The Minimum Duration to Realocate Data in a Cluster

Juliette Fournis d'Albiat
Irisa

This report corresponds to the work realized during my internship within the KerData research team at Inria Rennes Bretagne Atlantique, under the advisory of Nathanaël Cheriere and Gabriel Antoniu.

Abstract

In 2020, Big Data is an exciting subject. A greatest issue is to process Big Data. With this purpose, MapReduce paradigm has emerged in 2004 and distributed data file systems have been developed. In order to achieve efficiency, this systems should allocate ressources following web utilization. A solution to manage resource allocation is developing decommission. Although there this several works which work on ressource allocation, by now decommission hasn't been studied in details. In fact, decommission requires to reallocate Data and this operation is time-consuming. In this paper, we develop a model giving a bottleneck of decommission time. Our model aims at genericity, indeed we assume that data distribution do not have a particular shape. We also present experimental results of our model, which have been obtained on real-world cluster Grid5000 [8].

1 Introduction

The term "Big Data" refers to the data which is too large to be processed by traditional tools and procedures. This notion emerged in the 1990s and gained momentum in the 2000s, when the term Big Data became popular.
For instance, in an single minute in 2019, 4.5 Million of videos are viewed on YouTube, 3.8 Million of Search Queries are done on Google and 188 Million of emails are sent in a minute [7]. Big Data is more than ever a challenge. These data need to be stored, classified, sorted and analyzed.
With this goal, Google has proposed in 2004 the MapReduce paradigm. MapReduce [4] is a programming model for processing large data set using a distributed file system for data storage. It's based on two main operations, the *map* and *reduce* operations, which can be written by programmers. Programmers can focus on writing these two operations and the framework manage optimization, parallelization, and fault tolerance separately from programming.
To implement this programming paradigm and achieve good performance, many systems have been developed. Most of them use a distributed file system where data is co-located with computation. That reduce latency and makes easier to process data at large scales.
The main one is Hadoop [8]. Hadoop manages optimization, parallelization and fault tolerance). It relies on a distributed file system HDFS (Hadoop Distributed File System). HDFS has a master/slave architecture. The master holds metadata information about stored data and directs requests. The slaves serve read and write requests.
One of the challenges of distributed systems is *elasticity*. Elasticity is defined as the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner,

such that at each point in time the available resources match the current demand as closely as possible". By now, elasticity remains a challenge to achieve in clouds. Indeed while cloud utilization fluctuates, the size of infrastructures is fixed. Nevertheless, elasticity can be a means to achieve good performance.It mainly allows to adapt the resources used by the system to its needs and enables to manage power consumption by adapting the size of the cluster to the client demand. It also enables a better management of the cluster by quickly reallocating resources to other jobs. Elasticity in a storage cluster can be achieved by implementing decommission and commission of physical resource.

- Commissioning nodes consist in adding adding nodes to the cluster and bringing them to a working condition.
- Decommissioning nodes consists in removing nodes from the system's allocation.

While elasticity mechanism have been implemented in practice to react to failures, it has not really been used to optimize resource usage since the commission and decommission operations are assumed to take too long to be useful in practice.

1.1 Related Work

Our goal is to bring elasticity to storage clusters. This question has been previously explored. We present some of related works.

- Sierra [9] is a distributed file storage developed in order to achieve power proportionality (the number of nodes used in the cluster is proportional to the cluster utilization). It powers down servers during throughput without migrating data. So it doesn't use decommission. The limits of Sierra are that it has to keep at least one replica of each file to be able to maintain write availability. This requirement limits elasticity, the cluster can not shrink under $1/r$ of its maximum size where r is the decommission factor. Sierra lacks of agility. The authors don't try to minimize the time to powering off nodes. Also the data layout when turning servers back induces significant migration overhead, impairing system agility.
- Rabbit [1] is a distributed file system that arranges its data layout to achieve power proportionality. As Sierra, it powers down servers (it doesn't use decommission). It dispatches primaries in such a way that it can shrink to a small size (≈ 10 percent of total cluster's size). But, to achieve this, because of the data layout, Rabbit loses writing bandwidth even when the cluster is at its maximum powering, because primaries replicas (replicas in charge to schedule writing operations) are located on 10 percent of nodes. Besides this system induces significant cleanup overhead when shrinking the cluster thus it lacks of agility.
- SpringFS [6] is a distributed file system which uses Sierra [9] and Rabbit [1] ideas but which works on agility. The authors want *to minimize the time to resize their system*. To that goal, depending on workload, their system behaves like Rabbit [1] or Sierra [9]. They provide a continuum between this two systems which minimize their agility impairing.
- KoalaF [5] focuses on sharing resources among applications according to the variation of their workloads over time. It designs a resource manager. The authors' goal is to minimize overheads due to communication between the cluster manager and frameworks.

- The work of Cheriere and Antoniu [2] presents a model for the decommission time. The model developed establishes a lower bound of the decommission's duration and the article also describes experiments wich evaluate the accuracy of the model. However, this paper relies on strong hypotheses about the data distribution to develop the decommission model. It assumes that all cluster nodes have the same amount of data. Besides it assumes that any two nodes share exactly the same amount of data.
  To establish the model, the authors consider two cases. First they assume the performance of decommission is limited by storage, then they explore the case where it is limited by the I/O speed. Results (in term of relative error) are better with the first assumption. In order to evaluate their model the author also develops a modular benchmark Pufferbench [3]. This benchmark evaluates how fast one can scale up and down distributed storage systems. It has been designed in order to be easily customized. We modified the Pufferbench code and used it to assess our model.

These articles also point out the importance of being able to resize clusters quickly. But a theoretical model giving a lower bound of the duration of cluster resizing would be relevant in order to assess and improve distributed file systems.

1.2 Contribution overview

This study provides a mathematical model which gives a lower bound for the decommission's duration in a distributed file system. The model aims at a largest scope as possible. It offers genericity by considering that data distribution does not have a particular shape in the cluster. This approach improves the previous model for decommission which considered a fixed and uniform distributed distribution of data. This paper makes the following contributions:

- In section 2.1, we present the main assumptions on which our model is based.
- In section 2.1.1, we develop the mathematical model that we used to design the lower bound of duration of decommission. This section is composed of two major parts corresponding to two possible bottlenecks. The first part deals with a network bottleneck while the second part deals with a storage bottleneck.
- In section 3, we evaluate the accuracy of our model by experiments in a real cluster system and analyze the results

2 Contribution : modelling the decommission time for non-uniform data distributions and validation of the model

We consider a DFS deployed on a number of nodes (servers) linked through a network.

2.0.1 Definition of the decommission

Decommission is a main operation in a data cluster. It stands for removing nodes from the cluster. The key point is that data must not be lost after the removal of nodes. Consider a cluster of N nodes. Each node has a number of piece of data D. Each piece of data is replicated r times (it's often the case of practice because of fault tolerance or optimization's purposes). Decommissioning x nodes consists in shrinking the cluster from N nodes to (N - x) nodes and recreating the x nodes content. To recreate data, the x leaving nodes will send data to other (remaining) nodes. But because each piece of data is replicated r times in the cluster, there will be several nodes (not only the one which will be removed) that will be able to send information needed to be recreated. Because all data should be recreated during the operation, the number of data replicas r should be constant.

2.1 Assumptions on the cluster

In this section, we introduce the main hypotheses to establish our model :

Assumption 1: Homogeneous cluster
All nodes have the same characteristics, in particular the same network throughput S_{Net} and the same throughput for reading, writing to storage devices (S_{Read}, S_{Write}).

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 between signals generated by data transmission.

Assumption 3: Ideal storage
The device must share its I/O time between reads and writes and thus can not 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).

These assumptions enable us to represent the reality in a simple manner. They are often verified in practice.

2.1.1 Distribution

We consider a cluster of N nodes. We wish to decommission x nodes of a cluster. The x nodes to decommission have an amount of data D to recreate on the cluster. The data stored in these x nodes are replicated r times in the cluster. We know the file distribution of the data that we want to recreate among the cluster F_{rep}. F_{rep} gives for a node n in the cluster, the amount of data that it shares with the nodes in decommission (that is to say the data to recreate). So the x nodes to decommission have an amount D of data to recreate and there is also a group of nodes in the cluster which has $(r-1)D$ copies of the D data to recreate.

2.2 A model for the decommission time when the network is the bottleneck

In this part we assume that the network is the bottleneck.
Assumption 4: Bottleneck network
The network is the component that limits the speed of the operation. The storage throughput is greater than the network throughput.

We want to calculate t_{dec}, a lower bound to decommission time. We have assumed with Assumption 2 that the network was full duplex, data can be sent and received at the same time and there is no interference, thus

$$t_{dec} = max(t_{send}, t_{rec}) \tag{1}$$

Where t_{send} is the minimal time to send the data of the decommissioning nodes.
And t_{rec} is the minimal time to receive the data of the decommissioning nodes.

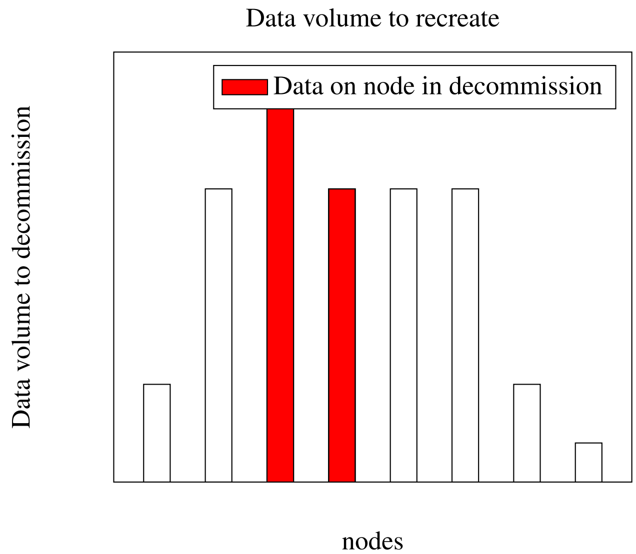

Figure 1: An example of data distribution F_{rep} for eight nodes. *Data in red is exactly the data to recreate in the cluster. Data in white is the copy of data to recreate*

The decommission can not complete before all the data is sent and received, thus, we need to determine the minimal duration of both.

2.2.1 Calculating t_{rec}

We want to calculate t_{dec} a lower bound to decommission time.
We know that the time for decommission is the maximum of the time taken by each node of the cluster during the decommission.

$$t_{rec} = \max_{node} t_n \tag{2}$$

This equation shows that we should minimize the max, which can be done by distribute uniformly the work between nodes in order to have for all nodes n, n'.

$$t_n = t_{n'}$$

Where t_n is the time spent by node n to work during decommission.

Because we want a lower bound, we focus on the strategy that minimizes the duration to send data. The fastest way to distribute data is to send it uniformly across the cluster. All nodes will receive the same amount of data excepting the nodes in decommission. Since nodes in decommission should leave the cluster, at the end of the decommission they should not have any data and so they shouldn't receive data. Then there are $N - x$ nodes which will receive data We assumed that there is a amount D of data to decommission and that the speed of the network is S_{net} . It follows that

$$t_{rec} = \frac{D}{S_{net}(N-x)} \tag{3}$$

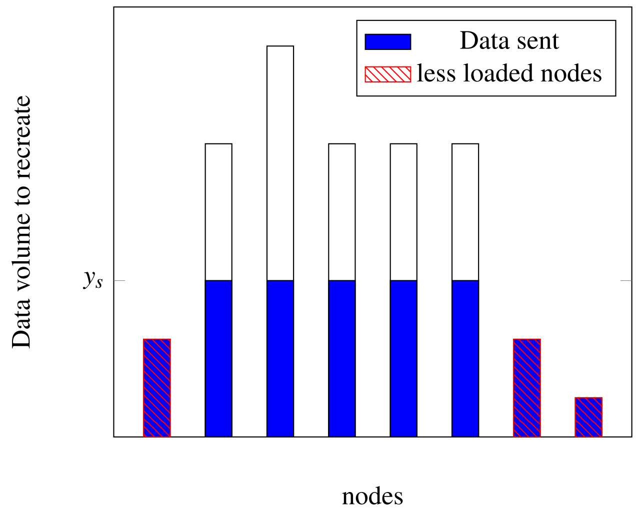

Figure 2: An example of data that should be sent by cluster's nodes

2.2.2 Calculating t_{send}

The time taken to send all the data is the maximum time taken by nodes which send data.
The total data to send is D.
According to this requirements, we should find k and y_s the mean amount of data sent by the most loaded nodes such as

$$\Sigma_{i=1}^{k} F_{rep}(n_i) + \Sigma_{x=k}^{N} y_s = D$$

- *$n_1, n_2, ..., n_k$ are the k nodes that have less data than y s ..*
- *y_s is the data sent by the most loaded nodes.*

The idea bellow this formula is to distribute data as uniformly as possible.
Nevertheless the k less loaded nodes will send less data than others because they haven't enough data.
All remaining nodes will send the same amount of data.
Then we have

$$t_{send} = \frac{y_s}{S_{net}} \tag{4}$$

2.3 A model for decommission time when the storage is the bottleneck

In this part we assume that the storage is the bottleneck thus not the network anymore.
Assumption 5: Bottleneck network
The network is the component that limits the speed of the operation. Storage throughput is greater than S_{net}

Like in section 2.2 we have

$$t_{dec} = \max_{node} t_n$$

Here $t_n = t_{n_{read}} + t_{n_{write}}$ is the time spend by node n to read and write data during decommission. t_n is a sum because of assumption 3 (ideal storage).
Moreover, we know that we should write a quantity

$$D_{write} = D$$

equal to the data hosted in the nodes in decommission.
Nevertheless we don't have $D_{read} = D$ because a node when it has read data to recreate can send this data to multiple nodes. So it should only read an amount

$$D_{read} = \frac{1}{1+\varepsilon} D$$

equal to the <u>*unique*</u> data of the nodes in decommission.
The additional factor $\frac{1}{1+\varepsilon}$ represents the ratio

$$\frac{\text{unique data}}{\text{total data}}$$

among data to replicate.
The ratio variates between 1 and $\frac{1}{r}$ and its mean value is given by

$$\sum_{i=1}^{r} iP(X = i)$$

where $P(X = i)$ is the probability to have exactly i replicas of the same data on the nodes to decommission.

2.3.1 Balancing read and write

Because nodes host different data, we differentiate nodes in decommission from each other

- The <u>decommissioning nodes</u> should only read data
- The <u>least loaded nodes</u> will not be able to read as much object data than others because they have too few replicas.

Ideally each node should work equally (eq: 2). And t_n should be the same for all nodes n. Thus the decommission time should just be the mean time that nodes have to work to write D_{write} and read D_{read}. Thus

$$t_{dec} = \frac{1}{N}\left(\frac{D_{read}}{S_{read}} + \frac{D_{read}}{S_{write}}\right) \tag{5}$$

Where N is the number of nodes in the cluster.
There will be many situations where this formula is valid, an example of such situation is shown by Figure 3 nevertheless depending on the data distribution, it's not always possible to balance work equally between nodes.

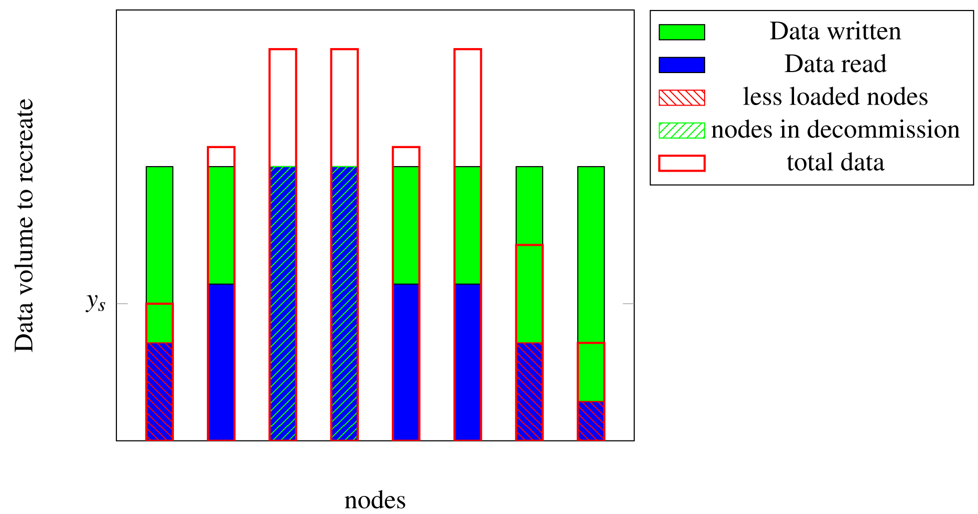

Figure 3: An example of work distribution in the cluster when each node can work equally

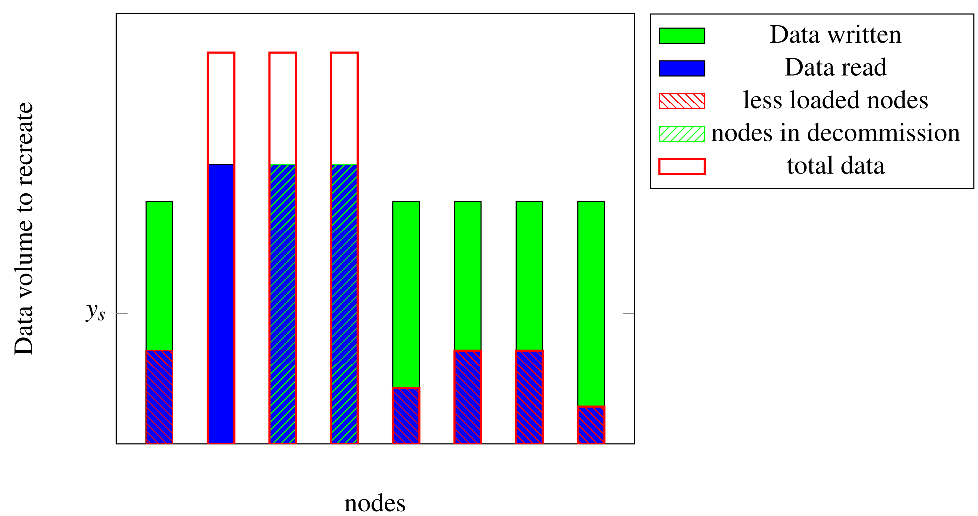

Figure 4: An example of work distribution in the cluster when the writting operation is the bottleneck

2.3.2 Two cases where work can't be distributed uniformly

The writing operation is the bottleneck
The less loaded nodes in the cluster can not be able to write enough data to balance the fact that they will read less data. An example of such situation is shown by the figure 4
The reading operation is the bottleneck
The nodes in decommission will no write data because they should leave the cluster. Thus they can't send

enough data to work as others nodes. Then

$$t_{dec} = \frac{D_{read}}{S_{read}x}$$

An example of such situation is shown by the figure 5

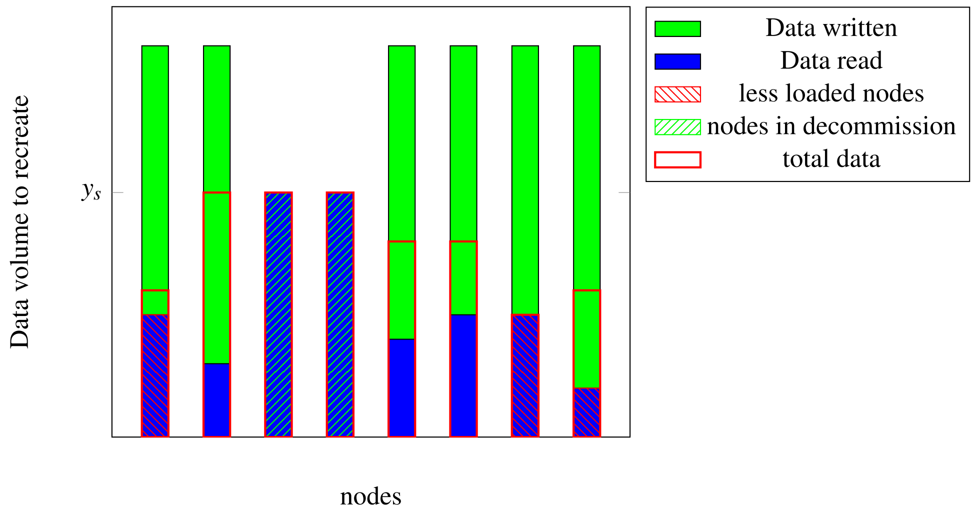

Figure 5: An example of work distribution in the cluster when the reading operation is the bottleneck

In the most general case, a lower bound for decommission time is

$$t_{dec} = \max_n \left(\frac{x_{n_{read}}}{S_{read}} + \frac{x_{n_{write}}}{S_{write}}\right)$$

where $x_{n_{read}}$ and $x_{n_{write}}$ are the data read and write by node n.

3 Validity of the Solution

In this section we assess our formulas. The purpose is to prove the validity of our assumptions and of our model. Although we provided a lower bound, this lower bound is achievable in real world and thus useful.

3.1 A case-study to assess the results

In order to assess our model, we create a simple case of distribution. We worked with 20 nodes.

- We distributed 40 GiB of data in the nodes to decommission
- We left 6 nodes empty.
- We distributed the remaining data to decommission equally
- We completed by data which not need to be decommissioned so that each node had 40 GiB of data

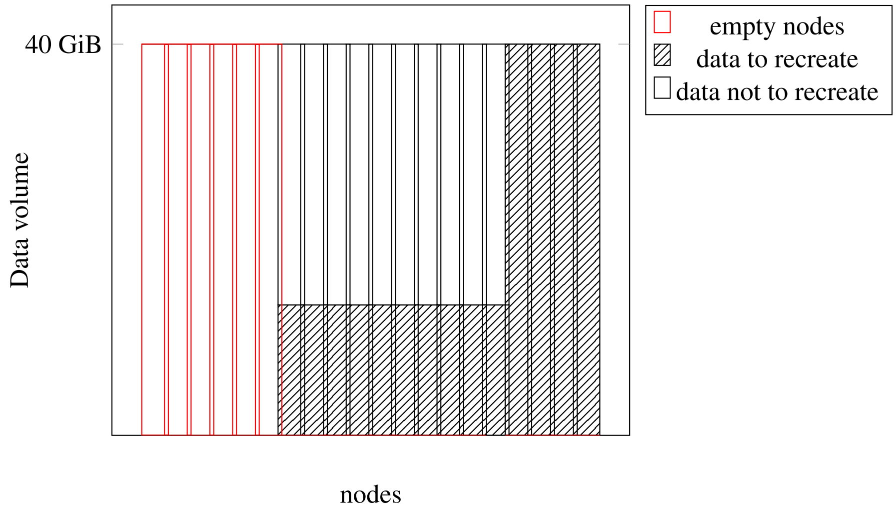

Figure 6: Data Distribution F_{rep} for experiments

3.2 Pufferbench

In order to assess our model through practical experiments, we used Pufferbench [3]. Pufferbench is a benchmark designed to evaluate decommission and commission duration. Pufferbench has been originally developed to validate assumptions of [2] but it has been designed to be easily adapted to other hypotheses. Thus we needed to rewrite a part of the Pufferbench code. Then we deployed Pufferbench on a cluster used in the scientific communitie to develop distributed tools.
In this section we will first describe our algorithmic contribution and our extension of Pufferbench.
Then we will present our experimental results and compare them to the theoretical model.

3.2.1 Developing Pufferbench

Pufferbench [3] is separated in several components (Figure 7). We focused especially on two components :

- The DataDistributionGenerator
  This component is in charge of distributing data in the cluster.
  It was previously designed to distribute the data uniformly or randomly.
  We developed it to enable it to distribute data has described in 3.1.

- The DataTransfertScheduler
  This component is the "brain" of the decommission, it distributes read/write/send/receive operations.
  It was previously designed to behave like [8] data transfert scheduler.
  We developed it in order that it schedules operations to minimize decommission time.

3.2.2 Experiments Material

For our experiments in Grid5000 [8] we use the Paravance cluster of Rennes. We reserved all this cluster to evict network interferences. The computers of Paravance have 128GB of RAM, two disks(HDD) of 558

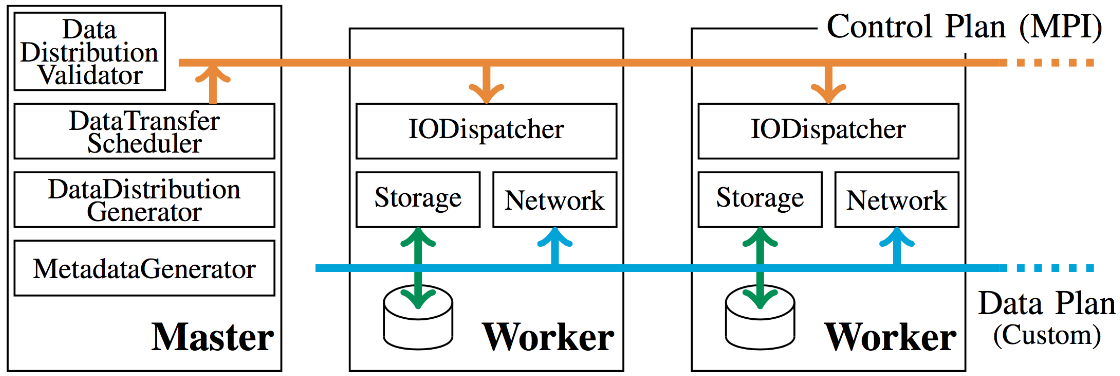

Figure 7: Pufferbench structure

GB and the network throughput is 10Gb/s. Data was transferred in blocks of size 128MiB. To test the case where the network is the bottleneck we used the cache. Thanks to the cache, IO speed is much more greater than network speed, that assures that network will be the bottleneck and not the disk. The network speed was 10 Gb/s. To test the case where the storage is the bottleneck we disables the cache.We used a benchmark to measure the disk speed. We found that it is 210 MB/s for reading operations and 190 MB/s for writing operations.

3.2.3 Experimentals Results

We obtain a maximal error between the results and the model of 10 percent when network is the bottleneck (This error has been obtained when there is only one node to decommission and the operation is fast, it takes only three seconds). The initial model also achieved good performance. Indeed in most of clusters the network speed limits operations thus clusters are optimized to send packets quickly.
When the storage is the bottleneck results are less satisfying (Fig 9). On graph we just give the mean maximum error. We repeated each experiment nine times, and we have calculated the maximum error for these nine experiments. We have a mean maximum error of 20 percent on the number of nodes that are decommissioned. The I/O throughput is very fluctuating, and in our model we considered it stable. We think that explains the difference between our experimental results and our model.
The error is much less important when the number of nodes in decommissioned increases. That is explained by the fact that latency becomes negligible only when operation duration is long enough and the standard deviation is weaker. The error should further decrease when decommissioning more nodes in bigger cluster. In practice many clusters have much more than 20 nodes. For example, the Yahoo cluster using HDFS, has 4,000 nodes.

4 Conclusion

Today decommission is used as maintenance operation. In fact, it requires to reallocate data and that is time-consuming. In order to evaluate its potential to improve future systems we have developed a model giving a lower bound of decommission operation.

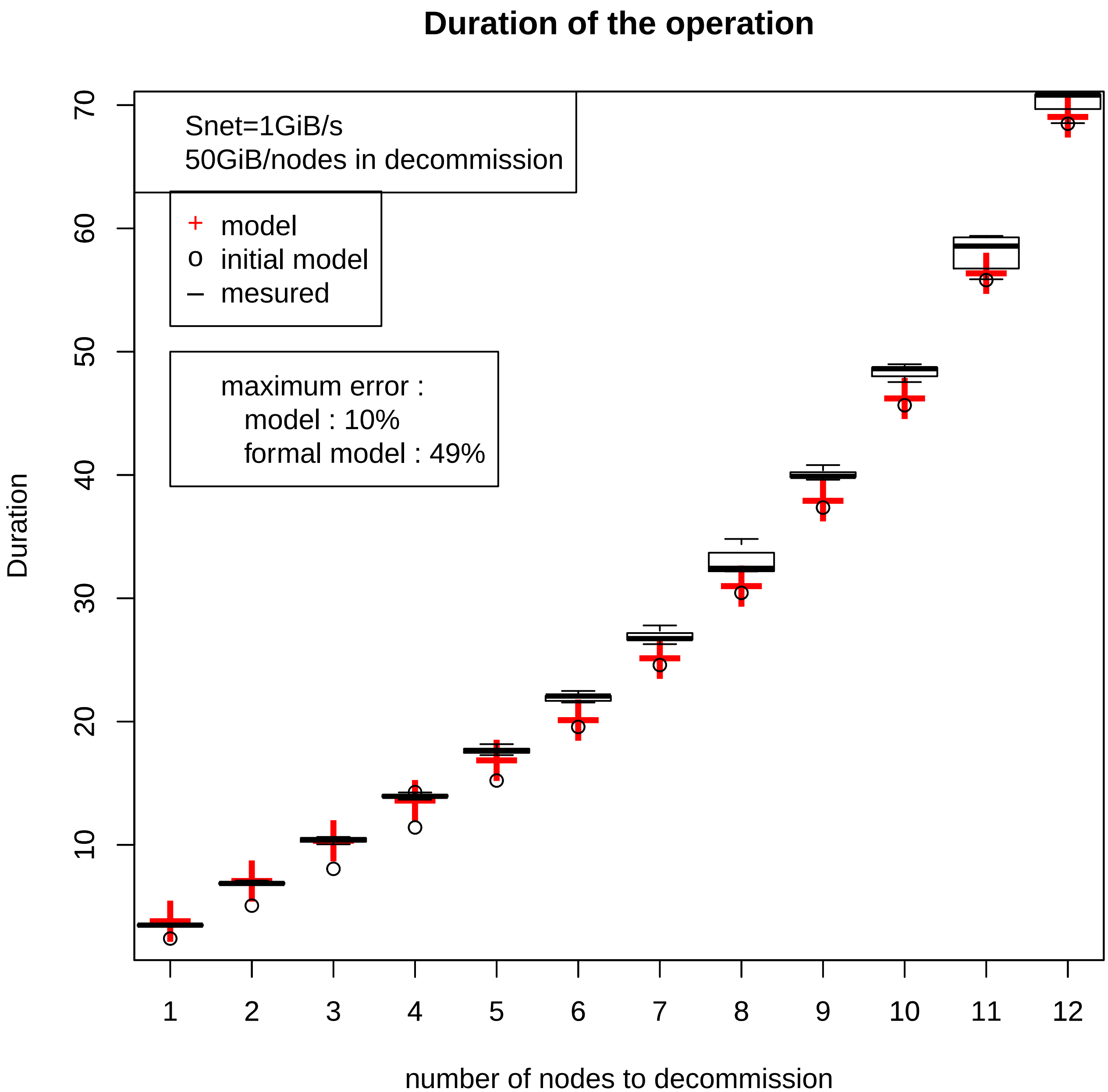

Figure 8: Decommission duration when the network is the bottleneck

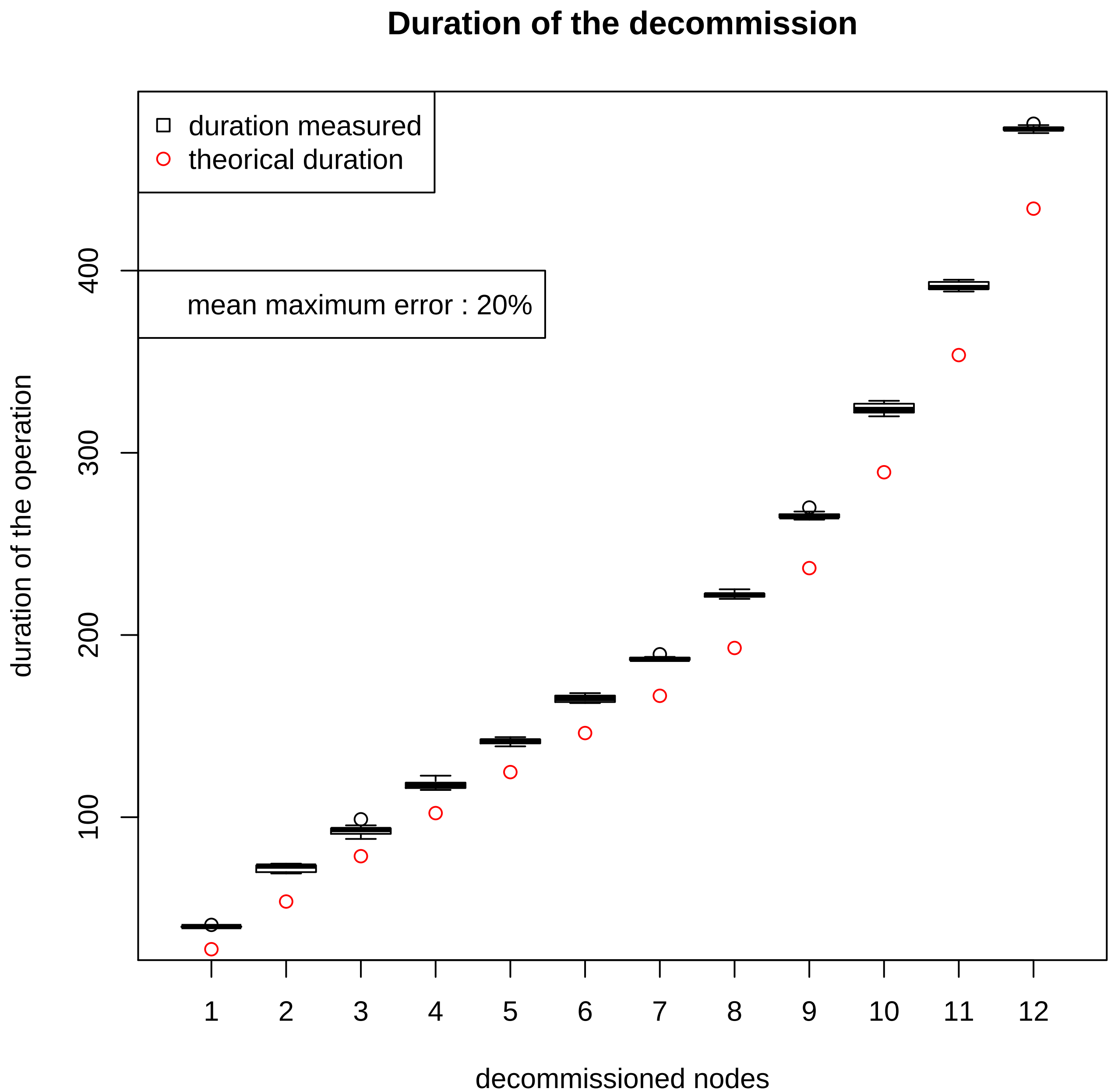

Figure 9: Decommission duration when the storage is the bottleneck

4.1 Future Work

There is still a lot of remaining work related to decommission to prove its interest.

4.1.1 Implementing decommission in a real world

- It needs a study of the energetic consumption of the operation. Explaining the cost and the gain of the operation.
- It needs an application that studies theoretically the impact of the decommission on the application's performance. In this study we have seen that the cost of the decommission is proportional to the amount of data to move. The decommission must be more interesting for applications that use few data compared to computational power. It is also more relevant when there is huge and slow variations.
- It needs to be properly implemented and to be optimized in real world distributed file systems.

4.1.2 Developing strategies to optimize decommission

Because the interest of decommission is highly related to the size of the data to displace, to be relevant each application that would use decommission should focuse on data layout. Some nodes in a cluster will produce less data than others. That would be faster to decommission least loaded nodes. In the second case of this study (subsection 2.3), we have also seen that when data should be read, it's better to decommission nodes which share the same replicas. Also because all data in nodes has to be read on disk, the reading time can be reduced at its minimum (there is no need to find data).

4.1.3 Improve the decommission model

With our experiments we have seen that our model fit better the real world when the network is the bottleneck. Thus it's possible to improve model when the storage is the bottleneck. Particularly that can relevant to study IO throughput variation and consider a model assume a variable speed.
Also there should be situations where there is no network or storage bottleneck but a mixed of it. This situation is also relevant.

4.2 Conclusion

The decommission is a basic operation in cluster management. Thus this operation needs to be study in details. It could bring real solution for storage elasticity. This work focused on studying it in a theoretical and generic manner. We have provided a theoretical lower bound for the duration of the decommission which is independent to data distribution. We first developed a theoretical model then we assessed our results experimentally. The study also enables a better understanding of the decommission and how to optimize this operation.
Nevertheless decommission study isn't a closed matter. And needs practical case of implementation before to be adopted.

References

[1] Hrishikesh Amur, James Cipar, Varun Gupta, Gregory R. Ganger, Michael A. Kozuch, and Karsten Schwan. Robust and Flexible Power-Proportional Storage. *ACM Symposium on Cloud Computing (SoCC)*, pages 217–228, 2010.

[2] Nathanaël Cheriere and Gabriel Antoniu. How Fast Can One Scale Down a Distributed File System? In *IEEE International Conference on Big Data (BigData)*, 2017.

[3] Nathanaël Cheriere, Matthieu Dorier, and Gabriel 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)*, pages 35–44, 2018.

[4] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communication of the ACM*, 51(1):107–113, 2008.

[5] Aleksandra Kuzmanovska, Rudolf H. Mak, and Dick Epema. KOALA-F: A Resource Manager for Scheduling Frameworks in Clusters. In *IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid)*, pages 592–595, 2016.

[6] Xu Lianghong, Cipar James, Krevat Elie, Tumanov Alexey, Gupta Nitin, Kozuch Michael, and Ganger Gregory. SpringFS: Bridging Agility and Performance in Elastic Distributed Storage. In *USENIX Conference on File and Storage Technologies (FAST)*, pages 243–255, 2014.

[7] LoriLewis and OfficiallyChadd. What happened in 2019. `www.visualcapitalist.com/what-happens-in-an-internet-minute-in-2019`.

[8] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. *IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[9] Eno Thereska, Austin Donnelly, and Dushyanth Narayanan. Sierra: Practical Power-Proportionality for Data center Storage. *6th Conference on Computer Systems (EuroSys)*, page 169, 2011.