Supporting Dynamic Allocation of Heterogeneous Storage Resources on HPC Systems

Julien Monniot, François Tessier, Matthieu Robert, Gabriel Antoniu

To cite this version:

Julien Monniot, François Tessier, Matthieu Robert, Gabriel Antoniu. Supporting Dynamic Allocation of Heterogeneous Storage Resources on HPC Systems. Concurrency and Computation: Practice and Experience, 2023, Special Issue:S2 World 2020. CUG 2021 & 2022. PN_HCP. HeteroPar 2022, 35 (28), pp.1-16. 10.1002/cpe.7890 . hal-03922866v2

HAL Id: hal-03922866
https://hal.science/hal-03922866v2

Submitted on 13 Nov 2023

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

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

Distributed under a Creative Commons Attribution 4.0 International License

Supporting Dynamic Allocation of Heterogeneous Storage Resources on HPC Systems

Julien Monniot, François Tessier, Matthieu Robert, and Gabriel Antoniu

Univ Rennes, Inria, CNRS, IRISA Rennes, France
Corresponding author: julien.monniot@inria.fr

Abstract. Scaling up large-scale scientific applications on supercomputing facilities is largely dependent on the ability to scale up efficiently data storage and retrieval. However, there is an ever-widening gap between I/O and computing performance. To address this gap, an increasingly popular approach consists in introducing new intermediate storage tiers (node-local storage, burst-buffers, ...) between the compute nodes and the traditional global shared parallel file-system. Unfortunately, without advanced techniques to allocate and size these resources, they remain underutilized. In this paper, we investigate how heterogeneous storage resources can be allocated on an HPC platform, just like compute resources. To this purpose, we introduce StorAlloc, a simulator used as a testbed for assessing storage-aware job scheduling algorithms and evaluating various storage infrastructures. We illustrate its usefullness by showing through a large series of experiments how this tool can be used to size a burst-buffer partition on a top-tier supercomputer by using the job history of a production year.

Keywords: Intermediate storage resources · Storage disaggregation · Simulation · Job scheduling.

1 Introduction

The execution of highly complex scientific applications usually leverages large-scale compute-intensive infrastructures (typically, supercomputers). Traditionally, such High-Performance Computing (HPC) systems have been designed with the main objective of improving computing power. However, scientific applications have recently evolved from compute-intensive codes towards complex data-centric workflows bridging the domains of modeling, simulation, data analytics and AI. The *data deluge* engendered by these workloads has been observed in major supercomputing centers: the National Energy Research Scientific Computing Center (NERSC), USA, noticed that the volume of data stored by applications has been multiplied by 41 over the past ten years while the annual growth rate is estimated to 30% [29]. While absorbing these data requires extended I/O performance and advanced technologies, storage systems on HPC platforms have been mostly based on an old paradigm: a centralized parallel file-system shared by all the computing resources. This storage architecture, not very adaptable from a

user perspective, suffers a relative performance decrease: a study of the top three supercomputers from the Top500 ranking between 2009 and 2022, as depicted in Figure 1, shows that the ratio of I/O bandwidth of the parallel file-system (PFS) to computing power has been divided by 25 during that period.

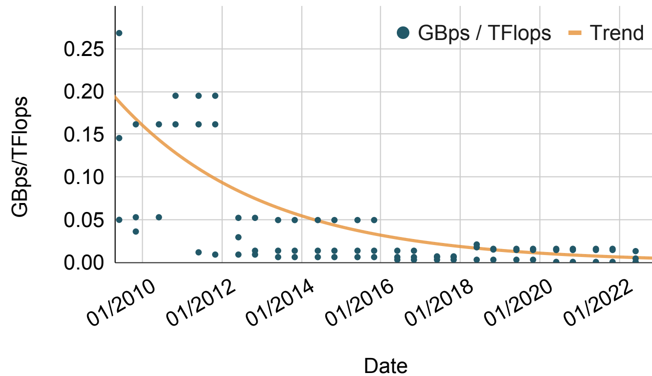

Fig. 1. Ratio of PFS I/O bandwidth (GBps) to computing power (TFlops) of the top 3 supercomputers of the Top500 from 2009 to 2022

To mitigate this gap, recently proposed approaches rely on new tiers of intermediate storage, such as node-local disks or burst buffers [19], backed by diverse technologies (Flash memory, NVDIMM, NVMeoF, CXL storage ...), and placed between the compute nodes and the global shared parallel file-system. This storage disaggregation offers new alternatives to a centralized storage system. However, to fully exploit the benefits of such approaches, advanced techniques for sizing and allocating these resources have yet to be devised.

Some of the limitations which make it difficult to investigate new methods for allocating such storage resources are the difficult access to hardware (with enough privileges) or the limited range of technologies deployed on the studied system. Simulation is one means to overcome these constraints. At the cost of a loss of accuracy, ideally as moderate as possible [13], simulation offers much better flexibility for representing a wide variety of storage architectures and can be used to estimate the benefits of storage infrastructures before their actual deployment.

This paper proposes to explore how storage resources can be allocated on HPC systems, i.e. to investigate which method (scheduling algorithm) and with which efficiency (metric) a set of I/O intensive jobs can be scheduled on a pool of heterogeneous storage resources. To this purpose, we introduce StorAlloc, a Discrete-Event Simulator (DES) of a batch scheduler able to play (or replay) the scheduling of I/O intensive jobs on intermediate storage resources.

A preliminary description of our approach has been made in a previous workshop paper [31]. This extended paper aims to provide a more refined view of our contribution. New experimental results have been added while previous results have been reinforced through additional experiments. Supplementary discussions of related work and extended explanations of our approach have also been added.

The remainder of this paper is organized as follows. Section 2 details the main motivational facts underlying our work. We then discuss the gaps that our contribution aims to fill in related work (Section 3). The architecture of StorAlloc is described in Section 4. Then we evaluate the tool on a set of basic scheduling algorithms and on multiple models of infrastructures featuring heterogeneous storage resources (Section 5). From our simulations, we can conclude on the right sizing of intermediate storage resources among a set of architectures and analyze the utilization rate of the underlying disks. Based on this use-case and thanks to the data that StorAlloc collects during the simulations, we provide an in-depth study of the scheduling algorithms and the storage layouts. Finally, Section 6 summarizes our contribution and discusses open research directions.

2 Context and Motivation

For many years, supercomputers have followed a hyper-centralized paradigm regarding storage: a unique global shared parallel file-system such as Lustre [3], BeeGFS [1] or Spectrum Scale (formerly GPFS [35]), used as a staging area from which data is read or written by applications or workflow components. These file-systems, although increasingly powerful, suffer the drawbacks of any highly centralized system: contention and interference make them very prone to performance variability [22]. In order to overcome this problem, we have seen the emergence of new storage systems, closer to the computing nodes. Node-local SSDs, burst buffers or dedicated storage nodes with network-attached storage technology (NVMeoF, CXL), to name a few, are all technologies that provide fast storage, albeit with limited capacity, various data lifetime, cost and performance, and different means of access.

This last point in particular makes the use of these resources complicated. To illustrate this, Table 1 presents the multiple ways of accessing resources for a subset of storage tiers that tend to become popular on large-scale systems. The usual scope of the storage space and the commonly deployed data manager, if any, are also listed.

This variety, which would require working on new levels of abstraction, also raises another problem: how to preempt all or part of these storage resources so as to make them available for the duration of an I/O-intensive job's execution, as we do for compute nodes? Allocation methods exist for storage tiers but they are numerous and not interoperable: storage allocated at the same time as the compute node, dedicated APIs integrated or not into the job scheduler, complex low-level configurations. Thus, while it is common on HPC systems to get access exclusively to compute nodes (usually through a job scheduler), the allocation of those intermediate levels of storage remains minor in practice and

	Access	Scope	Data manager
Global storage system	Mount point	System-wide	Parallel file-system
Node-local disk	Mount point	Node	File-system
NVDIMM - FSDAX	Mount point	Node	DAX-enabled file-system
NVDIMM - DEVDAX	Direct access	Node	Raw persistent memory
CXL storage	Direct access	Configurable	Raw storage/memory space
Burst buffer	Middleware	Job	(Parallel) file-system
Network-attached storage	API	Node(s)	Raw storage space

Table 1. Type of access, scope and default data management system on a subset of storage resources that tend to be democratized on large-scale systems.

often limited to homogeneous resources. In order to use these new storage tiers to their full potential, new allocation techniques must be devised and deployed on supercomputers.

The development of such solutions would, however, require access to intermediate storage resources with enough rights to repurpose them, which is usually not possible on deployed infrastructures for various reasons such as security or maintenance efforts. In addition, such experimentation can easily disrupt other users' workloads on production systems. An alternative approach is to use simulations as a way to reproduce with a certain degree of accuracy the behavior of a system with a very low footprint. While experiments on real systems would be limited to the embedded technologies, a simulator can also evaluate new types of architectures combining existing and emerging storage tiers, for example to make decisions about their sizing or their design. Using traces from actual HPC workloads, a number of uses cases can thus be modelled, ranging from realistic scenarios where replaying the allocations can help better understand past and current resource usage, to more theoretical ones where improbable platform configurations can be studied as well. Several simulators already exist for scheduling jobs on compute nodes or for optimizing I/O, yet very few has been done to model and allocate storage resources. Therefore, in this paper, we propose StorAlloc, a simulator of a storage-aware job scheduler whose main objective is to explore heterogeneous storage resource allocation on supercomputers.

3 Related work

3.1 Simulation frameworks

To the best of our knowledge, there is no tool whose goal is to simulate the scheduling of jobs on heterogeneous storage resources of a supercomputer. Simulators allowing to play or replay the execution of parallel and distributed applications on HPC systems exist and have been studied for many years. However, most frameworks essentially address the computational aspect. SimGrid [8], for example, is a powerful framework for simulating the scheduling and execution

of a large number of applications on real or made-up infrastructure models. The I/O aspect is limited to simulating data movement but storage resource allocation is absent from the framework, although preliminary work was started a few years ago [25]. Similarly, CODES [12], while initially presented as a simulation framework for storage systems, has eventually specialized in network topology simulation offering various built-in models for high-radix networks but showing the same limitations as SimGrid in terms of storage model.

Like StorAlloc, a few SimGrid-derived simulators have job scheduling oriented approaches. This is the case of batsim [15] or Wrench [9] for example. However, the support of heterogeneous storage levels as allocatable resources is not implemented: batsim does not include a model for disk capacity while Wrench only manipulates files located on an abstract storage location without considering the underlying hardware. Another recent work [23] built upon CODES evaluates burst buffers placement through the study of network topologies, I/O workload patterns and job scheduling techniques. Again the focus is mostly on data movement, while underlying resources and dynamic allocation are not in scope. The above-mentioned solutions also differ from StorAlloc in their monolithic design. As described in Section 4, StorAlloc has all its components decoupled. Therefore, the servers can be distributed on multiple nodes while the simulator component can be disabled to turn StorAlloc into a real storage-aware job scheduler.

3.2 Scheduling models and studies of intermediate resources

A large number of solutions for optimizing HPC storage systems performance consider the issue from the point of view of I/O requests. That is, they offer algorithms to orchestrate individual I/Os or I/O phases of concurrently executing applications. They can be specifically directed at the application level [17, 37, 27, 28], the I/O forwarding level [5, 36], or can be integrated into middlewares in charge of offloading such I/O requests [21]. Another common approach consists of considering the availability of I/O resources as a constraint when placing the scheduled tasks onto the compute nodes, without subsequently interfering with the way the applications do I/Os [20, 6]. In these approaches, the storage system is often considered as statically configured or even represented as a black box, and applications and their I/Os are the parameters to be acted upon. Our work takes an opposite stance. For each job, previously scheduled based on its compute requirements, we want to allocate an optimal set of storage nodes and disks that will fit the job's storage and I/O needs without needing to change the way the application does I/Os.

Allocating storage resources involves partitioning and sizing these resources to meet the needs of submitted jobs. In that field, models such as burst buffers have been studied [4, 34, 18, 24]. This research is complementary to StorAlloc. The proposed techniques are the basis of storage-aware job scheduling algorithms that could be evaluated in our simulator. For instance, the Harmonia dynamic I/O scheduler [24] considers I/O interference in the allocation process. It targets

burst-buffer systems and offers multiple scheduling policies (such as maximizing buffer efficiency and improving application bandwidth) which achieve better performances than vendor software for these resources.

More generally, a better understanding and management of storage resources in HPC is decisive to improve data management at extreme scale. This has been highlighted in an analysis of the NERSC systems [30]. The authors demonstrate, among other findings, that the parallel file system on this platform receives at least 8 times more reads and 16 times more writes than the burst buffers. On the same system, the authors also point out that some specific I/O forwarding nodes remain vastly unutilized by applications. More recently, a study [14] also shows how much work is still needed to better use intermediate storage tiers, both in terms of storage API and workflow configuration. This work hints that richer and more complex software between hardware resources and scientific applications is necessary to correctly address the needs of the workflows. Our work ultimately aims to address this kind of issues by streamlining the way storage is allocated to the users.

Finally, it is necessary to note that our approach can be similar to what has been done in the world of Cloud Computing for several years although these fully virtualized systems with very limited low-level access are different from HPC systems. On Clouds, the allocation of storage tiers as well as compute or network resources is more familiar. Work comparable to StorAlloc has been done to simulate the allocation of resources between different users [33, 7] in virtualized environment but these works are outdated and have very limited storage support. A variety of works have also addressed the issue of storage scheduling, but they tend to focus on requirements that are not necessarily in phase with HPC needs, such as SLAs, end-user costs or Cloud-specific infrastructures [26, 10, 32]. Some of the methods employed in these works could however prove useful for the next iterations of our work.

4 Architecture

StorAlloc is a tool able to simulate the scheduling of I/O-intensive jobs on heterogeneous storage resources available on a HPC system. In this section, we present its design and discuss implementation choices.

4.1 General design

The objective of StorAlloc is to provide a simple way to develop and evaluate storage-aware job scheduling algorithms targeting heterogeneous storage resources (any kind of disk-based storage can be described). Therefore, StorAlloc has been designed following the basic principles of a job scheduler, *i.e.* a middleware allowing clients to request resources available on a supercomputer. Extending from this architecture, we then added the ability to run it as a simulator, using a single code base.

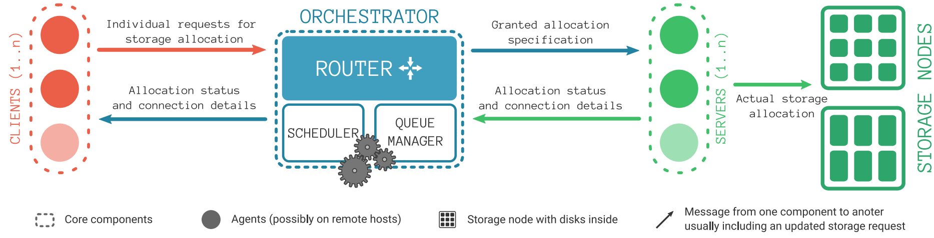

Fig. 2. StorAlloc core scheduling components

Core components StorAlloc's design is based on the composability of several components, which can be run together and extended in order to provide the desired behavior. Figure 2 depicts the core components: one or multiple *server* and *client* agents are communicating through a central *orchestrator*. The clients request storage allocations to the orchestrator and expect connection settings to the newly allocated storage space in return. The server components declare a pool of available resources under their responsibility to the orchestrator and perform the storage management operations when needed (partitioning, rights granting, exposure on the network, releasing). In between, the orchestrator handles routing messages between components, keeps track of running and pending allocations and hosts the scheduler process.

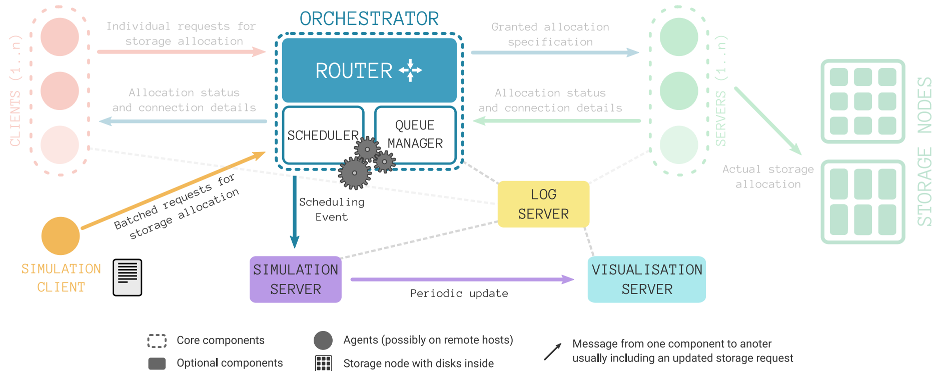

Fig. 3. StorAlloc simulation components

Additional components In addition to these core components, we have extended StorAlloc with two simulation units (client and server), a visualisation server for real-time plotting during simulation and an external log aggregator. They are presented on Figure 3. The simulation client offers an efficient way to serve requests to the orchestrator from a dataset file. Its counterpart, the

simulation server, accumulates scheduling decisions in order to build a Discrete Event Simulation, and outputs results to file. The visualisation server provides a convenient way to setup a live display of major metrics while the log server aggregates logs from all components in a single place, even when they are all distributed on several nodes.

The architecture of the tool makes it possible to add additional elements if necessary. All of these components are interconnected using a message-based protocol we have defined. They can be deployed across a set of hosts, or run on a single machine. While the former case is intended to properly map clients and servers onto an actual HPC platform, the latter is sufficient for simulations. The current design only allows for one orchestrator component to be running at any time. This constraint creates a single point of failure when deployed as a middleware in a production setting, and will be addressed in further developments.

In the following sections, we detail design choices for StorAlloc. In particular, we explain the general functioning of the scheduler, a central component in our simulator. Then we describe the storage abstraction layer used to characterize the pool of resources. In 4.4, we present the simulation capability with a focus on the real-time collection of scheduling data. We end this section with some technical considerations about StorAlloc.

4.2 Scheduling storage requests

We define a storage request as a triple consisting of a capacity in GB, an allocation time in minutes and a submission time in a *datetime* format. The scheduling of storage requests takes place in a *scheduler* sub-component of the *orchestrator*, as depicted in Figure 2. This sub-component receives requests through messages from clients and process them asynchronously in the receiving order. The scheduler has access to both the entire list of available storage resources and the list of currently allocated requests. Any algorithm can thus make a resource allocation decision backed by a full view of the platform state. So far, four algorithms have been implemented in StorAlloc as listed below:

- *random*: storage resources are picked randomly with a chance of failure;
- *round-robin*: storage space is allocated in a round-robin manner;
- *worst-fit*: disks are filled until no more space is available;
- *best-bandwidth*: nodes and disks on nodes are selected according to the best remaining bandwidth, considering a permanent maximum I/O regime for the existing allocations.

At launch time, the scheduler chooses one of these algorithms through a user-defined parameter. The scheduling algorithms share a common interface which accepts a storage request and a list of available storage resources, and returns an identifier for the resource(s) on which the desired storage space will be allocated. A request can also be refused (no space left for instance). In this case, we assume that the job falls back to a traditional parallel file-system, instead of using the intermediate storage tiers available through StorAlloc.

Only two out of four algorithms have been implemented with the capacity to refuse requests (*round-robin* and *best-bandwidth*). For each new request, they lookup on-going and planned allocations and try to determine if the new allocation has a chance to fit on any of the available resources. They do so in a deliberately conservative manner: deciding on future resource availability has to be fast and doesn't return an exact result. This is driving us to implement policies with a small bias towards refusing requests, instead of risking to allow too many requests that have good chances to fail once forwarded to the storage servers. The other two algorithms, *random* and *worst-fit*, always accept requests, but do not provide any guarantee on the availability of the selected resources. This implementation choice entails a pronounced difference in the behaviours of our algorithms, which is reflected in the analysis and help us check the coherency of our results.

At a higher level, the scheduling of storage requests can also be adjusted by leveraging two strategies presented in Table 2. They are meant to help allocate requests when resources are constrained. The first, called "Split", is to divide an allocation into several sub-allocations of equal size in order to distribute them over multiple nodes and disks. The second strategy, called "Requeued", consists of defining criteria for resubmitting unsuccessful allocations. This strategy only applies to algorithms able to proactively refuse requests. Both strategies have default values but are configurable in StorAlloc. The impact of these strategies, independent of the scheduling algorithms, is evaluated in Section 5. Again, we make the assumption that in case of allocation failure, I/O will be performed on the global shared parallel file system.

	Default setting	**Comment**
Split	Threshold at 200GB	Split requests whose capacity > threshold and allocate the parts on multiples resources.
Requeued	5 retries, one every 5m	Postpone starting time and retry a refused allocation.

Table 2. Optional scheduling strategies

Discussion on algorithms choice Let us note that the four algorithms used throughout this work, while being relatively simple, are still rather comparable to the level of complexity found in current production systems. For instance, in Lustre [3], objects may be internally load-balanced between OSTs using *round-robin*, and the policy switches to a basic weighted allocator only if inbalance between OSTs exceeds 20%. Similarly, in BeeGFS [1], files can be stripped in a RAID0 fashion (same mechanism as round-robin) between targets inside a storage pool. In both cases, a user must manually provide a chunk size and decide which paths in the filesystem should use stripping. In OpenStack, the Cinder scheduler [2], selects storage hosts in a fairly simple manner: the first step is to filter the list of hosts based on a capacity or placement criteria for instance. Then

in a second step, remaining storage hosts are weighted according to their already allocated capacity, system health, or simply by volume number. Eventually the storage host with the best weight is chosen. This process is similar to the method used in *best-bandwidth*, using the criteria of storage access bandwidth as weight. The main purpose of our current work being to evaluate the usefulness of a simulator such as StorAlloc, we chose to limit ourselves to the four algorithms listed above, and the evaluation of more complex or innovative scheduling algorithms (eg. multi-objective optimizations based on genetic algorithms) will be left to future works.

4.3 Storage abstraction

Because the available storage tiers can be extremely heterogeneous, an abstraction layer is needed to allow scheduling algorithms to accommodate the variety of technologies without needing to know the technical details of each level. In StorAlloc, storage platforms are represented through a hierarchy of three objects: *servers*, *nodes* and *disks*. Servers are top-level StorAlloc components which act as an interface between the orchestrator and one or many storage nodes. Nodes embed at least one disk. Nodes and disks may be of heterogeneous nature (number of disks, disk capacity, read and write bandwidth, node's network bandwidth). Disk or node characteristics don't have a fixed format. Anyone can define new fields, and it's up to the selected scheduling algorithm to make use of the information in order to achieve its own optimization goal. Whenever required by a parent server, a node should be able to setup and expose a specific partition of its storage resources, whose ownership will be transferred to a client. In simulation mode, servers passively accept requests without taking any action, but we still ensure that any allocation would be *legal* in terms of available resources.

It has to be noted that when defining a storage layout, we consider the network to be flat i.e., with all the nodes connected to the same switch and at equal distance from each other. This is motivated by the fact that dynamic routing policies are unpredictable, either because the vendor does not provide enough details (such as on the Cray XC40 Theta platform which provided the input data used in Section 5 [11]) or because there are too many factors involved in packet routing decisions to be accurately modeled. Hence we only define the bandwidth at the node and disk levels and let the scheduling algorithm model the impact of concurrent allocations on these resources.

4.4 Simulation

A longer-term goal of StorAlloc is to provide a single code base for a storage-aware job scheduler and its simulator. Therefore, we have designed our simulation server with a "component in the middle" approach. The core components run as if they were actually deployed on a real system except that, if the simulation mode is enabled, the requests are rerouted to the simulation server which stacks them until a specific message triggers the actual execution of the simulation.This mode of operation allows us to seamlessly switch from the theoretical

case to real allocations if StorAlloc needs to be tested on an actual platform. Then, the simulation is unrolled and go through the scheduler, using a discrete event simulation (DES) model [16]. During that phase, we replay the allocation decisions that were made by the scheduler onto the simulated storage. StorAlloc traces any possible allocation failures, and collect data measuring the impact of scheduling to feed a visualization server in real-time and output a result file. We have defined a number of indicators that allow us to study the behavior of the storage infrastructure. These indicators are defined below:

- *sum_ cap* is the total capacity that should be allocated and deallocated during a simulation:
$$sum_cap = \sum_{j \in J} V_j$$
- *mean_ disk_ use* is the mean disk usage (in % of C_{d_i}):
$$mean_disk_use = \frac{1}{T} \sum_{t=0}^{T-1} \frac{U_{d_i}(t) * 100}{C_{d_i}},\ \forall d_i \in D$$
- *max_ disk_ use* is the maximum disk usage:
$$max_disk_use = \max\{\, U_{d_i}(t)\ : t = 1..T\},\ \forall d_i \in D$$
- *mean_ alloc* is the mean number of concurrent allocations:
$$mean_alloc = \frac{1}{T} \sum_{t=0}^{T-1} A_{d_i}(t),\ \forall d_i \in D$$
- *max_ alloc* is the maximum number of concurrent allocations:
$$max_alloc = \max\{\, A_{d_i}(t)\ : t = 1..T\},\ \forall d_i \in D$$

Where:

T: The simulation duration, in seconds
J: The set of jobs scheduled on the platform
V_j: The storage space that needs to be allocated for a job j, with $j \in J$
D: The set of disks available on the storage infrastructure
d_i: The ith disk of the platform, with $i \in D$
C_{d_i}: Total capacity of disk i
$U_{d_i}(t)$: Used space on disk i at time t
$A_{d_i}(t)$: Number of allocations on disk i at time t

At the request level, although we only record their cardinality, we also define the following sets:

R_{Delay}: The set of delayed requests during the simulation (and D_r the total delay time)
$R_{Requeued}$: The set of requeued requests during the simulation
R_{Split}: The set of splitted requests during the simulation
$R_{Allocated}$: The set of allocated requests during the simulation
$R_{Refused}$: The set of refused requests during the simulation
R_{Failed}: The set of failed requests during the simulation

4.5 Implementation details

The simulator presented in this paper is implemented using Python3. Our messaging protocol relies on ZeroMQ, while the DES model used for the simulation comes from the SimPy library [1]. These choices allow us to limit the size of the source code by relying on robust on-shelf implementations. Our simulation is nevertheless constrained by the use of DES instead of more complex and accurate model: the potential impact of allocations choices, for instance on the runtime of a job, cannot be taken into account due to the lack of any form of feedback loop. However we believe this limitation, which will be addressed in future works, is acceptable when experimenting with static traces and focusing on the storage system usage. The source code of StorAlloc can be found at https://github.com/hephtaicie/storalloc.

5 Evaluation

In this section, we evaluate the benefits of our simulator to assess storage-aware job scheduling algorithms on heterogeneous resources. To do so, we run multiple configurations and show their impact on the storage tiers.

5.1 Simulation setup

To simulate storage requests from clients representative of real applications, we used a dataset composed of one year (2020) of Darshan [2] logs on Theta, a 11.7 PFlops Cray XC40 supercomputer at Argonne National Laboratory [3]. We extracted from these traces jobs spending at least 10% of their run time doing IO, and reading or writing at least 10 GB of data. It resulted in about 24 000 jobs out of approximately 624 000 jobs, each one translating into a storage request in StorAlloc: the requested capacity is conservatively based on the maximum of either read or write volume while the allocation time uses the initial job duration. Using Darshan traces as input data allows us to work with multiple HPC platforms which offer datasets of traces generated with this broadly available library. As the traces are pre-processed, other formats could also be adapted to be used with StorAlloc, thus making the simulator accessible for a large array of use case on various platforms.

Request ordering. We submitted these requests to the simulation process with the same ordering as in the original Darshan file, which is the order in which the traced jobs were initially scheduled on their host system. Consequently, the overall scheduling process could be seen as a two-step operation: a first scheduling

[1] Resp. https://zeromq.org/ and https://simpy.readthedocs.io/en/latest/

[2] Darshan I/O monitoring tool. https://www.mcs.anl.gov/research/projects/darshan/

[3] 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.

happened independently from StorAlloc, based on compute criteria only, and StorAlloc is in charge of a second operation, where we allocate storage resources based on I/O criteria. One of our future plans is to extend our simulator and use it to help design algorithms that rely on both criteria to make scheduling decisions.

In order to have a good overview of what can be observed with our simulator, we have run 256 different simulation setups based on the settings presented in Table 3. The average simulation time is around 5m9s per run, in a range of $[4m50s; 6m51s]$ on a single core of a Intel Xeon E5-2630 processor. This variability is due to the difference in complexity of the algorithms and the activation or not of the requeuing and splitting strategies. It has to be noted that total capacity, individual disk size and layout parameters have been chosen based on the study of our dataset, and allow us to simulate platforms which bracket a wide range of practical scenarios, from under-provisioned to over-provisioned storage resources.

Settings	**Tested values**	**Comment**
Algorithm	Random, round-robin worst-fit, best-bandwidth	See Section 4.2
Total capacity	8TB, 16TB, 32TB, 64TB	Disk sizes are 1TB, 2TB, 4TB and 8TB respectively.
Storage Layout	Single node, single disk (*1N1D*) Single node, multi disks (*1NnD*) Multi nodes, single disk (*nN1D*) Multi nodes, multi disks (*nNnD*)	*1N1D* serves as baseline.
Requeued	Enabled or disabled	When enabled, new attempts every 5m, until a 60m delay.
Split	200 GB or disabled	When disabled, some requests will be too large for any of the disks.

Table 3. Simulation settings

5.2 Analysis

We present here results plotted based on StorAlloc simulation data. From these figures, we can conclude on a right sizing of the intermediate storage architecture while we can compare the efficiency of the tested scheduling algorithms. For this analysis, platforms and algorithms have been chosen to reflect a variety of behaviors.

Platform sizing In our dataset, the sum of all the storage capacities requested by clients, called *sum_cap*, reaches 1.6 PB. In Figure 4, we plot the percentage

of this value achieved by each of the 256 runs of our simulation according to storage layouts and algorithms, grouped by platform capacity and split strategy.

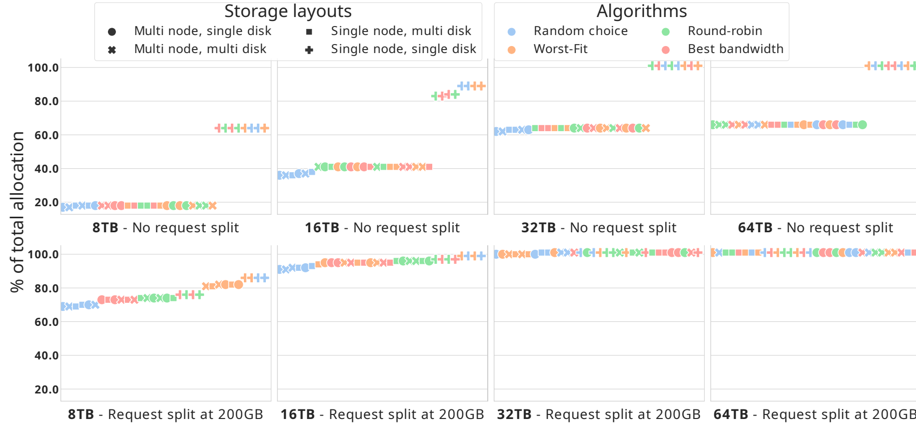

Fig. 4. Percentage of *sum_cap* (sum of the requested capacities in the entire dataset) per simulation run, grouped by capacity and split strategy. Ordered in each plot by total allocation volume (GB)

On the top row (no request split), only the *1N1D* layout at 32TB and 64TB capacities reaches 100% of *sum_cap*. However this layout is merely a baseline which shouldn't be used, as it leads to a high concurrency and consequently a very low I/O bandwidth. From this result, we can also conclude that never more than 32TB are needed at the same time in our dataset. This information must be balanced by the fact that we exclude from Theta's traces several hundreds of thousands of jobs that we do not consider I/O intensive. The best results with other layouts peak slightly above 60%, which hints towards an underprovisioning of storage resources.

Impact of request splitting on the allocation success rate The bottom row depicts the same analysis when using the split strategy with chunks of 200GB. We see that all layouts reach a 100% of *sum_cap* for 64TB. With a capacity of 32TB, the result is similar, although suggesting an ordering of the algorithms: *worst-fit* and *random* unsurprisingly rank last most of the time because they still fail on a few allocations. More generally, request splitting allows a better use of resources and requires less storage space (even the 16TB platform reaches between 91% and 96% of *sum_cap*). In the following plots, we present data from simulation runs exclusively using the split strategy, as it has proven to bring more efficiency in the resource utilization, and even stands out as one of the leading criteria for maximizing request allocations. Additionally, with many simulations runs reaching over 95% of *sum_cap*, split-enabled setups offer the

opportunity to study the behaviour of our algorithms at a boundary between sufficient and insufficient storage capacity.

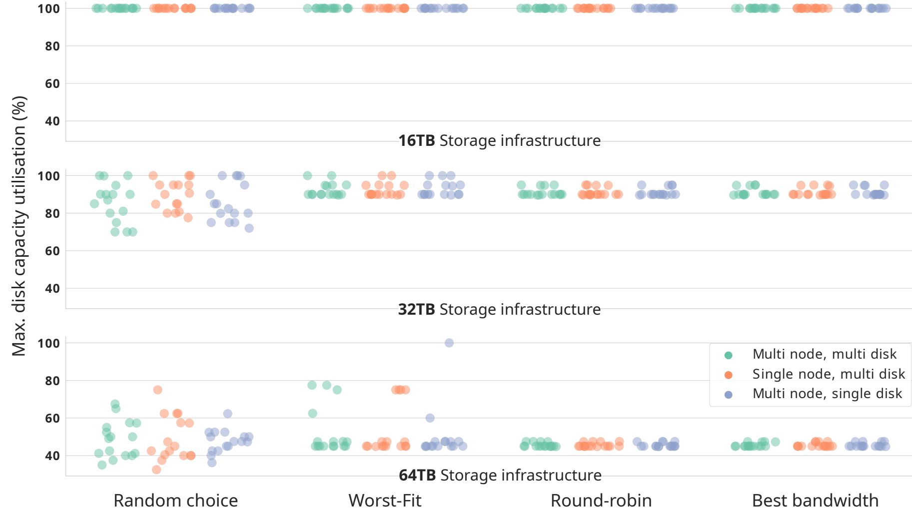

Fig. 5. Maximum disk capacity utilisation (% of capacity), for 16TB, 32TB and 64TB platforms with split strategy and 200G threshold. The *1N1D* layout has been removed.

Impact of request splitting on disk utilization The above results give little information, however, about the use of the disks composing the modeled platform. Figure 5 proposes to study this. Here, we plot the maximum disk utilization, called *max_disk_use*, for 16TB, 32TB and 64TB infrastructures (excluding *1N1D* layout). As expected, the disk utilization rate correlates with the ability to absorb split requests for storage space (Figure 4). For instance, with 16TB of storage capacity, all disks end up full at least once during the simulation. This is an expected result, granted that Figure 4 already exposes this capacity choice as being slightly underprovisioned. The 32TB case presents a different outcome: although disk utilization is globally still very high, only *random* and *worst-fit* algorithms now have disks reaching a 100% of their total capacity. The other two algorithms would instead make a 32TB platform suitable, although with a very small margin before entirely filling up disks. Eventually, it is possible to quantify a potential underutilization, as seen for the 64TB platform where the best algorithms now never use more than 50% of *max_disk_use*. The *worst-fit* algorithm is specifically intended for maximising the use of a single disk from a single node, which explains that it still reaches 100% of *max_disk_use* for one of its disks.

This first analysis shows that a 32TB intermediate storage tier can handle at least 99% of all the I/O intensive jobs in our dataset, as long as the requests are split into 200GB blocks. In that case, some of the targeted disks are used at their full capacity at least once, leaving little flexibility in case of a sudden overload, while the average disk utilization rate is however still very low (1.53%). This is explained by the sparsity of the jobs studied spread over a whole year. Finally, the different layouts tested (*1NnD*, *nN1D*, *nNnD*) behave in much the same way. Nevertheless, they have an impact on the available aggregated bandwidth as long as the scheduling algorithms can efficiently take advantage of the storage disaggregation, as shown in the rest of this paper.

Comparison of scheduling algorithms *fairness* We have implemented four different storage-aware job scheduling algorithms in StorAlloc, as described in 4.2. To evaluate their efficiency, we propose to define a *fairness* metric that looks at the maximum and average number of concurrent allocations per disk allocated by each algorithm. This metric provides information on the balancing of the distribution of requests and consequently on the potential bandwidth available for the allocated jobs: in a permanent maximum I/O regime hypothesis (all jobs with continuous I/O operations), the less allocations are concurrent on resources, the more I/O bandwidth is available.

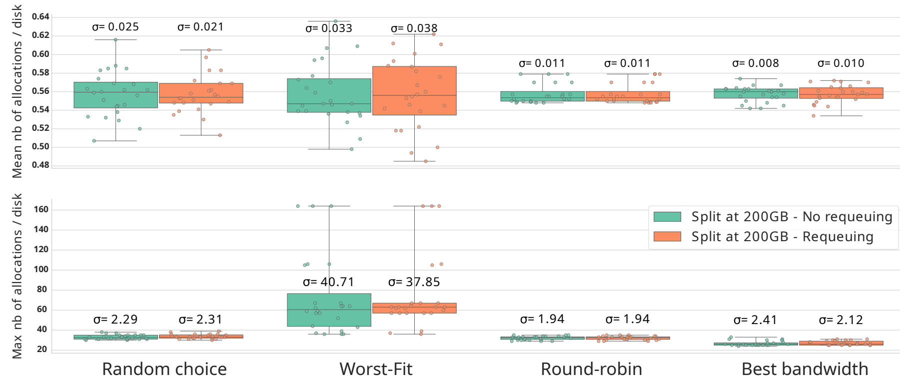

Fig. 6. *mean_alloc* (top) and *max_alloc* (bottom) per disk, grouped by algorithms, for 32TB platform and split strategy. Storage layout *1N1D* excluded. Dots plot the mean and max number of allocations of each disk separately. σ denotes the standard deviation of each group.

Figure 6 depicts this *fairness* for our four algorithms, in the case of split requests. First, we can see that the general variability (standard deviation) in both the mean and max numbers of allocations per disk are lower for *round-robin* and *best-bandwidth* than for *random* and *worst-fit*. As expected, *worst-fit* stands

out, as its design clearly goes against fairness. We also observe that *round-robin* and *best-bandwidth* have a quite similar fairness, with a slight advantage to *best-bandwidth*. This latter is the most advanced algorithm as it takes into account existing allocations on disks and nodes to make a decision. In terms of maximum number of allocations per disk, *best-bandwidth* is almost as stable as *round-robin* and *random*, and also leads to the smallest maximums. In other words, this algorithm can be expected to provide the best average bandwidth to jobs in the permanent regime case.

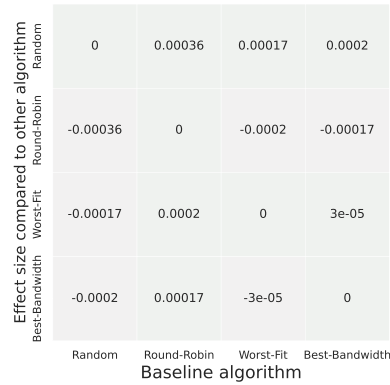

Fig. 7. Cohens d metric for *mean_ alloc*

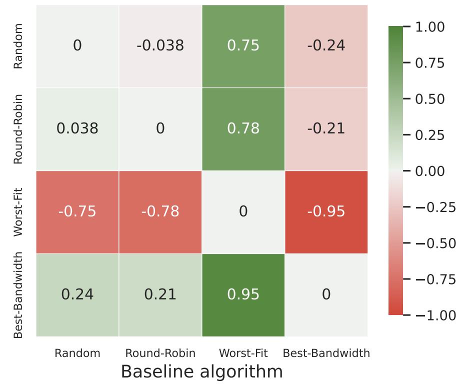

Fig. 8. Cohens d metric for *max_ alloc*

For further analysis, we present the effect sizes between all four algorithms for *mean_ alloc* (Fig 7) and *max_ alloc* (Fig 8), calculated using Cohen's d metric. Cohens d indicates the standardized difference between the means of two groups. This quantitative measurement of the strength of the relationship between two variables is helpful to confirm the initial conclusions drawn from Fig 6.

Fig 7 shows that the choice of algorithm has no effect on the mean number of allocations between disks. This is expected as the platform capacity and layout is the same for all four algorithms and they all manage to grant almost all allocations. On the other hand, Fig 8 validates the observation from Fig 6 that *best-bandwidth* has a strong effect on the maximum number of allocations per disk compared to *worst-fit*, and a small but perceptible effect when compared to *round-robin* or *random* algorithms. However, we also observe that in this experimental context, *round-robin* doesn't actually present any clear effect when compared to the *random* baseline.

Impact of allocation failures and refused requests Our fairness metric must also be balanced with the ability of an algorithm to allocate all the requests sent to it. As noted earlier, two negative outputs for requests are the failure of an allocation that was accepted by the scheduler and the refusal of a request which

is expected to fail. As mentioned in 5.1, only *best-bandwidth* and *round-robin* may refuse requests, but all algorithms may lead to failures. Actively refusing large requests might help with *fairness*, due to less allocations to deal with, but it obviously goes against the primary goals of our scheduler. Figure 9 presents the maximum number of requests in $R_{Split} \cap R_{Failed}$ and $R_{Split} \cap R_{Refused}$ among all simulation runs for a given capacity and a given algorithm.

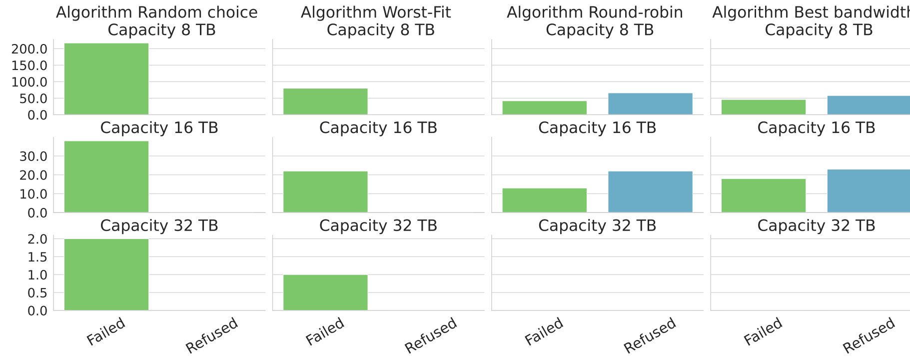

Fig. 9. Maximum number of failed and refused split requests, grouped by algorithm and platform capacity: 8, 16 and 32TB (no failed or refused requests at 64TB).

When the storage resources are very constrained (8TB), we observe up to 217 failures for *random*. In comparison, *best-bandwidth* and *round-robin* both have a combined number of failed and refused requests more than twice as low. This is comparable to the results of *worst-fit*, and shows that refusing requests doesn't have a negative impact compared to other solutions. In the 16TB case, however, there is no clear winner although *random* and *worst-fit* appear to be slightly ahead. Indeed, in this case the relatively high number of refused requests for *best-bandwidth* and *round-robin* still doesn't prevent them from experiencing failures. We can assume that the policy they use to refuse requests is not ideal when storage is only slightly underprovisioned and many requests could either succeed or fail by a small margin. Nevertheless, looking back at Figure 4, we see that for the 16TB case, *random* clearly leads to fewer allocations than *best-bandwidth*, even though it records fewer missed allocations (38 failures for the former, and 41 failed or refused requests for the latter). The advantage for *best-bandwidth* comes here from selecting which requests should be refused rather than facing random failures. As for the 32TB case, no more refused requests are recorded, and only *random* and *worst-fit* algorithms still end up with a handful of failures.

As a result, we show that it is possible to compare our algorithms not only in terms of fairness but also regarding their allocation capabilities. This lets us make sure that aiming for the best fairness, or other comparable goals, doesn't unnecessarily affect the basic capabilities of our algorithms. In this situation,

best-bandwidth and *round-robin* are once again better choices than the other two algorithms.

Behavior in the face of more heterogeneity The previous observations led us to conclude that a possible solution to allocate all the requests in our dataset was to use a 32TB platform, in conjunction with the *best-bandwidth* algorithm and a split request strategy. Requeuing doesn't seem necessary in this case. Using these parameters, we now run a final simulation with a different platform that presents more heterogeneity than our other simulated systems. It is composed of two types of nodes, as described in Table 4. These nodes have different numbers of disks, disk size, disk bandwidth, and network link bandwidth. Considering past results, an algorithm like *best-bandwidth* should be able to strongly minimize the use of HDD nodes, and prefer the SSD partition, which presents enough storage to accommodate most requests.

Disks / node	Disk bandwidth	# of nodes	Total capacity	Network
5 x 1.6TB SSD	2.93Gbps	2	16TB	200Gbps
2 x 2TB HDD	0.08Gbps	4	16TB	100Gbps

Table 4. Heterogeneous platform description

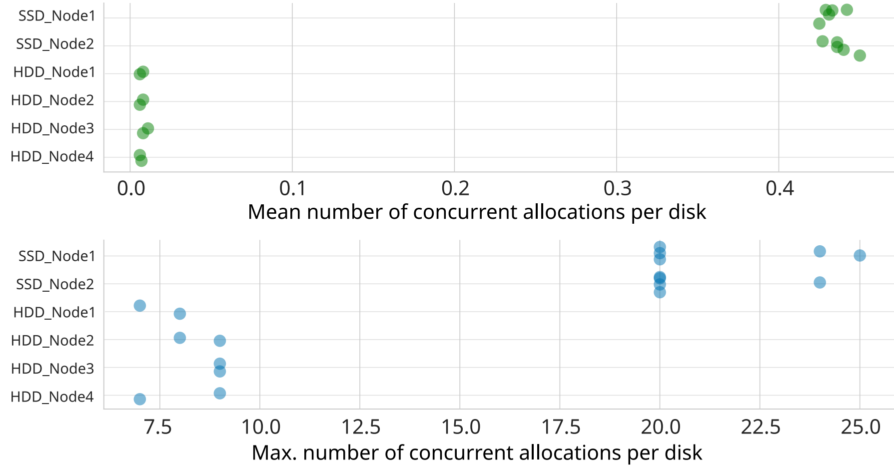

Fig. 10. *mean_alloc* and *max_alloc* per disk, grouped by node, when using a 32TB heterogeneous platform composed of nodes using either SSDs or HDDs.

Figure 10 displays the mean and maximum number of allocations per disk during the simulation, grouped by node. We can conclude on a much more frequent use of SSD nodes, whose disks support up to 25 concurrent allocations

when HDDs see at most 9 allocations per disk. Similarly, mean number of allocations is much higher for SSDs than for HDDs. In that regard, *best-bandwidth* worked as expected and indeed strongly favored the SSD partition. The offloading of some allocations onto the HDDs was also expected as, as shown in Figure 4, *best-bandwidth* wasn't able to allocate 100% of *sum_cap* on a 16TB platform which is the size of the SSD-based partition. Nevertheless, *best-bandwidth* proved to be resilient to this new level of heterogeneity. It once again reached 97% of *sum_cap*, with no more than 17 failed split requests, and all SSD disks ended-up full before the HDDs were used. These results are compatible with the observations made in the previous sections, and show that the algorithm is adaptable and performs well in this new setting.

Using this experimental setting, we can also compare effect sizes between algorithms when the layout changes. *Best-bandwidth* is the only algorithm in our study able to distinguish between SSDs and HDDs when granting allocations. Consequently, we expect that using it instead of any other algorithm will have a much greater effect than what was reported in Fig 8 (same capacity HDDs only).

First of all, we plot the maximum number of allocations per disk for all four algorithms, grouping data by disk type, in Fig 11. We show that only *best-bandwidth* and *worst-fit* grant more allocations on SSDs than on HDDs. However in the case of *worst-fit*, the decision is entirely based on the smaller capacity of the SSDs, and we can observe that HDDs still receive a considerable amount of allocations. In general, *best-bandwidth* appears to display some of the smallest maximums for the number of allocations per disk.

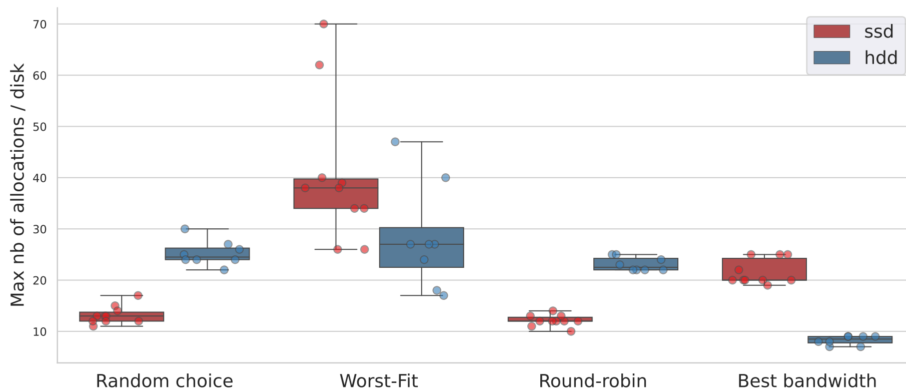

Fig. 11. Maximum number of allocations per disk, grouped by algorithm and disk type (32TB platform with heterogeneous disk layout). Given the platform capacity, all algorithms almost manage to grant every allocation from the dataset.

In Table 5, we confirm the observations from Fig 11, using a Cohen's d metric computed between *best-bandwidth* and each one of the three other algorithms,

for the maximum number of allocations per disk. We observe that Cohen's d is always greater or comparable to the values presented in Fig 8. In particular, we show a very large effect when looking only at SSDs or only HDDs disks, which support the initial design goal of the *best-bandwidth* algorithm.

Best-bandwidth compared to	Random	Round-robin	Worst-fit
Values from Fig 8 (HDD-only)	0.24	0.21	0.95
All nodes / all disks	0.42	0.21	1.77
SSD nodes only	1.18	0.99	1.66
HDD nodes only	-3.68	-5.20	2.59

Table 5. Cohen's d metric for compared use of *Best-Bandwidth* and other algorithms, computed on the *max_ alloc* metric.

6 Conclusion

This paper builds upon our previous work on StorAlloc, and offers a detailed perspective on this tool. StorAlloc is a DES-based simulator integrated on top of a job scheduler architecture. It helps investigate scheduling strategies for I/O intensive jobs on heterogeneous storage resources distributed across a HPC system. Through an extensible design and flexible configuration settings, this tool can be used to model diverse storage infrastructures, and to implement various scheduling strategies. In this work, we have explored a large parameter space, using multiple simulated platforms and algorithms. We demonstrated how StorAlloc can efficiently ingest a consequent number of allocation requests generated from production traces and output storage-related metrics. They provide valuable insights for storage platform sizing and scheduling algorithms evaluation. In particular, we have shown that a 32TB burst-buffer partition on Theta, a top-tier supercomputer at Argonne National Laboratory, would be sufficient to absorb the I/O burden of the I/O intensive jobs running on the system. Eventually, we proposed to assess the reliability of one of our algorithms, *best-bandwidth*, capturing its behaviour through multiple metrics and a statistical analysis.

In future work, we plan to carry out additional experimental campaigns using other datasets, along with new metrics, infrastructures and storage-aware scheduling algorithms. We are also currently evaluating the benefits we can obtain from simulation frameworks such as Wrench [9] for the implementation of our simulation component. In that regard, we started to work on the necessary foundations for porting features from StorAlloc to Wrench. We hope this will allow us to eventually combine compute and storage criteria into a single scheduling algorithm and analyse its behaviour with advanced feedback, thanks to the state-of-the-art models in Wrench. Another goal will be to add support for Cloud storage in order to simulate emerging hybrid workflows distributed across HPC and Cloud infrastructures, using storage as a staging area for coupling components.

References

1. BeeGFS. https://www.beegfs.io/c/, accessed: 2022-12-12
2. Cinder block scheduler. https://docs.openstack.org/cinder/pike/configuration/block-storage/schedulers.html, accessed: 2023-03-30
3. Lustre filesystem website. https://www.lustre.org/
4. Aupy, G., Beaumont, O., Eyraud-Dubois, L.: Sizing and Partitioning Strategies for Burst-Buffers to Reduce IO Contention. In: IPDPS 2019 - 33rd IEEE International Parallel and Distributed Processing Symposium. Rio de Janeiro, Brazil (May 2019), https://hal.inria.fr/hal-02141616
5. Bez, J.L., Boito, F.Z., Schnorr, L.M., Navaux, P.O.A., Méhaut, J.F.: Twins: Server access coordination in the i/o forwarding layer. In: 2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP). p. 116–123 (Mar 2017). https://doi.org/10.1109/PDP.2017.61
6. Bleuse, R., Dogeas, K., Lucarelli, G., Mounié, G., Trystram, D.: Interference-Aware Scheduling Using Geometric Constraints, Lecture Notes in Computer Science, vol. 11014, p. 205–217. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-96983-1_15, https://link.springer.com/10.1007/978-3-319-96983-1_15
7. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience **41**(1), 23–50 (2011). https://doi.org/https://doi.org/10.1002/spe.995, https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.995
8. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. Journal of Parallel and Distributed Computing **74**(10), 2899–2917 (Jun 2014), http://hal.inria.fr/hal-01017319
9. Casanova, H., Ferreira da Silva, R., Tanaka, R., Pandey, S., Jethwani, G., Koch, W., Albrecht, S., Oeth, J., Suter, F.: Developing Accurate and Scalable Simulators of Production Workflow Management Systems with WRENCH. Future Generation Computer Systems **112**, 162–175 (2020). https://doi.org/10.1016/j.future.2020.05.030
10. Chikhaoui, A., Lemarchand, L., Boukhalfa, K., Boukhobza, J.: Multi-objective optimization of data placement in a storage-as-a-service federated cloud. ACM Transactions on Storage **17**(3), 1–32 (Aug 2021). https://doi.org/10.1145/3452741
11. Chunduri, S., Harms, K., Groves, T., Mendygral, P., Zarins, J., Weiland, M., Ghadar, Y.: Performance evaluation of adaptive routing on dragonfly-based production systems. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 340–349. IEEE, USA (2021), https://doi.org/10.1109/IPDPS49936.2021.00042
12. Cope, J., Liu, N., Lang, S., Carns, P., Carothers, C., , Ross, R.: Codes: Enabling co-design of multi-layer exascale storage architectures. In: Workshop on Emerging Supercomputing Technologies (WEST 2011) (2011)
13. Cornebize, T.: High Performance Computing : towards better Performance Predictions and Experiments. Theses, Université Grenoble Alpes [2020-....] (Jun 2021), https://theses.hal.science/tel-03328956
14. Daley, C.S., Ghoshal, D., Lockwood, G.K., Dosanjh, S., Ramakrishnan, L., Wright, N.J.: Performance characterization of scientific workflows for the optimal use of burst buffers. Future Generation Computer Systems **110**, 468–480 (2020). https://doi.org/10.1016/j.future.2017.12.022

15. Dutot, P.F., Mercier, M., Poquet, M., Richard, O.: Batsim: A realistic language-independent resources and jobs management systems simulator. In: Desai, N., Cirne, W. (eds.) Job Scheduling Strategies for Parallel Processing. pp. 178–197. Springer International Publishing, Cham (2017)
16. Fishman, G.S.: Principles of discrete event simulation. [book review] (1 1978), https://www.osti.gov/biblio/6893405
17. Gainaru, A., Aupy, G., Benoit, A., Cappello, F., Robert, Y., Snir, M.: Scheduling the i/o of hpc applications under congestion. In: 2015 IEEE International Parallel and Distributed Processing Symposium. p. 1013–1022. IEEE, Hyderabad, India (May 2015). https://doi.org/10.1109/IPDPS.2015.116, http://ieeexplore.ieee.org/document/7161586/
18. Gainaru, A., Aupy, G., Benoit, A., Cappello, F., Robert, Y., Snir, M.: Scheduling the i/o of hpc applications under congestion. In: 2015 IEEE International Parallel and Distributed Processing Symposium. p. 1013–1022. IEEE, Hyderabad, India (2015). https://doi.org/10.1109/IPDPS.2015.116, http://ieeexplore.ieee.org/document/7161586/
19. Henseler, D., Landsteiner, B., Petesch, D., Wright, C., Wright, N.J.: Architecture and design of Cray DataWarp. In: Proceedings of 2016 Cray User Group (CUG) Meeting (2016)
20. Herbein, S., Ahn, D.H., Lipari, D., Scogland, T.R., Stearman, M., Grondona, M., Garlick, J., Springmeyer, B., Taufer, M.: Scalable i/o-aware job scheduling for burst buffer enabled hpc clusters. In: Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing. p. 69–80. ACM, Kyoto Japan (May 2016). https://doi.org/10.1145/2907294.2907316, https://dl.acm.org/doi/10.1145/2907294.2907316
21. Isaila, F., Garcia Blas, F.J., Carretero, J., Liao, W.k., Choudhary, A.: A scalable message passing interface implementation of an ad-hoc parallel i/o system. The International Journal of High Performance Computing Applications **24**(2), 164–184 (May 2010). https://doi.org/10.1177/1094342009347890
22. Jay, L., Zheng, F., Liu, Q., Klasky, S., Oldfield, R., Kordenbrock, T., Schwan, K., Wolf, M.: Managing variability in the io performance of petascale storage systems. In: In 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–12. New Orleans, LA, USA (2010), https://doi.org/10.1109/SC.2010.32
23. Khetawat, H., Zimmer, C., Mueller, F., Atchley, S., Vazhkudai, S.S., Mubarak, M.: Evaluating burst buffer placement in hpc systems. In: IEEE International Conference on Cluster Computing (CLUSTER). p. 1–11 (2019). https://doi.org/10.1109, https://doi.org/10.1109/CLUSTER.2019.8891051
24. Kougkas, A., Devarajan, H., Sun, X.H., Lofstead, J.: Harmonia: An interference-aware dynamic i/o scheduler for shared non-volatile burst buffers. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER). p. 290–301. IEEE, Belfast (2018). https://doi.org/10.1109/CLUSTER.2018.00046, https://ieeexplore.ieee.org/document/8514889/
25. Lebre, A., Legrand, A., Suter, F., Veyre, P.: Adding storage simulation capacities to the simgrid toolkit: Concepts, models, and api. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. pp. 251–260 (2015). https://doi.org/10.1109/CCGrid.2015.134
26. Lee, B.H., Song, T.G., Kim, D.H.: Block storage scheduling based on sla in cloud storage systems. In: Proceedings of the Sixth International Conference on Emerging Databases: Technologies, Applications, and Theory. p. 72–76. EDB '16, Association for Computing Machinery, New

York, NY, USA (Oct 2016). https://doi.org/10.1145/3007818.3007825, https://dl.acm.org/doi/10.1145/3007818.3007825
27. Li, Y., Lu, X., Miller, E.L., Long, D.D.E.: Ascar: Automating contention management for high-performance storage systems. In: 2015 31st Symposium on Mass Storage Systems and Technologies (MSST). p. 1–16 (May 2015). https://doi.org/10.1109/MSST.2015.7208287
28. Liu, J., Chen, Y., Zhuang, Y.: Hierarchical i/o scheduling for collective i/o. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. p. 211–218 (May 2013). https://doi.org/10.1109/CCGrid.2013.30
29. Lockwood, G., Hazen, D., Koziol, Q., Canon, R., Antypas, K., Balewski, J.: Storage 2020: A Vision for the Future of HPC Storage. In: Report: LBNL-2001072. Lawrence Berkeley National Laboratory (2017), https://escholarship.org/uc/item/744479dp
30. Lockwood, G.K., Snyder, S., Byna, S., Carns, P., Wright, N.J.: Understanding data motion in the modern hpc data center. In: 2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW). pp. 74–83 (2019). https://doi.org/10.1109/PDSW49588.2019.00012
31. Monniot, J., Tessier, F., Robert, M., Antoniu, G.: StorAlloc: A Simulator for Job Scheduling on Heterogeneous Storage Resources. In: HeteroPar 2022. Glasgow, United Kingdom (Aug 2022), https://hal.inria.fr/hal-03683568
32. Negru, C., Pop, F., Mocanu, M., Cristea, V., Hangan, A., Vacariu, L.: Cost-aware cloud storage service allocation for distributed data gathering. In: 2016 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR). p. 1–5 (May 2016). https://doi.org/10.1109/AQTR.2016.7501280
33. Núñez, A., Vázquez-Poletti, J., Caminero, A., Castañé, G., Carretero, J., Llorente, I.: Icancloud: A flexible and scalable cloud infrastructure simulator. Journal of Grid Computing **10**, 185–209 (03 2012). https://doi.org/10.1007/s10723-012-9208-5
34. Ruiu, P., Caragnano, G., Graglia, L.: Automatic dynamic allocation of cloud storage for scientific applications. In: 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems. pp. 209–216 (2015). https://doi.org/10.1109/CISIS.2015.30
35. Schmuck, F., Haskin, R.: Gpfs: A shared-disk file system for large computing clusters. In: Proceedings of the 1st USENIX Conference on File and Storage Technologies. p. 19–es. FAST '02, USENIX Association, USA (2002)
36. Vishwanath, V., Hereld, M., Iskra, K., Kimpe, D., Morozov, V., Papka, M.E., Ross, R., Yoshii, K.: Accelerating i/o forwarding in ibm blue gene/p systems. In: SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. p. 1–10 (Nov 2010). https://doi.org/10.1109/SC.2010.8
37. Zhou, Z., Yang, X., Zhao, D., Rich, P., Tang, W., Wang, J., Lan, Z.: I/o-aware batch scheduling for petascale computing systems. In: 2015 IEEE International Conference on Cluster Computing. p. 254–263 (Sep 2015). https://doi.org/10.1109/CLUSTER.2015.45