2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 979-8-3503-3766-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/IPDPS54959.2023.00035

SRC: Mitigate I/O Throughput Degradation in Network Congestion Control of Disaggregated Storage Systems

Danlin Jia[‡], Yiming Xie[*], Li Wang[*], Xiaoqian Zhang[†], Allen Yang[†], Xuebin Yao[‡],
Mahsa Bayati[‡], Pradeep Subedi[‡], Bo Sheng[†] and Ningfang Mi[*]
[*]Department of Electrical and Computer Engineering, Northeastern University, Boston, USA
[†]Department of Computer Science, University of Massachusetts Boston, Boston, USA
[‡]Samsung Semiconductor Inc., San Jose, CA, USA

***Abstract*—The industry has adopted disaggregated storage systems to provide high-quality services for hyper-scale architectures. This infrastructure enables organizations to access storage resources that can be independently managed, configured, and scaled. It is supported by the recent advances of all-flash arrays and NVMe-over-Fabric protocol, enabling remote access to NVMe devices over different network fabrics. A surge of research has been proposed to mitigate network congestion in traditional remote direct memory access protocol (RDMA). However, NVMe-oF raises new challenges in congestion control for disaggregated storage systems.**

In this work, we investigate the performance degradation of the read throughput on storage nodes caused by traditional network congestion control mechanisms. We design a storage-side rate control (SRC) to relieve network congestion while avoiding performance degradation on storage nodes. First, we design an I/O throughput control mechanism in the NVMe driver layer to enable throughput control on storage nodes. Second, we construct a throughput prediction model to learn a mapping function between workload characteristics and I/O throughput. Third, we deploy SRC on storage nodes to cooperate with traditional network congestion control on an NVMe-over-RDMA architecture. Finally, we evaluate SRC with varying workloads, SSD configurations, and network topologies. The experimental results show that SRC achieves significant performance improvement.

***Index Terms*—Disaggregated Storage System, Network Congestion Control, Storage Throughput Prediction and Control, NVMe-oF**

I. Introduction

In recent years, resource disaggregation has been commonly adopted in hyper-scale cloud systems and data centers to enable easy configuration, isolation, and scale-up of resources. Storage disaggregation has drawn significant attention since the emergence of high I/O speed storage media and serial buses, such as NVMe (Non-Volatile Memory express) SSDs with PCIe Gen 4 [1]. The advanced storage solution can provide up to 8 GB/s throughput but requires high CPU utilization to take advantage of it. Disaggregated storage systems decouple the compute and storage nodes, which provides flexibility in system management.

While possessing the desired features of enterprise storage systems, storage disaggregation relies on the support of networking frameworks, NVMe-over-Fabric (NVMe-oF) protocol has been developed to enable the submission of NVMe commands to a remote NVMe SSD through different network fabrics. It serves as a fundamental building block for disaggregated storage systems. There are multiple options for the underlying network fabric for NVMe-oF, such as Ethernet, Fibre Channel, and InfiniBand [2], and this paper considers the RoCE protocol [3], the Remote Direct Memory Access (RDMA) over Converged Ethernet protocol, that guarantees lossless data delivery. More details will be reviewed in Sec. II. When the I/O requests and the corresponding data blocks are transferred between computing nodes and storage hosts, the network performance becomes a critical component in storage operations. A large-scale disaggregated storage system would demand extremely low network latency, especially when processing a heavy I/O workload. On the other hand, network congestion, one of the top challenges in the networking field, would degrade the storage system's performance.

In the literature, there have been solutions for network congestion control in data center networks that yield high throughput and microsecond-level latency [4]–[6]. However, traditional network congestion control mechanisms of RoCE only consider network performance optimization but omit the performance degradation on storage nodes. When network congestion happens, the control mechanism sends a signal to the data sender to limit the data sending rate by buffering the data in the network socket. However, storage devices are unaware of the data throttling happening at network protocols and keep the same request processing rate to retrieve data, e.g., in read requests. Consequently, the high performance of the storage devices suffers from the bottleneck of network congestion control and results in overall performance degradation.

In this work, we propose a storage-side rate control (SRC) mechanism to mitigate the performance degradation of storage devices caused by network congestion. SRC cooperates with the network congestion control mechanism to reduce the data sending rate of storage nodes by precisely controlling read and write throughput. We first design a throughput control mechanism that prioritizes read and write requests when a congestion signal is received. The goal of the throughput control is to mitigate the degradation of aggregated throughput on storage nodes. We further develop a throughput prediction model to learn the mapping between workload characteristics to read and write throughput while using our control mechanism. Finally, SRC utilizes the learned knowledge of the throughput prediction model to help limit the data-sending rate on storage nodes and mitigate performance degradation. As we know,

This work was partially supported by the National Science Foundation Awards CNS-2008072 and the Samsung Semiconductor Inc. Research Grant.

1530-2075/23/$31.00 ©2023 IEEE
DOI 10.1109/IPDPS54959.2023.00035

SRC is the first work that systematically investigates and optimizes storage performance affected by network congestion of disaggregated storage systems. In summary, the contributions of this work are as follows.

- **Investigate the architecture of disaggregated storage systems.** We explore the current network congestion control mechanisms in NVMe-oF with RoCE. We study the current stack of NVMe-oF to understand the bottleneck incurred in the network congestion control mechanism.
- **Design a separate submission queue.** We develop a throughput control mechanism on storage nodes that submits I/O requests to different queues (read or write). A weighted round-robin is used to prioritize read and write requests.
- **Develop a throughput prediction model.** We construct a Random Forest Regression model to predict the read and write throughput of a black-box SSD, given an I/O workload. We further involve the parameters of the throughput control mechanism in the prediction model.
- **Build and evaluate the storage-side control architecture.** We propose a dynamic adjustment algorithm that combines throughput control and prediction to adjust the data sending rate. We finally construct a simulated system by adopting widely adopted network and storage simulators to evaluate our design.

In the remainder of this paper, we introduce the background and our motivation in Sec. II. We present the detailed approach and the algorithms of SRC in Sec. III. Sec. IV evaluates SRC across different workloads and system specifications. The conclusion and future work are presented in Sec. V.

II. Background and Motivation

A. Disaggregated Storage Systems

Disaggregated storage is one type of resource disaggregation infrastructure to enable organizations to access storage resources that can be managed, configured, and scaled independently. In a disaggregated storage system, storage resources can be isolated from compute resources physically and transmit data over the network fabrics. Such a composable infrastructure provides elasticity, scalability, and efficiency in modern data center management. Specifically, administrators can scale out storage resources with the slightest consideration of the characteristics of computing nodes because storage resources are no longer tightly attached to computing resources. Therefore, disaggregated storage systems provide a promising solution for skewed data and over-provisioning [7]. Consequently, the network plays a more critical role in deploying disaggregated storage systems because resource pools rely on network fabrics to transfer massive amounts of data.

1) NVMe-over-Fabric: NVMe-over-Fabric (NVMe-oF) [8] is an extension of the NVMe standard, which enables access to remote NVMe devices over different types of network fabrics. NVMe-oF is commonly adopted in disaggregated storage systems since it provides $< 10\mu s$ latency by eliminating protocol translations in the traditional remote data access protocols, such as from SCSI to NVMe [9]. NVMe-oF exposes the parallel submission queues of NVMe drives to remote nodes transparently via network fabrics so that remote users can submit NVMe commands as sent to their local drives.

There are four significant fabrics in NVMe-oF, i.e., TCP/IP, RDMA-based (RoCE and iWARP), Fiber Channel, and InfiniBand [8]. NVMe-oF using TCP/IP is supported after Linux Kernel 5.0 and is also present in the Storage Performance Development Kit (SPDK). Compared to the traditional TCP/IP protocol, Remote Direct Memory Access (RDMA) supports data transfer without CPU involvement, which can significantly reduce remote data access latency. RoCE requires specific hardware support on network switches. iWARP is an RDMA protocol on TCP/IP that can be deployed on existing software transport frameworks to fit RDMA at the cost of implementation of a complex mix of layers. Consequently, iWARP fails to deliver high throughput, low latency, and low CPU utilization as RoCE [10]. Fiber Channel is a facto standard for enterprise Storage Area Networking (SAN) solutions that support traditional SCSI and NVMe protocols [11]. InfiniBand is primarily proposed to construct I/O Area Network (IAN) and is now mainly used to build a preferred network interconnection technology for GPU servers [2].

2) NVMe-over-RDMA: Our study focuses on congestion control of NVMe-over-RDMA protocols. RDMA is designed natively for direct memory access to remote memory. NVMe-oF extends RDMA to the storage domain by transferring NVMe commands directly over fabrics. Fig. 1 shows the data flow in NVMe over RDMA. We denote storage nodes as *Target*s and the nodes sending I/O requests as *Initiator*s. In Fig. 1, we simplify the architecture with N Initiators and M Targets to a 1:1 system to clearly illustrate the structures of Target and Initiator. The outbound flows represent I/O requests and data sent from Initiators to Targets, while the inbound flows represent data sent from Targets back to Initiators.

Specifically, user applications submit NVMe commands to the transaction queue (TXQ) of RDMA Driver via NVMe-oF Initiator Driver. When RDMA Driver on Targets receives commands, NVMe-oF Target Driver delivers commands to the submission queue (SQ) of the NVMe Driver. The queue depth (QD) parameter controls how many commands can be fetched to SSD flash at once. Given upon data-processing is finished, SSD sends data (for read requests) and completion acknowledgment (for write requests) to the completion queue (CQ) of the NVMe Driver. The entries in CQ are sequentially transferred into TXQ of RDMA Driver on Targets. Ultimately, Initiators forward the received completion entries to user applications. It is worth noting that most inbound flows are data retrieved by read requests, while outbound flows are data sent by write requests majorly.

B. Traditional Network Congestion Control

Network congestion control in RDMA has drawn research concentrations since it was proposed. The existing congestion control mechanisms can be categorized as rate control-based and packet scheduling-based. The rate control-based mechanisms can be further divided into end-to-end and hop-by-

hop [5]. The end-to-end control is a transport layer solution, and the TCP congestion control [12] is one of the most traditional end-to-end control algorithms. On the other hand, the hop-by-hop control or flow control is a link-layer solution. In the case of congestion, where the queue length is more than a threshold, the switch informs its upstream device to stop sending (called Priority Flow Control or PFC [13]) or to slow down (called Quantized Congestion Notification or QCN [14]) to resolve the congestion.

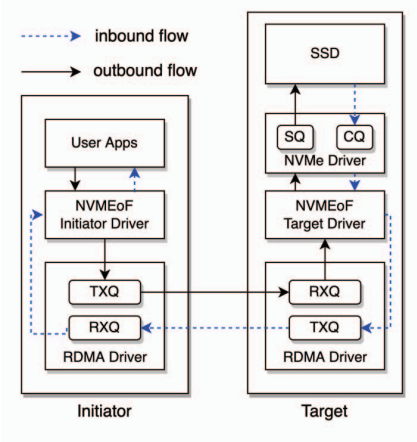


Fig. 1. Architecture of NVMe over RDMA.

Additionally, a packet marking mechanism, named ECN [15], can be used with PFC and QCN. ECN marks a packet by flipping the 1-bit in that packet's header if the queue length at the congestion point exceeds a threshold. Many current congestion control solutions combine PFC, QCN, and ECN to control network traffic on end-to-end and hop-by-hop levels. For example, DCTCP [16] is built based on TCP and ECN. DCQCN [4] is another significant congestion control scheme built upon QCN and can be used with PFC to ensure a lossless network [4]. PCN [5] is a more recent scheme built upon DCQCN and PFC. In this work, we use DCQCN as our network congestion control mechanism.

The extension of RDMA to NVMe-oF raises new challenges in network congestion control. Traditional network congestion control policies regard all nodes in the network topology homogeneously. When a Target/Initiator node receives a quantized congestion notification (QCN), DCQCN stacks data in the data transaction queue (TXQ) to limit the data sending rate. However, an NVMe-oF architecture differentiates Initiators and Targets, which imposes different effects on the read and write throughput.

For example, for congestions caused by outbound flows (i.e., data from Initiators to Targets for write requests), DCQCN buffers the data in the TXQ on Initiators while SSDs on Targets can still serve to write requests at their regular (not reduced) speed. On the other hand, for congestion caused by inbound flows (i.e., data retrieved from Targets to Initiators for read requests), DCQNC attempts to throttle the data sending rate from Targets. In this case, although SSDs continuously process read requests and send data to the TXQ on Targets, DCQNC limits the data sending rate of RDMA drives on Targets to relieve the network congestion. Therefore, the TXQ on Targets becomes the bottleneck of read throughput, which wastes the high throughput of SSDs.

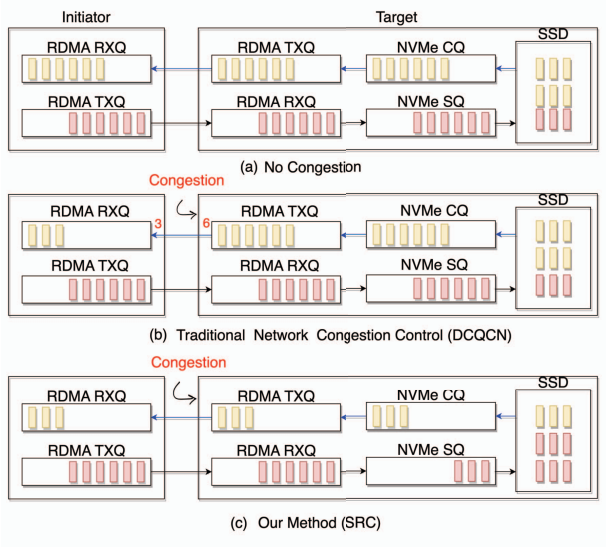


Fig. 2. Motivation examples of data transmission under (a) no congestion, (b) DCQCN, and (c) the proposed SRC. The red bars indicate write requests and the yellow bars show read requests.

Fig. 2-a shows a demo case of no congestion happening in the data transmission of NVMe-oF. Assume that an SSD can process 3 write requests and 6 read requests per time unit, and RDMA can transfer data of 6 requests per time unit when there is no congestion. Under such a situation, the overall throughput (including both read and write requests) is 9 I/Os per time unit. When outflow congestion happens, see Fig. 2-b, DCQCN cuts the data sending rate, for example, by half. Therefore, 3 of 6 read requests are stuck in RDMA TXQ due to network congestion. Although SSDs can continue processing 6 read requests per time unit, RDMA will only send Initiator the data of 3 requests according to DCQCN's control notification. The overall throughput is consequently reduced from 9 to 6.

C. Challenges and Related Work

Fig. 2-b shows that the existing congestion control mechanisms (e.g., DCQCN) rely on the RDMA at Targets to control the sending rate, which ignores the actual processing on the backend SSDs and sacrifices (or wastes) the high throughput of SSDs. The overall performance is thus degraded. An example of read throughput degradation in DCQCN is presented in Sec. IV-D. To address this issue, we propose a new Storage-side Rate Control (SRC) mechanism that moves the control of sending rates from the RDMA to the SSDs by enabling the processing (or throughput) control of read and write requests on the SSD devices. Specifically, SCR attempts to limit the throughput of read requests while increasing the write throughput to avoid the overall throughput degradation of

traditional congestion control mechanisms. We illustrate how SRC works for congestion control in Fig. 2-c. Upon receiving the congestion notification, SRC adjusts the processing rate of read and write requests by prioritizing write requests with the completion of 3 read requests but 6 write requests per time unit. As a result, the sending rate of inbound flows (i.e., data retrieved by read requests) is reduced to half while the overall throughput maintains the same (i.e., 9 per time unit) as the situation with no congestion.

To achieve our goal, we need to address the following three challenges. First, we need a mechanism to separately change the priority of read and write requests when congestion happens. To our knowledge, no such mechanism exists in the present NVMe-oF protocol which can control the individual read and write throughput. We can deploy the priority mechanism on either storage nodes (i.e., active model) or compute nodes (i.e., passive model). We choose the former because existing storage and memory systems have mainly adopted the active model [17].

Second, we need a dynamic priority adjustment mechanism to perform throughput control at run-time. An intuitive method is to monitor the current system status and reactively adjust the request priority. However, such a method suffers from slow response and control delay. We use a throughput prediction model to help make decisions on reducing the data sending rate. A few works have been proposed to predict SSD performance. For example, SSDcheck [18] develops a performance model for black-box SSDs, focusing on the impact of write buffer and garbage collection (GC) on I/O latency. However, this work lacks consideration of workload characteristics and interference between read and write requests. Huang [19] adopts a decision-tree machine-learning model to predict latency and throughput for read and write I/Os. [20] applies multiple machine learning algorithms to predict I/O performance for container-based virtualization and [21] develops a feature selection mechanism to speed up the training of SSD performance models. In this paper, we derive a new throughput prediction model and use machine learning algorithms to learn the selected key features of workload characteristics as input for predicting the individual throughput of read and write requests of SSDs.

Finally, we need to integrate our priority adjustment and throughput prediction systematically. Our throughput prediction model needs to consider workload characteristics and learn the impact of priority adjustment on SSD read/write throughput. We need to design an algorithm to utilize the learned knowledge from the throughput prediction model to adjust read and write priorities effectively.

III. Design of SRC

In this paper, we develop a new Storage-side Rate Control (SRC) solution to mitigate throughput degradation caused by traditional congestion control.

Fig. 3 shows the integration of SRC design into NVMe-oF. SRC is dedicated to assisting traditional network congestion control mechanisms (e.g., DCQCN) to relieve congestion

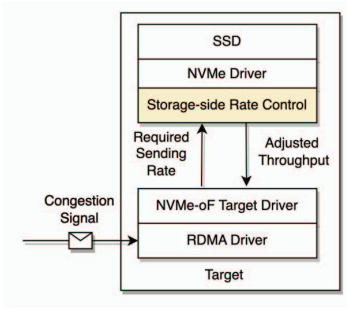


Fig. 3. Architecture of Storage-side Rate Control (SRC).

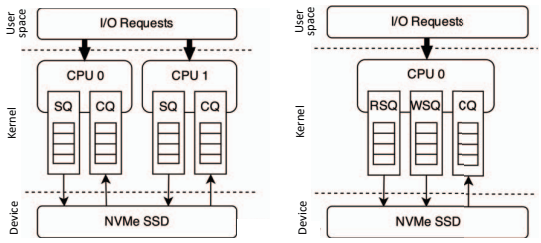
while avoiding performance degradation on Targets. When a congestion notification is received by RDMA Driver, instead of solely relying on RDMA Driver to limit the data sending rate, we transmit the notification to SRC with a required data sending rate calculated by RDMA Driver. Then SRC adjusts the priority of read and write requests and specifically allocates more storage bandwidth to write requests. Therefore, read throughput is reduced by SRC to meet the required data sending rate, while write throughput increases to avoid wasting I/O bandwidth and degrading overall throughput.

A. Separate Submission Queue

We first develop a separate submission queue (SSQ) mechanism on NVMe Driver to enable read and write requests to be submitted to the separate queues based on their I/O types. We then can control the respective throughput of read and write requests by adopting a weighted round-robin algorithm between the read and write submission queues. Fig. 4-a shows the default NVMe queuing mechanism, where the NVMe controller creates multiple submission queues (SQs) and completion queues (CQs) to submit I/O requests and receive completion acknowledgment or retrieved data, respectively. Theoretically, each CPU can spawn as many as 64K SQ and CQ, and multiple SQs can share a single CQ or have their corresponding CQs.

A typical NVMe queuing design is that each CPU has one pair of SQ and CQ to take advantage of parallel processing queues in NVMe SSDs, as shown in Fig. 4-a. When I/O requests submitted from applications in the user space are passed into the kernel space, without any I/O scheduling policy, the NVMe driver converts I/O requests to NVMe commands and enqueues them in a FIFO order. A parameter named queue depth (QD) decides how many NVMe commands can be fetched from a single SQ simultaneously to the storage device. Once QD commands are ready, the NVMe driver sets a register to ring the doorbell to let SSD fetch these commands. After a set of commands is processed, the corresponding completion entries will be enqueued into the CQ along with pointers to either the physical data addresses (for read) or competition acknowledgments (for write).

Fig. 4-b shows our proposed SSQ mechanism to isolate read and write I/O requests and facilitate individual read and



(a) Default (b) SSQ

Fig. 4. Submission Queues in NVMe Driver.

write throughput control. We construct two submission queues (i.e., RSQ and WSQ) for read and write requests, respectively. Meanwhile, we allow these two SQs to share the same CQ. We then use a weighted round-robin (WRR) algorithm provided by NVMe protocol [8] to fetch commands from RSQ and WSQ. Unlike the traditional round-robin, WRR sets a particular weight for each SQ so that the SQ with a larger weight can feed more commands when the storage device starts to fetch I/O commands. Specifically, we can assign different numbers of tokens to RSQ and WSQ to control the read and write weights. When a command is fetched from an SQ, one of this SQ's tokens is taken. If no token exists for an SQ, we reset its token number to continue I/O fetching.

We continue using queue depth (QD) to control the total number of commands that can be fetched simultaneously. In our proposed SSQ mechanism, QD is partitioned into RSQ and WSQ based on the weight ratio (i.e., the ratio of the write weight to the read weight). Therefore, the number of write and read commands that will be processed in parallel on SSDs follows the weight ratio. As SSD firmware grants an equal priority to read and write commands to utilize the internal parallelism, we can dynamically use the weight ratio to control the read and write throughput. We also notice that when SSD starts to fetch, one of the SQs may be empty, i.e., no commands are waiting. In this case, the SSQ mechanism turns to fetch the commands from the other non-empty SQ without manipulating the tokens. This indicates that the I/O workload (e.g., read-to-write request ratio) might also affect the throughput control in our mechanism. We will evaluate the impact of workload characteristics on our design in Sec. IV.

Data consistency is further considered in our proposed SSQ mechanism. Separating I/O submission queues breaks the sequentiality of I/O flows. For instance, a write-after-read operation may retrieve outdated data if RSQ has higher weights than WSQ. To solve this problem, we implement a consistency-checking mechanism to guarantee that dependent flows are executed in their demanded order. If a request R_t attempts to read or write the same logical block address (LBA) as the previous request $R_{t-\tau}$, we then locate where $R_{t-\tau}$ is (if waiting in SQ) and submit R_t to the same SQ. In this way, no matter whether R_t and $R_{t-\tau}$ have the same or different I/O types, our mechanism places them in the same queue to ensure the sequentiality of I/O flows. Meanwhile, to preserve the demanded weight ratio, our mechanism removes one token from the corresponding SQ that holds the same I/O type as R_t when this request command is fetched.

B. Throughput Prediction Model

To precisely control the data sending rate when network congestion happens, we derive a mapping function between the demanded data sending rate and the current system's status. Given a black-box SSD and a sequence of I/O requests, we build a throughput prediction model (TPM) to learn the relationship between SSD throughput and SSQ weight ratio. Formula 1 shows our target function, where we denote the characteristics of an I/O workload as Ch and the SSQ's weight ratio w. We consider both read throughput $TPUT_R$ and write throughput $TPUT_W$ as the prediction outputs because read and write requests usually interfere with each other while sharing the internal storage resources. Since SRC aims to help limit read throughput, we further impose an $w \geq 1$ constraint on Eq. 1.

$$TPUT_{R,W} = F(Ch, w) \tag{1}$$

In order to deepen the understanding of the impact of Ch and w on throughput, we conduct experiments on an NVMe SSD simulator (MQSim [22]) with various workload characteristics (i.e., average inter-arrival time and average request size) and weight ratios. Fig. 5 shows the read and write throughout across weight ratios under different workloads. For simplicity, we fix the read weight to 1 but increase the write weight to increase w. Each row represents workloads with the same average inter-arrival time, e.g., $10 \sim 25\mu$s. Each column represents workloads with the same average request size, e.g., $10 \sim 40$KB. Read and write requests have the same characteristics.

The results in Fig. 5 reveal the following observations. First, when w equals one, read and write requests have the same throughput because read and write requests share the same internal resources (i.e., fetch queues and backend channels) inside NVMe SSD. Second, read throughput decreases and write throughput increases as we increase w under moderate or heavy workloads (i.e., large request size and short inter-arrival time), see plots on the top-right side in Fig. 5. However, we also observe that when the workload becomes light, such as the one at the most left-bottom corner in Fig. 5, the effectiveness of w fades out, i.e., no changes in read/write throughput when w is increasing. This is because the request flow is relatively idle when the average interval time increases. Consequently, the number of commands waiting in RSQ or WSQ becomes low, which limits the possibility of fetching commands successfully under the WRR policy. Thus, WRR becomes ineffective under light workloads and performs the same as round-robin. Finally, we observe that both read and write throughput keep increasing and flattening as the workload's intensity increases. Heavy workloads can fully utilize storage resources and provide the maximum throughput. However, workloads with high write contention can easily saturate I/O bandwidth, which limits the effectiveness of WRR.

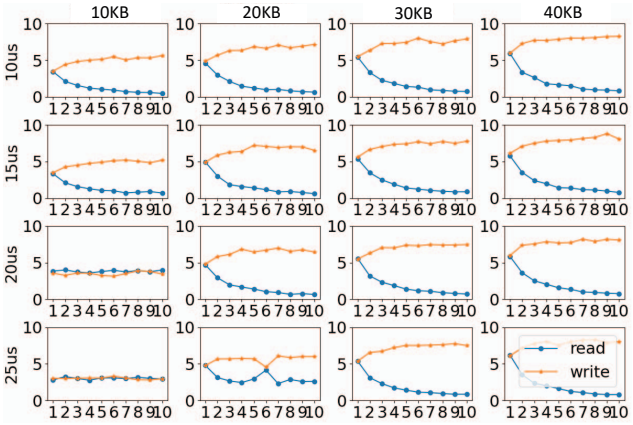


Fig. 5. I/O throughput across various weight ratios under different workloads. X-axis shows the weight ratio while y-axis gives the read/write throughput.

We also investigate the impact of the workload characteristics on throughput, including 1) the ratio of read requests to write requests, 2) the squared coefficient of variation (SCV) of request size and inter-arrival time for read and write requests, and 3) the arrival flow speed, defined as the data size arrived per time unit for read and write requests. We implement a feature extractor to analyze the workload and get the corresponding characteristics. All these extracted characteristics are included in Ch in Eq. 1.

To build our prediction model, we need to collect data samples under different workload characteristics and corresponding performance metrics (i.e., read and write throughput). We first run extensive experiments with various workloads and weight ratios to collect training data samples. We use five statistical machine learning algorithms to learn the mapping function in formula 1, including Linear Regression, Polynomial Regression, K-Nearest Neighbor, Decision Tree Regression, and Random Forest Regression. Given the features extracted from the workloads and the weight ratio, these algorithms attempt to calculate a predictive function that maps the input to the demanded output with the least prediction errors. We use the coefficient of determination to qualify the prediction accuracy of five regression algorithms.

Table I shows the corresponding regression accuracy, where Random Forest Regression achieves the highest prediction accuracy, e.g., 0.94. Thus, we adopt the Random Forest Regression algorithm in our throughput prediction model. Random Forest Regression is an extension of Decision Tree Regression. While the latter builds up a tree-like structure where each node has splitting criteria over a feature, the former constructs multiple tree-like structures to deliver a so-called ensemble learning method that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model. We apply the Breiman feature importance equation [23] to calculate the weights of each feature in Ch and find that the read and write arrival flow speed plays the most crucial role in TPM with a weight of 0.39 out of 1.

TABLE I
REGRESSION ACCURACY.

Model	Accuracy
Linear Regression	0.77
Polynomial Regression	0.74
K-Nearest Neighbor	0.86
Decision Tree Regression	0.89
Random Forest Regression	0.94

C. Systematic Design of SRC

Fig. 6 shows how the above SSQ and TPM are integrated into the proposed SRC to control read and write throughput. First, we develop a new separate submission queue (SSQ) mechanism on NVMe Driver, which adds read and write requests to their corresponding SQ when the NVMe-oF Target Driver delivers them. When NVMe SSD starts fetching I/O commands, the SSQ mechanism uses a weighted round-robin algorithm to fetch read and write commands from RSQ and WSQ, respectively. As described in Sec. III-A, tokens are given to each SQ based on the predefined read and write weights. The solid red lines in Fig. 6 show the corresponding request flows.

When a control notification is received from Network Congestion Control, our throughput prediction model (TPM) component learns the weights of RSQ and WSQ to obtain the demanded data sending rate. The TPM component then uses the learned mapping function to get the new weights and sends them to the SSQ mechanism to adjust the actual SSQ weights. Workload Monitor is also implemented to profile the workload characteristics in a user-specific time window δ (e.g., 10 ms) and extract selected features for learning. Specifically, we use a prediction window to catch all request flows within the time window. We learn the mapping function between throughput and write weight ratio using workload characteristics collected in the previous prediction window. Given the required data sending rate by the network congestion control mechanism, SRC chooses the write weight ratio with an absolute minimum distance to adjust SSQ's weights. The dashed black lines in Fig. 6 show the corresponding control flows.

Algorithm 1 further shows the dynamic sending rate control approach in SRC. The inputs of Alg. 1 include a set of congestion events, the running workload, and the prediction window. The congestion events can be either pause or retrieval events. The "DynamicAdjustment" procedure triggers the "PredictWeightRatio" procedure to get the proper weight ratio w_i for each congestion event (i.e., e_i). Specifically, e_i contains the information of a desired data sending rate r and the current timestamp t. The "DynamicAdjustment" procedure collects workload characteristics within the previous time interval $[t-\delta, t]$ and passes the workload characteristics with the desired data sending rate to the "PredictWeightRatio" procedure. After a proper w_i is returned, SRC adjusts SSQ's weights accordingly. It is worth noticing that the returned w_i

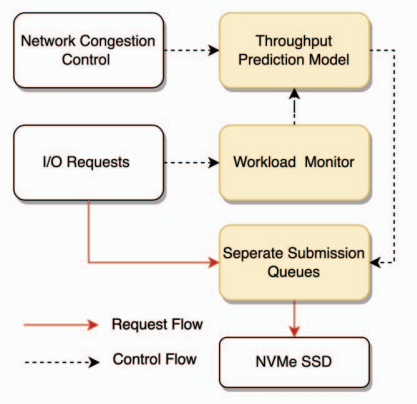


Fig. 6. Workflow of Storage-side Rate Control (SRC).

might be larger (resp. smaller) than the current weight ratio in SSQ if pause (resp. retrieval) events happen.

The inputs of "PredictWeightRatio" include the demanded data sending rate sent from the network congestion control mechanism and the workload characteristics extracted by the workload monitor. The output of Algorithm 1 is the weight ratio that SSQ should adjust to. In the "PredictWeightRatio" procedure, we first initialize a set of variables (lines 11 to 13). Particularly, the weight ratios w and w^* are initialized as 1, i.e., read and write SQs have the same priority. We then use TPM to predict the read throughput that SSD can provide under a given workload Ch and weight ratio $w = 1$ (line 14). If the predicted read throughput is already lower than the data sending rate, we directly return the new weight ratio w^* as 1 (lines 15 to 17). Otherwise, we start to search for a proper weight ratio by increasing w so that the predicted read throughput can be decreased. Since our goal is to find the predicted read throughput closest to the demanded data sending rate, we use min_dis to track the minimum absolute distance between them and update w^* to the corresponding w (lines 24 to 27). We continuously increase the weight ratio until the predicted read throughput is converging, i.e., under the relative distance between the throughputs (i.e., pre_tput and cur_tput) under the previous and current weight ratios is less than a threshold τ, e.g., 10% (line 28). After going through all possible weight ratios, we return w^* that has the smallest absolute distance from the predicted read throughput to the demanded data sending rate.

Finally, our TPM is a discrete prediction model, while the demanded data sending rate can be continuous. Although Algorithm 1 delivers the new weights with a minor prediction error, SRC may still suffer from the variation (i.e., discrete to continuous) in matching the demanded data sending rate. However, we also notice that the variation has a limited impact on network congestion control because traditional network congestion control algorithms intuitively decide the demanded data sending rate and use a feedback control flow to approach the congestion control. For example, when a congestion signal is received, DCQCN cuts the data sending rate by 75% and

Algorithm 1 SRC Dynamic Weight Adjustment

Input: Congestion events E. I/O workload *WL*. Prediction time window δ.
Result: A series of weight ratios *WR*.

```
1  Procedure DynamicAdjustment(E, WL, δ)
2      for e_i in E do
3          r ← demanded data sending rate in e_i
4          t ← timestamp of e_i
5          Ch' ← Ch in time interval [t − δ, t]
6          w_i ← PredictWeightRatio(r, Ch')
7          put w_i in WR and adjust SSQ's weights
8      end
9      return WR
10 Procedure PredictWeightRatio(r, Ch)
11     w ← 1, w* ← 1
12     pre_tput ← 0, cur_tput ← 0
13     min_dis ← INF_MAX
14     TPUT_{R,W} ← TPM(Ch, w)
15     if TPUT_R < r then
16         return w*
17     endif
18     min_dis ← |TPUT_R − r|
19     do
20         w ← w + 1
21         pre_tput ← TPUT_R
22         TPUT_{R,W} ← TPM(Ch, w)
23         dis ← |TPUT_R − r|
24         if min_dis > dis then
25             min_dis ← dis
26             w* ← w
27         endif
28         cur_tput ← TPUT_R
29     while |pre_tput − cur_tput| / pre_tput ≥ τ;
30     return w*
```

repeats the same procedure until the congestion is mitigated. Therefore, in SRC, a slight offset of the demanded data sending rate does not make non-negligible effects on congestion control and thus can be ignored.

IV. Evaluation

A. Testbed

We construct our experiments on a simulated disaggregated storage system that systematically integrates a network simulator (i.e., NS3-RDMA [24]) with a storage simulator (i.e., MQSim [22]). NS3 has been widely used to evaluate rate control-based schemes, e.g., DCQCN, TIMELY, and PCN. We build up a Clos network upon NS3, a multistage switching architecture involving two layers of network switches and an array of Initiators and Targets. Specifically, we have four pods, and each pod consists of two leaf switches, four top-of-rack (ToR) switches, and 64 nodes. We set the link capability as 40Gbps, and specify the link delay as 1 μs. We denote half of the nodes (i.e., 128 nodes) as Initiators and the rest (i.e., 128 nodes) as Targets. We choose DCQCN as the network congestion mechanism in our evaluation. We further launch multiple SSD instances on each Target using MQSim to simulate a flash array. MQSim simulates end-to-end latency of SSD, providing $2.9\%-4.9\%$ error rates for read and write [22],

respectively. We also evaluate our design on different types of SSDs as specified in Table II.

TABLE II
MQSIM PARAMETER CONFIGURATION

	SSD-A	SSD-B	SSD-C
Queue Depth	128	512	512
Write Cache	256MB	256MB	512 MB
CMT	2MB	2MB	8 MB
Page Capacity	16KB	16KB	8 KB
Read Latency	75 μs	2 μs	30 μs
Write Latency	300 μs	100 μs	200 μs

We generate two types of workload traces in our evaluation. The first type of workload is called micro traces, where the inter-arrival time and request sizes are drawn from exponential distributions. The other type of workload is generated based-on real storage repositories, e.g., SNIA IOTTA Repository [25], denoted as synthetic traces. Specifically, we first extract the statistics, e.g., the average, SCV, skewness, and auto-correlation of inter-arrival time and request size, of real storage traces, such as Fujitsu VDI traces [26] and Tencent CBS traces [27]. Then we use the Q Toolbox [28] to create an MMPP (Markov-modulated Poisson Process), a two-phase MAP process that can be used to generate inter-arrival time and request size with bursts for the synthetic traces.

B. Evaluation Method

We define two metrics, i.e., pause number and aggregated throughput, to evaluate the network congestion control and system performance. The pause number is the number of congestion signals received by Targets system-widely, and a large pause number represents heavy network congestion. Since our goal is to mitigate the performance degradation of Targets, our evaluation focuses on congestion caused by read requests. When congestion happens, we expect our SRC to reduce read throughput to the demanded data sending rate and meanwhile increase write throughput. Therefore, we use aggregated throughput (i.e., the summation of read throughput received at Initiators and write throughput obtained at Targets, see Fig. 2) to represent the SSD performance. Our baseline is executing a workload with DCQCN only. Another round of experiments on the same workload but with SRC activated (denoted DCQCN-SRC) is performed to compare with DCQCN-only. We omit the start (first 10%) and tail (last 10%) of experimental results across the timeline to avoid inaccuracy in the warmup and wrapup stages.

C. TPM Accuracy

We first evaluate the accuracy of our TPM across different workload characteristics of the synthetic workloads. We assume the specification is unknown to us for any given SSD and thus train TPM with extensive training samples. To ensure TPM can provide high enough accuracy for SRC to adjust read throughput, we produce cross-validation with micro and synthetic workloads on SSD-A in Table II. In cross-validation, we shuffle the whole data set and use the partial data set for training and the rest for validation to avoid using the same data set for training and validation. The accuracy shown in Table I is collected using micro traces only, i.e., 60% for training and the rest for validation. We observe that the accuracy is as high as 0.94 under Random Forest Regression.

Now, we extend our TPM evaluation to more realistic workloads. We first classify the synthetic workloads into four categories according to their spatial and temporal statistics. Each data subset represents one combination with low or high variation (e.g., SCV) in request size and inter-arrival time, see Table III. Then we use the synthetic traces in a data subset for validation and the remaining synthetic traces and all micro traces for training. The accuracy under Random Forest Regression is shown in Table III. We find that a reliable prediction accuracy (i.e., 0.89 - 0.98) is achieved by our TPM for different types of workloads. Similar accuracy is also obtained for the other two types of SSDs in Table II.

TABLE III
CROSS-VALIDATION ACCURACY USING RANDOM FOREST REGRESSION.

Data Subset	Accuracy
low size SCV + low inter-arrival SCV	0.89
low size SCV + high inter-arrival SCV	0.98
high size SCV + low inter-arrival SCV	0.96
high size SCV + high inter-arrival SCV	0.95

D. Throughput Control

The main goal of our SRC design is to control the read throughput to the demanded data sending rate while maintaining high aggregated I/O throughput. We first use a synthetic workload generated based on the characteristics of the Fujitsu VDI trace that has more intensive read requests than write requests. The average request sizes for read and write are 44 KB and 23 KB, respectively. The average inter-arrival time for read and write is similar, around $10\mu s$. The read traffic load [1] going through the network is around 35.2 Gbps. We also consider one Initiator and two Targets, where each Target processes $5,000$ read and $5,000$ write requests. We have three types of SSDs, as shown in Table II. Here, we only present the results of SSD-A as an example for the evaluation.

Fig. 7 shows the read, write, and aggregated throughput of DCQCN with and without SRC. We also record the pause number received per millisecond in Fig. 8. The read throughput, see blue bars in Fig. 7, is reduced from around 5 Gpbs to about 1 Gbps at the beginning stage because heavy network congestion happens. Correspondingly, we can find a dramatic boost in pause number from Fig. 8. We also observe that such a drop in read throughput is consistent under both DCQCN-only and DCQCN-SRC, which indicates that our SRC successfully controls the read throughput to the demand data sending rate. At the same time, the aggregated throughput (see the blue+orange bars in Fig. **??**) under DCQCN-only dramatically drops from 7.5 Gbps to 2.5 Gbps. As discussed

[1]The traffic load is calculated by dividing the average request size by the average inter-arrival time.

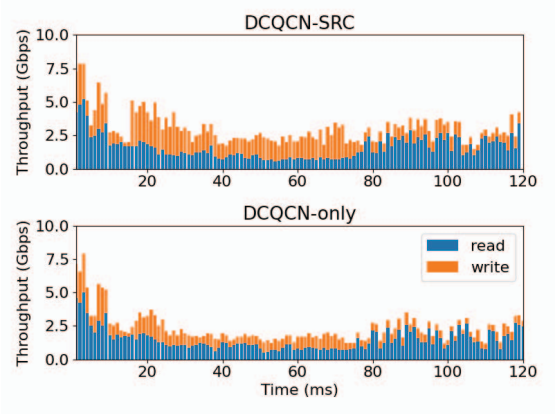
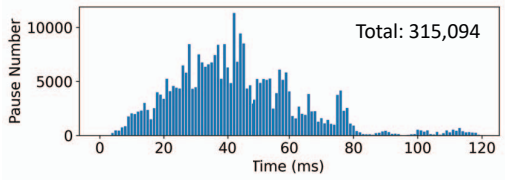
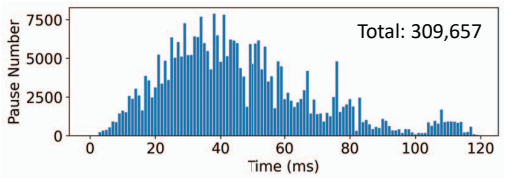


Fig. 7. Runtime throughput under DCQCN-only and DCQCN-SRC. The blue bars and orange bars show the read and write throughput, respectively. The entire bars give the aggregated throughput.



(a) DCQCN-only



(b) DCQCN-SRC

Fig. 8. Pause number under DCQCN-only and DCQCN-SRC.

in Sec. II-B, traditional DCQCN relies on TXQ on the RDMA driver to control the read throughput, which ignores the SSD's processing throughput. In contrast, SRC controls the read throughput at the storage side, which allows it to further control (i.e., increase) the write throughput. As a result, the aggregated throughput under DCQCN-SRC is just slightly decreased. At around 80ms, the congestion is relieved, and the read throughput goes up to about 2 Gbps. However, the read throughput dominates the aggregated throughput since the write intensity is much lighter (only half of) than the read intensity in this workload.

E. Dynamic Control of SRC

In our SRC design, we dynamically adjust the weights for RSQ and WSQ when a congestion event happens, as shown in Alg. 1. A delay in controlling the data-sending rate upon receiving the control notification is unavoidable. To investigate how the dynamic control takes effect, we generate a sequence of synthetic network congestion events with different demanded data sending rates to evaluate SRC's convergence speed in the throughput adjustment on SSD-B. Fig. 9 shows the runtime adjustment of read and write throughput when the synthetic congestion events happen. Each vertical dashed line represents receiving a congestion signal. We use SSD-B as our storage device

In Fig. 9, the first event (pause) happens at around 60 ms, where the demanded data sending rate is set as 6 Gbps. The "PredictWeightRatio" procedure is triggered and returns the new weight ratio 3. SRC then adjusts the corresponding weights for RSQ and WSQ, which makes the read throughput drop between 5 Gbps and 7.5 Gbps after 7 ms. Later at 100 ms, another pause signal is received with a 3 Gbps demanded data sending rate, indicating that network congestion still exists. SRC sets the weight ratio to 5, and the read throughput reaches around 2.5 Gbps after 10 ms. After congestion is relieved, SRC receives a retrieval signal, which requests to raise the data sending rate to 6 Gbps. It takes SRC around 12 ms to converge. Finally, another retrieval signal is received, and SRC controls the read throughput back to 10 Gbps after around 8 ms. We also generate a long trace with hundreds of connection events. The average control delay is around 7.3 ms. We state that such a control delay has a limited impact on network congestion because a typical latency of network flows with tens of KB data is tens of milliseconds.

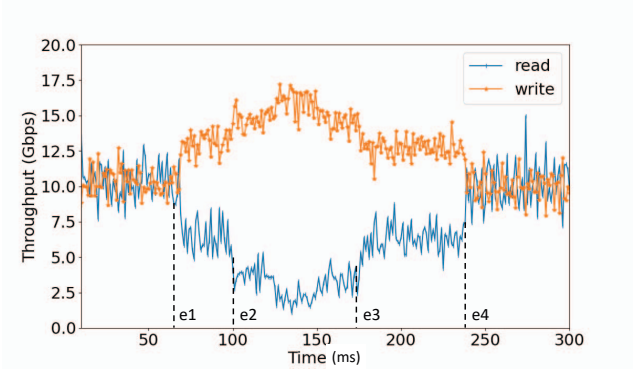


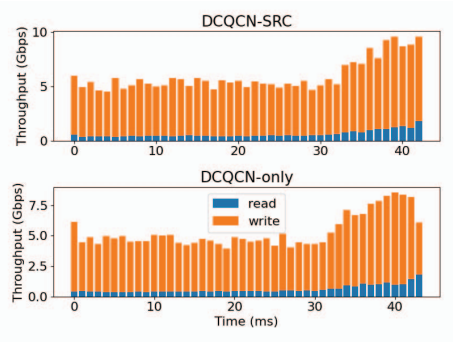
Fig. 9. Dynamic throughput adjustment under SRC.

F. Sensitivity Analysis

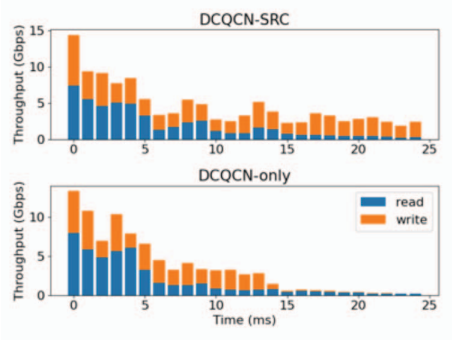
1) Workload Intensity: To investigate the effect of SRC on different types of workloads, we generate three micro workloads by adjusting the average arrival rate and request size but keeping the same network topology. That is, there are one Initiator and two Targets, and each Target has multiple SSD-A devices. Specifically, we increase both the average request size (44 KB) and average arrival rate (100 /ms) to mimic an intensive workload that transfers large data flows frequently. On the opposite, we generate a light workload by reducing the average request size to 22 KB and the average arrival rate to 60 /ms. We also define a moderate workload with an average request size of 32 KB and an average arrival rate of 80 /ms.

Fig. 10 shows the throughput (read, write, and aggregated) across time of DCQCN-only and DCQCS-SRC under these

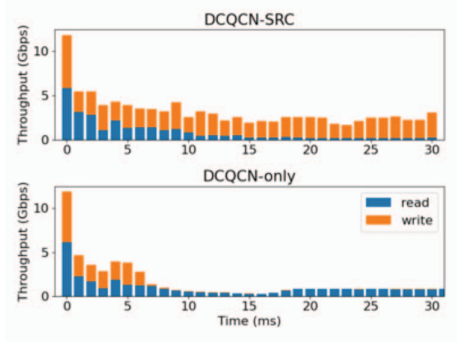
three workloads. We find no visible difference in the throughput between DCQCS-SRC and DCQCN-only when we have the light workload, see Fig. 10-a. This is because there are a limited number of requests in both network and SSD submission queues. On the other hand, compared with DCQCN-only, DCQCS-SRC obtains a significant increase in write throughput under moderate and heavy workloads, particularly when a congestion event happens (about 14 ms in Fig. 10-b and 7 ms in Fig. 10-c). Meanwhile, SRC successfully controls the read throughput to the demanded data sending rate that DCQCN notifies. We can observe that the read throughput under DCQCN-SRC aligns well with that under DCQCS-only.



(a) Light



(b) Moderate



(c) Heavy

Fig. 10. Workload intensity investigation under DCQCN-only and DCQCN-SRC.

2) In-cast Ratio: In-cast ratio, defined as the ratio of Targets to Initiators, is an essential factor in modern congestion control algorithms. Thus, we further conduct experiments to investigate the impact of the in-cast ratio on the control performance. In this set of experiments, we keep the same network traffic load (i.e., around 38 Gbps) but change the in-cast ratios. Note that a larger in-cast ratio indicates relatively more Targets. Table IV summarizes the aggregated throughput under DCQCN with and without SRC and the corresponding improvement of DCQCN-SRC over DCQCN-only.

We first observe that DCQCN-SRC improves the aggregated throughput by 33% when the in-cast ratio is 2:1. This is because when there are fewer Targets, each Target receives more requests to queue in the SQs. Therefore, SSD can successfully fetch commands in the weighted round-robin mode. Whereas, such an improvement becomes less when the in-cast ratio is increased from 2:1 to 4:1. We observe that a large in-cast ratio (e.g., 4:1) makes the total traffic load be distributed among more Targets so that each Target's workload intensity decreases. In this case, the weighted round-robin downgrades to the round-robin, as discussed in Sec. III-B, and thus our SRC cannot take effect. However, this case can be addressed by designing a data distribution mechanism that attempts to find a data distribution policy with the lowest network traffic [29]. We further increase the number of Initiators to change the in-cast ratio from 4:1 to 4:4. As shown in the table, DCQCN-SRC performs similarly to DCQCN-Only. This is because, given the same total traffic load, more Initiators can relieve network congestion and then avoid I/O throughput degradation under DCQCN-Only.

TABLE IV
IN-CAST RATIO ANALYSIS.

In-cast Ratio	DCQCN-SRC	DCQCN-Only	Improvement
2:1	3.2 Gbps	2.4 Gbps	33%
3:1	5.4 Gbps	4.6 Gbps	17%
4:1	8.2 Gbps	7.8 Gbps	5%
4:4	9.5 Gbps	9.2 Gbps	3%

V. CONCLUSION

Performance degradation of storage nodes caused by traditional network congestion control mechanisms is evident in disaggregated storage systems. In this work, we propose a storage-side rate control mechanism to help mitigate performance degradation when network congestion happens. We develop a throughput control mechanism using a weighted round-robin separate submission queue. We further develop a throughput prediction model to learn the mapping between workload characteristics to read and write throughput, considering our throughput control mechanism. Finally, we develop a dynamic adjustment system to utilize the throughput prediction model and separate submission queue to help limit the data sending rate on storage nodes. In the future, we plan to construct a small-scale disaggregated storage system, extend our design as an I/O scheduler in the block layer on Targets,

and explore the direction of integrating our design in SPDK, an NVMe driver in user space.

REFERENCES

[1] A. Verma and P. K. Dahiya, "Pcie bus: A state-of-the-art-review," *IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)*, vol. 7, pp. 24–28, 2017.

[2] G. Pfister, "Aspects of the infiniband architecture," *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pp. 369–371, 2001.

[3] IBTA, "RDMA over Converged Ethernet (RoCE)," https://cw.infinibandta.org/document/dl/7781, 2014.

[4] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*.

[5] W. Cheng, K. Qian, W. Jiang, T. Zhang, and F. Ren, "Re-architecting congestion management in lossless ethernet," in *NSDI*, 2020.

[6] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: a receiver-driven low-latency transport protocol using network priorities," *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018.

[7] Z. Guz, H. Li, A. Shayesteh, and V. Balakrishnan, "Nvme-over-fabrics performance characterization and the path to low-overhead flash disaggregation," in *Proceedings of the 10th ACM International Systems and Storage Conference*, 2017, pp. 1–9.

[8] NVM Express., "NVM Express over Fabric 1.0." http://www.nvmexpress.org/, 2016.

[9] M. Vuppalapati, J. Miron, R. Agarwal, D. Truong, A. Motivala, and T. Cruanes, "Building an elastic query engine on disaggregated storage," in *NSDI 20*, 2020.

[10] Mellanox, "RoCE vs. iWARP Competitive Analysis," https://network.nvidia.com/sites/default/files/pdf/whitepapers/WP_RoCE_vs_iWARP.pdf, 2017.

[11] Mellanox, "Comparative I/O Analysis," https://network.nvidia.com/related-docs/whitepapers/IOcompare_WP_140.pdf, 2005.

[12] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling tcp throughput: a simple model and its empirical validation," in *SIGCOMM '98*, 1998.

[13] IEEE 802.1Qbb, "Priority-based Flow Control," https://1.ieee802.org/dcb/802-1qbb/.

[14] IEEE 802.1Qau, "Congestion Notification," https://1.ieee802.org/dcb/802-1qau/.

[15] S. Floyd, "Tcp and explicit congestion notification," *Comput. Commun. Rev.*, vol. 24, pp. 8–23, 1994.

[16] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *SIGCOMM '10*, 2010.

[17] S.-Y. Tsai, Y. Shan, and Y. Zhang, "Disaggregating persistent memory and controlling them remotely: An exploration of passive disaggregated key-value stores," in *USENIX ATC' 20*, 2020.

[18] J. Kim, J. Kim, P. Park, J. Kim, and J. Kim, "Ssd performance modeling using bottleneck analysis," *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 80–83, 2018.

[19] H. H. Huang, S. Li, A. Szalay, and A. Terzis, "Performance modeling and analysis of flash-based storage devices," in *MSST'11*. IEEE, 2011, pp. 1–11.

[20] J.-E. Dartois, J. Boukhobza, A. Knefati, and O. Barais, "Investigating machine learning algorithms for modeling ssd i/o performance for container-based virtualization," *IEEE transactions on cloud computing*, vol. 9, no. 3, pp. 1103–1116, 2019.

[21] M. Tarihi, S. Azadvar, A. Tavakkol, H. Asadi, and H. Sarbazi-Azad, "Quick generation of ssd performance models using machine learning," *IEEE Transactions on Emerging Topics in Computing*, 2021.

[22] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu, "Mqsim: A framework for enabling realistic studies of modern multi-queue ssd devices," in *FAST*, 2018.

[23] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[24] Y. Zhu, M. Ghobadi, V. Misra, and J. Padhye, "Ecn or delay: Lessons learnt from analysis of dcqcn and timely," in *CoNEXT'16*, September 2016. [Online]. Available: https://www.microsoft.com/en-us/research/publication/ecn-delay-lessons-learnt-analysis-dcqcn-timely/

[25] SNIA IOTTA, http://iotta.snia.org.

[26] SNIA, "Block I/O Traces," http://iotta.snia.org/tracetypes/3, 2017.

[27] SNIA, "Tencent Block Storage," http://iotta.snia.org/traces/27917, 2020.

[28] G. Casale, E. Z. Zhang, and E. Smirni, "Kpc-toolbox: Simple yet effective trace fitting using markovian arrival processes," *Fifth International Conference on Quantitative Evaluation of Systems*, 2008.

[29] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. T. Morris, "Efficient replica maintenance for distributed storage systems." in *NSDI 2016*, vol. 6.