Distributed and Dependable Software-Defined Storage Control Plane for HPC

Mariana Miranda
HASLab, INESC TEC & University of Minho
mariana.m.miranda@inesctec.pt

***Abstract*—The Software-Defined Storage (SDS) paradigm has emerged as a way to ease the orchestration and management complexities of storage systems. This work aims to mitigate the storage performance issues that large-scale HPC infrastructures are currently facing by developing a scalable and dependable control plane that can be integrated into an SDS design to take full advantage of the tools this paradigm offers. The proposed solution will enable system administrators to define storage policies (*e.g.*, I/O prioritization, rate limiting) and, based on them, the control plane will orchestrate the storage system to provide better QoS for data-centric applications.**

***Index Terms*—HPC, SDS, Scalability, Dependability**

I. Motivation and Problem Statement

High-Performance Computing (HPC) has become an essential tool in many scientific and industrial advancements [1], as it allows conducting experiments and generating insights in areas that would otherwise be considered impossible or too expensive. This is feasible due to the scale of modern infrastructures, where hundreds to thousands of compute and storage nodes handle multiple applications concurrently.

The shift towards data-driven scientific discoveries has made many HPC applications more data-intensive than ever before [2]. This results in even more concurrent processes trying to access the HPC's shared storage resources, causing I/O interference and performance degradation [3]. In addition, the complexity of these infrastructures hinders the end-to-end control of storage I/O flows and the implementation of global optimizations that could improve its overall performance [4]. These problems become even more aggravating as we move towards the exascale era, since it will enable these systems to run more jobs, worsening the I/O bottleneck as a result.

The storage performance bottlenecks that HPC systems are currently facing go hand-in-hand with the improvements that software-defined storage (SDS) aims to deliver. This paradigm proposes a separation between the control layer and data storage, resulting in a design with two main components: the control plane and data plane [5]. The control plane is a logically centralized entity that handles the control logic, namely coordinates the enforcement of storage objectives (*e.g.,* I/O prioritization, bandwidth aggregation policies). The data plane applies the control logic defined by the control plane over the I/O flows of applications. This decoupling between the control layer and the actual data storage is what gives SDS its advantage over traditional storage systems [6], as it allows for the control plane to have system-wide visibility over HPC storage resources, thus facilitating their orchestration and management.

Although the control plane is a crucial component of the SDS stack and can be seen as the intelligence of the system, its design is usually overlooked in most of the research done on SDS systems. The literature mainly focuses on how storage objectives should be applied by the data plane or contemplates an oversimplified version of a control plane [7]–[11].

Moreover, they generally do not explore how the control plane behaves when handling a large-scale system, or what the procedure is for when a failure occurs. Nevertheless, the scalability and dependability of the control plane are fundamental properties to bear in mind for all types of infrastructures that a control plane may be managing. In our case, HPC systems have specific requirements that differ from those imposed by cloud-based systems. For instance, HPC infrastructures are not only large, but also susceptible to bursty workloads. This requires the control plane to adapt to oscillations in workload, in addition to scaling for the substantial size of an HPC infrastructure. Also, some applications are expected to run for large periods (*i.e.,* days or weeks) with stable and predictable performance; thus, failed or slow SDS components should not compromise their execution.

As a result, our main goal is to design and implement a control plane that can address the current HPC infrastructures requirements of scalability and dependability (with a focus on availability and reliability).

II. Overview of the proposed work

With this work, we aim at providing a scalable and dependable control plane suitable for current HPC infrastructures. Additionally, to provide a full-fledged SDS system, the control plane will be used in conjunction with current state-of-the-art solutions on the data plane field [12], [13].

To ensure the scalability and dependability requirements of HPC infrastructures, we intend to explore a distributed control plane design. Namely, a physically centralized control plane could be a single point of failure and would not be able to tackle the scale of HPC infrastructures. Thus, we envision a design partitioned into multiple distributed controllers, which requires the exploration of synchronization protocols. Moreover, to tackle the issue of dependability, we will consider fault-tolerance protocols to ensure that if a controller fails, such does not affect the system's availability and performance.

In addition, we will provide control algorithms that enable the control plane to deliver accurate enforcement strategies for the desired policies at the storage infrastructure. These possible algorithms span over a large range of functionalities, such as I/O prioritization [7], [14], bandwidth guarantees [7], [15], latency control [14], routing [16], among many others.

III. Current Prototype

The current prototype[1] follows a hierarchical design, where the controllers have different responsibilities depending on their control level. As shown in Figure 1, the current prototype is composed of two types of controllers - global and local.

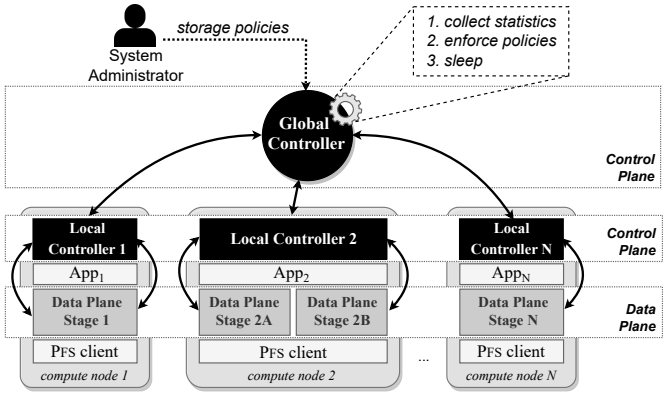

Fig. 1. Current control plane design.

The global controller has system-wide visibility and can holistically orchestrate the storage services. It does so by continuously collecting monitoring metrics from the system (*e.g.,* workflows' rate), and enforcing new policies to respond to workload variations or new rules set by system administrators. However, it only supports a single global controller, which can result in scalability, reliability, and availability issues.

The local controllers serve as a liaison between the global controller and the data plane stages (or stages, for short). A local controller is deployed per compute node and is responsible for managing locally deployed stages. Therefore, when the global controller sends a request to collect statistics or to enforce new storage policies, it is the local controllers' job to disseminate this request to the stages, and aggregate and send back the results to the global controller. This way, only the local controllers deal directly with the stages, offloading some of the global controller's work because it does not need to handle a large number of connections or know the specifics of the stages.

At the moment, this control plane prototype is already fully integrated with a state-of-the-art data plane [13]. Each data plane stage sits between the application and the file system client, and transparently intercepts I/O requests and enforces the storage policies indicated by the control plane (*e.g.*, rate limiting), before submitting the request to the PFS.

Initial testing of the current prototype showed that it can enforce simple control algorithms (*e.g.,* limit App_1 metadata to *X* IOPS) and manage up to 1,000 compute nodes. This would be suitable if we only consider a small-scale HPC center; however, it would have limitations for larger centers or when enforcing more complex control algorithms.

[1]Publicly available at dsrhaslab/cheferd.

IV. Next Steps

Moving forward, we will first focus on the scalability challenge, namely further assess the limits that our initial prototype is able to offer. Some initial testing showed some scalability limitations, namely, on the global controller component; thus, it is crucial to research how we can expand this current solution to the scale that HPC requires. In addition, the control plane allows us to impose a plethora of functionalities and control policies on the jobs running in the system, hence we seek to explore new use cases and control algorithms. Lastly, we will examine several fault-tolerance protocols and existing solutions, to extend the control plane design to be dependable.

V. Related work

Nowadays, we can find several studies that apply the SDS paradigm to a large range of storage infrastructures, from cloud computing [7]–[9], [14], [16], application-specific storage stacks [10], [11], [15], [17], [18], or even in HPC systems [4], [19]. Additionally, the SDS paradigm has been used to enforce several storage objectives such as compression [15], encryption [15], I/O prioritization [7], [14], among many others. Nevertheless, SDS is still in its initial stages, and there are many open questions that need to be explored.

For instance, the design of the control plane is one of them, namely its scalability and dependability is usually overlooked [7]–[11], [14]. As an example, a few solutions tackle the issue of scalability by configuring their systems to follow a hierarchical design instead of a flat one [4], [16], [19]. However, they usually do not explore how the control plane adapts to volatile workloads, which is crucial because these workloads can quickly saturate the storage system.

The dependability of the control plane is an even less explored topic. For instance, previous studies claim that to deal with the failure of a controller, one could simply impose a conservative default policy to be enforced until it is available again [7] or replicate the controller by resorting to a standard Paxos-like technique [16], although they do not actually implement or evaluate these techniques [7], [15], [16], [19].

Acknowledgments

This work was financed by the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through Ph.D. grant PD/BD/151403/2021, and realized within the scope of the project BigHPC – POCI-01-0247-FEDER-045924, funded by the ERDF - European Regional Development Fund, through the Operational Programme for Competitiveness and Internationalization (COMPETE 2020 Programme) and by National Funds through FCT, I.P. within the scope of the UT Austin Portugal Program. This research is being conducted towards a Ph.D. degree supervised by João Tiago Paulo (HASLab, INESC TEC & U. Minho) and José Orlando Pereira (HASLab, INESC TEC & U. Minho).

REFERENCES

[1] P. S. S. Commitee, *The Scientific Case for Computing in Europe 2018-2026*. Addison-Wesley Professional, 2018.

[2] N. Tavakoli, D. Dai, and Y. Chen, "Client-side straggler-aware i/o scheduler for object-based parallel file systems," *Parallel Computing*, vol. 82, pp. 3–18, 2019.

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

[4] F. Isaila, J. Carretero, and R. Ross, "Clarisse: A middleware for data-staging coordination and control on large-scale hpc platforms," in *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2016, pp. 346–355.

[5] R. Macedo, J. Paulo, J. Pereira, and A. Bessani, "A survey and classification of software-defined storage systems," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–38, 2020.

[6] A. Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, and A. Rindos, "Sdstorage: a software defined storage experimental framework," in *2015 IEEE International Conference on Cloud Engineering*. IEEE, 2015, pp. 341–346.

[7] 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 *Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013, pp. 182–196.

[8] M. Murugan, K. Kant, A. Raghavan, and D. H. Du, "flexStore: A software defined, energy adaptive distributed storage framework," in *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*. IEEE, 2014, pp. 81–90.

[9] 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 *Sixth ACM Symposium on Cloud Computing*, 2015, pp. 174–181.

[10] J. Mace, P. Bodik, R. Fonseca, and M. Musuvathi, "Retro: Targeted resource management in multi-tenant distributed systems," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 589–603.

[11] R. Gracia-Tinedo, P. García-López, M. Sánchez-Artigas, J. Sampé, Y. Moatti, E. Rom, D. Naor, R. Nou, T. Cortés, W. Oppermann *et al.*, "Iostack: Software-defined object storage," *IEEE Internet Computing*, vol. 20, no. 3, pp. 10–18, 2016.

[12] 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 (FAST 22)*. USENIX Association, 2022, pp. 413–428.

[13] R. Macedo, M. Miranda, Y. Tanimura, J. Haga, A. Ruhela, S. Harrell Lien, R. Todd Evans, J. Pereira, and J. Paulo, "Taming metadata-intensive HPC jobs through dynamic, application-agnostic QoS control," in *23rd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2023.

[14] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "Prioritymeister: Tail latency qos for shared networked storage," in *ACM Symposium on Cloud Computing*, 2014, pp. 1–14.

[15] 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 (FAST 17)*, 2017, pp. 243–256.

[16] 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 (FAST 16)*, 2016, pp. 197–212.

[17] R. Pontes, D. Burihabwa, F. Maia, J. Paulo, V. Schiavoni, P. Felber, H. Mercier, and R. Oliveira, "SafeFS: A modular architecture for secure user-space file systems: One fuse to rule them all," in *10th ACM International Systems and Storage Conference*, 2017, pp. 1–12.

[18] M. A. Sevilla, N. Watkins, I. Jimenez, P. Alvaro, S. Finkelstein, J. LeFevre, and C. Maltzahn, "Malacology: A programmable storage system," in *Twelfth European Conference on Computer Systems*, 2017, pp. 175–190.

[19] 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.