A survey on flash storage disaggregation: performance and quality of service considerations

Sudarsan Sivakumar
Vrije Universiteit, Amsterdam
s.sivakumar@student.vu.nl

Abstract

Disaggregated flash storage architectures have become increasingly important in cloud environments to enable flexible resource allocation. However, the performance of these disaggregated setups is hindered by the overhead introduced by the software stack. Additionally, ensuring consistent and reliable end-to-end quality of service (QoS) is challenging due to the inclusion of multiple resources (compute, network and storage) and the dynamic shifting of performance bottlenecks. This survey examines the mechanisms proposed to address these challenges in disaggregated flash storage environments.

1 Introduction

Designing server machines with the right balance of CPU, memory, and flash storage is difficult for cloud providers as well, because each application has unique and often dynamically varying requirements for each resource [2, 19]. To accommodate the difference in computing and storage capacity utilization, cloud providers are utilizing disaggregated storage setups. The disaggregated approach decouples storage and compute resources, this enables fine-grained and flexible allocation of storage resources as per user requirements. This reduces the total cost of ownership by avoiding the over-provisioning of resources.

Disk-based disaggregated storage architectures have been a common setup in cloud for a long time [8]. However, the introduction of flash storage has brought the disaggregated storage approach back into the spotlight. Flash storage has revolutionized the storage landscape by delivering the performance of up to one million IOPS with microsecond latencies [6, 10]. Flash storage has provided even more reasons to maintain a disaggregated storage architecture. The expensive flash controllers used in these devices encourage manufacturers to pack more capacity into a single flash module [2], driving the need for disaggregation. Additionally, the under-utilization of available IOPS by many applications can be more effectively addressed through the flexible resource allocation enabled by disaggregated storage [5, 22].

However, the high performance of flash storage has exposed the performance bottlenecks with the existing disaggregated storage software stack [19], which was unable to keep up with the capabilities of these new flash devices. The performance of Flash has made other resources in the disaggregation storage stack susceptible to becoming bottlenecks, affecting the end-to-end quality of service (QoS) experienced by applications.

To ensure consistent end-to-end QoS for disaggregated storage environments, it is necessary to regulate all the resources involved, including compute, and network. This highlights the requirement for a unified approach for QoS enforcement across the components of the disaggregated flash storage infrastructure.

The need for performance and reliable QoS in flash storage disaggregation has motivated a significant amount of research in this direction. To our knowledge, no survey has been published for this topic. This motivates us to compile this survey. The survey is organized as follows: In Section 3, we provide the background on the storage disaggregation software stack. Section 4 discusses performance optimization in different layers of the disaggregation software stack. Section 5 discusses the ensuing unified quality of service in flash storage disaggregation.

2 Study design

In this section, we define the goal of the survey and present the research questions. We also define the scope of the work and the methodology used to find research papers.

2.1 Research goal

The goal of this survey is to look into research done concerning performance and end-to-end unified quality of service in flash storage disaggregation as specified in the introduction. So, we will be answering the following research questions.

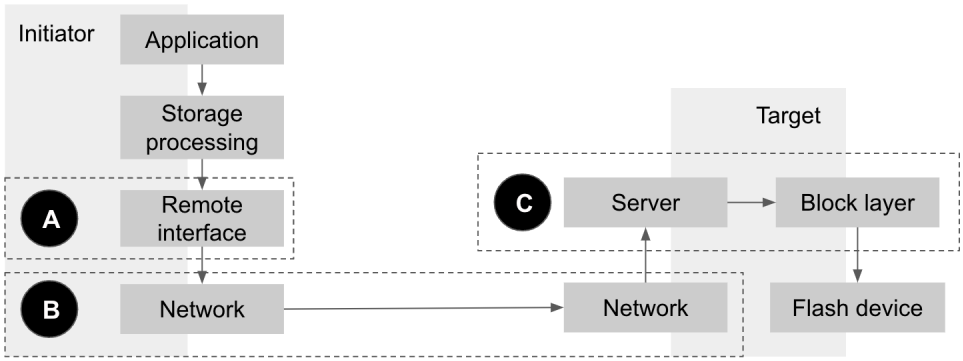

Figure 1: Storage disaggregation architecture with tracing the flow of request across the stack.

RQ1 What are the different mechanisms and their trade-offs for improving the performance of disaggregated flash?

RQ2 What are the challenges and different mechanisms to provide a unified network and storage QoS guarantee?

2.2 Scope

To limit the scope of this survey we considered the following exclusion criteria:

E1 Flash storage disaggregation in context of HPC.

E2 Application specific storage disaggregation.

E3 Disaggregated storage interface other than block or filesystem. We also exclude disaggregated filesystem like HDFS [26], that provides fault tolerance.

E4 QoS enforcement only at the end host.

2.3 Methodology

We use several methods to find papers to include in the survey. We first start with several seed papers and use the Snowball methodology to find related papers by looking at references in both directions. We also look at several conferences that include storage-related topics and look for papers that satisfy the inclusion criteria. The conferences considered are FAST, Hot-Storage, USENIX ATC, SYSTOR, NSDI, and EuroSys. We look through all conferences from 2010 to 2023 and look for relevant papers. Finally, we do a manual search using several relevant keywords.

3 Background

This section provides the necessary background before answer the research questions in this survey. Section 3.1 specifies the general architecture of the storage disaggregation. Section 3.2 provides the overview on exclusive processing per cores architecture for performance.

3.1 Storage disaggregation stack

Fig 1 depicts a schematic representation of a disaggregated storage architecture. This architecture consists of two main components: the initiator and the target. On the initiator side, we have the applications that generate the storage requests. Let's track remote block request propagation from application to the remote flash device. The application submits the request to the block layer in the initiator (Depicted as "Storage processing" in Fig 1), which then processes the request and submits it to the driver corresponding to the target block device (Depicted as "Remote interface").

Since the storage device is located remotely, the request is encapsulated in a packet and sent over the network to the target. On the target side, the server receives the network packet containing the request. The server does processing and then submits the request to its local block layer, which in turn forwards the request to the storage device, in this case, the flash device. Similar propagation of request across components holds for disaggregated flash storage with filesystem interface as well.

3.2 Processing pipeline per core

Since the advent of fast I/O devices, the compute has become the bottleneck in many storage systems. The traditional single-processing pipeline based designs often suffer from performance issues due to contention.

To address this challenge, the design of processing pipelines per core has been explored. This approach is also applicable to disaggregated flash storage, where separate processing pipelines are available for each core on the initiator by default, as shown in Fig 2.

In this disaggregated storage architecture, each I/O flow is processed separately in the core from which it originated. The

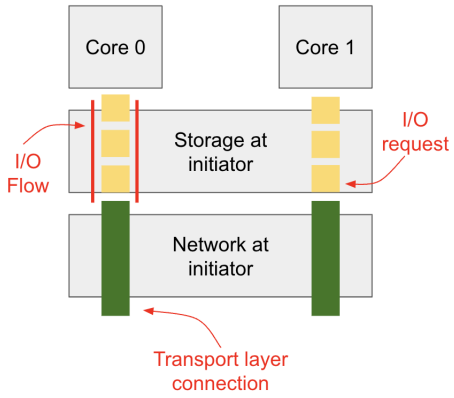

Figure 2: Dedicated storage processing pipeline for each core. In this case the processing pipeline refers to the storage and network processing of I/O.

processing of an I/O flow includes both storage processing and network processing for the disaggregated storage system.

By having dedicated processing pipelines for each I/O flow, the contention and performance issues associated with a single processing pipeline can be mitigated. This design approach allows for more efficient utilization of the available compute resources, leading to improved performance in disaggregated flash storage systems.

4 Performance in flash storage disaggregation

In this section, we will be answering the **RQ1** by looking into different performance optimization and the trade-off. In this context, performance refers to the latency and throughput of I/O operations. For better classification of mechanisms, we further abstract the components presented in Fig 1 into three components: Storage and network interface in the initiator **(marked as A)**, Network **(marked as B)** and Storage and network interface in the target **(marked as C)**.

4.1 Network

Transport protocol	Papers
TCP	i10 [17], Alibaba [23], LT-NoT [13], ReFlex [20]
RDMA	SPANoF [30], Pangu [11, 22]

Table 1: Transport protocols in the selected papers

Based on the survey, we discovered that there has been a focus on optimizing the transport protocol for the network component of the disaggregation stack. The transport protocol is expected to provide high performance and be scalable in the number of connections. Scalability is necessary because of dedicated processing pipeline for each core as we saw earlier.

TCP and RDMA are popular transport layer protocols for flash storage disaggregation. TCP can accommodate a large number of connections, but its compute overhead causes performance concerns, even with segmentation offloading, jumbo frames, and interrupt affinity. RDMA, on the other hand, delivers excellent performance by eliminating interrupt processing, context switching, and copying. However, RDMA's scalability concerns emerge from the number of connections it can support. This problem arises from the limited onboard memory in RNICs [30]. SpanoF [30] and Pangu [11, 22] do multiplexing of multiple storage I/O flows to network connections to address the scalability of RDMA. SpanoF and Pangu multiplex several storage I/O flows to network connections to improve RDMA scalability.

Table 1 specifies the papers considered for this literature survey and the transport protocol they use. All the papers that were specified under RDMA used the multiplexing optimization for scalability.

4.2 Storage and network interface in the initiator

In this section, we explore performance optimization techniques proposed in the literature, focusing on the storage and network interface at the initiator side. The key methods examined are batching of I/O request and hardware offloading implementations.

4.2.1 Batching of I/O request

To overcome the network processing cost, multiple I/Os are batched into a single packet to increase the overall throughput. However, this batching approach can also increase the latency of individual I/O operations, creating a trade-off between throughput and latency. This trade-off can be more easily negotiated if the I/O characteristics (throughput or latency sensitive) are known, allowing the cloud provider to perform the required optimization.

However, the processing pipeline per core attribute makes it difficult to apply characteristic-specific optimizations. For example, if latency-sensitive and throughput-sensitive applications are running on the same core, the requests will be mixed, leading to sub-optimal performance for both types of flows. The gathered literature has explored various techniques to address this challenge. The techniques form a trade-off with performance vs compute utilization.

- **Flushing on latency-sensitive requests:** One approach is to flush the batched requests when a latency-sensitive request is encountered. However, this can have a negative effect on overall performance. I.e, While flushing the batched requests on encountering a latency-sensitive

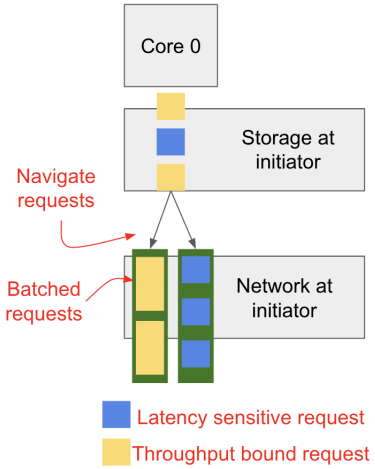

Figure 3: Dedicated processing pipelines for throughput and latency bound requests.

request may help improve the latency for that specific request, it can reduce overall system throughput by disrupting the batching mechanism. This technique trade-off performance for lower compute utilization.

- **Dedicated processing pipeline:** Another approach, as shown in Fig 3, involves building a special processing pipeline for requests that are performance and latency-sensitive. In this case, scheduling becomes challenging, as the system needs to provide high priority to threads handling pipelines of latency-bound requests while simultaneously ensuring that the throughput-bound requests do not starve. Implementation of this technique provides a choice of either lower compute utilization or performance by either scheduling pipeline threads in a round-robin manner, which trades off performance, or by scheduling them concurrently to achieve high performance.

- **Request steering:** Another approach, is to steer requests between cores. I/O processing on each cores can be specified to optimize for either latency or throughput. This technique utilizes more compute to improve performance. However, this approach can lead to slowness due to issues like data cache misses and congestion when accessing queues on separate cores, often favoring throughput-oriented requests over latency-sensitive ones. Frequent request steering may also impact the overall performance.

- **Application steering:** To address request steering challenges, the I/O layer can provide feedback to the scheduler based on a certain policy, allowing the application thread in charge of a particular flow to be shifted to a different core, effectively separating the competing flows. This technique utilizes more compute to improve performance.

By leveraging these techniques, a balance between throughput and latency performance of disaggregated storage can be achieved. Table 2 specifies the papers and the balancing technique they use.

Batching technique	Papers
Flushing	i10 [17]
Dedicated pipeline	i10 [17]
Request steering	Hwang et al. [18], LT-NoT [13], Gu et al. [14]
Application steering	Hwang et al. [18]

Table 2: Batching techniques in the selected papers.

4.2.2 Hardware offloading of storage virtualization services

Storage virtualization in cloud

Apart from providing raw storage, cloud providers often execute additional functions on the stored data, such as compression, erasure coding, and other storage-related operations, depending on the requirements. To avoid such complexities, the cloud providers abstract the storage resources. The hypervisor plays a crucial role in this virtualization process, providing the necessary abstraction and management capabilities to the cloud provider as depicted in Fig 4. The hypervisor abstracts the disaggregated storage and storage functions from the user application (e.g., a virtual machine in the case of Fig 4) and presents them as a block device.

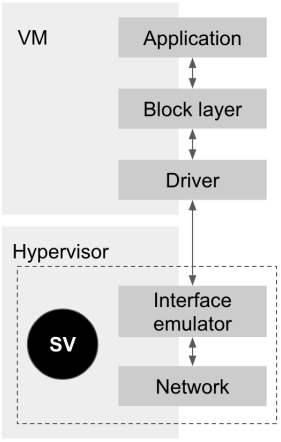

Figure 4: Hypervisor based block storage virtualization in cloud. SV represents storage virtualization components.

This virtualization layer allows the cloud provider to manage the underlying storage infrastructure and associated

services transparently, while the user application interacts with the storage as if it were a local block device. This also enables the cloud provider to optimize and modify the storage resources and services based on the workload requirements, without the user application being aware of the underlying complexities. Cloud providers frequently modify the control logic [31]. One such example is the change of primary storage node in case of replication.

Hardware offloading

The demand for bare metal services, the presence of limited compute availability, and the need for cloud service security and isolation drive cloud providers to seek hardware offloading (represented in Fig 5) of the virtualized service. The offloading should provide good performance as storage functions(compression, erasure coding, etc) are also executed. Apart from the performance requirements of hardware offloading, the mechanism must also be flexible enough to accommodate upgrades to control logic, as specified earlier.

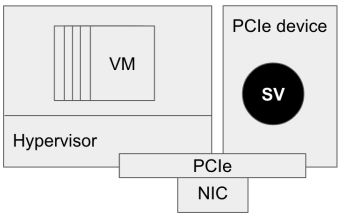

Figure 5: Offloading of storage virtualization. SV represents storage virtualization.

Batching technique	Papers
ARM DPU	DPFS [12], LeapIO [21]
Hybrid	Alibaba [23], SmartDS [31]

Table 3: Hardware offloading methodologies in the selected papers.

The gathered literature has explored various mechanisms to address the requirements for hardware offloading of virtualized services, each with its trade-offs. The initial approach was the usage of ARM-based DPUs (Data Processing Units), which provide flexibility in control logic but suffer from reduced performance compared to hypervisor-based virtualizations due to the lower computational throughput of the ARM cores. The next mechanism explored was offloading to FPGAs (Field-Programmable Gate Arrays), which can deliver high performance but lack the flexibility of the ARM-based DPUs and are also susceptible to bit flips and other integrity issues [23]. To achieve a balance between high performance and flexibility, cloud providers have adopted a hybrid strategy, dividing the processing between the control plane and the data plane. Control plane processing is handled on the CPU, allowing for greater flexibility in control logic updates, while data plane processing is offloaded to dedicated ASICs (Application-Specific Integrated Circuits), providing the necessary performance improvements. This hybrid approach enables cloud providers to maximize the performance of the virtualized services while maintaining the flexibility to adapt the control logic as needed, addressing the trade-offs inherent in the previous mechanisms.

Table 3 specifies the papers and the corresponding hardware offloading mechanism they propose.

4.3 Storage and network interface in the target

In this section, we explore performance optimization techniques proposed in the literature, focusing on the storage and network interface at the target side. The key methods examined are the request processing model and CPU-free target implementation. The request processing model investigates different strategies for handling I/O requests at the target to improve overall performance. Meanwhile, the CPU-free target implementation explores alternate to CPU based processing to enhance performance.

4.3.1 Request processing model in the target

The default request processing model of the target is called processor-sharing. Where the processing of different requests shares the compute as specified in Fig 6 (marked as A). The processor-sharing model has a performance downside because of interrupt handling, context switching, and cache pollution.

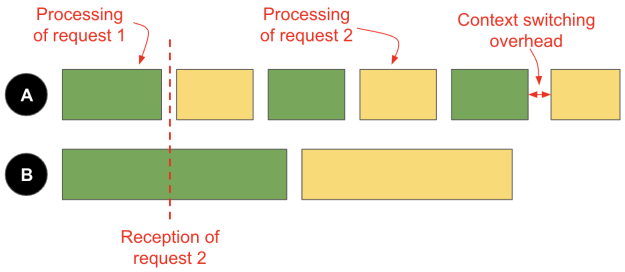

Figure 6: Target request processing model. (A) Processor sharing model. (B) Run to complete model.

An alternative approach to the processor sharing model is the run-to-completion (RTC) model. In the RTC model, the processing of requests follows a first-come, first-served (FCFS) basis, as depicted in Fig 6 (marked as B).

The key advantage of the RTC model is that it eliminates the overhead associated with context switching, thereby reducing the overall processing time. Additionally, this model can enhance data cache performance by maintaining better data locality.

However, the implementation of the RTC process model for flash storage disaggregation targets can have implications on the tail latencies. A larger I/O size request can potentially cause head-of-line (HoL) blocking, where subsequent requests are delayed due to the processing of the preceding request.

To mitigate this issue, the approach taken is to make the request to the flash device asynchronous. By handling the flash requests asynchronously as proposed in ReFlex [20] and Alibaba [23], the RTC model can be applied without the risk of HoL blocking, ensuring that the tail latencies are not adversely affected.

4.3.2 CPU free

The performance gains from Moore's law and Denard scaling have started to diminish, leading to a stagnation in CPU performance [3, 7, 9]. As a result, applications that rely heavily on CPU performance are facing bottleneck issues. To address this challenge, research efforts have been focusing on exploring alternatives to traditional CPUs [28]. Similarly, in the context of flash storage disaggregation, there is a search for CPU-free solutions to overcome the compute bottleneck. The gathered literature presents two CPU-free approaches. The desired solution is expected to not only provide high performance but also be portable, allowing these optimizations to be considered as an opportunity rather than a necessity.

CPU free design	Papers
NIC-PCIe-Flash	Flashnet [29]
NIC-Flash	QuickSAN [4] NVMeoF-FPGA [25]

Table 4: CPU free methodologies in the selected papers.

The first approach, proposed by Flashnet [29]. By design Flashnet enables direct access to the storage device through the RNIC (Remote Direct Memory Access Network Interface Card). This is achieved by co-designing the RNIC, flash controller, and file system stack, which streamlines the storage access process. The advantage of this approach is its portability. I.e, it can be used even without specalized RNICs. Even the implementation in the paper is done in software RDMA. However, this method has a limitation in terms of performance gain due to the PCI (Peripheral Component Interconnect) bottleneck [31]. This approach can be classified as NIC-PCIe-Flash.

The second approach involves embedding the NIC directly into the flash device, as specified in QuickSAN [4] and NVMeoF-FPGA [25]. This approach provides high performance, but it sacrifices portability as it requires specialized hardware and the respective papers haven't specified about it as well. In this case, the solution is classified as NIC-Flash.

4.4 Summary

In this section, we reviewed the literature on different components of the flash storage disaggregation stack, which have explored various mechanisms and their associated trade-offs for performance improvement. Table 5 provides an overview of these mechanisms and their respective trade-offs, as gathered from the literature.

5 Enforcing unified network and storage quality of service

Quality of service (QoS) is a critical requirement for applications running in cloud environments [16]. Applications running in cloud environments often require quality of service (QoS) guarantees to ensure smooth performance output. Service providers must enforce QoS for each application to meet these requirements.

Flash storage disaggregation utilizes multiple resources, such as compute, network, and storage. Performance variability caused by any one of these resources can affect the end-to-end performance of the application. The improved performance of storage media makes the end-to-end performance more sensitive to the performance variability of network and compute resources.

The QoS is typically enforced by schedulers that allocate resources across applications fairly. In a storage disaggregation setup, each resource has its own schedulers, policies, and workloads. For example, the process scheduler takes care of the compute resource by implementing its scheduling policies across processes, separate from the disaggregated storage processing. The interactions of these schedulers and policies across different stages can lead to unexpected end-to-end QoS experiences.

This motivates the need for unified QoS enforcement across resources in flash storage disaggregation. In this survey, the focus will be on the consideration of network resources in QoS, as this is the area where the available literature provides the most insights. This section will answer the **RQ2** about challenges and mechanisms to unified network and storage.

5.1 Challenge of semantic gap in unifying network and storage

One of the main challenges in unifying the network and storage schedulers for end-to-end QoS enforcement is the semantic gap between the two components. As shown in Figure 7, the network schedulers typically operate in terms of network packets, while the storage schedulers work with I/O requests and responses. This difference in the fundamental units of operation creates a semantic gap that needs to be addressed.

From the papers gathered, the enforcement of QoS policies happens at two different scales: per-packet or per-flow. In

Component	Mechanism	Trade-off	Papers
Network	Transport protocol	Performance vs scalability	6
Storage and network interface at the initiator	Batching techniques	Performance vs compute	5
	Hardware offloading	Performance vs flexibility	4
Storage and network interface at the target	Request processing model	Performance vs tail latency	2
	CPU free	Performance vs portability	2

Table 5: Performance optimization techniques and their trade-offs in the components of a disaggregated flash storage architecture, with the number of papers contributed for each mechanism.

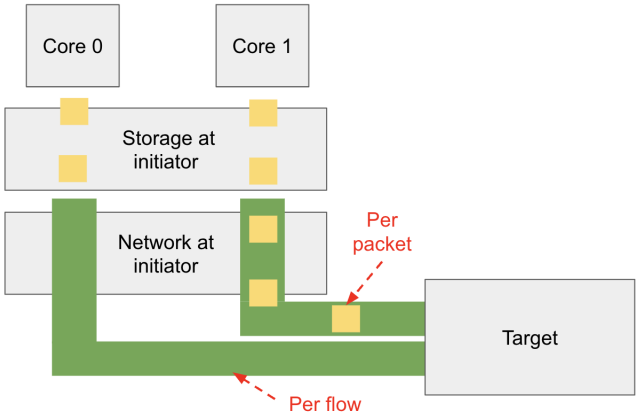

Figure 7: Common semantics for network and storage.

the per-packet approach, the I/O request or response is expected to be encapsulated within a network packet, and the flow is identified by the source and destination tuple. This per-packet, as used by the RackBlox [24], mechanism provides finer-grained control over QoS enforcement, as it allows for a more direct mapping of I/O semantics to the network packet. However, the per-packet approach requires deep packet inspection on the network switches, which can add complexity and processing overhead.

In contrast, the per-flow based mechanisms, as employed by systems like Pulsar [1], IOFlow [27] and Gupta et al. [15], rely on identifying the flow based on higher-level information, such as the source and destination addresses. While this per-flow method is simpler to manage, it may miss some of the granular I/O-level details that the per-packet approach can capture.

By understanding the trade-offs between these approaches and the underlying semantic gap, researchers and cloud providers can develop more effective strategies for unifying the network and storage schedulers to achieve end-to-end QoS enforcement in flash storage disaggregation environments.

5.2 Mechanisms for unifying network and storage

The mechanisms proposed in the gathered literature for unifying the schedulers of the various components of the flash storage disaggregation stack can be classified into two main types: centralized and decentralized. In a centralized setup, a server uses overall information, such as resource availability and application requirements, to construct non-conflicting policies for schedulers. This ensures unification across the different components. However, this centralized approach has limitations in terms of scalability, as it may not be able to handle a large number of applications and schedulers effectively. Examples of literature that have explored this centralized approach include IOFlow [27] and Pulsar [1].

On the other end, there exist multiple decentralized approaches. Gupta et al [15]. suggest modification of congestion control to prioritize certain flows over others, where application I/O flows with low latency or high throughput storage requirements can get similar support from the network. Another example is the RackBlox [24] system, which takes a complementary approach by performing end-switch to flash I/O scheduling. This helps alleviate delays caused in other stages of the disaggregation stack, effectively unifying the different components. The decentralized approaches aim to address the scalability limitations of the centralized approach by distributing the unification and coordination tasks across multiple components or layers of the system.

By exploring both centralized and decentralized mechanisms, the literature has provided a range of strategies for cloud providers to achieve unification and coherence across the various components and requirements of their virtualized services.

6 Conclusion

This survey examined the performance optimization (RQ1) and unified QoS challenges and mechanisms (RQ2) in disaggregated flash storage architectures, which have become increasingly important in cloud environments to accommodate dynamic resource requirements.

RQ1: What are the different mechanisms and their trade-offs for improving the performance of disaggregated flash

storage? The survey presented the optimizations made to the various components of the flash storage disaggregation stack. For the network component, the optimizations are discussed in Section 4.1. For the storage and network interface on the initiator side, the optimizations are covered in Section 4.2. For the storage and network interface on the target side, the optimizations are presented in Section 4.3.

RQ2: What are the challenges and mechanisms for providing a unified network and storage QoS guarantee in disaggregated flash storage? The survey covers the following: Section 5.1 discusses the challenge from semantics difference for unifying storage and network for end-to-end QoS across the disaggregated infrastructure. Section 5.2 then presents the different mechanisms and approaches proposed for unification.

References

[1] ANGEL, S., BALLANI, H., KARAGIANNIS, T., O'SHEA, G., AND THERESKA, E. End-to-end performance isolation through virtual datacenters. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (2014), pp. 233–248.

[2] BARROSO, L. A., HÖLZLE, U., AND RANGANATHAN, P. *The datacenter as a computer: Designing warehouse-scale machines*. Springer Nature, 2019.

[3] BORKAR, S., AND CHIEN, A. A. The future of microprocessors. *Commun. ACM 54*, 5 (may 2011), 67–77.

[4] CAULFIELD, A. M., AND SWANSON, S. Quicksan: a storage area network for fast, distributed, solid state disks. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2013), ISCA '13, Association for Computing Machinery, p. 464–474.

[5] CULLY, B., WIRES, J., MEYER, D., JAMIESON, K., FRASER, K., DEEGAN, T., STODDEN, D., LEFEBVRE, G., FERSTAY, D., AND WARFIELD, A. Strata:{High-Performance} scalable storage on virtualized non-volatile memory. In *12th USENIX Conference on File and Storage Technologies (FAST 14)* (2014), pp. 17–31.

[6] DELL INC. Poweredge pcie express flash ssd. http://www.dell.com/learn/us/en/04/campaigns/poweredge-express-flash, 2015. Accessed: 2024-04-10.

[7] ESMAEILZADEH, H., BLEM, E., ST. AMANT, R., SANKARALINGAM, K., AND BURGER, D. Dark silicon and the end of multicore scaling. *SIGARCH Comput. Archit. News 39*, 3 (jun 2011), 365–376.

[8] FACEBOOK INC. Open compute project. http://www.opencompute.org/projects, 2015. Accessed: 2024-04-10.

[9] FERDMAN, M., ADILEH, A., KOCBERBER, O., VOLOS, S., ALISAFAEE, M., JEVDJIC, D., KAYNAK, C., POPESCU, A. D., AILAMAKI, A., AND FALSAFI, B. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *SIGARCH Comput. Archit. News 40*, 1 (mar 2012), 37–48.

[10] FUSION IO. Atomic series server flash. http://www.fusionio.com/products/atomic-series, 2015. Accessed: 2024-04-10.

[11] GAO, Y., LI, Q., TANG, L., XI, Y., ZHANG, P., PENG, W., LI, B., WU, Y., LIU, S., YAN, L., FENG, F., ZHUANG, Y., LIU, F., LIU, P., LIU, X., WU, Z., WU, J., CAO, Z., TIAN, C., WU, J., ZHU, J., WANG, H., CAI, D., AND WU, J. When cloud storage meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)* (Apr. 2021), USENIX Association, pp. 519–533.

[12] GOOTZEN, P.-J., PFEFFERLE, J., STOICA, R., AND TRIVEDI, A. Dpfs: Dpu-powered file system virtualization. In *Proceedings of the 16th ACM International Conference on Systems and Storage* (New York, NY, USA, 2023), SYSTOR '23, Association for Computing Machinery, p. 1–7.

[13] GU, W., XIE, X., AND DONG, D. Ltnot: Realizing the trade-offs between latency and throughput in nvme over tcp. In *Algorithms and Architectures for Parallel Processing* (Cham, 2023), W. Meng, R. Lu, G. Min, and J. Vaidya, Eds., Springer Nature Switzerland, pp. 412–432.

[14] GU, W., XIE, X., ZHANG, W., AND DONG, D. A transformable nvmeof queue design for better differentiating read and write request processing. In *2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS)* (2023), IEEE, pp. 546–553.

[15] GUPTA, J., KANT, K., PAL, A., AND BISWAS, J. Configuring and coordinating end-to-end qos for emerging storage infrastructure. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems 9*, 1 (2024), 1–32.

[16] GUPTA, M., SINGH, D., AND GUPTA, B. Literature review: Improving the quality of services in cloud computing environment.

[17] HWANG, J., CAI, Q., TANG, A., AND AGARWAL, R. TCP RDMA: CPU-efficient remote storage access with i10. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)* (Santa Clara, CA, Feb. 2020), USENIX Association, pp. 127–140.

[18] HWANG, J., VUPPALAPATI, M., PETER, S., AND AGARWAL, R. Rearchitecting linux storage stack for µs latency and high throughput. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)* (July 2021), USENIX Association, pp. 113–128.

[19] KLIMOVIC, A., KOZYRAKIS, C., THERESKA, E., JOHN, B., AND KUMAR, S. Flash storage disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems* (New York, NY, USA, 2016), EuroSys '16, Association for Computing Machinery.

[20] KLIMOVIC, A., LITZ, H., AND KOZYRAKIS, C. Reflex: Remote flash local flash. *ACM SIGARCH Computer Architecture News 45*, 1 (2017), 345–359.

[21] LI, H., HAO, M., NOVAKOVIC, S., GOGTE, V., GOVINDAN, S., PORTS, D. R. K., ZHANG, I., BIANCHINI, R., GUNAWI, H. S., AND BADAM, A. Leapio: Efficient and portable virtual nvme storage on arm socs. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2020), ASPLOS '20, Association for Computing Machinery, p. 591–605.

[22] LI, Q., XIANG, Q., WANG, Y., SONG, H., WEN, R., YAO, W., DONG, Y., ZHAO, S., HUANG, S., ZHU, Z., WANG, H., LIU, S., CHEN, L., WU, Z., QIU, H., LIU, D., TIAN, G., HAN, C., LIU, S., WU, Y., LUO, Z., SHAO, Y., WU, J., CAO, Z., WU, Z., ZHU, J., WU, J., SHU, J., AND WU, J. More than capacity: Performance-oriented evolution of pangu in alibaba. In *21st USENIX Conference on File and Storage Technologies (FAST 23)* (Santa Clara, CA, Feb. 2023), USENIX Association, pp. 331–346.

[23] MIAO, R., ZHU, L., MA, S., QIAN, K., ZHUANG, S., LI, B., CHENG, S., GAO, J., ZHUANG, Y., ZHANG, P., LIU, R., SHI, C., FU, B., ZHU, J., WU, J., CAI, D., AND LIU, H. H. From luna to solar: the evolutions of the compute-to-storage networks in alibaba cloud. In *Proceedings of the ACM SIGCOMM 2022 Conference* (New York, NY, USA, 2022), SIGCOMM '22, Association for Computing Machinery, p. 753–766.

[24] REIDYS, B., XUE, Y., LI, D., SUKHWANI, B., HWU, W.-M., CHEN, D., ASAAD, S., AND HUANG, J. Rackblox: A software-defined rack-scale storage system with network-storage co-design. In *Proceedings of the 29th Symposium on Operating Systems Principles* (New York, NY, USA, 2023), SOSP '23, Association for Computing Machinery, p. 182–199.

[25] SAKALLEY, D. Using fpgas to accelerate nvme-of based storage networks. *Flash Memory Summit 2017* (2017), 8–11.

[26] SHVACHKO, K., KUANG, H., RADIA, S., AND CHANSLER, R. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)* (2010), Ieee, pp. 1–10.

[27] THERESKA, E., BALLANI, H., O'SHEA, G., KARAGIANNIS, T., ROWSTRON, A., TALPEY, T., BLACK, R., AND ZHU, T. Ioflow: a software-defined storage architecture. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, Association for Computing Machinery, p. 182–196.

[28] TRIVEDI, A., AND BRUNELLA, M. S. Cpu-free computing: A vision with a blueprint. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems* (New York, NY, USA, 2023), HOTOS '23, Association for Computing Machinery, p. 1–14.

[29] TRIVEDI, A., IOANNOU, N., METZLER, B., STUEDI, P., PFEFFERLE, J., KOURTIS, K., KOLTSIDAS, I., AND GROSS, T. R. Flashnet: Flash/network stack co-design. *ACM Trans. Storage 14*, 4 (dec 2018).

[30] XIAO, Y., XIE, X., LI, Q., QIAO, X., AND GU, W. Spanof: A scalable and performant architecture for nvmeof-based storage disaggregation with limited network resources. *Electronics 12*, 13 (2023).

[31] ZHANG, J., HUANG, H., ZHU, L., MA, S., RONG, D., HOU, Y., SUN, M., GU, C., CHENG, P., SHI, C., AND WANG, Z. Smartds: Middle-tier-centric smartnic enabling application-aware message split for disaggregated block storage. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2023), ISCA '23, Association for Computing Machinery.