StorAlloc: A Simulator for Job Scheduling on Heterogeneous Storage Resources

Julien Monniot(✉), François Tessier, Matthieu Robert, and Gabriel Antoniu

University of Rennes, Inria, CNRS, IRISA, Rennes, France
{julien.monniot,francois.tessier,matthieu.robert,
gabriel.antoniu}@inria.fr

Abstract. The ability of large-scale infrastructures to store and retrieve a massive amount of data is now decisive to scale up scientific applications. However, there is an ever-widening gap between I/O and computing performance. A way to mitigate this consists of deploying 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, in a similar way as compute resources. In that regard, we introduce StorAlloc, a simulator used as a testbed for assessing storage-aware job scheduling algorithms and evaluating various storage infrastructures.

1 Introduction

Running scientific applications at scale requires the power of a large infrastructure such as a High-Performance Computing (HPC) system. For years, HPC systems have been designed with the main objective of improving computing power. However, nowadays the corpus of compute-centric applications has evolved towards complex data-centric workflows across the domains of modeling, simulation, AI and data analytics. The *data deluge* engendered by these workloads has been observed in major supercomputing centers: the National Energy Research Scientific Computing Center, 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% [12]. Yet, during the same period, we have observed a relative performance decrease of storage systems: a study of the top three supercomputers from the Top500 ranking between 2011 and 2021 shows that the ratio of I/O bandwidth to computing power has been divided by 9.6.

An attempt to mitigate this gap has led to the emergence of new tiers of intermediate storage, such as node-local disks or burst buffers [9], backed by diverse technologies (Flash memory, NVDIMM, NVMeoF, ...), and placed between the compute nodes and the global shared parallel file-system. Although this storage disaggregation offers new alternatives to a centralized storage system, advanced techniques for sizing and allocating these resources have yet to be devised to fully leverage them.

J. Singer et al. (Eds.): Euro-Par 2022 Workshops, LNCS 13835, pp. 211–222, 2023.
https://doi.org/10.1007/978-3-031-31209-0_16

Unfortunately, exploring methods for allocating storage resources on supercomputers suffers from several limitations such as a difficult access to the hardware with enough privileges or a panel of technologies reduced to those deployed on the studied system. Simulation is one way to overcome these constraints. At the cost of a loss of accuracy, ideally as moderate as possible, simulation offers much better flexibility for representing a wide variety of storage architectures and can be used to evaluate storage infrastructures before they are deployed.

In this paper, we propose to explore how storage resources can be allocated on HPC systems, i.e. with 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 do so, we introduce StorAlloc, a Discrete-Event Simulation-based (DES) simulator of a batch scheduler able to play (or replay) the scheduling of I/O intensive jobs on intermediate storage resources. We first present the architecture of StorAlloc, then we evaluate the tool on a set of basic scheduling algorithms and on multiple models of infrastructures featuring heterogeneous storage resources. From our simulations, we can conclude on the right sizing of intermediate storage resources among a set of architectures or analyze the utilization rate of the underlying disks.

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 [1] or Spectrum Scale (formerly GPFS [15]), 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 [10]. 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), 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

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.

	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
Burst buffer	Middleware	Job	(Parallel) file-system
Network-attached storage	API	Node(s)	Raw storage space

to get access exclusively to compute nodes (usually though a job scheduler), the allocation of those intermediate levels of storage remains minor in practice and often limited to homogeneous resources. In order to use these new levels of storage to their full potential, new allocation techniques must be invented 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. 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

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, it is the computational aspect that is essentially addressed. SimGrid [4], 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, although preliminary work was started a few years ago [11], storage resource allocation is absent from the framework. A few SimGrid derived simulators also have job scheduling oriented approaches. This is the case of batsim [7] or Wrench [5] for example. However, the full support of heterogeneous storage levels as allocatable resources is not implemented (disk capacity is not modeled in batsim for example). Another difference between these solutions and StorAlloc concerns the design of the tool as described in Sect. 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.

The world of Cloud Computing is more familiar with the allocation of storage tiers as well as compute or network resources. Work has been done to simulate the allocation of resources between different users [3,13] in virtualized environment but these works are outdated and have very limited storage support.

Finally, models for partitioning and sizing intermediate storage resources such as burst buffers have been studied [2,14]. These techniques are the basis of storage-aware job scheduling algorithms that could be evaluated in our simulator.

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.

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 the original architecture, we added the ability to run it as a simulator, using a single code base.

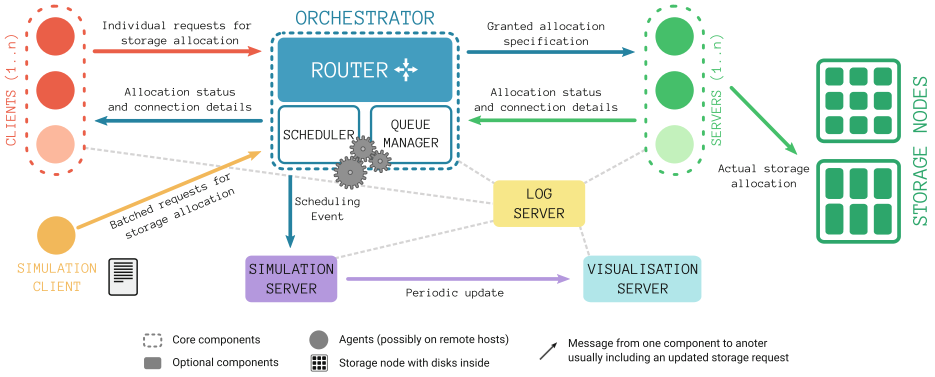
CLIENTS (1..n)
Individual requests for storage allocation
Allocation status and connection details
ORCHESTRATOR
ROUTER
SCHEDULER
QUEUE MANAGER
Granted allocation specification
Allocation status and connection details
SERVERS (1..n)
Actual storage allocation
STORAGE NODES
Batched requests for storage allocation
Scheduling Event
LOG SERVER
SIMULATION CLIENT
SIMULATION SERVER
Periodic update
VISUALISATION SERVER
Core components
Optional components
Agents (possibly on remote hosts)
Storage node with disks inside
Message from one component to anoter usually including an updated storage request

Fig. 1. StorAlloc Architecture

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 1 depicts the components already implemented and how they fit together. At the core, 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.

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. 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 Sect. 4.3, 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.

4.1 Scheduling of 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 Fig. 1. 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 naive 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.

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 impact of these strategies, independent of the scheduling algorithms, is evaluated in Sect. 5. Again, we make the assumption that in case of (possibly repeated) allocation failures, I/O will be performed on the global shared parallel file system.

Table 2. Optional scheduling strategies

	Default setting	Comment
Split	Threshold at 200 GB	Split requests with capacity over threshold and allocate the parts on multiples resources
Requeued	5 retries, one every 5 m	Postpone starting time and retry a failed allocation

4.2 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). Whenever required by a parent server, a node should be able to setup and expose a specific partition of their 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. 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 Sect. 5 [6]) 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.

4.3 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. Then, the simulation is unrolled and go through the scheduler, using a discrete event simulation (DES) model [8]. During that phase, data measuring the impact of scheduling is collected and feeds a visualization server in real-time. In particular, we measure the following indicators:

- Total allocated (and deallocated) volume.
- Mean and max number of simultaneously allocated requests (global, per node and per disk).
- Mean and max percentage of non-free disk space for each disk over the simulation.
- Number of requeued requests and total delay time during the simulation.
- Number of split requests if any.
- Request's status: allocated or refused.

4.4 Implementation Details

The proof of concept 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]. 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 thanks to metrics we have defined.

5.1 Simulation Setup

To simulate storage requests from clients representative of real applications, we used a dataset composed of one year of a Darshan[2] logs on Theta, a 11.7 PFlops

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

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

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 based on the maximum of either read or write volume while the allocation time uses the initial job duration.

In order to have a good overview of what can be observed with our simulator, we have run 192 different simulation setups based on the settings presented in Table 3. The average simulation time is around 25 m 48 s per run, in a range of [5 m 40 s; 1 h 29 m 57 s] on a single core of a Intel Core i7-1185G7 processor. This variability is due to the difference in complexity of the algorithms and the activation or not of the requeuing and splitting systems.

Table 3. Simulation settings

Settings	Tested values	Comment
Algorithm	Random, round-robin worst-fit, best-bandwidth	See Sect. 4.1
Total capacity	8 TB, 16 TB, 64 TB	Disk sizes are 1, 2 and 8 TB 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 5 m, until a 60 m delay
Split	200 GB or disabled	When disabled, some requests will be too large for any of the disks

5.2 Analysis

We present here results plotted from StorAlloc simulation data. From these figures, we can conclude on an approximation of a right sizing of the platform and we can compare the efficiency of the tested scheduling algorithms. For this analysis, platforms and algorithms have been chosen to reflect a variety of behavior.

Platform Sizing. In our dataset, the sum of all the storage capacities requested by clients, called *sum_cap*, reaches 1.6 PB. In Fig. 2, we plot the percentage of

[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.

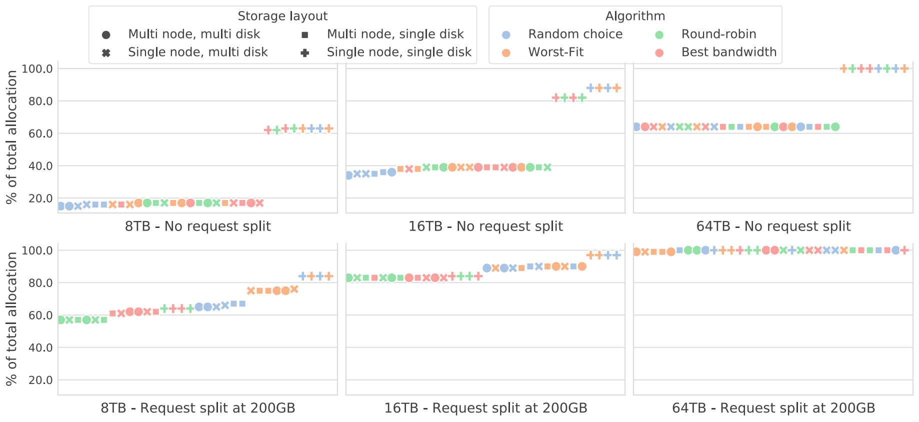

Fig. 2. Percentage of *sum_cap* (sum of the requested capacities in the entire dataset) per simulation run, grouped by capacity and split strategy.

this value achieved by each of the 192 runs of our simulation according to storage layouts and algorithms, grouped by platform capacity and split strategy.

On the top row (no request split), only the *1N1D* layout at 64 TB capacity 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 node bandwidth. From this result, we can also conclude that never more than 64 TB 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. The bottom row depicts the same analysis with requests split in chunks of 200 GB. We see that all layouts reach a 100% of *sum_cap* at least once for 64 TB. More generally, the splitting of requests allows a better use of resources and requires less storage space (the 16 TB platform reaches 90% of *sum_cap* for half of the runs). These results give little information, however, about the use of the disks composing the modeled platform.

Figure 3 proposes to study this. Here, we plot the maximum disk utilization, called *max_disk_use*, for both 16 TB and 64 TB infrastructures (excluding *1N1D* layout). As expected, the disk utilization rate correlates with the ability to absorb split requests for storage space (Fig. 2). Nevertheless it is possible to quantify a potential underutilization, as seen for the 64 TB platform where no more than 65% of disk capacity is ever used. The worst-fit algorithm is specifically intended for maximising the use of a single disk from a single node, which explains that it reaches 100% of *max_disk_use* for several disks.

This first analysis shows that a platform slightly larger than 16 TB can handle all the I/O intensive jobs in our dataset, as long as the requests are split into 200 GB blocks. In that case, the targeted disks are mostly used at their full capacity at least once, leaving little flexibility in case of a sudden overload,

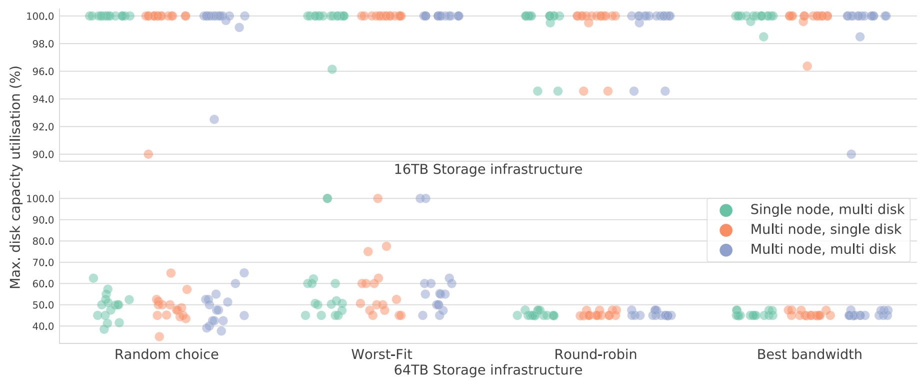

Fig. 3. Maximum disk capacity utilisation (% of capacity), for 16 TB and 64 TB platforms with request split threshold at 200G. The 1N1D layout has been removed.

while the average disk utilization rate is however very low (2.82%), which 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.

Scheduling Algorithms Comparison. We have implemented four different storage-aware job scheduling algorithms in StorAlloc, as described in Sect. 4.1. 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 (split or not) and consequently on the potential bandwidth available for the allocations: in a permanent maximum I/O regime hypothesis (all jobs with continuous I/O operations), the less allocations are concurrent on resources, the more bandwidth will be available.

Figure 4 depicts this *fairness* for our four algorithms. 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 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 to make a decision. In terms of maximum number of allocations per disk, best-bandwidth is the most stable, and also usually 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. Best-bandwidth behaves better than round-robin which, under the same conditions, tends to show more irregularities.

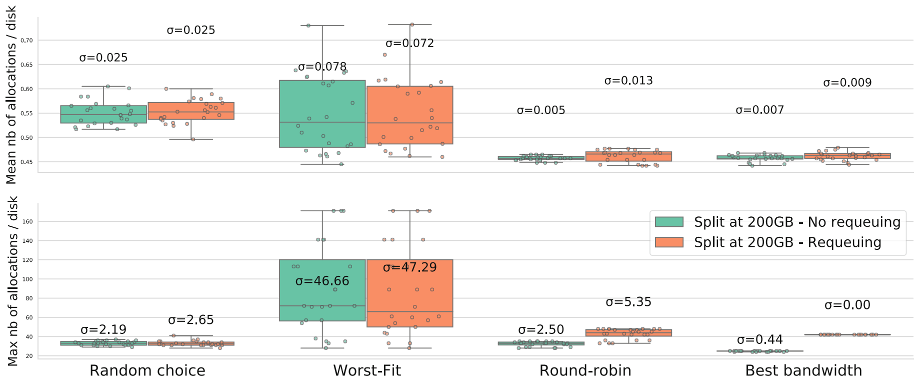

Fig. 4. Mean (top) and max (bottom) number of allocations per disk, grouped by algorithms, for 16 TB platform and split strategy. Storage layout 1N1D excluded. Dots plot the mean and max number of allocations of each disk separately.

Finally, we can see the impact of a queuing system on the number of allocations, i.e. fewer jobs are refused and have to fall back on the parallel file system.

6 Conclusion

In this paper, we have introduced StorAlloc, a DES-based simulator used to explore the scheduling of I/O intensive jobs on heterogeneous storage resources distributed across a HPC system. We have detailed its extensible design and configuration settings for modeling storage infrastructures and implementing various scheduling strategies. Our evaluation demonstrated how StorAlloc can ingest a large number of allocation requests generated from production traces and output storage-related metrics which provide valuable insights for storage platform sizing and scheduling algorithms evaluation. Building upon this preliminary work, we plan to extend this experimental campaign to more metrics, infrastructures and storage-aware scheduling algorithms. Another direction we want to take is to evaluate the benefits we could get from simulation frameworks such as Wrench [5] for the implementation of our simulation component. Finally, a longer term goal will be to explore how to combine computing and storage resources within the same request and provide suitable scheduling algorithms.

References

1. Lustre filesystem website. https://www.lustre.org/
2. 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 (2019). https://hal.inria.fr/hal-02141616

3. 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. Softw. Pract. Exp. **41**(1), 23–50 (2011). https://doi.org/10.1002/spe.995. https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.995
4. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. J. Parallel Distrib. Comput. **74**(10), 2899–2917 (2014). http://hal.inria.fr/hal-01017319
5. Casanova, H., et al.: Developing accurate and scalable simulators of production workflow management systems with WRENCH. Future Gener. Comput. Syst. **112**, 162–175 (2020). https://doi.org/10.1016/j.future.2020.05.030
6. Chunduri, S., et al.: Performance evaluation of adaptive routing on dragonfly-based production systems. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS), USA, pp. 340–349. IEEE (2021). https://doi.org/10.1109/IPDPS49936.2021.00042
7. 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.) JSSPP 2015-2016. LNCS, vol. 10353, pp. 178–197. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61756-5_10
8. Fishman, G.S.: Principles of Discrete Event Simulation (1978). [Book Review]. https://www.osti.gov/biblio/6893405
9. 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)
10. Jay, L., et al.: Managing variability in the IO performance of petascale storage systems. In: 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, pp. 1–12 (2010). https://doi.org/10.1109/SC.2010.32
11. 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
12. Lockwood, G., Hazen, D., Koziol, Q., Canon, R., Antypas, K., Balewski, J.: Storage 2020: a vision for the future of HPC storage. Report: LBNL-2001072, Lawrence Berkeley National Laboratory (2017). https://escholarship.org/uc/item/744479dp
13. Núñez, A., Vázquez-Poletti, J., Caminero, A., Castañé, G., Carretero, J., Llorente, I.: iCanCloud: a flexible and scalable cloud infrastructure simulator. J. Grid Comput. **10**, 185–209 (2012). https://doi.org/10.1007/s10723-012-9208-5
14. 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
15. 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, FAST 2002, USA, p. 19-es. USENIX Association (2002)