrac{xD}{S_{Net}(N-x)}$ by recurrence.

For $k = 1$, decommission in one step is a simple decommission so they have the same duration.

For $k > 1$, we assume that decommissioning in $k-1$ steps takes $t_{decom}(k-1) = \dfrac{xD}{S_{Net}(N-x)}$. Then, we denote as y the number of nodes decommissioned in the first step on the total of N nodes. Thus,

$$t_{decom}(k) = \frac{y \cdot D}{S_{Net}(N-y)} + t_{decom}(k-1)$$

Then, for the $k-1$ other steps, the cluster as a size of $N-y$ and each node hosts $\frac{N \cdot D}{N-y}$.

Thus,

$$
\begin{aligned}
t_{decom}(k) &= \frac{yD}{S_{Net}(N-y)} + \frac{(x-y)\frac{ND}{N-y}}{S_{Net}(N-y-(x-y))} \\
&= \frac{1}{S_{Net}}\left(\frac{yD}{N-y} + \frac{(x-y)\frac{ND}{N-y}}{N-x}\right) \\
&= \frac{1}{S_{Net}} \frac{(N-x)yD + (x-y)ND}{(N-y)(N-x)} \\
&= \frac{1}{S_{Net}} \frac{xD}{(N-x)}
\end{aligned}
$$

Thus, the time to decommission in k steps is $t_{decom}(k) = \frac{xD}{S_{Net}(N-x)}$.

QED

Property 27:

$$t_{decom} = \max\left(\frac{xD}{(N-x)min(S_{Write}, S_{Net})}, \frac{D}{min(S_{Read}, S_{Net})}\right)$$

Proof:

During the decommission, the total amount of data to transfer is xD.

$N-x$ remaining nodes can receive and write it at a throughput of $min(S_{Write}, S_{Net})$.

x leaving nodes can read and send it at a throughput of $min(S_{Read}, S_{Net})$.

QED