Folding a Cluster containing a Distributed File-System

Quentin Guilloteau, Olivier Richard, Raphaël Bleuse, and Eric Rutten

Univ. Grenoble Alpes, Inria, CNRS, LIG, F-38000 Grenoble France*

Abstract

The development, study, and experimental validation of cluster and grid middlewares are combersome, time and resource consuming. Researchers need to deploy their middlewares on multiple nodes to test and evaluate their modifications. But these middlewares are destine to be used in production with hundreds or thousands of machines, and deploying at this scale to perform tests and evaluations is too costly. In this paper, we investigate folding techniques to represent a full scale cluster on less physical machines. We evaluate the performance of a distributed I/O benchmark application with respect to the level of folding to determine guidelines for future folded experiments containing a distributed file-system.

1 Introduction

The increase of computing demands from scientists from all fields led to the development of computing cluster and grid architectures. And with these new architectures came new challenges. Due to their high prices, clusters are often shared among several research laboratories and teams. This sharing motivates the development of middlewares such as batch schedulers that are responsible to assign the user jobs to physical machines, manage the state of the resources, deal with reservation, etc. Such cluster, or grid, middlewares are complex applications of great research interest, and they must be tested before reaching production. However, they are usually destine to operate in an environment of hundreds or thousands of machines. Deploying a full scale environment is too costly to perform simple tests and very specific evaluations.

We thus want to answer the following question: **Can we reduce the number of physical machines needed to perform a full scale experiment while keeping a similar behavior as the full scale system?**

One solution to reduce the number of machines used for experiments would be to use simulation techniques. The system, the applications, and the middlewares are modeled and can then be executed on a single node. Beside reducing the number of machines used, simulators also reduce the execution time of the experiments. In the context of distributed systems and applications, projects such as Simgrid [8] and Batsim [9] are leading the way. One drawback of simulation is that the real middleware is not being executed, or not fully executed, but instead, a partial or modelled version in a modelled environment. In the case of cluster and grid middlewares, the applications are often far too complex to model them fully correctly.

Another approach is to *fold* the experiment by deploying more "virtual" resources on physical machines. In the case of a Resources and Jobs Management Systems (RJMS), like Slurm [23] or OAR [5], it is possible to define several resources, from the point of view of the RJMS, on a single machine, which would be completely transparent for the users. For example, one can deploy a 1000-node virtual cluster on 10 physical nodes by defining 100 virtual resources on each node. The advantage is that the experiment takes place on the real system (CPU, network, disk, etc.) and with the real middleware code. The drawback is that this folding can introduce noise in the experiment and degrade performance.

In this paper we evaluate the performance of a distributed I/O benchmark when we fold a computing cluster onto itself, and quantify the impact of folding a computing cluster containing a file system. We then give a *rule of thumb* for choosing the amount of folding for distributed experiences containing a distributed file-system.

Section 2 presents a motivating example for this study. Section 3 defines notions and concepts. We present the experimental protocol in Section 4 and perform the evaluation in Section 5.

*Firstname.Lastname@inria.fr

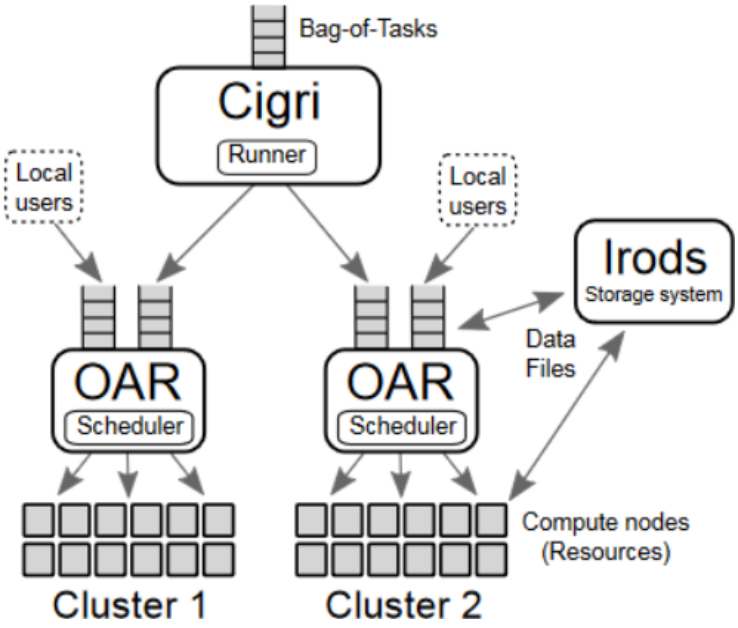


Figure 1: Interactions between *CiGri* and the schedulers *OAR* of the computing grid. *CiGri* users submit *Bag-of-Tasks* applications, whose jobs are then submitted to the different cluster schedulers of the computing grid. The computing clusters are shared with users with more priority, thus *CiGri* jobs must be killed if one premium user requires the resources.

2 Motivating Example: the *CiGri* middleware

CiGri [10] is a grid middleware in production at the French *Gricad* meso-center [1]. The goal of *CiGri* is to use the idle resources of the meso-center. It interacts with several clusters managed by *OAR* [5] batch schedulers.

Users of *CiGri* submit *Bag-of-Tasks* applications to the middleware. Such applications are composed of thousands of short, independent and similar tasks are classified as *embarrassingly parallel* which make them a good candidate for "filling the holes" in the cluster schedules. Monte-Carlo simulations or parameter sweeps are examples of *Bag-of-Tasks* applications.

Once the set of tasks submitted to *CiGri*, the middleware will submit sub sets of jobs to the different schedulers of the grid. The jobs are submitted with the lowest priority in order to allow premium users of the clusters to get the resources used by *CiGri* jobs if needed.

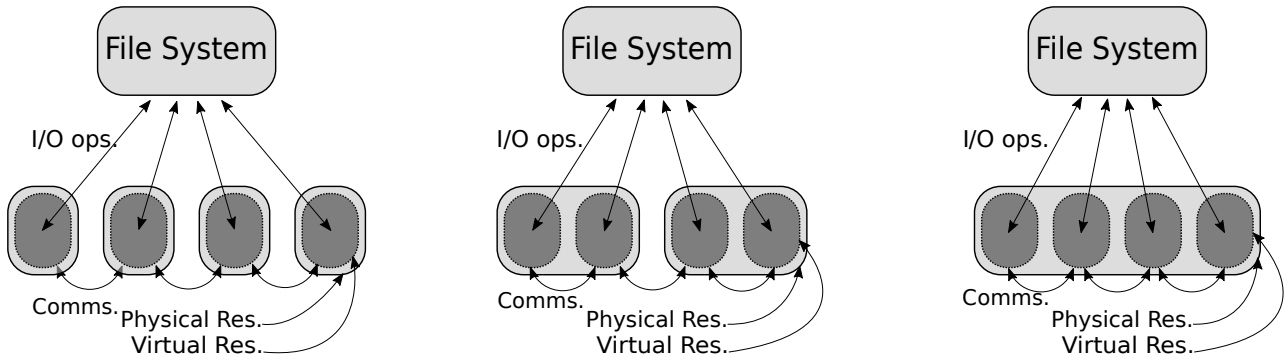
Figure 1 depicts the interactions between *CiGri* and different clusters of the grid.

One problem of *CiGri* is its submission algorithm. *CiGri* will submit a batch of jobs the one scheduler, and wait for the completion of the batch to submit again. This strategy can lead to an underutilization of the cluster resources. Moreover, one objective of *CiGri* is to harvest in a *non-intrusive* fashion. Meaning that the premium users of the different clusters must not notice the impact of the *CiGri* jobs on the platform. However, once executing, the *CiGri* jobs are using the shared resources of the cluster (file-system, network, etc.), which can have an impact on the performance of every user jobs.

In the context of research projects requiring to modify *CiGri* to control its job submission based on the cluster file-system load [14, 12, 13, 21, 22], we must deploy and perform experimentation evaluation of a modified version of *CiGri* on a realistic environment. It is unreasonable to deploy on the *entire Gricad* meso-center, or using simulation due to the complex software stack (*CiGri* + *OAR* + users jobs). We are thus interested in folding strategies to perform the evaluation of our *CiGri* modifications.

A study of the *CiGri* jobs running on the *Gricad* platform [11] gives a statistical description of the execution times of those jobs. This allows us to use a `sleep` model to represent the execution times. As `sleep` calls are extremely lightweight for a CPU to deal with, we are able to fold several virtual resources onto one physical resource. However, as explained earlier, no realistic job only does computation without reading and/or writing data. We can extend the previous job model by adding I/O operations before or after the `sleep` with the `dd` command. This addition makes it less obvious how much we can afford to fold.

[1] `https://gricad.univ-grenoble-alpes.fr/index_en.html`



(a) Folding with a factor of 1 (4 physical resources and 4 virtual resources).

(b) Folding with a factor of 2 (2 physical resources and 4 virtual resources).

(c) Folding with a factor of 4 (1 physical resource and 4 virtual resources).

Figure 2: Example of folding a deployment for a system with 4 resources. Figure 2a depicts the system deployed at full scale. Figure 2c represent the system completely folded. And Figure 2b shows an intermediate folded deployment.

The questions that we want to answer are thus the following:

1. **What is the minimum number of machines we need to deploy to emulate a full scale cluster with this job model while keeping the same performance in I/O?**
2. **Is there a trade-off between the number of physical resources used and the overhead of performances due to the folding?**

3 Definitions & Concepts

We define the **folding** of a deployment as the action of defining several "virtual resources" on a physical resource. A physical resource is a node from a cluster, and a virtual resource represents a resource on the full scale system from the point of view of the RJMS.

We also define the **folding factor** (f_{fold}) as the division of the number of physical resources in the deployment divided by the number of virtual resources. Intuitively, it represents the number of virtual resources for each physical resource:

$$f_{fold} = \frac{\#resource_{virtual}}{\#resource_{physical}} \in [1, +\infty[\tag{1}$$

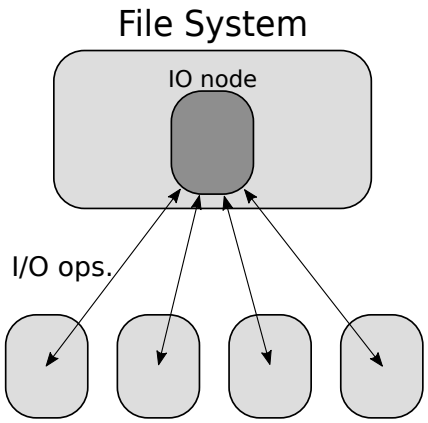
Figure 2 depicts an example of folding a deployment.

4 Methodology

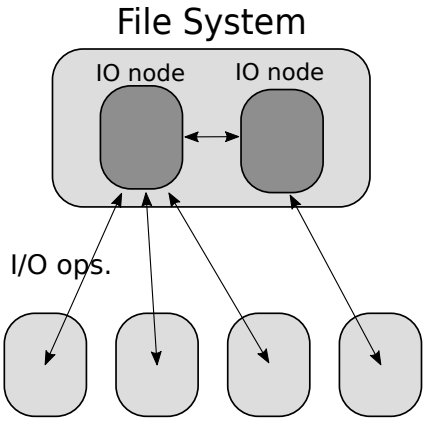
We aim at evaluating the variations in performance of a distributed application using a distributed file-system for different value of folding factors (f_{fold}). As explained in Section 2, the job model that we are using is a `sleep` time to represent the CPU bound phase, and a `dd` operation to represent the I/O phase. The sleep operation allows us to fold the CPU bound phase on the same machine easily without noise. We thus focus on the performance of the I/O operations in a folded deployment.

4.1 Experimental Setup

The following experiments were carried on the `gros` cluster, located in Nancy, of the *Grid'5000* [1] French test bed. The machines of this cluster have an Intel Xeon Gold 5220 CPU with 18 cores, 96 GiB of memory, a 2 x 25 Gbps (SR-IOV) network and a 480 GB SSD SATA Micron MTFDDAK480TDN disk. The reproducibility of the deployed environment was ensured by *NixOS Compose* [15].



(a) Distributed File system like NFS. Multiple clients query a single I/O node that process all the requests.



(b) Parallel File system like OrangeFS. Multiple clients query in parallel all the I/O nodes.

4.2 Benchmark application

To evaluate the performance of the cluster file system, we chose to use the IOR [17] benchmark. IOR is a MPI application to benchmark I/O performances. It is the most popular benchmark among research on I/O in HPC [3], and is also used in the context of the IO500 list [16].

IOR has a multitude of parameters, but give the main ones here. We used the POSIX protocol (`api=POSIX`), a transfer size (`transferSize`) of 1Mbytes and a segment count of 1 (`segmentCount`). We assigned a single file per IOR process (`filePerProc`), and checked the correct writing and reading afterwards (`checkWrite` and `checkRead`). The number of tasks (`numTasks`), which represent the number of IOR processes, and the block size (`blockSize`), which is the total size of the file to read/write in this case, are parameters of the following experiments. We used OpenMPI with the TCP backend.

4.3 Distributed File-Systems

For the chosen distributed file-systems, we used NFS and OrangeFS.

4.3.1 NFS (v4)

NFS [20] is a popular **distributed** file-system for small clusters. There is only one server. Clients mount the file-system and can perform POSIX operations. Figure 3a depicts the simplified architecture of a distributed file system like NFS. All the clients query the same I/O node for their files. NFS servers do have several workers that can manage the requests concurrently. The NFS export options used are: `*(rw,no_subtree_check,fsid=0,no_root_squash)`. The NFS server runs under the default configuration (8 workers).

4.3.2 OrangeFS

OrangeFS [4] (or PVFS2) is a **parallel** file-system. This means that there are several servers (also called I/O nodes) to answer the requests of the clients. Figure 3b depicts the simplified architecture of a parallel file-system like OrangeFS. The clients query a I/O node of the file system. If one stripe asked by the client is not present on the query I/O nodes, the file system indicate on which I/O node to find it. We used the default configuration recommended by the OrangeFS installer (the I/O nodes host both the metadata and the storage). Moreover, in our system (Section 2), we do not consider the problem of metadata.

4.4 Number of I/O nodes

In the case of PFS, like OrangeFS, we did not find any methodology nor "rule of thumb" to define the number of I/O nodes for a computing cluster. In the following, we will consider OrangeFS file-systems with 1, 2, or 4 I/O nodes. Note that in the case of NFS there is only one I/O node.

4.5 I/O load

We consider 5 different sizes of I/O operations to perform, both in writing and reading: 1Mbytes, 10Mbytes, 100Mbytes, 500Mbytes, and 1Gbytes. These file sizes will be the values of the IOR `blockSize` option.

4.6 Number of CPU nodes

As a first step, and because we aim at emulating clusters from regional meso-centers, we will thus consider small clusters of 8, 16, 24, and 32 nodes. These number of nodes will be the values of the IOR `numTasks` option: one tasks per node of the cluster.

4.7 Protocol

Let us consider a system with N CPU nodes. We first deploy the system at full scale, *i.e.,* N machines and the I/O nodes of the file system. As IOR is an MPI application, we compute the `hostfile` with one `slots` per compute node. We then start the IOR benchmark, that we repeat 5 times (the IOR `repetitions` option), and gather the performance reports. We remove one compute node from the `hostfile` and recompute the `slots` for the remaining nodes to keep the number of processes (`numTasks`) constant. This protocol is then repeated for all the different variations: number of CPU, size of I/O operations, type of file system, number of I/O nodes.

Figure 4 shows a visual representation of the experimental protocol for an experiment with 4 CPU nodes.

5 Evaluation

In this Section, we present the results of the experiments presented in the previous section. We first consider the file-system of the cluster to be NFS in Section 5.1, and then OrangeFS in Section 5.2.

5.1 Evaluation of NFS

Figure 5 shows the results of the experiments when using NFS. We also plot the 95% confidence intervals. Note that we **did not** remove the outliers. We notice that the writing performances (bottom row) does not seem to be affected by the folding of the deployment as it remains flat. However, for the reading performances (top row), we can see that the reading times increasing for higher folding factor. This means that the more we fold, the more we degrade the reading performances.

Take away # 1

Write operations on NFS do not seem to be affected by the folding of the deployment. On the other hand, read operation performances degrade the more the cluster is folded with a quadratic behavior.

We modeled the reading performances based on the size of the file to read and the number of CPU nodes involved. Figure 6 shows the results of a linear regression between the reading time and the folding ratio, file size and number of CPU nodes involved. We fitted a model with the following form:

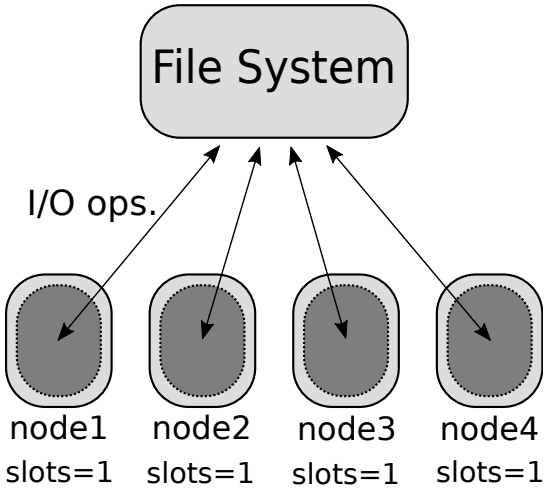
$$t_{read}(f_{fold}, f_{size}) \simeq \alpha + \beta_1 {f_{fold}}^2 + \beta_2 f_{size} + \gamma {f_{fold}}^2 f_{size} \tag{2}$$

with $\alpha, \beta_1, \beta_2, \gamma \in \mathbb{R}$ the coefficients of the model. The R^2 of the fitting is 0.9982.

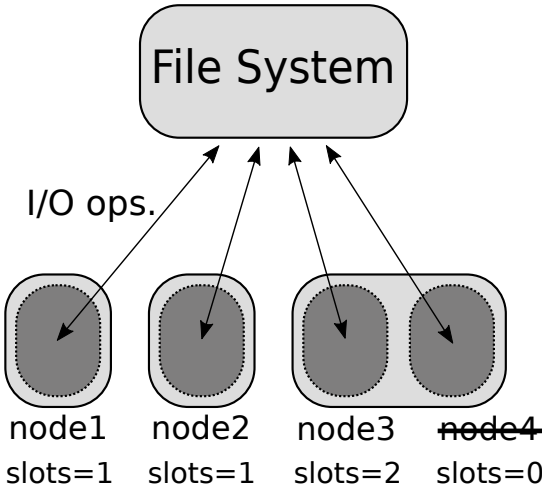
Figure 6 shows the fitting of the model on the NFS data. The bottom row represents the same information as the top one, but in log scale. We can see that the model fits all the file sizes but for 1M. We believe that the variations in performance for the 1M files are due to noise.

We are interested in knowing the maximum folding factor (f_{fold}) possible for a desired file size (f_{size}). Let $p > 1$ be the percentage of increased reading time compared to the full scale deployment containing nb_{cpu} CPU nodes. By using the definition of the model for t_{read} in Equation 2, we get:

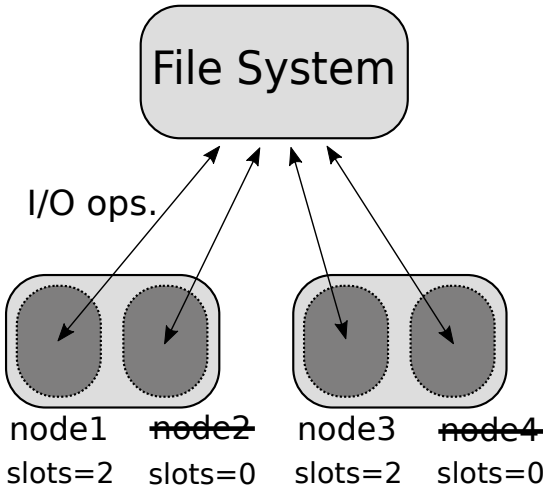
$$\frac{t_{read}(f, f_{size})}{t_{read}(nb_{cpu}, f_{size})} < p \implies f < \sqrt{\frac{p \times t_{read}(nb_{cpu}, f_{size}) - (\alpha + \beta_2 f_{size})}{\beta_1 + \gamma f_{size}}} \tag{3}$$



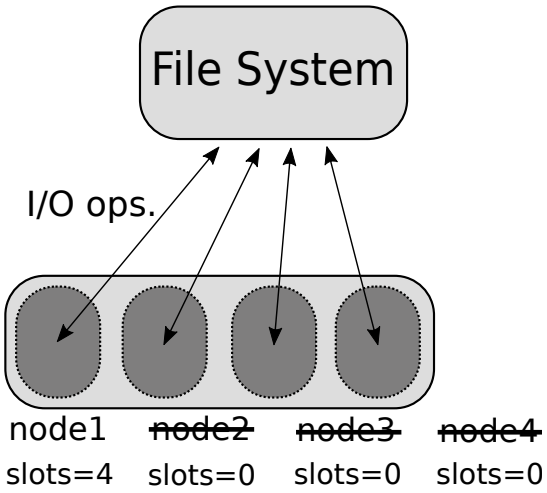
(a) Initial deployment at full scale: one process per node.



(b) We remove `node4` and add one extra `slot` to `node3`.



(c) We remove `node2` and add one extra `slot` to `node1`.



(d) We remove `node3` and reach a fully folded deployment.

Figure 4: Graphical representation of the experimental protocol presented in Section 4. We start with a full scale deployment, *i.e.,* one MPI process (viewed as a virtual resource) per physical node (Figure 4a), and remove from the `hostfile` the physical nodes one by one while keeping the number of MPI processes (*i.e.,* the number of virtual resources) constant. We recompute the number of `slots` per node to balance the processes (Figures 4b and 4c). The experiment stops when there is no more node to remove (*i.e.,* after Figure 4d).

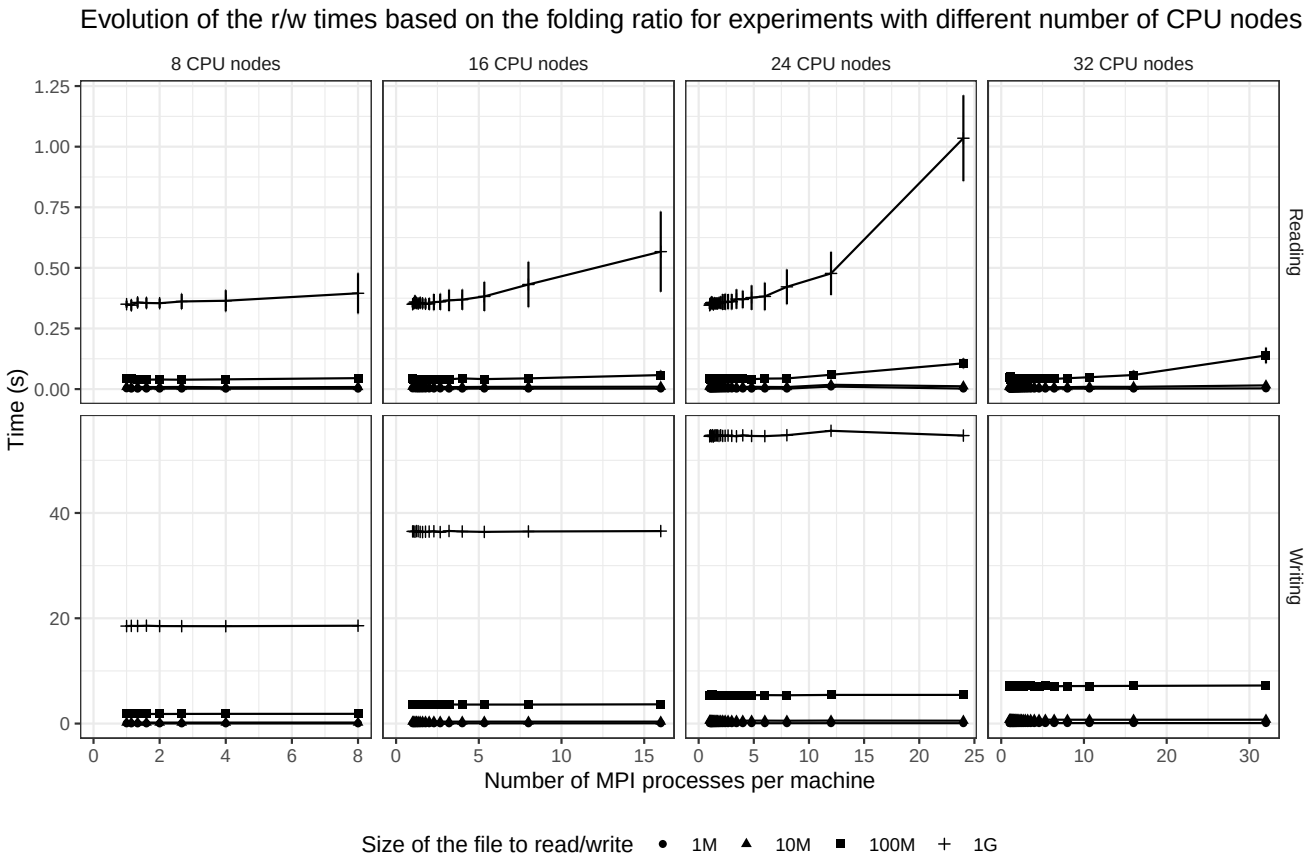


Figure 5: Evolution of the reading (top row) and writing (bottom row) times based on the folding factor (x-axis) for experiments with different cluster size (*i.e.,* number of CPU nodes) and different sizes of file to read and write (point shape). We observe that the writing performances are not affected by the folding, but that the reading ones are, and that the degradation has quadratic growth with respect to the folding factor.

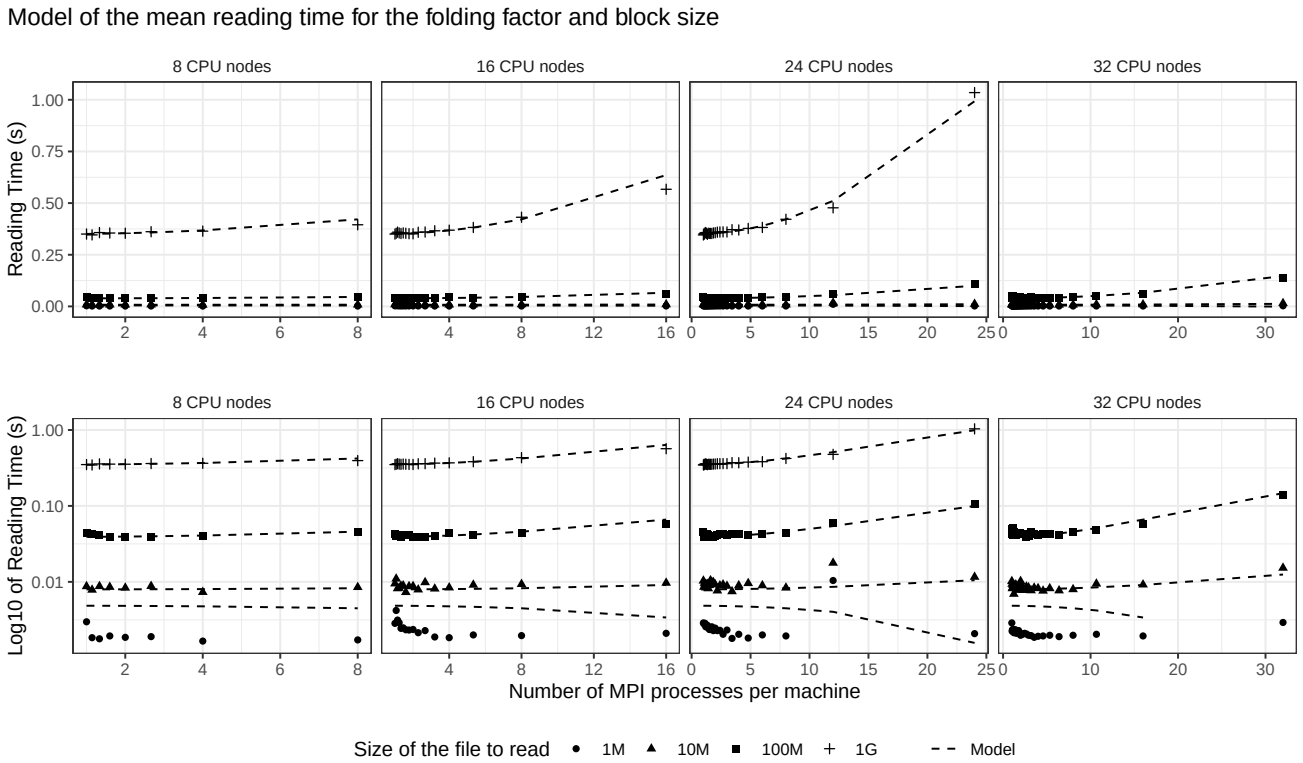
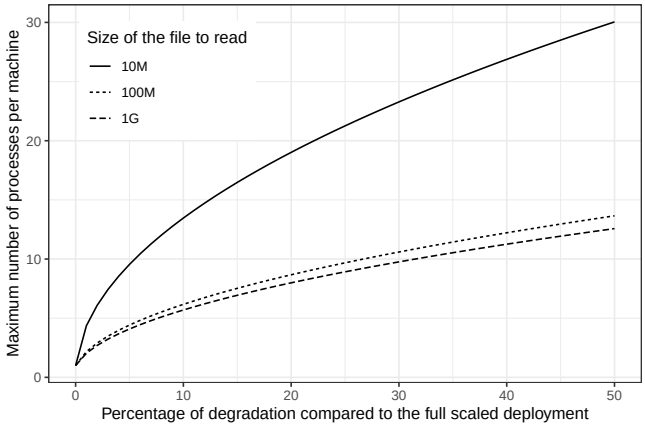
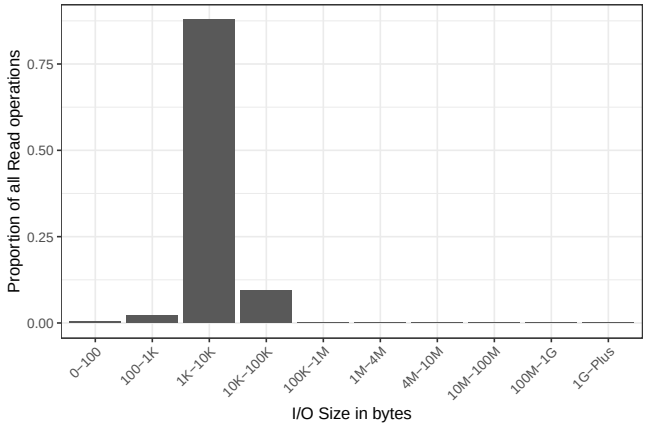


Figure 6: Linear regression modeling the reading times (y-axis) and the folding factor (x-axis), file size (point shape). The top row shows the fitting of the model on the data, and the bottom row the same data but in *log* scale. We can see that the model fits correctly the data for file sizes greater than 10M. 1M files does not seem to be affected by the folding, and their variation in performance seem to be due to noise.



(a) Maximum folding factor (f_{fold}) based on the accepted degradation of the reading time compared to the full scale deployment and to the size of the file to read.



(b) Proportion of the size of the read operations on ANL-Theta between 2019 and 2022. The majority is smaller than 100K. These values are extracted from Darshan [7, 6] logs.

Figure 7: Figure 7a shows the maximum folding factor to use to have a desired overhead on the reading times on NFS base on the file size. Figure 7b shows the distribution of the number of read requests per size on ANL-Theta.

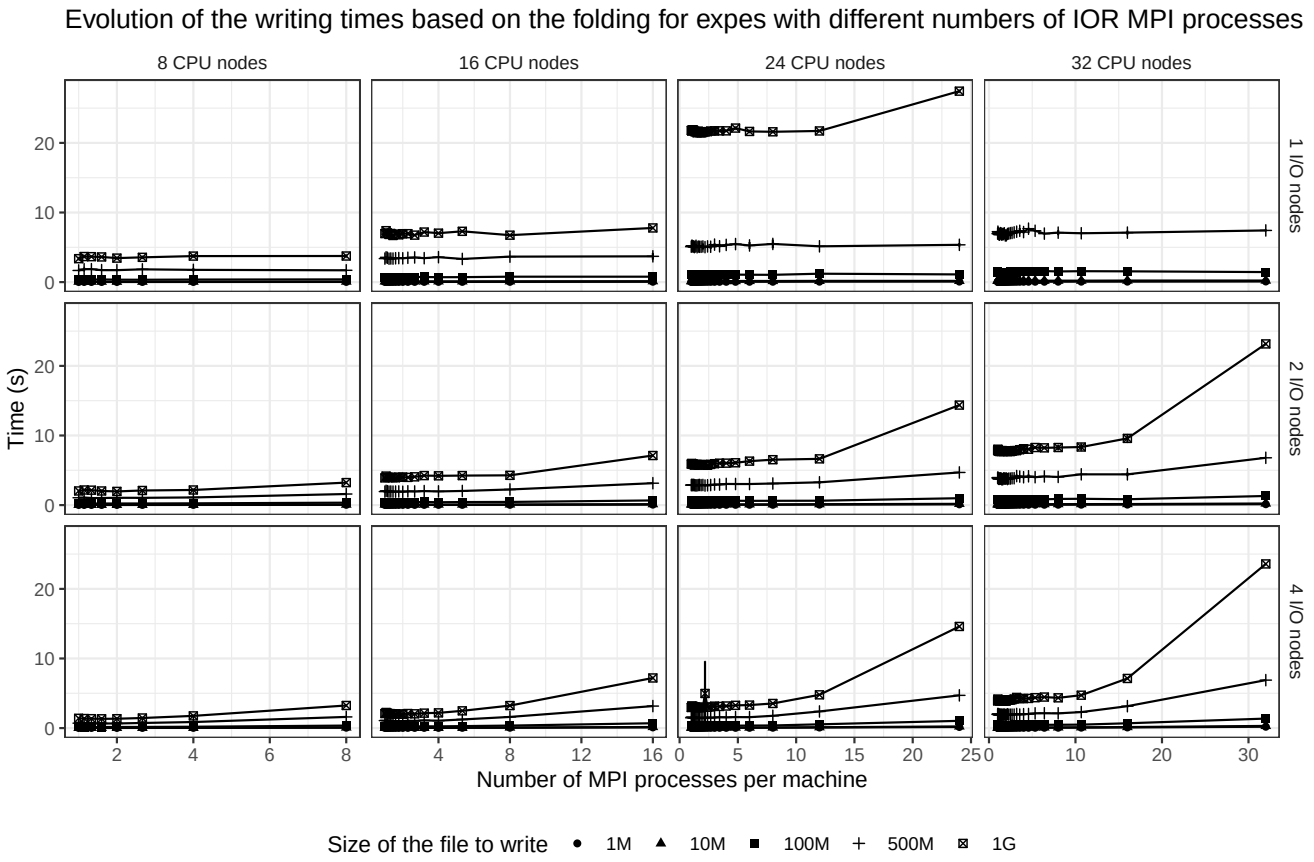


Figure 8: Evolution of the writing times with OrangeFS based on the folding factor (f_{fold}) for experiments with different number of CPU and I/O nodes and different sizes of file to write.

Take away # 2

For files of size 10M, folding 10 resources onto a single physical resource leads to a degradation of 5%. To reach the same degradation for file size of 100M or 1G, the maximum folding factor would be 5. (Figure 7a)

From the model defined previously, Figure 7a, and the Darshan [7, 6] logs from ANL-Theta between 2019 and 2022 (Figure 7b)[2], we can have an estimation of the overhead if we decided to rerun these Darshan logs (on NFS and with the job model considered in Section 2) with different folding factors. For example, a folding factor of 10 would lead to an overhead of 64 hours over 4 years (*i.e.,* an increase of 0.2%), while requiring 10 times fewer machines.

5.2 Evaluation of OrangeFS

In the case of OrangeFS, there is an extra dimension to explore: the number of I/O nodes in the file-system.

Figures 8 and 9 show respectively the evolution of the performance in reading and writing time of the IOR benchmark for different number of CPU nodes in the cluster and I/O nodes in the file-system, as well as different sizes of file to read/write.

We can see on Figure 8 that contrary to NFS, there is a significant loss of performance for the write operations when increasing the folding factor. This loss of performance appears more significant when there are more I/O nodes in the file-system.

Concerning the reading performances (Figure 9), we observe the same behavior as for NFS. High folding factors lead to an increase of reading time. The increase appears greater when there are more I/O nodes in the file-system.

[2]This data was generated from resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

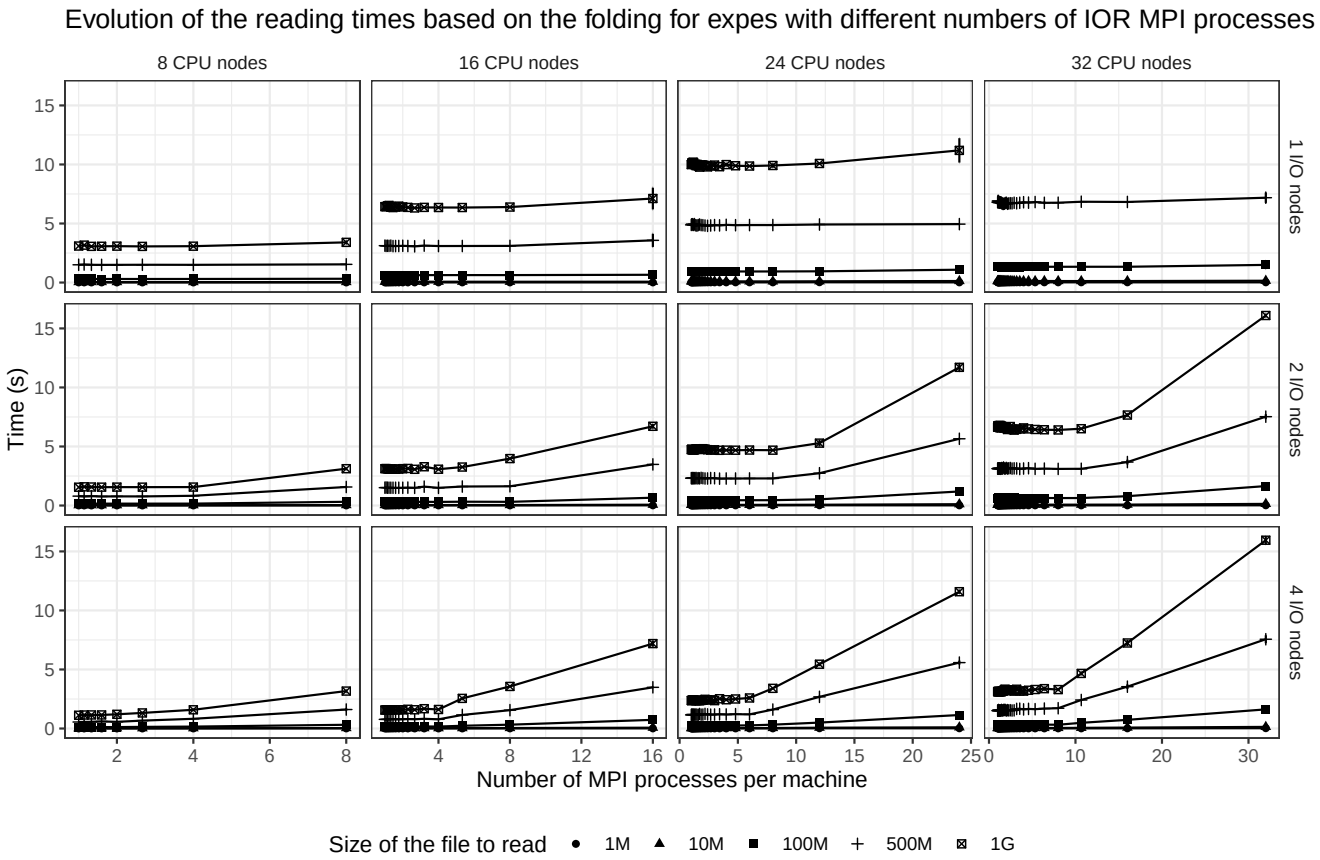


Figure 9: Evolution of the reading times with OrangeFS based on the folding factor (f_{fold}) for experiments with different number of CPU and IO nodes and different sizes of file to read.

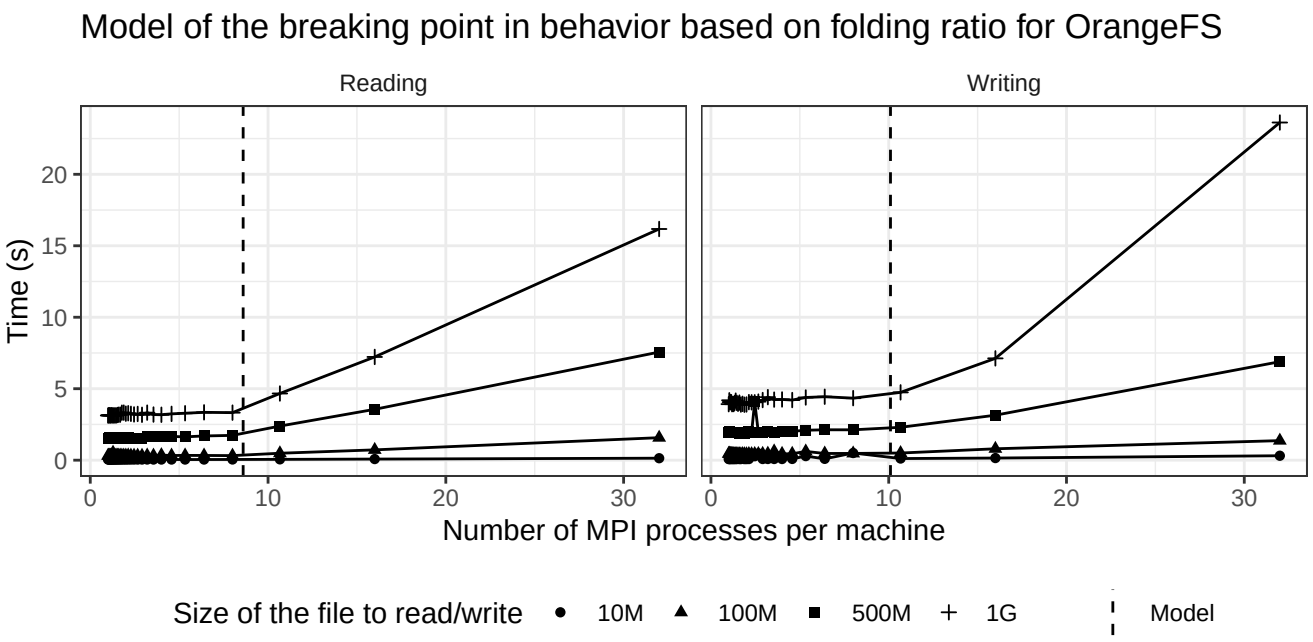


Figure 10: Model of the breaking point in behavior of performance in reading (left) and writing (right) for 32 CPU nodes (nb_{cpu}) and 4 I/O nodes (nb_{io}). The model (dashed line) comes from Equation 4.

The start of this increase seems to depend on the number of CPU nodes. The more CPU nodes, the later the increase starts. It is interesting to note that the variation when measuring the reading time during the experiments is smaller (thus more stable) than for NFS.

Take away # 3

The reading performance of a fully folded deployment **does not depend** on the number of I/O nodes in OrangeFS (Figure 9).

For both reading and writing performances there seem to be two behaviors. First a phase where the folding does not affect the performances, and then, from a given folding factor, the reading/writing times grow linearly. As we want to know the maximum folding factor until which the folded system behaves like a full scale system, we are now interested in finding this breakpoint where the behavior changes. Using segmented regression techniques [19], we found a model that we simplified to make it a *rule of thumb*:

$$\begin{aligned} f_{break,r} &\simeq 1 + 0.3 \times nb_{cpu} - 0.5 \times nb_{io} \\ f_{break,w} &\simeq 2 + 0.3 \times nb_{cpu} - 0.5 \times nb_{io} \end{aligned} \tag{4}$$

where, $f_{break,r}$ and $f_{break,w}$ are respectivelly the folding factor of the breakpoint for the reading and writing performances, nb_{cpu} and nb_{io} are the number of CPU and I/O nodes in the system.

Take away # 4

For writing and reading times on OrangeFS, there is a breakpoint in performance before which there is no degradation. We modeled *rules of thumb* to them in Equation 4. These *rules of thumb* are depicted in Figure 10.

As $f_{break,w}$ (Equation 4) will always be greater than $f_{break,r}$, **the overall breaking point in performance for OrangeFS is** $f_{break,r}$.

6 Conclusion

In this paper, we investigated the use of *folding* techniques to reduce the number of deployed machines for the large scale evaluation of applications such as grid and cluster middlewares. We have seen that *folding* requires a change in the job model We focused on the impact on the performances of the file-system. To do so, we took a distributed I/O benchmark, IOR, and ran it for several cluster sizes, I/O loads and distributed file-systems. We analyzed the results of the benchmark and reached to the following conclusions:

- Write operations on NFS are not subject to folding (Take away 1).
- The performance of read operations on NFS can be modeled with a quadratic relation, and this model can be used to determine the maximum folding for an accepted degradation (Take away 2).
- The performance of read operations in a fully folded cluster with OrangeFS do not depend on the number of I/O nodes (Take away 3).
- There are breaking point in reading and writing performance for OrangeFS when the fold the cluster. Equation 4 gives *rules of thumb* to estimate these breakpoints (Take away 4).

This study presents some limitations. The studied file-systems are not among the most popular in large HPC centers. It would be interesting to consider file-systems such as Lustre[18] of BeeGFS[2]. The reasons of this loss of performance due to folding are still unclear. The main suspect is the network. We did observe a speed similar (23 Gbps) to the one advertised on the NIC (25 Gbps). We think that the overhead can be due to the TCP protocol. Investigating different network protocol like InfinyBand or OmniPath would be interesting.

Acknowledgments

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

References

[1] Daniel Balouek et al. "Adding Virtualization Capabilities to the Grid'5000 Testbed". In: *Cloud Computing and Services Science*. Ed. by Ivan I. Ivanov et al. Vol. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, pp. 3–20. ISBN: 978-3-319-04518-4. DOI: 10.1007/978-3-319-04519-1_1.

[2] *BeeGFS File-System*. https://www.beegfs.io/c/. Accessed: 2023-01-19.

[3] Francieli Zanon Boito et al. "A checkpoint of research on parallel i/o for high-performance computing". In: *ACM Computing Surveys (CSUR)* 51.2 (2018), pp. 1–35.

[4] Michael Moore David Bonnie et al. "OrangeFS: Advancing PVFS". In: *USENIX Conference on File and Storage Technologies (FAST)*. 2011.

[5] N. Capit et al. "A batch scheduler with high level components". en. In: *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.* Cardiff, Wales, UK: IEEE, 2005, 776–783 Vol. 2. ISBN: 978-0-7803-9074-4. DOI: 10.1109/CCGRID.2005.1558641. URL: http://ieeexplore.ieee.org/document/1558641/ (visited on 05/25/2020).

[6] Philip Carns et al. "24/7 characterization of petascale I/O workloads". In: *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE. 2009, pp. 1–10.

[7] Philip Carns et al. "Understanding and improving computational science storage access through continuous characterization". In: *ACM Transactions on Storage (TOS)* 7.3 (2011), pp. 1–26.

[8] Henri Casanova et al. "Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms". In: *Journal of Parallel and Distributed Computing* 74.10 (June 2014), pp. 2899–2917. URL: http://hal.inria.fr/hal-01017319.

[9] Pierre-François Dutot et al. “Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator”. In: *20th Workshop on Job Scheduling Strategies for Parallel Processing*. Chicago, United States, May 2016. URL: https://hal.archives-ouvertes.fr/hal-01333471.

[10] Yiannis Georgiou, Olivier Richard, and Nicolas Capit. “Evaluations of the lightweight grid cigri upon the grid5000 platform”. In: *Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007)*. IEEE. 2007, pp. 279–286.

[11] Quentin Guilloteau, Olivier Richard, and Éric Rutten. “Étude des applications Bag-of-Tasks du méso-centre Gricad”. In: *COMPAS 2022 - Conférence francophone d’informatique en Parallélisme, Architecture et Système*. Amiens, France, July 2022, pp. 1–7. URL: https://hal.archives-ouvertes.fr/hal-03702246.

[12] Quentin Guilloteau et al. “Collecte de ressources libres dans une grille en préservant le système de fichiers : une approche autonomique”. In: *COMPAS 2021 - Conférence d’informatique en Parallélisme, Architecture et Système*. Lyon, France, July 2021, pp. 1–11. URL: https://hal.inria.fr/hal-03282727.

[13] Quentin Guilloteau et al. “Controlling the Injection of Best-Effort Tasks to Harvest Idle Computing Grid Resources”. In: *ICSTCC 2021 - 25th International Conference on System Theory, Control and Computing*. Iași, Romania, Oct. 2021, pp. 1–6. DOI: 10.1109/ICSTCC52150.2021.9607292. URL: https://hal.inria.fr/hal-03363709.

[14] Quentin Guilloteau et al. “Model-free control for resource harvesting in computing grids”. In: *Conference on Control Technology and Applications, CCTA 2022*. Trieste, Italy: IEEE, Aug. 2022. URL: https://hal.archives-ouvertes.fr/hal-03663273.

[15] Quentin Guilloteau et al. “Painless Transposition of Reproducible Distributed Environments with NixOS Compose”. In: *CLUSTER 2022 - IEEE International Conference on Cluster Computing*. Vol. CLUSTER 2022 - IEEE International Conference on Cluster Computing. Heidelberg, Germany, Sept. 2022, pp. 1–12. URL: https://hal.science/hal-03723771.

[16] *IO500 List*. https://io500.org/. Accessed: 2023-01-19.

[17] *IOR Benchmark*. https://github.com/hpc/ior. Accessed: 2023-01-19.

[18] *Lustre File-System*. https://www.lustre.org/. Accessed: 2023-01-19.

[19] Vito MR Muggeo. “Estimating regression models with unknown break-points”. In: *Statistics in medicine* 22.19 (2003), pp. 3055–3071.

[20] Brian Pawlowski et al. “The NFS version 4 protocol”. In: *In Proceedings of the 2nd International System Administration and Networking Conference (SANE 2000*. Citeseer. 2000.

[21] Emmanuel Stahl et al. “Towards a control-theory approach for minimizing unused grid resources”. In: *Proceedings of the 1st International Workshop on Autonomous Infrastructure for Science*. 2018, pp. 1–8.

[22] Agustin Gabriel Yabo et al. “A control-theory approach for cluster autonomic management: maximizing usage while avoiding overload”. In: *2019 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE. 2019, pp. 189–195.

[23] Andy B. Yoo, Morris A. Jette, and Mark Grondona. “SLURM: Simple Linux Utility for Resource Management”. en. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Gerhard Goos et al. Vol. 2862. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60. ISBN: 978-3-540-20405-3 978-3-540-39727-4. DOI: 10.1007/10968987_3. URL: http://link.springer.com/10.1007/10968987_3 (visited on 11/22/2021).