A Lower Bound for the Commission Times in Replication-Based Distributed Storage Systems

Nathanaël Cheriere, Matthieu Dorier, Gabriel Antoniu

Research report
A Lower Bound for the Commission Times in Replication-Based Distributed Storage Systems

Nathanaël Cheriere*, Matthieu Dorier†, Gabriel Antoniu*
*Univ Rennes, Inria, CNRS, IRISA, Rennes, France, nathanael.cheriere@irisa.fr, gabriel.antoniu@inria.fr
†Argonne National Laboratory, Lemont, IL, USA, mdorier@anl.gov

***Abstract*—Efficient resource utilization becomes a major concern as large-scale computing infrastructures such as supercomputers and clouds 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 cost.**

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 model the duration of the commission operation, for which we obtain theoretical lower bounds. We then consider HDFS as a use case, and we show that our lower bound can be used to evaluate the performance of the commission algorithms. We show that the commission in HDFS can be greatly accelerated. With the highlights provided by our lower bound, we suggest improvements to speed the commission in HDFS.

***Keywords*-Elastic Storage, Distributed File System, Malleable File System, Lower Bound, Commission**

I. 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, these frameworks are still limited by the fact that *the distributed storage system*[1] *they rely on is not malleable*. In frameworks where such a storage system is deployed in compute nodes along with the job, such as HDFS, this prevents any kind of malleability from the job altogether.

Most distributed storage systems already provide basic *commission* (adding nodes) and *decommission* (removing nodes) operations, usually for maintenance purposes. They are rarely used in practice for optimizing resource usage, however, since they are assumed to have high performance overhead. **With faster networks and faster storage devices (e.g., SSDs, NVRAM, or even in-memory file systems [6]), it is time to revisit this assumption and evaluate whether malleability can be a feature in future distributed storage systems.** Fast storage resizing can indeed favor a quick response to new requests for resources or to sudden variations in workload.

In this report, we focus on the cost of the commission operation in the context of distributed storage systems that leverage data replication. We devise theoretical, implementation-independent lower bounds of the time required to undertake this operation. These models provide a baseline to evaluate implementations of the commission. As a case study, we evaluate how HDFS, a representative state-of-the-art distributed file system with a commission mechanism already implemented, compares with these theoretical bounds. We show that the commission mechanism used by HDFS is not optimized to be fast. We suggest modifications that would improve commission times in HDFS.

This work extends our previous work [7], which was focusing solely on the decommission. Decommission is only half of the operations needed for complete malleability. Thus, in this work we complete our previous study by providing a lower bound for the commission, as well as the study of the commission mechanism implemented in HDFS.

II. Context and motivation

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

A. 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.

[1]This work can easily be applied not only to distributed file systems but also to various kinds of distributed storage systems. Because of this, we use the denomination *Distributed Storage System* in this paper.

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.

B. 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. Past efforts to bring malleability to HPC jobs [4], [9] could be completed by providing a malleable file system able to efficiently leverage the presence of storage devices on compute nodes; for example, Theta [10], at Argonne National Laboratory, features such devices.

III. 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 [11], [12]. Rare, however are papers focusing on the malleability of file systems.

Among them, the SCADS Director [13] 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 node commission but do not study its efficiency. Their algorithm includes a decision mechanism for determining which nodes to add. In contrast, we assume that the file system must follow commands from an independent resource manager, and we propose a lower bound for the commission time that can be integrated in external scheduling strategies.

Lim, Babu, and Chase [14] 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 operation. Both [13] and [14] 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 [15], Sierra [16], or SpringFS [17] shut down nodes to save energy. Because 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 our work we consider a more general case of node commission, the newly added nodes do not host any data and must receive some to be able to service requests. 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 [18], for HDFS [19], or for hash tables [20]. To the best of our knowledge, however, no studies focus on the minimum duration of data rebalancing.

IV. Hypotheses

To be able to compute a lower bound, we make several hypotheses regarding the initial state of the storage system.

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

Because this work is the complement of a previous one [7], the scope of the study is mainly determined by the decommission mechanism. This 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.

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. We also exclude from our scope systems using erasure coding for fault tolerance, such as Pelican [21], since such mechanisms require CPU power to regenerate missing data, and a theoretical lower bound 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 [22] for Spark [23]. 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.

B. Hypotheses about the cluster

We make three hypotheses about the cluster in order to build a comprehensible lower bound.

Hypothesis 1: Cluster homogeneity

All nodes have the same characteristics, in particular the same network throughput (S_{Net}) and storage write and read throughputs (S_{Write},S_{Read}).

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

Hypothesis 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.

Since no network is ideal, the bisection bandwidth is a better metric to use for the network, and the interference problem can be taken into account by using a bandwidth measured in the presence of interference (or provided by some model that is interference-aware). Doing so would provide a better estimation of the time needed for both operations, but the time provided would not be a lower bound anymore.

Hypothesis 3: Ideal storage system

The writing speed is not higher than the reading speed ($S_{Write} \leq S_{Read}$). The device must share its I/O time between reads and writes and thus cannot sustain simultaneous reads and writes at maximum speed (during any span of time t, if a time $t_{Read} \leq t$ is spent reading, the storage cannot write for more than $t - t_{Read}$, and conversely).

Hypothesis 3 holds for most modern storage devices.

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

C. Hypotheses on the initial distribution of the data

The initial data distribution is important for the performance of the commission operation. Thus we make some hypotheses in this respect.

Hypothesis 4: Load-balancing

All N nodes initially host the same amount of data D.

Hypothesis 4 matches the load-balancing target of policies implemented in existing file systems, such as HDFS [19] or RAMCloud [6].

We distinguish between *data* and *unique data*. The data is the amount of space used to store information on the storage system, including the replicas. The unique data is the data without the redundancy of the replicas.

We denote any item stored on the cluster as an object (which, for a file system, can be a file or a chunk of a file). The amount of unique data is the size of all objects stored, while the amount of data stored is r times larger because of the replication of data.

Hypothesis 5: Data replication

Each object stored in the storage system is replicated on $r \geq 2$ distinct nodes. The probability of finding a given object on a node is uniform and independent.

We denote as *exclusive data* of a subset of nodes the data that has *all* its replicas on nodes included in the specified subset.

Hypothesis 6: Uniform data distribution

All sets of r distinct nodes host the same amount of exclusive data, independently of the choice of the r nodes. Since there are $\binom{N}{r}$ distinct sets of r distinct nodes in a system of N nodes, each such set of r nodes hosts exactly $ND/\left(r\binom{N}{r}\right)$ exclusive data.

The rationale behind this hypothesis is the effect of fault on load-balancing. The fault tolerance mechanism can accommodate $r-1$ simultaneous faults at most. In such a situation, in order to recover as quickly as possible, the remaining $n+1-r$ nodes should have to recreate replicas for the same, minimum amount of data regardless of which nodes crashed. Thus, each possible set of r distinct nodes has the same amount of exclusive data.

Since there is a total of $\frac{ND}{r}$ unique data and $\binom{N}{r}$ sets of r distinct nodes, each set of r nodes has exactly $ND/\left(r\binom{N}{r}\right)$ of exclusive data.

From Hypothesis 6, we deduce Property 1.

The probability of finding a given object on all the nodes in a set of r distinct nodes is uniform and independent of the chosen set. **(Prop. 1)**

D. Formalizing the problem

At the end of both the commission and decommission operations, the data distribution should satisfy the following objectives.

Objective 1: No data loss
No data can be lost during either operations.

Objective 2: Maintenance of the replication factor
Each object stored on the storage system is replicated on r distinct nodes. The probability of finding an object on a node is uniform and independent. Moreover, the replication factor of the objects should not drop below r during the operations.

Objective 3: Load-balancing
All nodes host the same amount of data D'.

Objective 4: Uniform data distribution
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 Hypotheses 5, 4, and 6 and are here to ensure a data distribution that is the same as if the cluster always had its new size.

Both the hypotheses and the objectives are rarely attained in practice. However, they reflect the goal of the load-balancing policies implemented in many current state-of-the-art distributed file systems such as HDFS [19] or RAMCloud [6].

V. Lower Bound for the Commission

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

A. 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 a lower bound t_{com} of the time needed to commission a set 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.

B. 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.

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) and the fact that each of the N nodes initially host D data (Hypothesis 4), we deduce the following.

Each node must host $D' = \frac{ND}{N+x}$ of data at the end of the commission. **(Prop. 2)**

Demonstration:
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

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.

$$D_{\rightarrow new} = xD' = \frac{xND}{N+x} \quad \textbf{(Prop. 3)}$$

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

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

QED

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.

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

Detail:

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

QED

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

Properties on the probabilities: The following are useful properties on the probabilities for computing the lower bound of the commission time.

$$\sum_{i=0}^{r} p_i = 1 \qquad \textbf{(Prop. 4)}$$

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

QED

$$\sum_{i=0}^{r} i p_i = \frac{xr}{N+x}. \qquad \textbf{(Prop. 5)}$$

Demonstration:
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:

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

Thus,

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

QED

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 $D_{old\to new}$.

$$D_{old\to new} = \frac{ND}{r}(1-p_0). \qquad \textbf{(Prop. 6)}$$

Demonstration:
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

1) `Cluster 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 cluster.`
2) `Divide {clusters} according the proportions in the new placement; i.e., from a given cluster` C `of objects, pick randomly a proportion` p_i `(for` 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.

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 $D_{old/new\to new}$.

$$D_{old/new->new} = \frac{ND}{rx}\left(\frac{rx}{N+x} + p_0 - 1\right). \qquad \textbf{(Prop. 7)}$$

Demonstration:
$D_{old/new\to new}$ 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

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.

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. 8)**

Demonstration:

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.

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

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

$$\forall 0 \leq i \leq r, d^i_A = \frac{iD^i_A}{x}$$

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

$$\forall 0 \leq i \leq r, dr^i_A = \frac{iD^i_A}{r}$$

Load-balancing (Objective 3)

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

$$\begin{aligned}
D'_{new} &= \sum_{A \in S^r_{old}} \sum_{i=0}^{r} d^i_A \\
&= \sum_{A \in S^r_{old}} \sum_{i=0}^{r} \frac{iD^i_A}{x} \\
&= \binom{N}{r} \sum_{i=0}^{r} \frac{NDip_i}{rx\binom{N}{r}} \\
&= \frac{ND}{xr} \sum_{i=0}^{r} ip_i \\
&= \frac{xD}{N+x} \textit{(with prop. 5)} \\
&= D'
\end{aligned}$$

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

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

$$\begin{aligned}
D'_{old} &= D - \sum_{A \in S^r_{old}(n)} \sum_{i=0}^{r} dr^i_A \\
&= D - \sum_{A \in S^r_{old}(n)} \sum_{i=0}^{r} \frac{iD^i_A}{r} \\
&= D - \sum_{A \in S^r_{old}(n)} \sum_{i=0}^{r} \frac{NDip_i}{r^2\binom{N}{r}} \\
&= D - \binom{N-1}{r-1} \frac{ND}{r^2\binom{N}{r}} \sum_{i=0}^{r} ip_i \\
&= D - \binom{N}{r} \frac{r}{N} \frac{ND}{r^2\binom{N}{r}} \frac{xr}{N+x} \textit{(with prop. 5)} \\
&= D - \frac{xD}{N+x} \\
&= D'
\end{aligned}$$

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,

$$D_{ex} = nb \times D^k_A \times p_{remain} \times p_{distr}.$$

After simplification,

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

Thus, the objective of uniformity is satisfied. **QED**

With this result, since no data transfers are required between old nodes, it will not have any impact on the lower bound of the time needed to commission nodes.

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.

C. A lower bound 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.

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 lower bound of the time 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 lower bound of the time needed to commission nodes.

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}.

$$T_{recv} = \frac{ND}{(N+x)S_{Net}} \quad \textbf{(Prop. 9)}$$

Demonstration:

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

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

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

QED

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 $T_{old\to new}$.

$$T_{old\to new} = \frac{D}{rS_{Net}}(1-p_0) \quad \textbf{(Prop. 10)}$$

Demonstration:

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

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

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.

$$T_{new\to new} = \frac{ND}{rxS_{Net}} \sum_{i=2}^{r}(i-1)p_i \quad \textbf{(Prop. 11)}$$

Demonstration:

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

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

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 $T_{\to new}^{balanced}$.

$$T_{\to new}^{balanced} = \frac{xND}{(N+x)^2 S_{Net}} \quad \textbf{(Prop. 12)}$$

Demonstration:

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.

$$T_{old\to new}^{balanced} = \frac{D_{\to new} - Y}{S_{Net}^{old}} = \frac{1}{NS_{Net}}\left(x\frac{ND}{N+x} - Y\right)$$

$$T_{new\to new}^{balanced} = \frac{Y}{S_{Net}^{new}} = \frac{1}{xS_{Net}}Y$$

Then $T_{old\to new}^{balanced} = T_{new\to new}^{balanced}$, and thus $Y = \frac{x2ND}{(N+x)^2}$

QED

5) Properties: A few useful properties can be deduced.

If $T_{old\to new} \leq T_{new\to new}$, *then*
$T_{old\to new} \leq T^{balanced}_{\to new} \leq T_{new\to new}$. **(Prop. 13)**

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

$$\begin{aligned}
T_{old\to new} &\leq T_{new\to new} \\
1-p_0 &\leq \frac{N}{x}\sum_{i=2}^{r}(i-1)p_i \\
\sum_{i=1}^{r} p_i &\leq \frac{N}{x}\sum_{i=1}^{r}(i-1)p_i \\
\sum_{i=1}^{r} p_i &\leq \frac{N}{x}\sum_{i=1}^{r} ip_i - \frac{N}{x}\sum_{i=1}^{r} p_i \\
\frac{x}{N}(1+\frac{N}{x})\sum_{i=1}^{r} p_i &\leq \sum_{i=1}^{r} ip_i \\
(\frac{x}{N}+1)\sum_{i=1}^{r} p_i &\leq \sum_{i=1}^{r} ip_i
\end{aligned}$$

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

$$\begin{aligned}
&T_{new\to new} - T^{balanced}_{\to new} \\
&= \frac{D}{S_{Net}}\left(\frac{N}{rx}\sum_{i=2}^{r}(i-1)p_i - \frac{xN}{(N+x)^2}\right) \\
&= \frac{D}{S_{Net}}\left(\frac{N}{rx}\sum_{i=2}^{r}(i-1)p_i - \frac{N}{r(N+x)}\sum_{i=0}^{r} ip_i\right) \text{ with prop. 5} \\
&= \frac{ND}{xrS_{Net}}\left(\sum_{i=1}^{r} ip_i - \sum_{i=1}^{r} p_i - \frac{x}{N+x}\sum_{i=1}^{r} ip_i\right) \\
&= \frac{ND}{xrS_{Net}}\left(\frac{N}{N+x}\sum_{i=1}^{r} ip_i - \sum_{i=1}^{r} p_i\right) \\
&\geq \frac{ND}{xrS_{Net}}\left(\frac{N}{N+x}\frac{N+x}{N}\sum_{i=1}^{r} p_i - \sum_{i=1}^{r} p_i\right) \text{ using the assumption} \\
&\geq 0
\end{aligned}$$

QED

If $T_{old\to new} \geq T_{new\to new}$, *then*

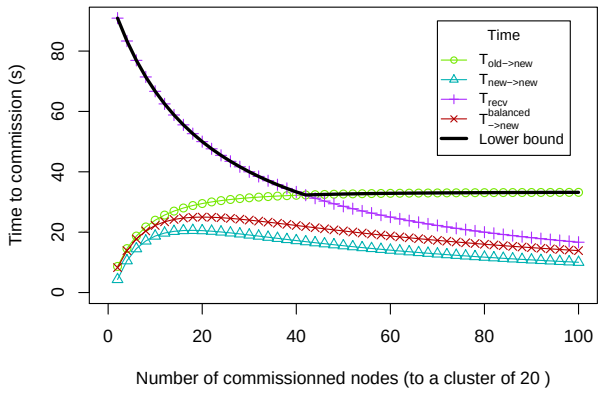

Figure 1. Important times when adding nodes to a cluster of 20 nodes each hosting 100 GB of data, $S_{Net} = 1$ *GBps*, $r = 3$.

$T_{old\to new} \geq T^{balanced}_{\to new} \geq T_{new\to new}$. **(Prop. 14)**

Demonstration:
Basically the same as Property 13. **QED**

$T_{recv} \geq T^{balanced}_{\to new}$ **(Prop. 15)**

Demonstration:

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

QED

6) Commission time with a network bottleneck: The commission, in the case of a network bottleneck, cannot be faster than t_{com}.

$t_{com} = max(T_{old\to new}, T_{recv})$ **(Prop. 16)**

Demonstration:
The commission time is the maximum between T_{recv} (time to receive data) and the time to send the data: $T_{old\to new}$ (if $T_{new\to new} \leq T_{old\to new}$) or $T^{balanced}_{old\to new}$ (if $T_{new\to new} \geq T_{old\to new}$).
After applying Properties 13, 14, and 15, we have

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

QED

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 lower bound 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.

D. A lower bound 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 lower bound of the time needed to finish each action depends on the characteristics of the storage devices.

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

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

$$T_{write} = \frac{ND}{(N+x)S_{Write}} \qquad \textbf{(Prop. 17)}$$

Demonstration:

$$T_{write} = \frac{D'}{S_{Write}}.$$

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

QED

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 $T_{old\to new}$.

$$T_{old\to new} = \frac{D(1-p_0)}{rS_{Read}} \qquad \textbf{(Prop. 18)}$$

Demonstration:

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

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

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 time needed for the commission is

$$t_{com} = max(T_{write}, T_{old\to new}). \qquad \textbf{(Prop. 19)}$$

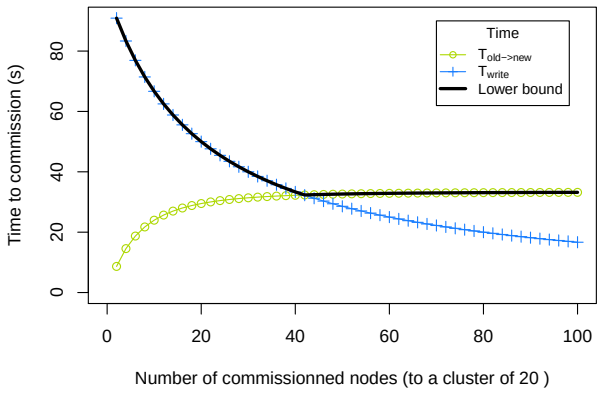

Figure 2. Important times when adding nodes to a cluster of 20 nodes each hosting 100 GB of data. $S_{Read} = S_{Write} = 1$ *GBps*, $r = 3$.

Demonstration:

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

In Figure 2, we can observe the different times that are important in constructing the lower bound 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.

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 (Hypothesis 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}).

$$T^{alldata}_{old\to new} = \frac{xD}{(N+x)S_{Read}} \qquad \textbf{(Prop. 20)}$$

Demonstration:

$$T^{alldata}_{old\to new} = \frac{D'}{S_{Read}} = \frac{xD}{(N+x)S_{Read}}.$$

QED

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.

$$T^{balanced}_{old\to new} = \frac{xND}{(N+x)^2}\frac{S_{Read}+S_{Write}}{S_{Read}S_{Write}} \qquad \textbf{(Prop. 21)}$$

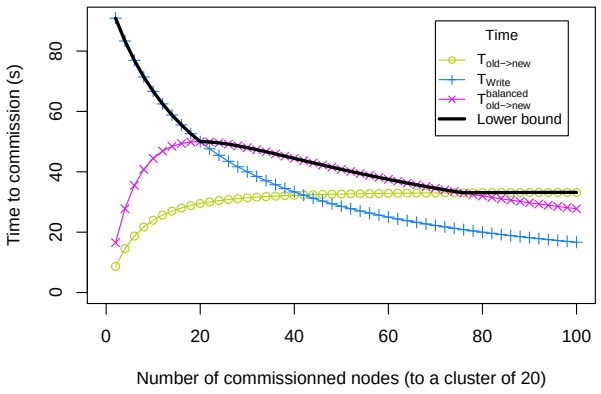

Figure 3. Important times when adding nodes to a cluster of 20 nodes each hosting 100 GB of data. $S_{Read} = S_{Write} = 1$ $GBps$, $r = 3$.

Demonstration:

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 $T_{old\to new}^{balanced} = \frac{D_{\to new}}{xyS_{Write}}$

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

QED

Thus, when no buffering is possible, the time needed to commission nodes is

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

(Prop. 22)

Demonstration:

If $T_{Write} \leq T_{old\to new}^{alldata}$, then the balanced strategy is used. Similarly to Property 13, 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

In Figure 3, we show the different times that are important in constructing the lower bound, 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; 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.

E. 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.

VI. Commission in HDFS

In this section we focus on the commission in HDFS and compare the obtained commission times with the lower bound from Section V.

1) Testbed: The experiments presented in this section have been performed on the Grid'5000 [24] experimental testbed. The *grisou* cluster from Nancy was used for the commission 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.

2) HDFS: We deployed HDFS and Hadoop 2.7.3. One node acted as both DataNode (slave of HDFS) and NameNode (master of HDFS) 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.

3) HDFS in memory: To experiment RAM-based storage with HDFS, we used the same setup as in the paper introducing Tachyon [22]: 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.

A. 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).

B. 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 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.

C. Commission in HDFS

Figures 4 and 5 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 4 presents the duration of the commission operation when the network is the bottleneck, while Figure 5 shows the duration of the commission operation when the storage (drives) is the bottleneck. Both figures show the same pattern: the lower bound and the observed results are opposite of each other. The lower bound 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.

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 lower bound, on the other hand, has been designed with commission speed as the primary objective.

1) Compute the *average* data per available node on the cluster; it is the amount of data each node should be hosting.
2) Compute the *threshold*; it is a percentage of the total capacity of the nodes, given by the user.
3) Cluster nodes according to the amount of data hosted:
   - if they host more than *average+threshold*, they are Over-Utilized;
   - else if they host more than *average*, they are Above-Average;
   - else if they host more than *average-threshold*, they are Below-Average;
   - else, if they host less than *average+threshold*, they are Under-Utilized.
4) Pair the nodes (source and target) with the following priority:
   - Over-Utilized and Under-Utilized,
   - Over-Utilized and Below-Average,
   - Under-Utilized and Above-Average.
5) Select data to move from the source to the target:
   - Target must not already host the same object.
   - Data must not already be scheduled to move.
6) Execute the moves, no more than *threshold* amount of data is moved during each iteration.
   - Replicas can be sent from the source directly
   - or from another node hosting the replica.
7) Wait for all transfers to finish.
8) Repeat from the first step 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.

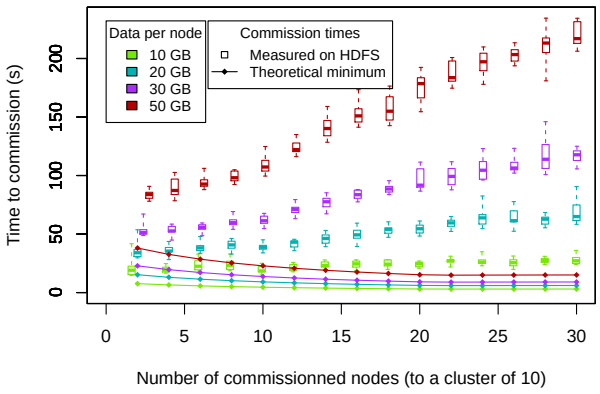

Figure 4. Commission time measured when there is a network bottleneck. The minimum theoretical time obtained with the lower bound is added.

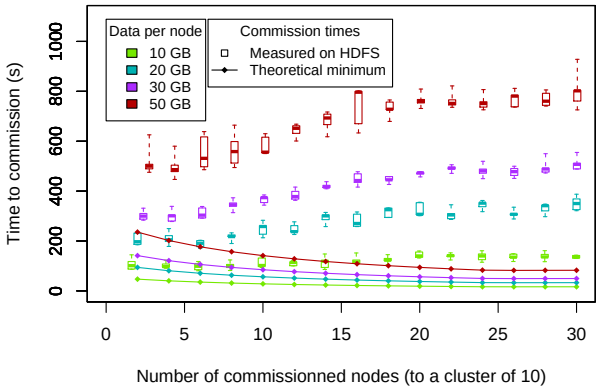

Figure 5. Commission time measured when there is a storage bottleneck. The minimum theoretical time obtained with the lower bound is added.

D. Hints to improve the commission mechanism

The lower bound 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.

VII. Conclusion

Efficient commission 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 lower bound for the duration of this operation, regardless of their implementation. Using these generic lower bounds, we study the commission of HDFS and highlight potential improvements thanks to the better understanding of the various possible bottlenecks of the operation. For the commission, we show that the implementation of the mechanism in HDFS is not optimized for speed and could be greatly accelerated.

These results opened various leads for future work. The next step is the development of a benchmark to evaluate the practical performance of both migration operations on a given platform. This benchmark could help design and optimize the malleability mechanisms before implementing them in a real distributed storage system.

Another challenge left for future work is the implementation of an efficient malleable distributed storage system in order to evaluate its benefits on real use cases.

Acknowledgment

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.

References

[1] A. Kuzmanovska, R. H. Mak, and D. Epema, "KOALA-F: A Resource Manager for Scheduling Frameworks in Clusters," *IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pp. 592–595, 2016.

[2] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv *et al.*, "Morpheus: Towards Automated SLOs for Enterprise Clusters," *USENIX Symposium on Operating Systems Design and Implementation*, pp. 117–134, 2016.

[3] S. S. Vadhiyar and J. J. Dongarra, "SRS: A Framework for Developing Malleable and Migratable Parallel Applications For Distributed Systems," *Parallel Processing Letters*, vol. 13, no. 2, pp. 291–312, 2003.

[4] L. V. Kale, S. Kumar, and J. Desouza, "A Malleable-Job System for Timeshared Parallel Machines," *IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

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

[6] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières *et al.*, "The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 4, pp. 92–105, 2010.

[7] N. Cheriere and 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 *et al.*, "Parallel Scripting for Applications at the PetaScale and Beyond," *Computer*, vol. 10, pp. 50–60, 2009.

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

[10] "Theta," www.alcf.anl.gov/theta, Accessed 19/06/17.

[11] K. Jansen and L. Porkolab, "Linear-Time Approximation Schemes for Scheduling Malleable Parallel Tasks," *Algorithmica*, vol. 32, pp. 507–520, 2002.

[12] G. Mounie, C. Rapine, and D. Trystram, "Efficient Approximation Algorithms for Scheduling Malleable Tasks," *ACM Symposium on Parallel Algorithms and Architectures*, vol. 3, pp. 23–32, 1999.

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

[14] H. C. Lim, S. Babu, and J. S. Chase, "Automated Control for Elastic Storage," *International Conference on Autonomic Computing*, pp. 1–10, 2010.

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

[16] E. Thereska, A. Donnelly, and D. Narayanan, "Sierra: Practical Power-Proportionality for Data center Storage," *Conference on Computer Systems*, p. 169, 2011.

[17] L. Xu, J. Cipar, E. Krevat, A. Tumanov, N. Gupta, C. Mellon *et al.*, "SpringFS : Bridging Agility and Performance in Elastic Distributed Storage," *USENIX Conference on File and Storage Technologies*, pp. 243–255, 2014.

[18] A. Miranda and T. Cortes, "CRAID: Online RAID Upgrades Using Dynamic Hot Data Reorganization." in *FAST*, vol. 14, 2014, pp. 133–146.

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

[20] P. B. Godfrey and 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.

[21] S. Balakrishnan, R. Black, A. Donnelly, P. England, A. Glass, D. Harper *et al.*, "Pelican: A Building Block for Exascale Cold Data Storage," in *Operating Systems Design and Implementation*, 2014, pp. 351–365.

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

[23] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," *HotCloud*, vol. 10, no. 10, p. 95, 2010.

[24] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine *et al.*, "Adding Virtualization Capabilities to the Grid'5000 Testbed," in *Cloud Computing and Services Science*, 2013, vol. 367, pp. 3–20.