Allocation and Placement Algorithms for Scheduling Distributed I/O Resources in HPC Systems

Alexis Bandet, Francieli Zanon Boito, Guillaume Pallez

Allocation and Placement Algorithms for Scheduling Distributed I/O Resources in HPC Systems

Alexis Bandet, Francieli Boito, Guillaume Pallez

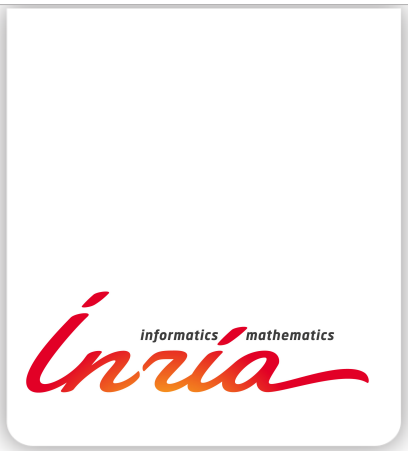
RESEARCH REPORT

N° 9549

May 2024

Project-Teams TADaaM and KerData

ISSN 0249-6399 ISRN INRIA/RR--9549--FR+ENG



Allocation and Placement Algorithms for Scheduling Distributed I/O Resources in HPC Systems

Alexis Bandet*, Francieli Boito*, Guillaume Pallez†‡

Project-Teams TADaaM and KerData

Research Report n° 9549 — May 2024 — 27 pages

Abstract: This paper presents a comprehensive investigation on optimizing I/O performance in the access to distributed I/O resources in high-performance computing (HPC) environments. I/O resources, such as the I/O forwarding nodes and object storage targets (OST), are shared by applications. Each application has access to a subset of them, and multiple applications can access the same resources. We propose heuristics to schedule these distributed I/O resources in two steps: for each application, determining *how many* (allocation) and *which* (placement) resources to use. We discuss a wide range of information about applications' characteristics that can be used by the scheduling algorithms. Despite the fact that a higher level of application knowledge is associated with better performance, our comprehensive analysis indicates that strategic decision-making with limited information can still yield significant enhancements in most scenarios. Moreover, we demonstrate the robustness of our solutions in scenarios where information is limited or inaccurate. This research provides insights into the trade-offs between the depth of application characterization and the practicality of scheduling I/O resources.

Key-words: HPC, parallel I/O, parallel file system, object storage targets, I/O forwarding, scheduling, resource allocation

* Inria Bordeaux, Université de Bordeaux, Labri - TADaam
† Inria, Univ. Rennes - KerData
‡ {alexis.bandet, francieli.zanon-boito, guillaume.pallez} @inria.fr

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Algorithmes d'allocation et de placement pour l'ordonnancement des ressources d'E/S distribuées dans les systèmes HPC

Résumé : Cet article présente une étude approfondie sur l'optimisation des performances d'E/S dans l'accès aux ressources d'E/S distribuées dans les systèmes de calcul à hautes performances. Les ressources d'E/S, telles que les nœuds de transfert d'E/S et les cibles de stockage d'objets (OST), sont partagées par les applications. Chaque application a accès à un sous-ensemble de ces ressources, et plusieurs applications peuvent accéder aux mêmes ressources. Nous proposons des heuristiques pour l'ordonnancement de ces ressources d'E/S distribuées en deux étapes : pour chaque application, déterminer *comment* (allocation) et *quelles* (placement) les ressources à utiliser. Nous discutons d'un large éventail d'informations sur les caractéristiques des applications qui peuvent être utilisées par les algorithmes d'ordonnancement. Malgré le fait qu'un niveau plus élevé de connaissance des applications est associé à des meilleures performances, notre analyse indique que la prise de décision stratégique avec des informations limitées peut encore produire des améliorations significatives dans la plupart des scénarios. En outre, nous démontrons la robustesse de nos solutions dans les scénarios où les informations sont limitées ou inexactes. Cette recherche permet de mieux comprendre les compromis entre la profondeur de la caractérisation de l'application et l'aspect pratique de l'ordonnancement des ressources d'E/S.

Mots-clés : calcul à hautes performances, E/S parallèles, système de fichiers parallèle, cibles de stockage d'objets, ordonnancement, allocation de ressources

Contents

1 Introduction

In large high-performance computing (HPC) platforms, applications access persistent data by performing I/O operations to a remote shared parallel file system (PFS), such as Lustre or BeeGFS. Because of the persistent gap between processing and I/O speeds, and with processing power ever increasing, many HPC applications spend a large portion of their time on I/O. This access is often synchronous — meaning the application occupies the compute resources while waiting to complete I/O transfer. Therefore, improving I/O performance promotes a more efficient usage of the expensive and power-hungry HPC compute resources.

Parallel file systems cut files into fixed-size stripes and distribute them across a number of storage targets (OSTs) for parallel access. Depending on the files that they access, all compute nodes may require access to the same OSTs at the same time. Thus, to mitigate contention, a layer of I/O forwarding nodes (or simply "I/O nodes") is sometimes placed between compute nodes and the PFS [2]. Both OSTs and I/O nodes are I/O resources, and it is important to notice both are potentially shared by running applications. Other shared resources include burst-buffer nodes, present in some systems [6, 26].

When applications access the same I/O resources concurrently, their I/O performance can be slowed down [34, 37, 32], and hence they occupy compute resources for longer. In addition to wasting resources, the fact that I/O performance depends on what others are doing in the system leads to higher performance variability [16, 27], which makes execution time less predictable and, consequently, complicates resource management [11, 21].

Many techniques have been proposed to mitigate contention, mainly on scheduling the accesses to the PFS [20, 18, 11], burst-buffers [3] and I/O-aware batch scheduling [24, 8].

However, these efforts usually see the shared I/O infrastructure as a single resource capable of a certain bandwidth, whereas in practice it is a distributed set of resources from which each application can use a subset. In addition, using X% of the OSTs, for example, does **not** grant a job X% of the PFS' peak performance [15, 10]. Indeed, as we discuss in Section 2, depending on their characteristics, each application will be impacted differently by the number of used I/O resources [5, **?**, 17].

In this paper, we present a comprehensive study of the problem of scheduling shared I/O resources— I/O nodes, OSTs, etc — to HPC applications with the goal of mitigating contention and improving I/O and system performance. We tackle this problem by proposing heuristics to answer two questions: 1) *how many* resources should we give each application (allocation heuristics), and 2) *which* resources should be given to each application (placement heuristics). These questions are not independent, as using more resources often means sharing them. Nonetheless, our two-step approach allows for simpler heuristics that would be usable in practice. Moreover, it allows for studying the impact of these two steps separately. Our main contributions are:

- We accurately model the problem of scheduling distributed I/O resources, and propose allocation and placement algorithms.

- An important aspect, which impacts how "implementable" algorithms are, is their requirements input-wise. Indeed this information is often not available or at least imprecise. We discuss the quality of various input parameters and study their impact with the goal of answering questions about the importance of accurate application description, and how robust the heuristics are to inaccurate information.

The rest of this paper is organized as follows: Section 2 further motivates this work by providing background on the relationship between the number of I/O resources and performance.

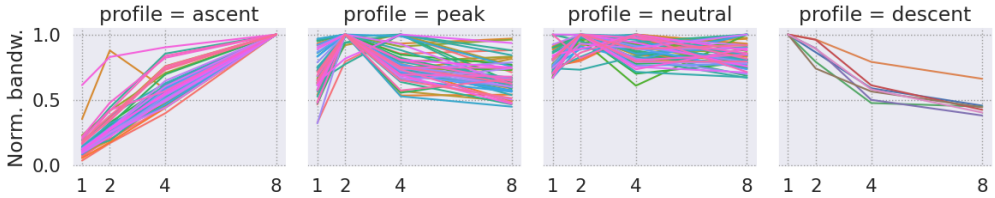


Figure 1: Normalized bandwidth as a function of the number of I/O nodes for 189 different applications, grouped by behavior. Data from [5].

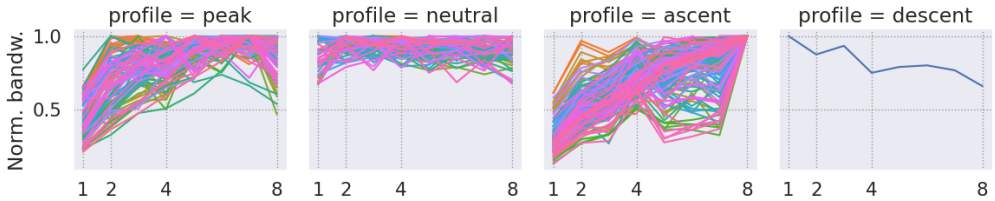


Figure 2: Normalized bandwidth as a function of the number of OSTs for 301 benchmark configurations, grouped by behavior. Section 5.1 details how these results were obtained.

Section 3 formally states the studied problem, and then Section 4 discusses the proposed heuristics. The evaluation methodology is detailed in Section 5, and results in Section 6. Section 7 discusses related work, and Section 8 concludes this paper.

2 Motivation

For both I/O nodes and OSTs, I/O performance depends on the number of I/O resources, and the impact of the latter on the former depends on their access pattern. In the case of I/O nodes, that was shown to be the case by Bez et al. [5]. We plotted data from their work in Figure 1, where we see four different behaviors: increasing the number of I/O nodes can have no impact on performance, improve it, decrease it, or increase it until a point from where having more I/O nodes starts to harm it.

Boito et al. [10] studied the impact of the number of OSTs on performance and concluded that the more, the better. However, their analysis was limited to a single access pattern. A previous study by Chowdhury et al. [15] with the same file system, but using other applications, concluded the number of OSTs had little or no impact. On the other hand, Xu et al. [33] concluded that for collective reads using more OSTs actually *degraded* performance. Figure 2 shows our own results, described in Section 5.1.

In practice, file systems are configured with a fixed number of OSTs per file, which is typically rather small to minimize sharing [30, 15]. Moreover, in most systems the number of I/O nodes assigned to a job, either depends on its number of compute nodes, which is not necessarily linked to its I/O behavior, or in some cases is a fixed number whatever the size of the job (it used to be seven on Theta). In all cases, sharing I/O resources is generally considered something to be avoided, as it may harm performance. For example, Bez et al. [7] proposed an algorithm to schedule I/O nodes to jobs which aims at providing exclusive access to the applications with higher I/O performance.

Nonetheless, over the years studies of the HPC I/O workload have consistently pointed to

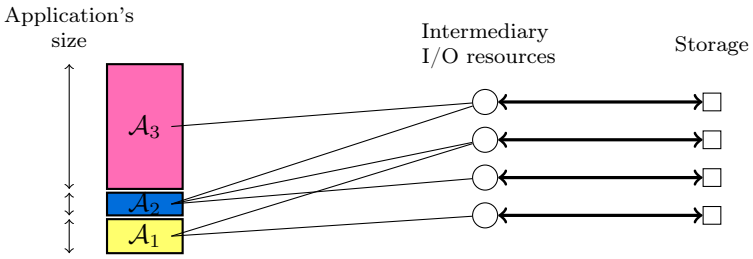


Figure 3: Representation of the architecture: three applications $\mathcal{A}_1$ (resp. $\mathcal{A}_2$, $\mathcal{A}_3$) use two (resp. three, one) I/O resources. The number of I/O resources is not necessarily correlated to the size (number of compute nodes) of the application.

a bursty behavior, with most applications actually spending only a small portion of their time on I/O operations [31, 13, 34, 25, 36]. That means an exclusive allocation of I/O resources to applications, if made for their whole execution, would not be the most efficient approach. The alternative would be to dynamically change the scheduling of I/O resources whenever application behavior changes. However, that would require a quickly available characterization of current behavior, which is in general not available, and fast implementation of the calculated schedule. Since the behavior may change very frequently for each job, the desired granularity may be only a few milliseconds. For these reasons, in this paper we focus on algorithms (presented in Section 4) that make decisions for entire executions and that do *not* focus on exclusive access, as we believe this approach is more suitable for use in practice.

3 Model

This section describes the platform (shown in Fig. 3) and application models we use in this study.

3.1 Platform model

We assume that we have a parallel platform composed of compute nodes and remote shared storage. To access this storage, each application must communicate first through distributed I/O resources, which can be I/O nodes, OSTs, burst buffer nodes, etc. There are N I/O resources. We consider that access to compute nodes are exclusive (congestion-free), hence in this work we focus on a performance model for I/O.

Capacity sharing When multiple applications access concurrently an I/O resource, they share equally its capacity. In other words, the bandwidth of each I/O resource is divided by the number of applications using it at this time.

3.2 Application model and I/O behavior

K applications run concurrently on the platform. Each application is a series of phases that alternate between computation and I/O subphases [14, 19, 20]. We study the synchronous I/O case where compute and I/O subphases cannot overlap. The key parameters describing an application and used in the rest of this paper are summarized in Table 1.

The length of each I/O subphase depends on the number of I/O resources allocated to application $\mathcal{A}_j$. The bandwidth of $\mathcal{A}_j$ as a function of I/O resources is given by: $n \mapsto b_j(n)$. If $\mathcal{A}_j$

Table 1: Key parameters for application $\mathcal{A}_j$.

Q_j	Number of compute resources
p_j	Number of phases (compute then I/O)
$t_{\text{cpu}}^{(i)}$	Length of the compute subphase of phase $i \leq p_j$
$v_{\text{io}}^{(i)}$	Volume of I/O transferred in phase $i \leq p_j$
T_{cpu}^j	Accumulated compute time over all compute (sub)phases
V_{io}^j	Accumulated volume of I/O over all I/O (sub)phases
b_j	I/O bandwidth as a function of the number of I/O resources

uses n_j I/O resources, when there is no congestion, its aggregated I/O time (for amount of data V_{io}) is: $\text{T\_cf}_{\text{io}}^j(n) = \frac{V_{\text{io}}}{b_j(n)}$.

When there is no congestion, during a total time $T_{\text{cpu}}^j + \text{T\_cf}_{\text{io}}^j(n)$, $\mathcal{A}_j$ occupies n I/O resources during a time $\text{T\_cf}_{\text{io}}^j(n)$, and we have:

$$\texttt{I/O-Stress}(j,n) = \frac{n \cdot \text{T\_cf}_{\text{io}}^j(n)}{T_{\text{cpu}}^j + \text{T\_cf}_{\text{io}}^j(n)}$$

In the rest of this paper, for clarity and when there is no ambiguity, we remove the index j when talking about an application variable.

3.2.1 Characteristic values

For application $\mathcal{A}_j$, we define two characteristic values n_{perf}^j and n_{sys}^j as:

$$n_{\text{perf}}^j = \text{argmin}_n \text{T\_cf}_{\text{io}}^j(n) = \text{argmax}_n b_j(n) \quad (1)$$

$$n_{\text{sys}}^j = \text{argmin}_n \texttt{I/O-Stress}(j,n) \quad (2)$$

n_{perf}^j corresponds to the number of I/O resources that minimizes the I/O transfer time of $\mathcal{A}_j$; n_{sys}^j corresponds to the number of I/O resources that minimizes the stress (`I/O-Stress`) on the system due to $\mathcal{A}_j$.

Lemma 1. *For all applications, $n_{sys} \leq n_{perf}$.*

The proof is presented in Appendix A.

3.2.2 Independence of I/O transfers

An I/O subphase over n I/O resources can be seen as n independent I/O transfers. This means that if an application is using multiple I/O resources, and its performance is slowed-down on one of those, then the other transfers are *not* slowed down. However, an I/O subphase only ends when all its I/O transfers are over (See Fig. 4).

3.3 Measuring performance

In this Section, we start by describing a solution (Section 3.3.1), before presenting how the performance of various solutions are measured (Section 3.3.2).

Figure 4: Independence of I/O transfers with $n = 2$

3.3.1 Defining a schedule

A solution is described by two elements:

- the number of I/O resources each application uses ($\pi = (n_1, \ldots, n_K)$);
- a mapping of the applications over the I/O resources.

At a given time t, and given a schedule $\pi = (n_1, \ldots, n_K)$, we define its I/O load as:

$$\texttt{I/O-load}(\pi) = \frac{1}{N} \sum_{i=1}^{K} \texttt{I/O-Stress}(i, \pi(i)) \tag{3}$$

Intuitively, this is a lower bound on the expectation of I/O time occupied by π per unit of time. By definition, `I/O-load` is minimized for the schedule $\pi_{\text{sys}} = (n^1_{\text{sys}}, \ldots, n^K_{\text{sys}})$. If `I/O-load` > 1, then the system is *saturated*. In such a scenario, typical I/O time needed by applications exceeds system capabilities.

3.3.2 Measuring a schedule's performance

Given a schedule π, we define the following optimization criteria to evaluate it.

`Mean-I/O-SlowDown` Assume application $\mathcal{A}_j$ transferred an amount of data V in total I/O time T using n_j I/O resources, then we call its `I/O-SlowDown` ρ_j the ratio of time taken to perform its I/O operations compared to the time they take when running in isolation using the number of I/O resources that minimizes this time (n^j_{perf}).

$$\rho_j = T / \left(V / b_j(n^j_{\text{perf}}) \right) \tag{4}$$

The `Mean-I/O-SlowDown` is the average of these values:

$$\texttt{Mean-I/O-SlowDown} = \frac{\sum_{j \leq K} \rho_j}{K} \tag{5}$$

To have a better qualitative understanding of the `I/O-SlowDown` and `Mean-I/O-SlowDown`, we can separate them into two components: one for the slowdown caused by not using the number of I/O resources that minimizes I/O time (ρ^{io}_j), and another for the slowdown due to congestion (ρ^{con}_j):

$$\rho_j = \rho^{io}_j + \rho^{con}_j \qquad \text{where } \rho^{io}_j = b_j(n^j_{\text{perf}}) / b_j(n_j)$$

I/O occupancy Given a schedule π, the `I/O-res-occ` O_i of an I/O resource i is the measure of the proportion of time this resource is occupied performing I/O operations. Given this value, we can define the *spread* of `I/O-res-occ`, as it represents the load imbalance over the different I/O resources:

$$\texttt{I/O-spread} = \max_i O_i - \min_i O_i \tag{6}$$

Machine utilization The previous metrics focus on I/O performance. Nonetheless, from a system administrator perspective, it may be interesting to favor I/O of applications that use more compute nodes/cores in order to improve the utilization of these resources. We define hence `Machine-Idletime` to represent the proportion of time when the compute nodes are *not* being use for computing. Let L^i be the total I/O time of application $\mathcal{A}_i$, in a time interval $[0, t]$ and exclusively using Q_i of the Q^{cpu} available compute resources. Then:

$$\texttt{Machine-Idletime} = \frac{\sum_{i=1}^{K} L^i.Q_i}{t \cdot Q^{cpu}}$$

4 Algorithms for scheduling I/O resources

As mentioned in Section 3.3, a solution is defined by answers to two questions: the number of I/O resources each application will use and the mapping of applications over the multiple I/O resources. In this section we present heuristics to provide the answers - respectively allocation (Section 4.1) and placement (Section 4.2) algorithms.

While an application is accurately described by the elements described in Section 3, in practice this data can be hard to collect and inaccurate. Thus, in Section 4.3 we further discuss the input required by the different heuristics.

4.1 Allocation algorithms

We propose five allocation policies:

- **R*andom***: each application receives a randomly picked number of I/O resources. This serves as a baseline.
- **S*tatic*:** application $\mathcal{A}_j$, running on Q_j compute nodes (out of Q^{cpu} in the system) receives $\frac{Q_j \cdot N}{Q^{cpu}}$ I/O resources, rounded to the closest positive integer. For the case of I/O nodes, this policy represents what happens in HPC systems where the mapping from compute nodes to them is static.
- **B*estBdw* (BBA)** allocates n^j_{perf} to each $\mathcal{A}_j$, i.e., the number of I/O resources that minimizes its I/O time. On the one hand, this policy minimizes ρ^{io}. However, in some cases it may increase `I/O-load` and make applications share more I/O resources, which leads to more congestion and hence to an increased ρ^{con}.
- **N*sys-Allocator* (NSYSA)** gives π_{sys}, i.e., it allocates n^j_{sys} to each $\mathcal{A}_j$ to minimize the `I/O-load`.
- **T*CPU-Allocator* (TA)**, detailed in Algorithm 1, starts at π_{sys} and then repeatedly increases the number of I/O resources of the application that maximizes the utilization of compute resources (the sum of `CPULoad`) while respecting the constraint that `I/O-load` must be smaller or equal to 1 (so the I/O system is not saturated).

$$\texttt{CPULoad}(i, n) = Q_i \cdot \frac{T^i_{\text{cpu}}}{T^i_{\text{cpu}} + \text{T\_cf}^i_{\text{io}}(n)}$$

TA aims at being a compromise between BBA and NSYSA. Note than when the `I/O-load` of π_{perf} is below 1, then it behaves as BBA.

Data: K applications
Result: An allocation π
$\pi \leftarrow$ initialized as π_{sys};
done $\leftarrow$ False;
while *not done* **do**
  $\tilde{\pi} \leftarrow \pi$;
  loadDiff $\leftarrow$ an array of size K, filled with -1;
  initIOLoad $\leftarrow$ `I/O-load`(π);
  for i *from 1 to* K **do**
    $n \leftarrow \pi(i)$;
    while $n \neq n^i_{perf}$ & *loadDiff*$(i) < 0$ **do**
      $n \leftarrow n + 1$;
      if *initIOLoad* $+ (\texttt{I/O-Stress}(i,n)/N - \texttt{I/O-Stress}(i,\pi(i))/N) \leq 1$ **then**
        loadDiff$(i) \leftarrow \texttt{CPULoad}(i,n) - \texttt{CPULoad}(i,\tilde{\pi}(i))$;
        $\tilde{\pi}(i) \leftarrow n$;
  $idx \leftarrow$ argmax(loadDiff);
  if *loadDiff*$(idx) < 0$ **then**
    done $\leftarrow$ True;
  else
    $\pi(idx) \leftarrow \tilde{\pi}(idx)$;
return π;

Algorithm 1: TCPU-Allocator (TA)

4.2 Placement algorithms

Three placement algorithms are considered.

- **R*andom-Placement* (RandP)** randomly assigns I/O resources to applications. This policy reflects what happens in practice in many HPC systems, where I/O behavior is not taken into consideration for placement.
- **G*reedy-Non-Clairvoyant* (GNC)** aims at providing a balanced number of applications per I/O resource. It sorts applications by decreasing number of I/O resources (computed by the allocation algorithm), then it places each of them going over the I/O resources in a round-robin fashion.
- **G*reedy-Clairvoyant* (GC)** strives for a balanced load across the I/O resources. It sorts applications by decreasing congestion-free I/O ratio $\text{T\_cf}_{\text{io}}/(T_{\text{cpu}}+\text{T\_cf}_{\text{io}})$, then it greedily places them on the I/O resources the least stressed.

4.3 On the difficulty of instantiating an algorithm

The eight described algorithms use different information about applications, as summarized in Table 2. The colors represent how easy to obtain we consider these values to be:

- Easy (green): the number of compute resources used by each application can be obtained from the resource manager. Similarly to the total number of available compute and I/O resources, it is easily obtained and reliable.

Table 2: Heuristics and their input

		Allocation					Placement		
		Random	Static	BBA	NSYSA	TA	RandP	GNC	GC
Easy	Q_j		x			x			
Medium	V_{io}^j				x	x			x
	T_{cpu}^j				x	x			x
	n_{perf}			x					
Hard	b_j				x	x			x

- Medium (orange): aggregated information such as the total amount of transferred data and compute time of an application can be obtained from previous runs, for example with profiling tools such as Darshan [13], or provided by the user. In both cases, the actual observed values could vary and this data is only semi-reliable. n_{perf}, is considered in this category because it does not require the whole evaluation and bandwidth values.

- Hard (red): for an application $\mathcal{A}_j$, obtaining the bandwidth as a function of the number of I/O resources (b_j) requires multiple previous runs and is naturally sensitive to variability. The system could accumulate this information over time (so it would only be available to *some* of the running applications), or the user could provide it (less reliable).

Still, rather than using the exact b_j for each application, we believe a more realistic alternative is to approximate it by a general behavior (e.g. the profiles seen in Figure 1). Given an application general I/O characteristics, such as read/write ratio, request size, etc. (medium difficulty in our classification above), it could be matched to a benchmark for which the profile is known [12]. We explore this approach, and the robustness of the studied heuristics, in Section 6.2.

5 Evaluation methodology

The evaluation in this paper relies on data from real executions, described in Section 5.1. The evaluation is done on a time-based simulator, detailed in Section 5.2. Sections 5.3 and 5.4 describe the workload generation and evaluation protocols.

5.1 Datasets

We use two sets of data with several applications $\mathcal{A}_j$ and b_j, i.e., bandwidth measurements for different numbers of I/O resources.

1. The first, obtained for 189 applications at the MareNostrum supercomputer and shown in Fig. 1, was made publicly available by Bez et al. [5] and is used for the use case of scheduling I/O nodes.

2. The second dataset (use case of OST scheduling) [9] is generated by running the IOR benchmark with different configurations in the PlaFRIM experimental platform, using numbers of OSTs for BeeGFS varying from 1 to 8. The I/O infrastructure of this platform has been detailed in [10]. The 301 configurations all write to a shared file, while covering various values for numbers of nodes, processes per node, request size, contiguous vs. 1D-strided file layout, and total amount of data. These results were shown in Fig. 2.

5.2 Simulator design

We developed a time-based simulator, available at `https://gitlab.inria.fr/hpc_io/ionode_simulator` along with instructions to reproduce our results. It consists of three phases:

1. the workload generation, described in Section 5.3;
2. the execution of the algorithms presented in Section 4. Note that we also implemented MCKP from the state of the art in I/O nodes scheduling [7]. It performs both steps (allocation + placement).
3. The evaluation of the results according to the objectives presented in Section 3.3.2. We detail how these objectives are measured in Section 5.4.

At each unit of time, the simulator examines the status of each application (compute or I/O). When there is a *collision* on an I/O resource (i.e. two or more applications try to do I/O), the colliding applications will alternate in performing I/O for one unit of time each until their I/O subphases are over. This approximates the fair-share I/O scheduling algorithm because all simulations are performed for thousands of units of time.

5.3 Workload generation protocol

We make the hypothesis that the algorithms' behavior depends on the I/O stress on the system. Hence in our generation of the application sets, we cover various `I/O-load` values.

In all the experiments we consider that we have $N = 20$ I/O resources, and $Q^{cpu} = 480$ compute resources. The number of applications K depends on the experiment, so does the target `I/O-load`: Θ. Given K and Θ, we generate $\mathcal{A}_j$ as follows:

- we pick an I/O bandwidth profile uniformly at random in the dataset;
- the number of phases p[j] is picked uniformly at random in $\{2, 3, 4, \cdots, 20\}$;
- Q_j is computed so that 10% (resp. 30%, 60%) of applications use 75% (resp. 20%, 5%) of the compute resources (large, medium, and small applications);
- for all applications, we set $T^j_{\text{cpu}} + \text{T_cf}^j_{\text{io}}(1) = 5000$s (common horizon). Then, we set $T^j_{\text{cpu}}/\text{T\_cf}^j_{\text{io}}(1) = X$, where X is picked uniformly at random in $[0, b]$. The bound b is computed so that $\mathbb{E}\left(\texttt{I/O-Stress}(j, 1)\right) = \frac{\Theta N}{K}$: b is the solution to $\log(1 + b) = b\frac{\Theta N}{K}$ (computed analytically). Consequently, we have

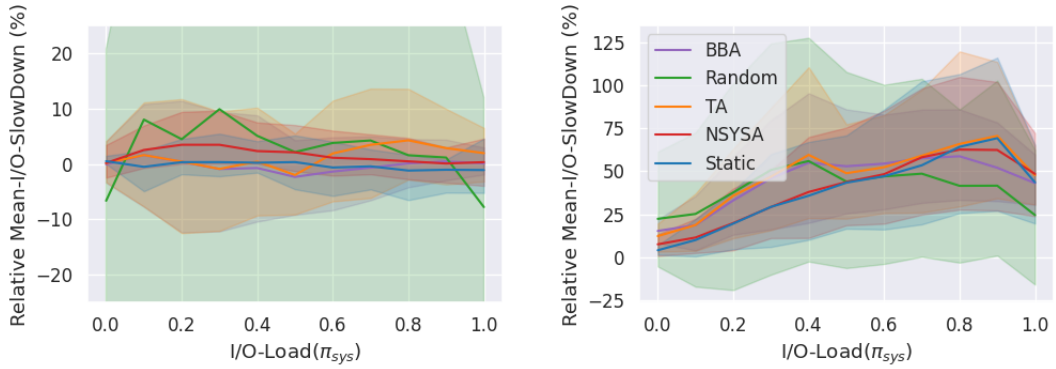
$$V_{\text{io}}[j] = 5000 \cdot b_j(1)/(1 + X); \quad T^j_{\text{cpu}} = 5000\,(1 - 1/(1 + X))$$

By construction, this generation has the property $\texttt{I/O-load}(\pi_1) = \Theta$.

In a second step and for the evaluation, we categorize the sets of applications that we have generated by their minimum `I/O-load`, i.e., $\texttt{I/O-load}(\pi_{\text{sys}}) \leq \Theta$.

5.4 Evaluation protocol

Each studied scenario is evaluated with over 100 different application sets, randomly generated as described above. The metrics are measured on an interval where the state of the system is constant (i.e. no application finishes), but of a sufficient length (i.e. each application should have performed at least a complete phase compute+I/O).



(a) GNC / GC

(b) RandP / GC

Figure 5: Relative `Mean-I/O-SlowDown` for different algorithm combinations when `I/O-load`(π_{sys}) increases. Placement are compared with GC. Lines show the mean value, and the area around them is the percentile interval (10^{th}-90^{th}).

6 Results

In this Section we provide various elements to compare the different heuristics. We focus the discussion on the impact of the input that each algorithm considers. Specifically:

- in Section 6.1, we compare the algorithms in a setup where we trust their inputs;
- then, in Section 6.2, we loosen the quality of the input information to study the robustness of our algorithms;
- finally, after using the I/O nodes case study in all previous sections, in Section 6.3 we show that the results hold with another bandwidth model (the OST case study).

6.1 Evaluation with accurate input data

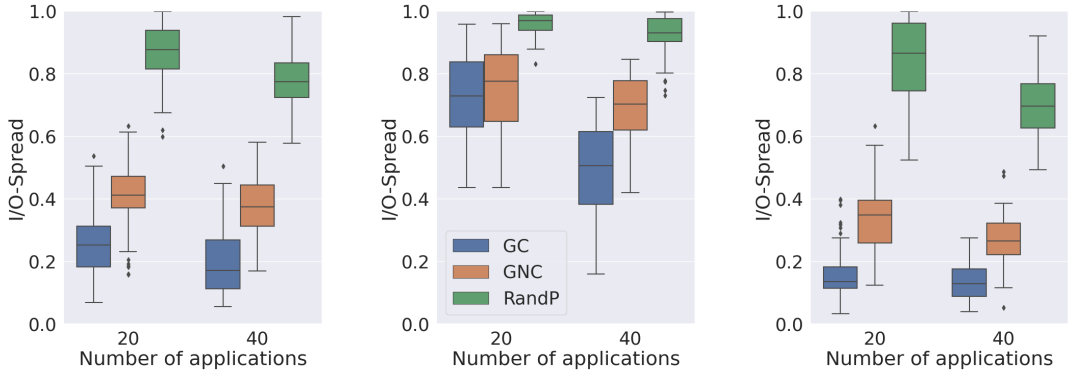
In this Section, we evaluate the different solutions based on the optimization criteria presented in Section 3.3. As already discussed, the comparison between the algorithms is performed as a function of `I/O-load`(π_{sys}), which corresponds to the level of stress on the I/O system.

6.1.1 User observed performance

In Fig. 5, we present the performance ratio in terms of `Mean-I/O-SlowDown` when using GNC (Fig. 5a) or RandP (Fig. 5b) instead of the more informed heuristic GC for all allocation algorithms. To read Fig. 5a (resp. Fig. 5b), when the TA line is at 2%, that means that on average, TA-GNC (resp. TA-RandP) has a `Mean-I/O-SlowDown` 2% worse than that of TA-GC.

The first key observation that we can make is that choosing either GC or GNC for placement has very little impact on performance (except for Random allocation). Nonetheless, this is not true for RandP, which confirms that placement plays a part in the performance. That shows that efficient I/O scheduling can be done with limited information.

Note that GC and GNC have a different behavior. We can verify this by studying the `I/O-spread`, which represents the load *imbalance* between I/O resources (Eq. (6)). We show



(a) BBA (b) NSYSA (c) Random

Figure 6: `I/O-spread` for different algorithms when `I/O-load`(π_{sys}) $= 0.5 \pm 0.05$.

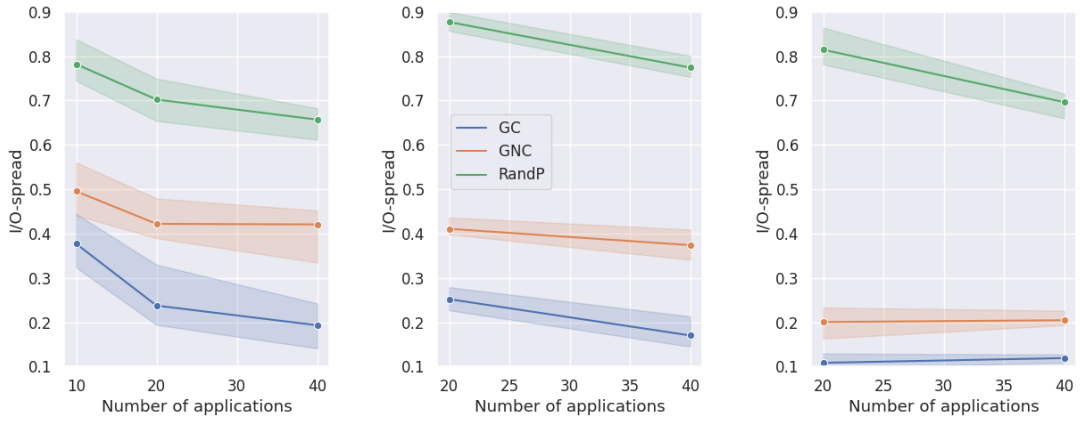
in Fig. 6 the `I/O-spread` for the three placement algorithms with several allocation algorithms. This confirms that GC balances I/O better over the I/O resources. GNC, which balances the number of applications without considering the load/stress they impose, still balances I/O better than RandP. That happens because occupancy is correlated to the number of applications. In Fig. 7, we confirm that this holds for different values of `I/O-load`.

The fact that the I/O performance is similar between GC and GNC, even if they behave differently, hints that as far as we manage to stay below a certain level of I/O stress per I/O resource, improving the load balance between I/O resources does not matter. Hence, we conclude that placement can be done greedily with limited information. *In the rest of this section, we only use* G*NC as a placement algorithm.*

To compare the allocation heuristics, we decompose the `I/O-SlowDown` into their I/O and congestion components in Fig. 8, where we show their behavior when `I/O-load`(π_{sys}) varies. By design, BBA decreases the portion of I/O time due to I/O resources allocation (ρ^{io}) to its bare minimum, at the cost of a much higher congestion overhead (ρ^{con}) than that of NSYSA or Static. As `I/O-load`(π_{sys}) increases, this cost increases linearly and may become a problem at very high `I/O-load` values. On the contrary, the congestion overhead with allocation algorithms that take into account the system `I/O-load` (NSYSA and TA) do not vary as much when `I/O-load`(π_{sys}) increases. Nonetheless, that comes at the cost of increased I/O time due to I/O resources allocation. Finally, MCKP [5] performs worse than all the other studied policies for these objectives. Indeed, it seeks to optimize the sum of applications' bandwidths due to I/O resources allocation, and has hence a tendency of favoring a few applications (the highest bandwidth ones) in detriment of others.

6.1.2 Machine utilization

Up to now we focused on the average I/O behavior, a user-oriented objective. This disadvantages more unfair algorithms such as Static, which gives more I/O resources to applications that occupy a larger fraction of the compute resources, possibly in detriment of smaller applications that are more numerous (and hence have a higher impact in a quantitative objective like `Mean-I/O-`



(a) `I/O-load`(π_{sys}) $= 0.2 \pm 0.05$ (b) `I/O-load`(π_{sys}) $= 0.5 \pm 0.05$ (c) `I/O-load`(π_{sys}) $= 0.8 \pm 0.05$

Figure 7: `I/O-spread` for different placement algorithms using BBA. Lines connect the mean values, and the area around each line shows the 95% confidence interval.

`SlowDown`). From a system administrator perspective, it may be more interesting to have an application that occupies a large part of the machine to perform its I/O fast, even if it at the cost of worse performance for smaller applications.

To evaluate this, we plot the `Machine-Idletime` of the compute nodes (i.e. the time when applications are performing I/O) in Fig. 9. We can make two observations:

- At low values of `I/O-load`(π_{sys}), the relative allocation algorithm performance in terms of `Machine-Idletime` are the same as those for `Mean-I/O-SlowDown` (BBA and TA perform the best). This is not surprising: I/O has less impact.

- However, at larger `I/O-load`(π_{sys}), we start to see a real difference of performance between TA and BBA, even though their respective I/O performance were similar, giving an advantage to the heuristic that considers more information (TA).

Static is an interesting heuristic: it uses little information about the applications and still can get good machine utilization results when the `I/O-load` is high.

6.2 Robustness to the quality of the input

In Section 6.1, we were able to demonstrate the fact that under low `I/O-load`, BBA was the most efficient allocation algorithm, whereas under a heavier `I/O-load`, one should rather use TA which provides the same I/O performance from an application perspective but improves on system utilization.

As shown in Section 4.3, there is however an important difference between these two algorithms: their input. In particular, TA requires a full I/O profile of the applications whereas BBA only requires the number of I/O nodes that provides the maximum bandwidth.

In this section, we run the same simulations as before, but giving algorithms partial (and potentially wrong) information to study how robust they are. Specifically, all applications from a given I/O performance profile (Ascent, Descent, Peak, Neutral) are said to have the same

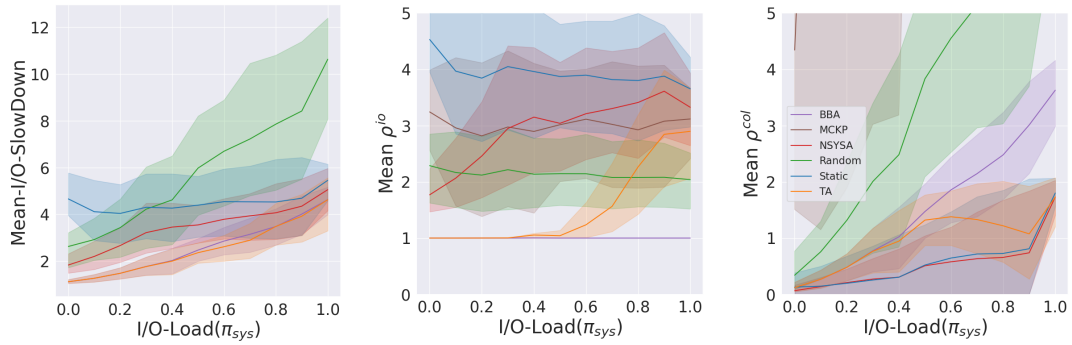
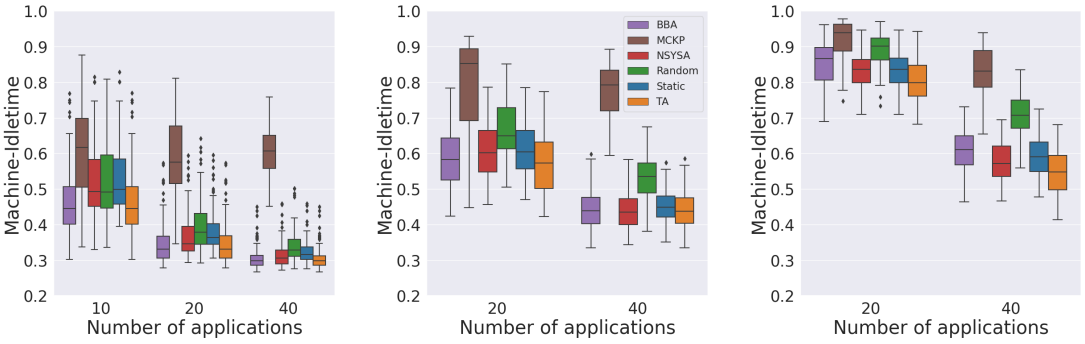


Figure 8: `Mean-I/O-SlowDown` (left) separated into its two main components: I/O resources allocation ρ^{io} and congestion ρ^{con}. The lines show the mean value, and the area around them is the percentile interval (10^{th}-90^{th}).



(a) `I/O-load`(π_{sys}) $= 0.2 \pm 0.05$ (b) `I/O-load`(π_{sys}) $= 0.5 \pm 0.05$ (c) `I/O-load`(π_{sys}) $= 0.8 \pm 0.05$

Figure 9: `Machine-Idletime` for allocation algorithms at increasing `I/O-load`(π_{sys}). The y axes do *not* start at 0. The lower the better.

bandwidth function b_j. This function is obtained by interpolating all curves within a profile to the same function (see Fig. 10).

In Fig. 11, we study the performance of three combinations of algorithms (TA+GNC; BBA+GNC; BBA+GC) with poor input accuracy. To do so, we use all generated scenarios, and plot them as a function of `I/O-load`(π_{sys}). Note that GNC is not impacted by the lack of accuracy: its behavior does not change. Similarly, BBA is impacted by the lack of accuracy only for some of the applications with a bandwidth profile `Peak` and `Neutral`.

Specifically, in Fig. 11a, we compare their performance to what was observed with accurate information: if the value is 2%, for example, it means that the algorithm with inaccurate information performed 2% worse than with accurate information.

We can observe that they have very similar performance. As expected, the allocation algorithm where the behavior differs the most is TA, which is the one that requires the most information. Yet this difference is still negligible (less than 1%). With respect to the mapping algorithm, GC performs almost identically to GNC except in cases with extremely low `I/O-load`, i.e. when the `Mean-I/O-SlowDown` is close to 1, the error in prediction is most impactful.

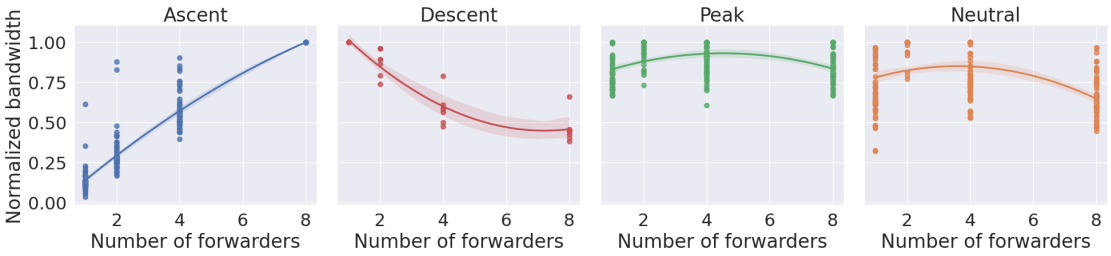
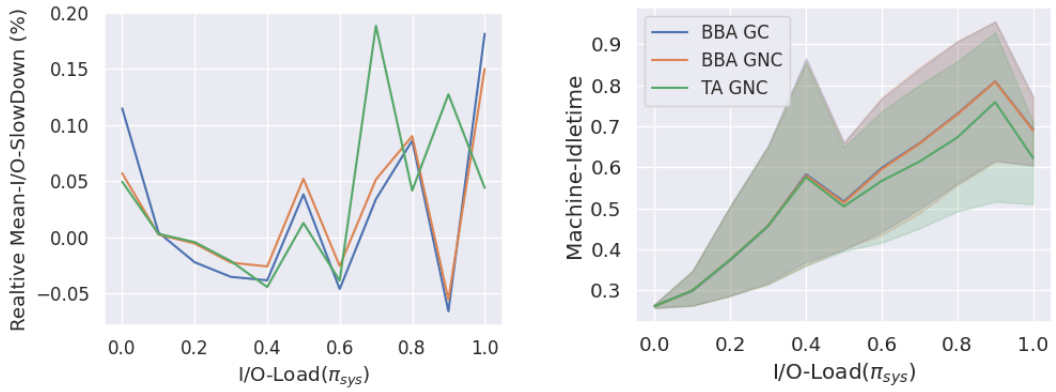


Figure 10: Polynomial regression over all application profiles.



(a) `Mean-I/O-SlowDown` compared to performance with exact information (%)

(b) `Machine-Idletime`

Figure 11: Performance of three scheduling solutions with partial input information

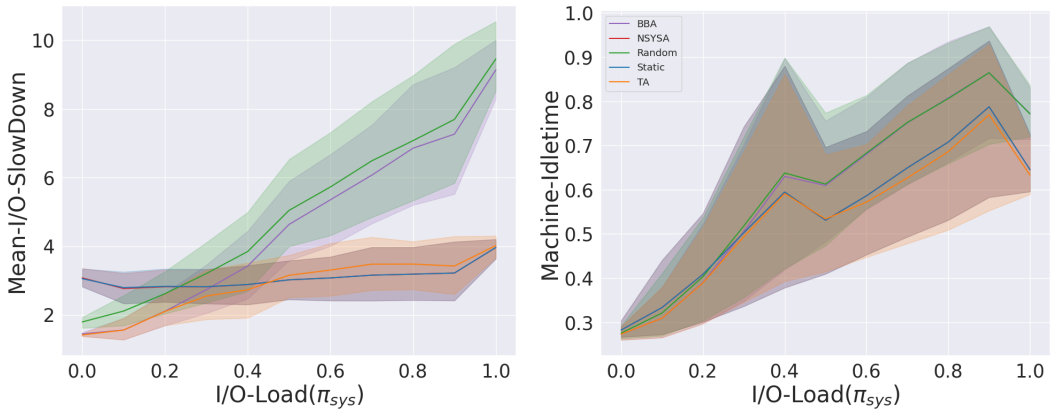
We confirm their absolute impact by plotting their `Machine-Idletime` (Fig. 11b) which confirms that even with inaccurate information TA is the best solution. More generally, we observe that the various algorithms are quite robust to some inaccuracy in I/O behavior, and that the main claims of our work with respect to choosing allocation and placement algorithms hold.

6.3 Scheduling of OSTs

In this final set of experiments, we evaluate our algorithms on a different context: OST scheduling. The main difference with I/O nodes lies on the performance obtained depending on the number of resources allocated to the application (see Fig. 1 and 2).

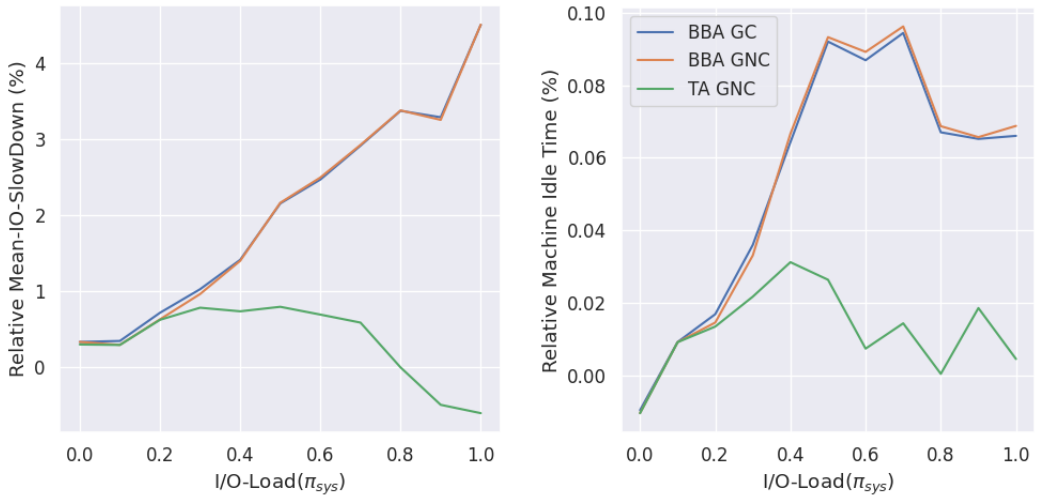
Figure 12 presents `Mean-I/O-SlowDown` and `Machine-Idletime` results using GNC as the placement policy, and Figure 13 shows relative difference between results obtained for the OST and the I/O nodes cases. BBA shows an increase in this difference as the `I/O-load` increases, which points to this case (of OST scheduling) being more sensitive to contention. Nonetheless, all differences are very low, below 5%, hence we conclude the strategies behave similarly in the two studied cases. All obtained results for the OST case are presented in Appendix B.

Overall the main observation is the same as previously: when there is little I/O stress on the system, then BBA performs better than I/O stress-aware algorithms such as NSYSA. Then



(a) `Mean-I/O-SlowDown` (b) `Machine-Idletime`

Figure 12: Results for the OST case study. Lines show the mean values, and the area around them the interval between the 10^{th} and 90^{th} percentiles.



(a) `Mean-I/O-SlowDown` (b) `Machine-Idletime`

Figure 13: Results with different scheduling solutions for the OST case, compared to the I/O nodes case.

as the stress increases, its performance quickly degrades. The main difference here is that the cut-off point is much earlier. These results confirm that TA is an excellent alternative that is able to match the performance of BBA when needed, while being I/O stress aware.

Static's performance are excellent for this use case. This is interesting because i) for OSTs, Static is not what is typically used in practice; and ii) it is quite natural and easy to implement. Unfortunately, the fact that this behavior is highly dependent on application profile (i.e. we did not see the same behavior with the I/O nodes dataset) makes the Static allocation algorithm unreliable.

7 Related work

7.1 Scheduling of I/O nodes

Some systems have a static organization of the I/O nodes layer, with each of them connected to a subset of processing nodes, which are unable to communicate with the other I/O nodes [28]. However, other machines — a notable example being the Sunway TaihuLight [22] — allow for real-time reconfiguration. In this paper, we have purposely not discussed the technical implementation aspects of I/O nodes scheduling because we aimed at studying the heuristics' potential benefits and limitations. Still, we believe the obtained results justify that systems *should* allow for this scheduling to be possible. Furthermore, at the same time, for a static system, allocation and placement decisions could be approximated by a resource manager when placing the applications in the compute nodes.

Yu et al. [38] document the load imbalance problem of I/O nodes and propose a strategy where they are statically assigned to applications in an exclusive way. Whenever the load of an application's I/O nodes it too high, it is allowed to temporarily use the I/O nodes assigned to others. Differently from us, they do not study the selection of *which* I/O nodes should be shared, hence our work is complementary to theirs.

In the work by Ji et al. [22], less than half of the I/O nodes are statically assigned to applications, and then historical data is used to decide if more nodes, from a pool of available ones, should be used. This decision on the number of I/O nodes each application should use is taken only based on the number of compute nodes that perform I/O operations, while it has since been shown (see Section 2) that other characteristics impact b_j in a more complicated way. Their approach also allocates more I/O nodes to applications to avoid sharing them with others of incompatible access patterns. Differently, we focus on more generic placement strategies, that require less information, and our results for Greedy-Non-Clairvoyant prove how effective they can be. Our techniques can be used in the absence of detailed application information and interference models.

Bez et al. [5] propose MCKP, which does allocation and placement by optimizing the sum of applications' bandwidths and favoring exclusive access as much as possible. We argue that exclusive access may be wasteful as many applications are not I/O intensive, and instead further explore the placement aspect. In Section 6, we compare our heuristics to MCKP.

7.2 Scheduling of OSTs

Yang et al. [35] reduce the problem of placing applications on I/O nodes *and* OSTs to the maximum flow problem, using their I/O load and the current monitored load of the resources. This is similar to what Greedy-Clairvoyant does, with the difference that we do not place applications on all layers of I/O resources at the same time. The strategy by Wang et al. [29] also considers all layers at once. The OST with the lowest-cost path is selected for each of the application's

compute nodes. The cost of a path depends on manually-set weights given to layers according to their importance for performance. Moreover, while they focus on improving load balance in the context of each application, our approach considers a global view of all concurrent applications.

The challenge in determining the best number of OSTs for an application is often tackled in the literature by having systems try different configurations over multiple runs. For example, Kim et al. [23] propose DCA-IO to automatically tune PFS parameters. If no information is available for an application, it receives the number of OSTs historically observed to be the best for the number of compute resources it uses. On the other hand, if the application is known, it will receive more OSTs at each execution until a local maximum is found.

Another way of obtaining such information is to train models on aggregated application metrics (more easily obtained). The auto-tuning framework proposed by Behzad et al. [4] adapts the number of OSTs by keeping a database of I/O patterns, extracted from applications, and using non-linear regression models to find the best values. Agarwal et al. [1] use Bayesian optimization and a pre-trained I/O performance model to recommend the best number of OSTs to each application. These approaches could be used together with our proposed heuristics to provide the required information.

8 Conclusion

In this work we have investigated the problem of allocating subsets of distributed I/O resources to applications in order to optimize their I/O performance and the platform utilization.

Our contributions include both allocation and placement algorithms. In their design, we have taken into account a trade-off between simplicity and efficiency. To this regard we have shown that the placement algorithm can be quite naive: balancing the absolute number of applications per I/O resource, without considering their I/O load, leads to results as good as I/O-aware placement. This naive algorithm gives more leeway to optimize placement based on other reasons (such as proximity to the applications).

In contrast, we have shown that the allocation algorithm is more important for I/O performance, and one should use a more fine-tuned algorithm rather than a naive approach such as peak bandwidth or a static approach that allocates a number of I/O resources proportional to the number of compute resources.

An important contribution of our work is the robustness study: indeed, I/O behavior has been shown to be quite volatile and hard to predict. Hence a very efficient heuristic that is not robust to volatility in its input can become quite useless. In this work we have studied different types of input that algorithms could use, and the limits of each algorithm based on these inputs. In addition, we have shown that our presented heuristics are robust to inaccuracy in input information. We believe that this opens avenues in terms of I/O behavior prediction, which is a hard problem: indeed exact information may not be needed for some of the I/O scheduling algorithms. This should considerably simplify the design of I/O analysis tools.

Acknowledgements

As part of the "France 2030" initiative, this work has benefited from a State grant managed by the French national research agency (*Agence Nationale de la Recherche*) attributed to the Exa-DoST project, and bearing the reference ANR-22-EXNU-0004. It was also supported by the Adaptive multitier intelligent data manager for Exascale (ADMIRE) project, funded by the European Union's Horizon 2020 JTI-EuroHPC Research and Innovation Programme (grant 956748). Experiments were carried out using the PlaFRIM experimental testbed, supported by

Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and *Conseil Régional d'Aquitaine* (see `https://www.plafrim.fr`).

References

[1] M. Agarwal, D. Singhvi, P. Malakar, and S. Byna. Active learning-based automatic tuning and prediction of parallel i/o performance. In *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*, pages 20–29, 2019.

[2] G. Almási, R. Bellofatto, J. Brunheroto, C. Caşcaval, J. G. Castaños, L. Ceze, P. Crumley, C. C. Erway, J. Gagliano, D. Lieber, X. Martorell, J. E. Moreira, A. Sanomiya, and K. Strauss. An overview of the blue gene/l system software organization. In *Euro-Par 2003 Parallel Processing*, pages 543–555, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[3] G. Aupy, O. Beaumont, and L. Eyraud-Dubois. Sizing and partitioning strategies for burst-buffers to reduce io contention. In *2019 IEEE international parallel and distributed processing symposium (IPDPS)*, pages 631–640. IEEE, 2019.

[4] B. Behzad, S. Byna, Prabhat, and M. Snir. Optimizing i/o performance of hpc applications with autotuning. *ACM Trans. Parallel Comput.*, 5(4), mar 2019.

[5] J. L. Bez, F. Z. Boito, A. Miranda, R. Nou, T. Cortes, and P. O. A. Navaux. Towards On-Demand I/O Forwarding in HPC Platforms. In *2020 IEEE/ACM Fifth International Parallel Data Systems Workshop (PDSW)*, pages 7–14, Nov. 2020.

[6] J. L. Bez, A. M. Karimi, A. K. Paul, B. Xie, S. Byna, P. Carns, S. Oral, F. Wang, and J. Hanley. Access patterns and performance behaviors of multi-layer supercomputer i/o subsystems under production load. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '22, page 43–55, New York, NY, USA, 2022. Association for Computing Machinery.

[7] J. L. Bez, A. Miranda, R. Nou, F. Z. Boito, T. Cortes, and P. Navaux. Arbitration Policies for On-Demand User-Level I/O Forwarding on HPC Platforms. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 577–586, May 2021. ISSN: 1530-2075.

[8] R. Bleuse, K. Dogeas, G. Lucarelli, G. Mounié, and D. Trystram. Interference-aware scheduling using geometric constraints. In *Euro-Par'18*, pages 205–217. Springer, 2018.

[9] F. Boito. Write performance with different numbers of OSTs for BeeGFS in PlaFRIM (Zenodo data set), available at `https://doi.org/10.5281/zenodo.10518127`, Jan. 2024.

[10] F. Boito, G. Pallez, and L. Teylo. The role of storage target allocation in applications' I/O performance with BeeGFS. In *CLUSTER 2022 - IEEE International Conference on Cluster Computing*, Heidelberg, Germany, Sept. 2022.

[11] F. Boito, G. Pallez, L. Teylo, and N. Vidal. IO-SETS: Simple and efficient approaches for I/O bandwidth management. *IEEE Transactions on Parallel and Distributed Systems*, 34(10):2783 – 2796, Aug. 2023.

[12] F. Z. Boito. Estimation of the impact of I/O forwarding on application performance. [Research Report] RR-9366, Inria. 2020.

[13] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross. Understanding and Improving Computational Science Storage Access through Continuous Characterization. *ACM Transactions on Storage*, 7(3):8:1–8:26, Oct. 2011.

[14] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley. 24/7 Characterization of petascale I/O workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10, New Orleans, LA, USA, 2009. IEEE.

[15] F. Chowdhury, Y. Zhu, T. Heer, S. Paredes, A. Moody, R. Goldstone, K. Mohror, and W. Yu. I/o characterization and performance evaluation of beegfs for deep learning. In *Proceedings of the 48th International Conference on Parallel Processing*, ICPP '19, New York, NY, USA, 2019. Association for Computing Machinery.

[16] E. Costa, T. Patel, B. Schwaller, J. M. Brandt, and D. Tiwari. Systematically inferring i/o performance variability by examining repetitive job behavior. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[17] H. Devarajan and K. Mohror. Extracting and characterizing i/o behavior of hpc workloads. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 243–255, 2022.

[18] M. Dorier, G. Antoniu, R. Ross, D. Kimpe, and S. Ibrahim. Calciom: Mitigating i/o interference in hpc systems through cross-application coordination. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 155–164, 2014.

[19] M. Dorier, S. Ibrahim, G. Antoniu, and R. Ross. Using Formal Grammars to Predict I/O Behaviors in HPC: The Omnisc'IO Approach. *IEEE Transactions on Parallel and Distributed Systems*, 27(8):2435–2449, Aug. 2016.

[20] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir. Scheduling the I/O of HPC Applications Under Congestion. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 1013–1022, May 2015. ISSN: 1530-2075.

[21] A. Gainaru, B. Goglin, V. Honoré, and G. Pallez. Profiles of upcoming hpc applications and their impact on reservation strategies. *IEEE Transactions on Parallel and Distributed Systems*, 32(5):1178–1190, 2021.

[22] X. Ji, B. Yang, T. Zhang, X. Ma, X. Zhu, X. Wang, N. El-Sayed, J. Zhai, W. Liu, and W. Xue. Automatic, application-aware i/o forwarding resource allocation. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies*, FAST'19, pages 265–279. USENIX Association, 2019.

[23] S. Kim, A. Sim, K. Wu, S. Byna, T. Wang, Y. Son, and H. Eom. Dca-io: A dynamic i/o control scheme for parallel and distributed file systems. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 351–360, 2019.

[24] Y. Liu, R. Gunasekaran, X. Ma, and S. S. Vazhkudai. Server-Side Log Data Analytics for I/O Workload Characterization and Coordination on Large Shared Storage Systems. In *SC'16*, pages 819–829, Nov 2016.

[25] Z. Liu, R. Lewis, R. Kettimuthu, K. Harms, P. Carns, N. Rao, I. Foster, and M. E. Papka. Characterization and identification of HPC applications at leadership computing facility. In *Proceedings of the 34th ACM International Conference on Supercomputing*, pages 1–12, Barcelona Spain, June 2020. ACM.

[26] A. Lopez, S. Valat, S. Narasimhamurthy, and M. Golasowski. Ephemeral data access environment: Concept and architecture. Technical report, IO-SEA project public deliverable, 2022.

[27] S. Oral, J. Simmons, J. Hill, D. Leverman, F. Wang, M. Ezell, R. Miller, D. Fuller, R. Gunasekaran, Y. Kim, S. Gupta, D. T. S. S. Vazhkudai, J. H. Rogers, D. Dillow, G. M. Shipman, and A. S. Bland. Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 217–228, 2014.

[28] V. Vishwan, M. Hereld, K. Iskra, D. Kimpe, V. Morozov, M. E. Papka, R. Ross, and K. Yoshii. Accelerating I/O Forwarding in IBM Blue Gene/P Systems. In *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–10, New Orleans, LA, USA, Nov. 2010. IEEE.

[29] F. Wang, S. Oral, S. Gupta, D. Tiwari, and S. S. Vazhkudai. Improving large-scale storage system performance via topology-aware and balanced data placement. In *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 656–663, 2014.

[30] F. Wang, H. Sim, C. Harr, and S. Oral. Diving into petascale production file systems through large scale profiling and analysis. In *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, PDSW-DISCS '17, page 37–42, New York, NY, USA, 2017. Association for Computing Machinery.

[31] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, and D. Long. File system workload analysis for large scale scientific computing applications. In *Proceedings of the Twenty-first Symposium on Mass Storage Systems (MSST)*, 2004.

[32] T. Wang, S. Byna, G. K. Lockwood, S. Snyder, P. Carns, S. Kim, and N. J. Wright. A zoom-in analysis of i/o logs to detect root causes of i/o performance bottlenecks. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 102–111, 2019.

[33] C. Xu, S. Byna, V. Venkatesan, R. Sisneros, O. Kulkarni, M. Chaarawi, and K. Chadalavada. Lioprof: exposing lustre file system behavior for i/o middleware. In *2016 Cray User Group Meeting*, 2016.

[34] B. Yang, X. Ji, X. Ma, X. Wang, T. Zhang, X. Zhu, N. El-Sayed, H. Lan, Y. Yang, J. Zhai, W. Liu, and W. Xue. End-to-end I/O monitoring on a leading supercomputer. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 379–394, Boston, MA, Feb. 2019. USENIX Association.

[35] B. Yang, Y. Zou, W. Liu, and W. Xue. An end-to-end and adaptive i/o optimization tool for modern hpc storage systems. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1294–1304, 2022.

[36] W. Yang, X. Liao, D. Dong, and J. Yu. A quantitative study of the spatiotemporal i/o burstiness of hpc application. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1349–1359, 2022.

[37] O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, and G. Antoniu. On the root causes of cross-application i/o interference in hpc storage systems. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 750–759, 2016.

[38] J. Yu, G. Liu, W. Dong, X. Li, J. Zhang, and F. Sun. On the load imbalance problem of I/O forwarding layer in HPC systems. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 2424–2428, Dec. 2017.

A Proof of $n_{\text{sys}} \leq n_{\text{perf}}$

Proof of Lemma 1. By definition of n_{perf} and n_{sys} we have the following:

$$\begin{aligned} \text{T\_cf}_{\text{io}}(n_{\text{perf}}) &\leq \text{T\_cf}_{\text{io}}(n_{\text{sys}}) && (n_{\text{perf}}) \\ \frac{n_{\text{perf}}\text{T\_cf}_{\text{io}}(n_{\text{perf}})}{T_{\text{cpu}} + \text{T\_cf}_{\text{io}}(n_{\text{perf}})} &\geq \frac{n_{\text{sys}}\text{T\_cf}_{\text{io}}(n_{\text{sys}})}{T_{\text{cpu}} + \text{T\_cf}_{\text{io}}(n_{\text{sys}})} && (n_{\text{sys}}) \end{aligned}$$

For c constant, the function $x \mapsto \frac{x}{c+x}$ is increasing (its derivative $x \mapsto \frac{c}{(c+x)^2}$ is positive). Hence

$$\text{T\_cf}_{\text{io}}(n_{\text{perf}}) \leq \text{T\_cf}_{\text{io}}(n_{\text{sys}}) \implies \frac{\text{T\_cf}_{\text{io}}(n_{\text{perf}})}{T_{\text{cpu}} + \text{T\_cf}_{\text{io}}(n_{\text{perf}})} \leq \frac{n_{\text{sys}}\text{T\_cf}_{\text{io}}(n_{\text{sys}})}{T_{\text{cpu}} + \text{T\_cf}_{\text{io}}(n_{\text{sys}})}$$

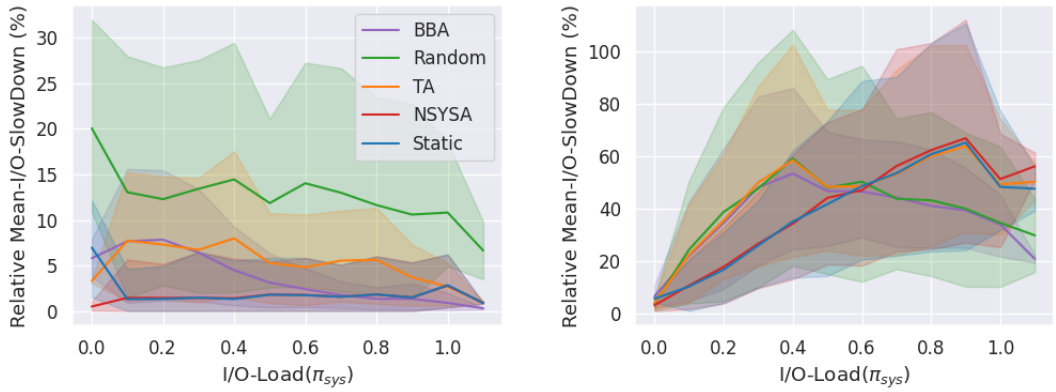
Hence,

$$\frac{n_{\text{perf}}\text{T\_cf}_{\text{io}}(n_{\text{perf}})}{T_{\text{cpu}} + \text{T\_cf}_{\text{io}}(n_{\text{perf}})} \geq \frac{n_{\text{sys}}\text{T\_cf}_{\text{io}}(n_{\text{sys}})}{T_{\text{cpu}} + \text{T\_cf}_{\text{io}}(n_{\text{sys}})} \implies n_{\text{perf}} \geq n_{\text{sys}}$$

showing the result. □

B All results for the OST case

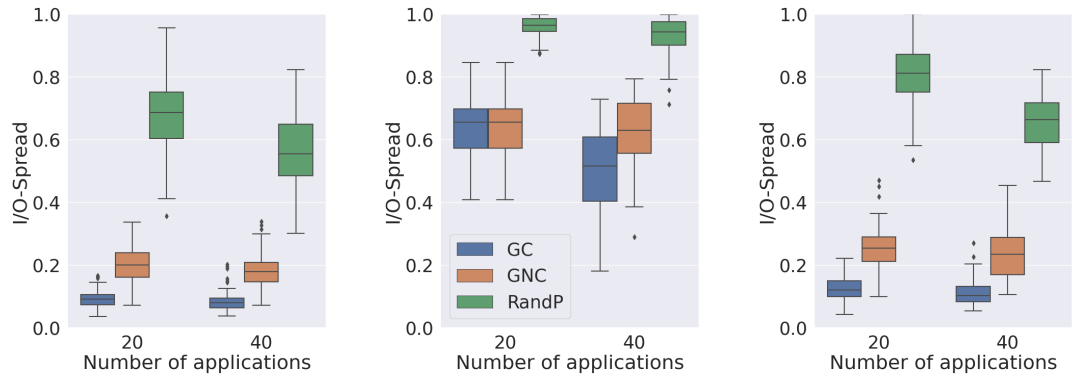
In this Section, we present the results obtained for the OST case, which correspond to the same experiments performed for the case of scheduling I/O nodes.



(a) GNC / GC

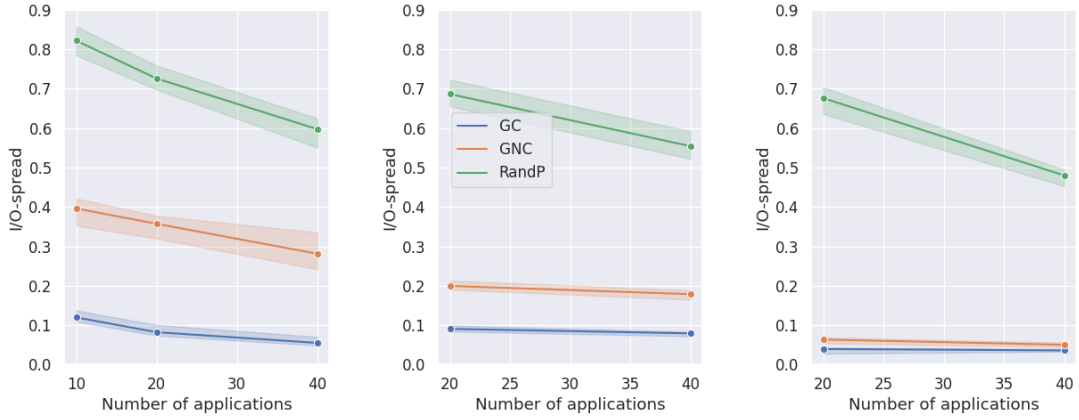
(b) RandP / GC

Figure 14: Relative `Mean-I/O-SlowDown` for different algorithm combinations when `I/O-load`(π_{sys}) increases. Placement are compared with GC. Lines show the mean value, and the area around them is the percentile interval (10^{th}-90^{th}). This is for the OST case, and corresponds to Figure 5 from the case of I/O nodes.



(a) BBA (b) NSYSA (c) Random

Figure 15: `I/O-spread` for different algorithms when `I/O-load`(π_{sys}) $= 0.5 \pm 0.05$. This is for the OST case, and corresponds to Figure 6 from the case of I/O nodes.



(a) `I/O-load`(π_{sys}) $= 0.2 \pm 0.05$ (b) `I/O-load`(π_{sys}) $= 0.5 \pm 0.05$ (c) `I/O-load`(π_{sys}) $= 0.8 \pm 0.05$

Figure 16: `I/O-spread` for different placement algorithms using BBA. Lines connect the mean values, and the area around each line shows the 95% confidence interval. This is for the OST case, and corresponds to Figure 7 from the case of I/O nodes.

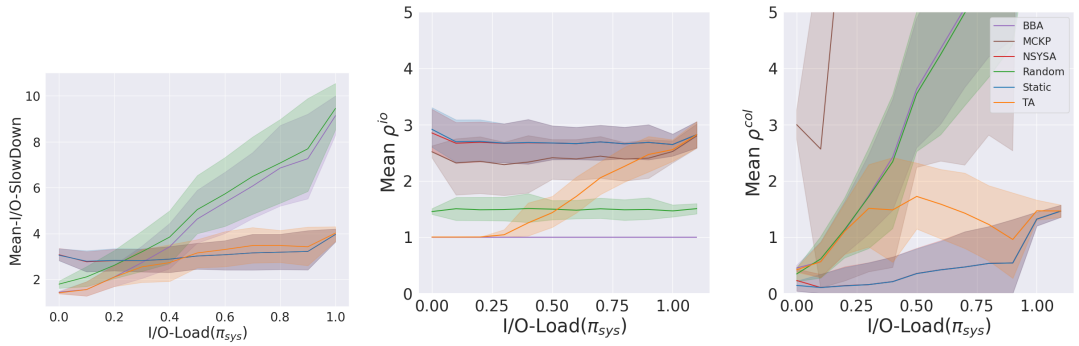
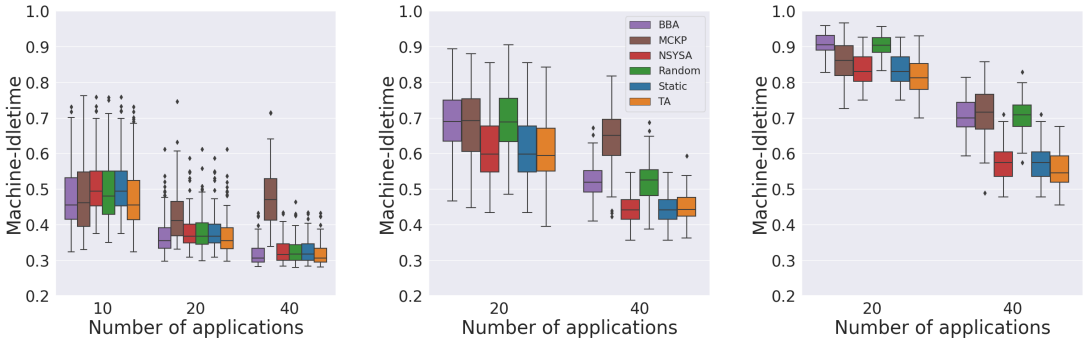
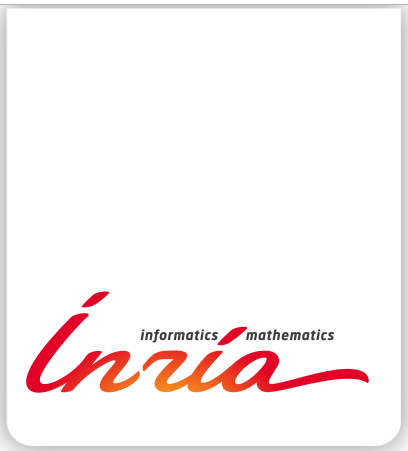


Figure 17: `Mean-I/O-SlowDown` (left) separated into its two main components: I/O resources allocation ρ^{io} and congestion ρ^{con}. The lines show the mean value, and the area around them is the percentile interval (10^{th}-90^{th}). This is for the OST case, and corresponds to Figure 8 from the case of I/O nodes.



(a) `I/O-load`(π_{sys}) $= 0.2 \pm 0.05$ (b) `I/O-load`(π_{sys}) $= 0.5 \pm 0.05$ (c) `I/O-load`(π_{sys}) $= 0.8 \pm 0.05$

Figure 18: `Machine-Idletime` for allocation algorithms at increasing `I/O-load`(π_{sys}). The y axes do *not* start at 0. The lower the better. This is for the OST case, and corresponds to Figure 9 from the case of I/O nodes.



RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399