Can Current SDS Controllers Scale To Modern HPC Infrastructures?

Mariana Miranda, Yusuke Tanimura†, Jason Haga†, Amit Ruhela⋆, Stephen Lien Harrell⋆
John Cazes⋆, Ricardo Macedo, José Pereira, João Paulo
INESC TEC & University of Minho †*AIST* ⋆*TACC & UTAustin*

***Abstract*—Modern supercomputers host numerous jobs that compete for shared storage resources, causing I/O interference and performance degradation. Solutions based on software-defined storage (SDS) emerged to address this issue by coordinating the storage environment through the enforcement of QoS policies. However, these often fail to consider the scale of modern HPC infrastructures.**

In this work, we explore the advantages and shortcomings of state-of-the-art SDS solutions and highlight the scale of current production clusters and their rising trends. Furthermore, we conduct the first experimental study that sheds new insights into the performance and scalability of flat and hierarchical SDS control plane designs.

Our results, using the Frontera supercomputer, show that a flat design with a single controller can scale up to 2,500 nodes with an average control cycle latency of 41 *ms*, while hierarchical designs can handle up to 10,000 nodes with an average latency ranging between 69 and 103 *ms*.

***Index Terms*—Software-defined storage, scalability, control plane, QoS**

I. Introduction

As HPC facilities move towards the exascale era, we are witnessing a significant increase in the computational power of supercomputers, due to better hardware but also due to the increase in the number of compute nodes. For instance, Frontier and Aurora supercomputers, operating at 1.206 and 1.012 exaFLOPS, have 9,408 and 10,624 compute nodes, respectively [1, 2]. This scale makes managing supercomputers increasingly harder, in some cases, compromising the performance and quality-of-service (QoS) for user jobs [3, 4].

Indeed, a known issue that we explore in this paper is related to I/O contention caused by having multiple jobs simultaneously accessing shared HPC storage resources [5]–[7], typically exposed through a Parallel File System (PFS), such as Lustre, BeeGFS, among others [8, 9].

Intuitively, as the number of compute nodes increases, so does the number of simultaneous jobs executing and accessing the shared PFS. Furthermore, as supercomputers are requested more and more to run data-centric workloads (*e.g.,* DL and LLM training), I/O contention becomes even more aggravated as these jobs run for long periods of time and require consecutive data and metadata accesses to the PFS [10]–[13]. Although numerous solutions are proposed to ensure better storage QoS for HPC jobs, they do not address several challenges that are particular to HPC infrastructures.

Intrusive to critical HPC components. Systems like GIFT [14], CALCioM [15], and TBF [4] mitigate storage performance interference and variability by modifying core layers of the HPC I/O stack, such as I/O libraries, job schedulers, and the PFS. These are intrusive approaches that require profound code refactoring over critical software components, increasing the work needed to maintain and port them to new platforms.

Static and uncoordinated control. On the other hand, solutions like OOOPS [16] transparently intercept and rate limit POSIX requests from the application side, thus not requiring changes to core layers of the HPC software stack. However, these operate as isolated instances that are agnostic of other jobs in the system, being unable to coordinate the requests submitted from all jobs to the PFS. Further, these also do not consider that HPC environments are highly dynamic, with jobs frequently entering and leaving the system, each with specific I/O patterns. These factors can lead to severe I/O contention, performance interference, and poor resource usage.

To overcome these challenges, several solutions based on software-defined storage (SDS) are proposed, providing non-intrusive, dynamic, and coordinated control over all HPC jobs, ensuring that storage QoS policies are met at all times [17, 18]. In detail, the SDS paradigm proposes the separation between how jobs' data is intercepted and manipulated, and how policies are ensured at the HPC infrastructure, which is achieved through two distinct planes of functionality — the *control plane* and the *data plane*, as shown in Fig. 1. The data plane is divided into stages residing at compute nodes and sitting between the job (*i.e.,* application) and the PFS client. Stages transparently intercept and rate limit I/O requests according to specific policies defined by the control plane (*e.g.*, limit I/O bandwidth and/or number of metadata operations submitted to the PFS). The control plane acts as an independent service with cluster-wide visibility. It orchestrates all stages, meticulously coordinating their actions through control algorithms to ensure holistic storage policies (*e.g.,* I/O priority, fairness) across the HPC infrastructure.

However, current research on SDS systems mainly focus on the data plane counterpart, leaving important decisions about the control plane's design unexplored, especially when it comes to its scalability. While several works express the importance of having a scalable design for the control plane, most of them handle it as an orthogonal challenge [19, 20]. In fact, the few solutions addressing this challenge do not provide empirical evidence of the benefits of their designs at scale, conducting experiments with only a few dozen nodes.

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00123

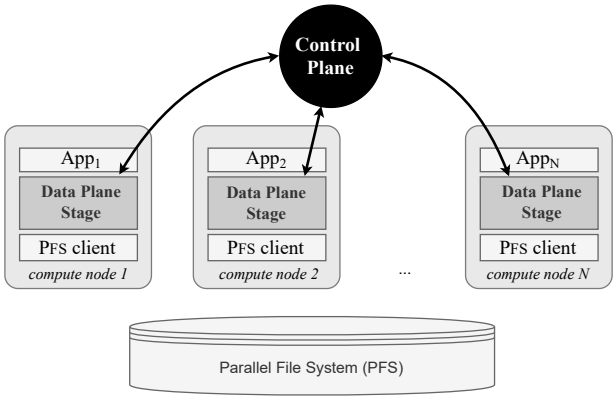

Fig. 1: *Typical SDS design applied over HPC infrastructures.*

As such, this paper presents the first experimental study that empirically demonstrates the scalability and performance implications of state-of-the-art control plane architectures. The main insight of this paper is that such an experimental study, while accounting for HPC infrastructures of different sizes, is key to better understand the current limitations of SDS systems and their applicability to large-scale supercomputers.

In detail, the paper provides the following contributions. We leverage the Cheferd control plane, a state-of-the-art storage controller, to implement *flat* and *hierarchically* distributed designs [18]. Then, we conduct an experimental evaluation under synthetic workloads comparing the scalability of both architectures for infrastructures of different sizes (*i.e.,* varying number of compute nodes). Our study is the first to assess the scale of these two designs for infrastructures with up to 10,000 nodes.

Our results show that a single-node flat control plane can ensure QoS for up to 2,500 nodes, with a latency under 41 *ms* for each control cycle.[1] For larger infrastructures, a hierarchical design is necessary. Our results show that this design can ensure QoS for up to 10,000 nodes, with latency ranging from 69 *ms* and 103 *ms* depending on the number of controllers used.

II. Motivation and Related Work

In this section, we overview the landscape of HPC infrastructures in terms of scale and maximum performance, and discuss the state-of-the-art of SDS control planes.

A. Modern HPC Infrastructures

To better understand the sheer magnitude of modern supercomputers, Table I highlights some of the HPC systems included in the Top500 [21] list (June 2024), which ranks the 500 supercomputers by their performance (*i.e.*, according to the results of running the *LINPACK Benchmark* [22]).

In the first place, the Frontier supercomputer [1], with 1.206 EFlop/s, is located at the Oak Ridge National Laboratory and comprises a total of 9,408 compute nodes. It is followed by the Aurora system [2] from the Argonne National Laboratory, with 1.012 EFlop/s and a total of 10,624 compute nodes.

TABLE I: *Supercomputers Top500 rank, peak performance, number of nodes, and installation year.*

System	Rank	Rmax (PFlop/s)	Number of nodes	Year
Frontier [1]	1	1,206	9,408	2021
Aurora [2]	2	1,012	10,624	2023
Fugaku [23]	4	442	158,976	2020
Summit [24]	9	148.6	4,608	2018
Frontera [25]	33	23.52	8,368	2019

Interestingly, the Fugaku supercomputer [23], achieving 442 PFlop/s in this benchmark, is one order of magnitude larger than the previous supercomputers, comprising 158,976 compute nodes.[2] Even when considering older petascale systems such as Summit [24] from Oak Ridge National Laboratory (148.6 PFlop/s) and Frontera [25] from TACC (23.52 PFlop/s), we can see that the number of compute nodes is in the order of thousands (4,608 and 8,368, respectively).

In summary, modern supercomputers are operating with thousands of compute nodes, and the tendency is for this number to increase in the future. Moreover, ARM-based systems are expected to further increase this scale, as observed with the Fugaku supercomputer, which operates with over a hundred thousand nodes.

Given this scale, it is clear why HPC centers have been struggling to efficiently manage the shared load at their PFS systems [4, 14, 16, 26], especially when a majority of these nodes are running demanding data-intensive workloads (*e.g.*, DL, LLM).

B. SDS and Scalability

The need to provide better storage services for distributed infrastructures, such as those of cloud computing and HPC infrastructures, motivated the emergence of distinct SDS solutions over the last years [17]. Interestingly, these solutions can be used to enforce a wide scope of objectives (*i.e.,* storage policies) such as I/O bandwidth guarantees [19, 27], I/O prioritization [19, 28]), I/O routing [29], caching [29, 30], data reduction [27], and data encryption [27].

Despite having disparate storage objectives, all these works follow a decoupled design (Fig. 1) where data plane stages mediate, in isolation, the I/O flow between applications and the storage system, while implementing the necessary mechanisms (*e.g.,* caching, I/O rate limiting, encryption, compression).

The control plane is then responsible for ensuring high-level storage policies (*e.g.,* ensure I/O fairness across jobs, prioritize some jobs, encrypt data for a given set of applications) by holistically monitoring and coordinating the different data plane stages placed across the compute nodes where jobs are running. The control plane must, therefore, implement control algorithms that collect I/O metrics from data plane stages, compute an optimal set of rules, and enforce such rules at

[1] A control cycle considers the time needed for the control plane to collect metrics from all data plane stages, compute the best strategy to ensure the desired QoS strategy, and enforcing the necessary rules over the stages.

[2] In some other benchmarks, such as HPL-AI, Fugaku reaches exascale performance.

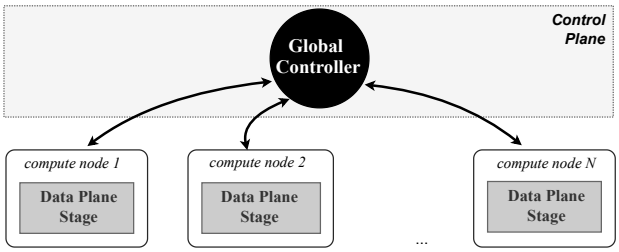

Fig. 2: *Centralized control plane design.*

the stages. For example, a control algorithm may receive the current IOPS handled by each stage, compute the fair share of IOPS for optimal usage of the underlying storage system, and enforce this rule individually at each stage. These three phases (*i.e.*, collect, compute, and enforce), are periodically repeated (*e.g.,* every second) to ensure storage policies are met at all times, even under workload and system variations. The periodicity of these control cycles determines how fast the control plane reacts to changes in the system, usually being set by the system administrator. The latency of these cycles determines how fast the control plane reconfigures new policies in the system.

Monitoring and coordinating large-scale HPC infrastructures (as those depicted in Table I) is, therefore, a complex task that must be handled by the control plane. However, this is often overlooked in the state-of-the-art. We next summarize the two main designs followed for the control plane of current solutions. Figs. 2 and 3 depict two examples of these designs.

Centralized and flat designs. Many solutions discuss that, although exposed as a logically centralized service, their control planes need to be physically distributed, implementation-wise, for scale and fault tolerance purposes [19]. However, most of these works leave the scalability details of their designs as orthogonal work. For instance, IOFlow's [19] centralized control plane is designed for small-to-medium data centers (*i.e.*, tens to hundreds of nodes), deferring the exploration of scaling for larger data centers to future work.

Crystal [27] implements a flat design in which the control plane can be composed of multiple controllers, each responsible for a different set of applications and storage policies. Controllers can be dynamically provisioned or removed as new applications or policies arrive or leave the infrastructure. As controllers manage independent storage policies, these do not need coordination among each other. Crystal's control plane scalability is evaluated only with tens of nodes.

On the other hand, in Mirador [31], coordination between controllers is required for I/O load balancing decisions across the nodes of cloud-based storage solutions. Since this work is directed at storage appliances (*i.e.,* NFS-based storage systems), the scale of the solution is significantly smaller than when considering large-scale HPC infrastructures.

Hierarchical designs. Few solutions deviate from a flat organization, designing the control plane as a hierarchy of controllers [18, 29, 32, 33]. This hierarchy follows a tree-based topology. Controllers at the top level of the tree are responsible

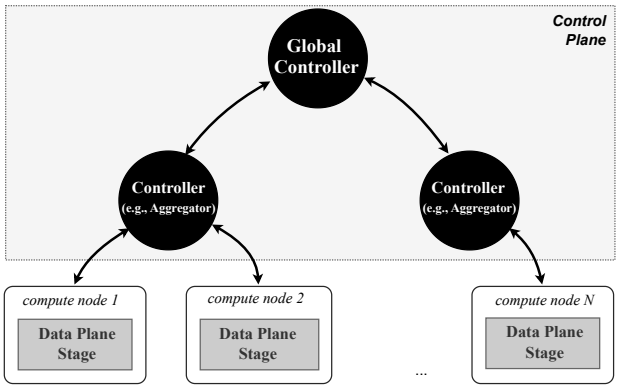

Fig. 3: *Hierarchical control plane design.*

for executing the control algorithms and propagating enforcement rules across controllers at the remaining levels of the hierarchy. Controllers at the bottom level directly communicate with the data plane stages, enforcing the QoS rules defined by the upper levels of the hierarchy. The latter can also perform part of the computation (*e.g.*, aggregate metrics from data plane stages or controllers at lower tree levels), reducing the work that controllers at the top tree level must perform.

CLARISSE [32] and SIREN [33] control planes are specifically designed for large-scale HPC infrastructures and both propose a hierarchical design to address the scalability and performance requirements of these infrastructures. However, the experimental evaluation of these systems does not surpass the 1,024 compute node mark.

Cheferd [18], also designed for HPC systems, provides the notion of local controllers, aggregating metrics collected at each compute node, and a global centralized controller receiving these metrics and ensuring global QoS objectives. Metric aggregation is only possible when multiple data plane stages reside at the same node, while different compute nodes are all managed by the global controller. Cheferd's scalability is again only tested for up to 1,000 nodes.

C. Summary

As supercomputers grow in size, it becomes fundamental to effectively control the I/O requests submitted by several thousands of compute nodes to the shared PFS. Although SDS is a promising solution to achieve such a goal, the scalability of the control plane has been largely overlooked in the literature. In this paper, we aim to give the first steps towards a better understanding of flat and hierarchical control plane designs and to answer the following open questions.

- What is the scalability of single-node, flat-based control planes?
- Can hierarchical designs ensure better scalability than flat-based ones?
- What is the performance impact of adding more controllers across the hierarchy?
- How are the different phases of the control cycle impacted by each control plane design?

III. Scalability study of SDS control planes

To answer the previous open research questions, we next detail the experimental methodology of our study and then present and discuss its main results.

A. Experimental setup

All experiments were conducted on compute nodes of the Frontera supercomputer [25], equipped with two 28-core Intel Xeon processors, 192 GiB of RAM, a single 240 GiB SSD, and a Mellanox InfiniBand HDR-100 network card, running CentOS 7.9 with the Linux kernel v3.10. Compute nodes are connected to a shared Lustre file system (PFS).

B. Designs under testing

Our study compares two control plane implementations mimicking state-of-the-art flat and hierarchical designs.

Flat. The implementation of the flat centralized design is based on a simplified version of the state-of-the-art Cheferd control plane [18]. As depicted in Fig. 2, the global controller is deployed in a single compute node and has system-wide visibility, being able to orchestrate I/O workflows at the HPC infrastructure. The control logic (*i.e.,* control algorithms) is implemented in a feedback control loop, where it continuously collects metrics from data plane stages (*e.g.,* I/O bandwidth of each compute node), verifies if QoS policies are being met, and computes and enforces new storage rules for the uncompliant data plane stages. Moreover, as depicted in Fig. 2, each compute node comprises a single data plane stage to manage its I/O requests.

Hierarchical. To achieve a hierarchical design, we extended the *Flat* design prototype by introducing a new layer of controllers, named *aggregators*. As depicted in Fig. 3, each of these controllers is deployed in individual compute nodes, logically sitting between the global controller and the data plane stages. Their role is to disseminate requests sent by the global controller to the stages, and to aggregate and send back those results to the global controller.

The goal of this design is to understand the performance implications of adding an extra level of controllers that independently aggregate the metrics from several compute nodes and send pre-processed information to the global controller, offloading some of the processing that the latter needs to do.

C. Workloads

Computation algorithm. At the global controller, we run the *proportional sharing without false allocation* (PSFA) state-of-the-art control algorithm [18]. Briefly, the algorithm allows assigning different rates of I/O operations (IOPS) per job (*i.e.*, it allows defining jobs with more IOPS than others to ensure different QoS levels), while guaranteeing that jobs running simultaneously never surpass the maximum rate of operations that can be handled efficiently by the PFS. The latter value is defined by system administrators. Furthermore, the algorithm is aware of the actual I/O rate submitted by each job, proportionally assigning leftover IOPS to active jobs whenever available, preventing under- and over-provisioning of shared storage resources.

Across all experiments, we run the algorithm under a stress workload, where the control plane continuously executes control cycles without interruption. In each control cycle, metrics are collected from data plane stages, the PSFA algorithm executes, and new rules are sent to all stages. This workload ensures that we stress test the control plane to better understand its scalability and performance characteristics, a common approach followed by other benchmarks [34]–[36].

Compute nodes workload. At the compute nodes, we implemented a lighter version of a data plane stage that mimics the behavior of a regular stage without the need to run real applications. We refer to them as virtual stages, and their goal is to reply back to the control plane every time it requests metrics (*i.e.*, IOPS for *data* and *metadata* operations in this study). Since the PSFA algorithm is working under a stress workload, regardless of the value of each collected metric, it must run its computation to all collected metrics and send enforcement actions to all virtual stages.

With virtual stages we can accurately mimic the payload of realistic deployments while running several instances of these at the same compute node to simulate larger infrastructures (*i.e.*, with thousands of compute nodes).

D. Methodology

Considering that we want to explore the scalability of storage controllers in modern HPC infrastructures with thousands of nodes, it is unfeasible to have a dedicated supercomputer of such size to run real applications. Thus, we instead simulate the necessary scale by deploying several virtual data plane stages per compute node (*i.e.,* 50 virtual stages). Aggregator controllers and the global controller run on separate compute nodes. The number of compute nodes used varies with the scale of each experiment.

Although we deploy several virtual data plane stages per compute node, from a logical point of view, either from aggregator controllers or even the global controller, these are treated as being deployed in separate compute nodes. Since virtual stages use few computational resources, our experiments are not impacted by having multiple instances in a single compute node.

For each experiment, after deploying the necessary controllers, the global controller executes the PSFA algorithm for at least 5 minutes. As further shown in Section IV, since the average control cycle latency is at the *ms* scale, the overall execution time is enough to get normalized results. Each test was repeated at least 3 times. For each test, we collected the total control cycles completed by control plane, the average latency of control cycles, and a breakdown of the latency across the different control phases (namely, collect statistics, compute PSFA, enforce rules).

Moreover, for each compute node where experiments were conducted, we collected resource usage metrics (using the REMORA resource usage collection tool [37]), namely CPU, memory, and network usage.

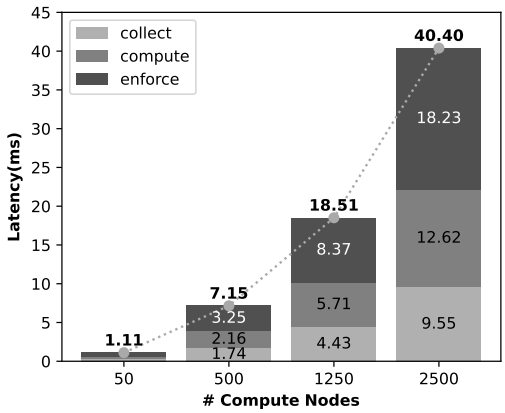

Fig. 4: *Average latency of control cycles for a flat control plane design with a single global controller managing an increasing number of compute nodes.*

IV. Experimental results

We start by presenting the results for the flat control plane design and then proceed to analyze the main findings for the hierarchical controller.

A. Flat controllers

To assess the performance and scalability of a single-node, flat-based control plane (Fig. 2), we measured the average latency of control cycles for an increasing number of compute nodes, namely 50, 500, 1,250 and 2,500.[3] The results for these experiments are depicted in Fig. 4, where we also breakdown the latency for the different phases of a control cycle. The standard deviation for all the results discussed in our study is below 6%.

As expected, having a single controller managing more nodes leads to an increase in the average latency of control cycles, increasing from 1.11 *ms* for 50 nodes, to 40.40 *ms* for 2,500 nodes. When observing the latency breakdown for each control cycle, the enforce phase is more demanding than the collect phase, not only because the message payload is larger (which involves more data being transmitted across the network) but also due to the need for coordinating to which compute node each storage rule should be submitted. Furthermore, the time used in each phase increases proportionally with the number of nodes.

Despite the significant increase in latency, a single controller running the PSFA algorithm is able to manage up to 2,500 compute nodes under 41 *ms*. However, besides the performance overhead induced by collecting metrics, computing, and enforcing rules over data plane stages, the control plane must also manage a large pool of network connections to these. Inevitably, a single controller is limited by the number of physical connections, which, in the case of the Frontera node where we deploy the controller, can only handle up to 2,500 concurrent connections. This is a limitation of the networking equipment, whose value may change but that will also be present in other systems.

[3] As discussed in Section III, we assume that each compute node is running a single data plane stage.

TABLE II: *Resource utilization for CPU, memory, and network consumption for a flat control plane with a single global controller.*

Controller	Resource	Setup			
		# Compute Nodes			
		50	*500*	*1250*	*2500*
Global	*CPU (%)*	6.07	9.58	10.39	10.34
	Memory (GB)	0.07	0.31	0.64	1.18
	Transmitted (MB/s)	5.67	8.74	8.74	9.73
	Received (MB/s)	3.74	5.75	5.74	5.36

Table II shows the resource usage (namely, CPU, memory, and network bandwidth) for the single global controller. We observe an increase in resource usage as the number of compute nodes handled by the controller increases. Nevertheless, even when considering 2,500 compute nodes, CPU and memory usage is below 11% and 1.2 GB. Also, network bandwidth is under 9.73 MB/s and 5.36 MB/s for sent and received data, respectively.

> **Observation #1.** *A flat control plane with a single global controller is a suitable design for small to medium infrastructures where one reacts to I/O workload variations at the tens of milliseconds or larger time windows.*

> **Observation #2.** *Having a single controller performing all the computation and managing all network connections to compute nodes leads to a scalability bottleneck, mainly set by the physical limitations of the hardware.*

B. Hierarchical controllers

To assess the performance and scalability of the hierarchical control plane (Fig. 3), we measured the latency of control cycles for 10,000 compute nodes with increasing number of aggregators, namely 4, 5, 10, and 20. Given that each Frontera node can handle up to 2,500 physical connections, we set the minimum number of aggregators to 4.

Scaling for 10,000 compute nodes. With the extra control level (*i.e.,* aggregators) one can now validate the scalability of the hierarchical design for up to 10,000 compute nodes. We start with the most straightforward setup, including 4 aggregator controllers, each collecting metrics and enforcing rules to disjoint sets of 2,500 compute nodes (stages). The global controller receives aggregated metrics from the aggregators, computes the PSFA algorithm, and sends back the sets of enforcement rules to the aggregator nodes.

As depicted in Fig. 5, with 4 aggregators, the control plane can scale up to 10,000 nodes with a control cycle of approximately 103 *ms*. As expected there is a latency increase mainly due to the amount of compute nodes being managed. Interestingly, although the number of nodes increases 4×, the latency only doubles when compared to the hierarchical design, with a single aggregator managing 2,500 nodes (Fig. 6).

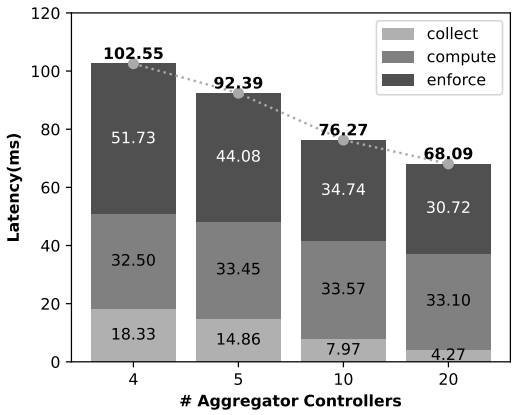

Fig. 5: *Average latency of control cycles for the hierarchical design managing 10,000 compute nodes with an increasing number of aggregator controllers.*

As shown in Table III, using 1 global controller and 4 aggregators increases the overall resource usage when compared to a single-node flat control plane (Table II). This is an expected result as the number of compute nodes being managed increases from 2,500 to 10,000. For instance, memory usage increases since more metadata needs to be stored at the global controller. Interestingly, CPU consumption is now split between the global and aggregator controllers, while the latter requires more CPU as aggregating metrics from 2,500 nodes is more compute-intensive than running the PSFA algorithm. Network bandwidth is also split across the two types of controllers. The global controller receives aggregated metrics, explaining the lower network bandwidth for received bytes. On the other hand, transmitted bandwidth is higher as this controller must calculate rules for all data plane stages and transmit these to the corresponding aggregator controllers.

> **Observation #3.** *An hierarchical design, with 4 aggregator controllers, is able to scale for large infrastructures (up to 10,000 nodes) and react to I/O workload variations at the hundreds of milliseconds or larger time windows.*

Increasing the number of aggregator controllers. We also compare the impact of increasing the number of aggregators in the hierarchical design, as shown in Fig. 5. For all experiments, the number of compute nodes remains the same, as we only vary the subset of nodes that each aggregator is responsible for. Specifically, with a setup of 4 aggregators, each must handle a disjoint set of 2,500 compute nodes, while for a setup with 20 aggregators, each handles a set of 500 nodes.

Results show that as the number of aggregators increases, while the latency for *compute* phase latency remains approximately the same, it decreases for the *collect* and *enforce* phases. This is expected as distributing the load across more controllers enables more parallelism in metrics collection and rule enforcement at the data plane stages.

With 10 aggregators, one can reduce the average latency of control cycles to under 80 *ms*, while with 20 aggregators latency is kept below 70 ms. However, this decrease in latency comes with the expense of requiring additional servers to run the control plane service.

TABLE III: *Resource utilization for CPU, memory, and network consumption for a hierarchical design managing 10,000 compute nodes. Results depict usage for the global controller and the average resource consumption per aggregator controller.*

Controller	Resource	Setup			
		# Aggregator Controllers			
		4	*5*	*10*	*20*
Global	*CPU (%)*	2.55	2.81	3.22	3.52
	Memory (GB)	3.52	3.56	3.53	3.60
	Transmitted (MB/s)	4.39	4.73	5.66	6.08
	Received (MB/s)	1.45	1.58	1.82	1.98
Aggregator	*CPU (%)*	3.95	3.4	1.94	0.95
	Memory (GB)	0.16	0.13	0.08	0.04
	Transmitted (MB/s)	4.53	4.13	2.4	1.31
	Received (MB/s)	2.53	2.31	1.34	0.73

Regarding resource usage, we observe that the global controller uses more resources when more aggregator controllers are employed (Table III). This is expected since the amount of work and metadata being handled by the global controller increases with the number of aggregators being managed. On the other hand, the work done at each aggregator decreases as the 10,000 compute nodes are distributed across more controllers. This trend is also visible for the consumption of memory and network bandwidth.

> **Observation #4.** *Increasing the number of aggregator controllers reduces the average latency of control cycles. For highly dynamic I/O workloads (e.g., burstiness) this can be an important feature to ensure sustained storage QoS.*

> **Observation #5.** *While adding more aggregators benefits latency, it requires further computational resources for the control plane service. Therefore, there is a trade-off between the amount of resources and control cycle latency that must be chosen according to the needs of the targeted infrastructure.*

Overhead of adding more control levels. We now analyze the overhead of adding aggregator controllers to the critical path of control cycles. Fig. 6 compares the performance of both control designs (single-node flat controller *vs.* hierarchical controller with a single aggregator) for 2,500 compute nodes.

As expected, there is an increase in latency, although small, when moving to the hierarchical design. Latency increases from approximately 41 *ms* to 53 *ms*, which stems from the *collect* and *enforce* phases due to the additional network hop going through the aggregator controller. Interestingly, we observed a decrease in the latency of the *compute* phase.

As depicted in Table IV, we observe a significant decrease in CPU consumption at the global controller, which is now moved to the aggregator, when considering the hierarchical design. This happens because metrics from the 2,500 compute nodes are now merged at the aggregator controller. As

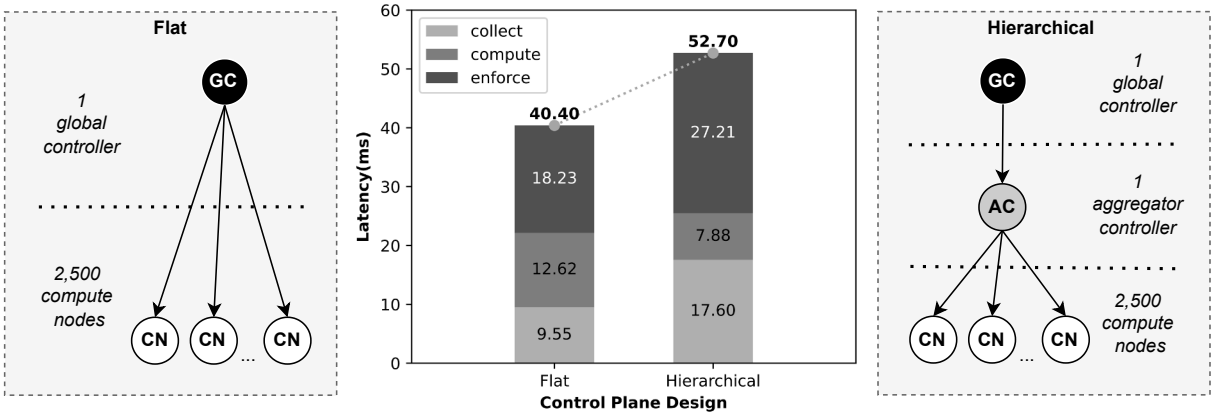

Fig. 6: *Average latency of control cycles for flat and hierarchical (with a single aggregator node) designs managing 2,500 compute nodes.*

TABLE IV: *Resource utilization for CPU, memory, and network consumption for the flat and hierarchical (with a single aggregator node) designs handling 2,500 compute nodes.*

Controller	Resource	Setup	
		Flat	***Hierarchical***
Global	*CPU (%)*	10.34	1.15
	Memory (GB)	1.18	0.92
	Transmitted (MB/s)	9.73	2.36
	Received (MB/s)	5.74	0.77
Aggregator	*CPU (%)*	-	7.83
	Memory (GB)	-	0.22
	Transmitted (MB/s)	-	8.65
	Received (MB/s)	-	4.98

previously discussed, this is more expensive, computation-wise, than running the PSFA algorithm. Network-wise we see a similar pattern to the one observed in the previous experiments, where the global controller receives less data as it was pre-processed by the aggregator controllers.

Observation #6. *For a small amount of compute nodes (up to 2,500), the hierarchical design increases the latency of control cycles up to 12.3 ms, when compared to the flat one. Nonetheless, the latency of control cycles remains under 55 ms.*

Observation #7. *Adding a dedicated control level for aggregating metrics reduces the latency of the compute phase.*

V. Discussion

This paper conducted the first study on the scalability of two widely used control plane designs. Interestingly, our results show that a single flat controller can run the PSFA algorithm for up to 2,500 compute nodes under an average latency of 41 *ms* per control cycle.

To avoid the scalability and hardware limitations of managing a large pool of network connections, one must move to a hierarchical design. Our results show that adding an extra control level responsible for metric aggregation and enforcement of rules (aggregator controller) incurs minimal performance overhead, increasing the average latency of control cycles by approximately 12 *ms*. With 4 aggregator controllers, the hierarchical design can scale up to 10,000 nodes while keeping control cycles latency under 103 *ms*. Interestingly, adding more aggregators further reduces the latency to under 69 *ms*.

Notably, adding more aggregators requires more computational resources, and therefore, this trade-off must be managed according to the I/O workloads and infrastructure requirements. Namely, under volatile and bursty workloads and/or when the control plane is required to quickly react and adapt the QoS rules across several data plane stages, then the control cycles ideally have the lowest possible latency. On the other hand, when it is enough to react to workload changes at larger time-window intervals (*e.g.,* seconds or minutes) and with some delay when adjusting the necessary QoS rules, then opting for the minimal amount of aggregators to support the full set of compute nodes is the best approach to minimize resource usage.

VI. Future Work

Based on these results, this preliminary study opens up multiple interesting research paths. First, there are several design options for distributed control planes that could be explored and then included in a future experimental study. As examples, one could include flat control designs with multiple controllers that coordinate their actions (*e.g.*, through coordination protocols), and can each orchestrate different sets of nodes while maintaining global visibility. As an alternative, one could also include hierarchical designs that further explore the processing logic that can be offloaded to aggregator nodes in order to be able to make independent decisions for the set of controllers/stages managed by these, thus decreasing the computational load from the controllers of the top levels of the tree.

Second, the study could be enriched with further workloads. We chose to use a synthetic, stress-like workload to understand the limits of flat and hierarchical designs, but it would be valuable to study such designs with real workloads and applications. This would allow us to determine whether our system

adapts well to dynamic workloads and assess the impact of the control plane on the execution of the applications themselves.

Another important aspect to explore, which has also been ignored in the literature, is the dependability of the control plane. This is an important aspect since, while the failure of controllers may not lead to unavailability of the infrastructure (*i.e.*, the stages will still be mediating I/O requests with possible outdated rules), it may compromise the guarantees expected for the storage policies in place.

Acknowledgments

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within the project UIDB/50014/2020 (DOI 10.54499/UIDB/50014/2020) and through the Ph.D. grant PD/BD/151403/2021.

References

[1] Frontier. [Online]. Available: https://www.olcf.ornl.gov/frontier/
[2] Aurora. [Online]. Available: http://aurora.alcf.anl.gov
[3] B. Yang, X. Ji, X. Ma, X. Wang, T. Zhang, X. Zhu, N. El-Sayed, H. Lan, Y. Yang, J. Zhai, W. Liu, and W. Xue, "End-to-end I/O Monitoring on a Leading Supercomputer," in *16th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2019, pp. 379–394.
[4] Y. Qian, X. Li, S. Ihara, L. Zeng, J. Kaiser, T. Süß, and A. Brinkmann, "A configurable rule based classful token bucket filter network request scheduler for the lustre file system," in *Proceedings of the International Conference for High-Performance Computing, Networking, Storage and Analysis*. ACM, 2017, pp. 1–12.
[5] C. S. Daley, D. Ghoshal, G. K. Lockwood, S. Dosanjh, L. Ramakrishnan, and N. J. Wright, "Performance Characterization of Scientific Workflows for the Optimal Use of Burst Buffers," *Future Generation Computer Systems*, vol. 110, pp. 468–480, 2017.
[6] M. Dorier, G. Antoniu, R. Ross, D. Kimpe, and S. Ibrahim, "CALCioM: Mitigating I/O interference in HPC systems through cross-application coordination," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 2014, pp. 155–164.
[7] T. Patel, S. Byna, G. K. Lockwood, and D. Tiwari, "Revisiting I/O Behavior in Large-Scale Storage Systems: The Expected and the Unexpected," in *International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2019.
[8] P. Braam, "The Lustre Storage Architecture," 2019.
[9] Beegfs. [Online]. Available: https://www.beegfs.io/c/
[10] Y. Gao, Y. He, X. Li, B. Zhao, H. Lin, Y. Liang, J. Zhong, H. Zhang, J. Wang, Y. Zeng, K. Gui, J. Tong, and M. Yang, "An Empirical Study on Low GPU Utilization of Deep Learning Jobs," in *IEEE/ACM 46th International Conference on Software Engineering*. ACM, 2024.
[11] M. Dantas, D. Leitão, P. Cui, R. Macedo, X. Liu, W. Xu, and J. Paulo, "Accelerating Deep Learning Training Through Transparent Storage Tiering," in *2022 IEEE/ACM 22nd International Symposium on Cluster, Cloud and Internet Computing*. IEEE, 2022.
[12] S. W. Chien, S. Markidis, C. P. Sishtla, L. Santos, P. Herman, S. Narasimhamurthy, and E. Laure, "Characterizing Deep-Learning I/O workloads in TensorFlow," in *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*. IEEE, 2018, pp. 54–63.
[13] F. Chowdhury, Y. Zhu, T. Heer, S. Paredes, A. Moody, R. Goldstone, K. Mohror, and W. Yu, "I/O Characterization and Performance Evaluation of BeeGFS for Deep Learning," in *48th International Conference on Parallel Processing*. ACM, 2019.
[14] T. Patel, R. Garg, and D. Tiwari, "GIFT: A coupon based Throttle-and-Reward mechanism for fair and efficient I/O bandwidth management on parallel storage systems," in *18th USENIX Conference on File and Storage Technologies*. USENIX Association, 2020, pp. 103–119.
[15] M. Dorier, G. Antoniu, R. Ross, D. Kimpe, and S. Ibrahim, "Calciom: Mitigating i/o interference in hpc systems through cross-application coordination," in *2014 IEEE 28th international parallel and distributed processing symposium*. IEEE, 2014, pp. 155–164.
[16] L. Huang and S. Liu, "OOOPS: An innovative tool for IO workload management on supercomputers," in *2020 IEEE 26th International Conference on Parallel and Distributed Systems*. IEEE, 2020, pp. 486–493.
[17] R. Macedo, J. Paulo, J. Pereira, and A. Bessani, "A Survey and Classification of Software-Defined Storage Systems," *ACM Computing Surveys*, vol. 53, no. 3, May 2020.
[18] R. Macedo, M. Miranda, Y. Tanimura, J. Haga, A. Ruhela, S. L. Harrell, R. T. Evans, J. Pereira, and J. Paulo, "Taming Metadata-intensive HPC Jobs Through Dynamic, Application-agnostic QoS Control," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing*. IEEE, 2023, pp. 47–61.
[19] E. Thereska, H. Ballani, G. O'Shea, T. Karagiannis, A. Rowstron, T. Talpey, R. Black, and T. Zhu, "IOFlow: A software-defined storage architecture," in *24th ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 182–196.
[20] R. Macedo, Y. Tanimura, J. Haga, V. Chidambaram, J. Pereira, and J. Paulo, "PAIO: General, Portable I/O Optimizations With Minor Application Modifications," in *20th USENIX Conference on File and Storage Technologies*. USENIX Association, 2022, pp. 413–428.
[21] Top 500. [Online]. Available: http://www.top500.org/
[22] Linpack benchmark. [Online]. Available: https://top500.org/project/linpack/
[23] Supercomputer fugaku. [Online]. Available: https://www.fujitsu.com/global/about/innovation/fugaku/
[24] Summit supercomputer. [Online]. Available: https://www.olcf.ornl.gov/summit/
[25] Frontera. [Online]. Available: https://www.tacc.utexas.edu/systems/frontera
[26] S. Liu, L. Huang, H. Liu, A. Ruhela, V. Trueheart, S. Lindsey, and Q. Yuan, "Practice Guideline for Heavy I/O Workloads with Lustre File Systems on TACC Supercomputers," in *Practice and Experience in Advanced Research Computing*. ACM, 2021.
[27] R. Gracia-Tinedo, J. Sampé, E. Zamora, M. Sánchez-Artigas, P. García-López, Y. Moatti, and E. Rom, "Crystal: Software-Defined Storage for Multi-tenant Object Stores," in *15th USENIX Conference on File and Storage Technologies*. USENIX Association, 2017, pp. 243–256.
[28] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "PriorityMeister: Tail Latency QoS for Shared Networked Storage," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–14.
[29] I. Stefanovici, B. Schroeder, G. O'Shea, and E. Thereska, "sRoute: Treating the storage stack like a network," in *14th USENIX Conference on File and Storage Technologies*. USENIX Association, 2016, pp. 197–212.
[30] I. Stefanovici, E. Thereska, G. O'Shea, B. Schroeder, H. Ballani, T. Karagiannis, A. Rowstron, and T. Talpey, "Software-defined caching: Managing caches in multi-tenant data centers," in *6th ACM Symposium on Cloud Computing*. ACM, 2015, pp. 174–181.
[31] J. Wires and A. Warfield, "Mirador: An Active Control Plane for Datacenter Storage," in *15th USENIX Conference on File and Storage Technologies*. USENIX Association, 2017, pp. 213–228.
[32] F. Isaila, J. Carretero, and R. Ross, "CLARISSE: A Middleware for Data-staging Coordination and Control on Large-Scale HPC Platforms," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2016, pp. 346–355.
[33] S. Karki, B. Nguyen, J. Feener, K. Davis, and X. Zhang, "Enforcing End-to-End I/O Policies for Scientific Workflows Using Software-Defined Storage Resource Enclaves," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 4, pp. 662–675, 2018.
[34] J. Axboe *et al.* Flexible i/o tester. [Online]. Available: https://github.com/axboe/fio
[35] H. Shan, K. Antypas, and J. Shalf, "Characterizing and Predicting the I/O Performance of HPC Applications Using a Parameterized Synthetic Benchmark," in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. IEEE, 2008.
[36] mdtest. [Online]. Available: https://sourceforge.net/projects/mdtest/
[37] C. Rosales, A. Gómez-Iglesias, and A. Predoehl, "Remora: a resource monitoring tool for everyone," in *2nd International Workshop on HPC User Support Tools*. ACM, 2015.