2021 IEEE International Performance, Computing, and Communications Conference (IPCCC) | 978-1-6654-4331-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/IPCCC51483.2021.9679397

SNIS: Storage-Network Iterative Simulation for Disaggregated Storage Systems

Danlin Jia*, Tengpeng Li†, Xiaoqian Zhang†, Li Wang*, Mahsa Bayati‡, Ron Lee‡, 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*—In recent years, designs and optimizations on disaggregated storage systems supported by cutting-edge storage and network techniques emerge dramatically. However, conducting experiments in a disaggregated architecture is often expensive. A comprehensive modeling system of disaggregated storage systems is indispensable for researchers to construct fast and reliable experiments. Modeling and evaluating the performance of disaggregated storage systems is a challenge for the following reasons. First, the performance of a disaggregated storage system depends on network protocols and storage solutions jointly. Second, the available trace datasets for generating the workload may not suffice the need for the simulation, as they are collected without considering the integration of network delay and storage processing time. This work proposes a storage-network iterative simulation (SNIS) for disaggregated storage systems by considering the issues above. Our simulation methodology integrates storage and network simulations to model the end-to-end performance of disaggregated storage systems and conducts multiple rounds of simulations to update arrival times of read/write requests. The evaluation results show that SNIS can converge to a relatively stable state after a certain number of iterations.**

I. Introduction

Efficient infrastructure is critically essential for large-scale enterprise storage systems (i.e., hyperscaler [1] and cloud storage [2]) to provide a high quality of service. One promising direction of solutions is to develop a disaggregated storage system where storage drives are physically apart from compute nodes. In such a system, compute and storage resources can be scaled independently for different needs, and resource management becomes more flexible. With other technologies such as software-defined storage and hyper-converged infrastructure, the disaggregation architecture represents a form of scale-out storage solution. All-flash arrays have emerged in data centers recently, yielding superior performance to traditional storage systems. Consequently, storage network protocols have also drawn much attention in the industry. The prevalent storage technique, Non-Volatile Memory Express (NVMe), has spawned its network protocol standard NVMe-oF which can be implemented on multiple underlying network protocols such as Ethernet, Fibre Channel, RoCE [3], InfiniBand [4] or TCP/IP. With all these advances, disaggregated storage infrastructure is becoming an important research field for both academia and industry.

This work was partially supported by the National Science Foundation Career Award CNS-1452751, the National Science Foundation Awards CNS-2008072, and the Samsung Semiconductor Inc. Research Grant.

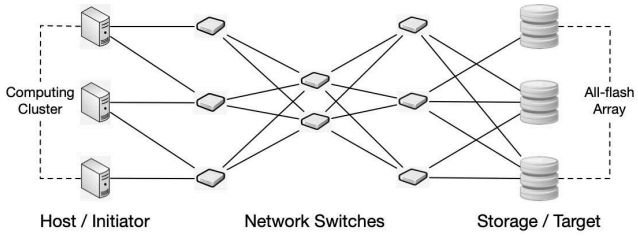


Fig. 1. Architecture of a disaggregated storage system.

As a large-scale system, conducting experiments in a disaggregated architecture is often expensive, and simulation is a well-accepted alternative. While simulators for both storage and network systems have been well developed, there is no integrated simulation environment that can holistically evaluate our targeted disaggregated storage systems. The lack of a comprehensive simulator connecting storage and network subsystems cannot be compensated by individually running storage and network simulation. For evaluation of a storage system, the most primary performance metric is I/O response time. In the disaggregated setting, the I/O response time consists of two overheads: storage overhead and network overhead. It is challenging to measure these two overheads and aggregate them for each request by simulating storage and network separately. The correlation and dependence between storage and network simulation hinder a straightforward integration for evaluating disaggregated storage systems. In addition, the available trace datasets for generating the workload may not suffice the need for the simulation. The legacy storage system traces are often collected at a central point with the timestamp of the request arrival. The network overhead is not considered, and some other networking factors usually are not detailed.

In this paper, we attempt to examine an iterative storage-network simulation approach named as SNIS to evaluate disaggregated storage systems. We integrate the existing storage and network simulators (i.e., an NS3 simulator for RDMA (NS3-RDMA [5]) and an MQSim [6] simulator for SSD) and conduct an iterative multi-round simulation process. We construct extensive experiments on traces with different characteristics and observe that our simulation method converges for all types of workloads. Our results show that for various workloads, SNIS leads to a relatively stable state after a certain number of iterations. In the remainder of this paper, we introduce the background and motivation in Sec. II. In Sec. III

978-1-6654-4331-9/21/$31.00 ©2021 IEEE

and Sec. IV, we present SNIS and the evaluation results. The conclusion and future work is shown in Sec. V.

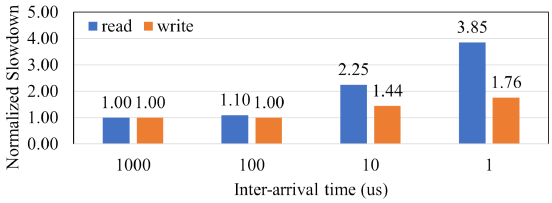
II. Background and Motivation

In a disaggregated storage system, the data storage is physically separated from the data consumer/generator. The data consumer/generator is denominated as a request initiator since they need to initiate a request to control the data. In a read request, the data is extracted from the storage and then sent to the initiator through the network. In reverse, the write request sends the data from the initiator to the storage and then writes the data to the disk in the storage. The completion time of both types of requests consists of two main components: **storage processing delay** and **network delay**. Both the storage processing delay and network delay are required to be scrutinized to improve the performance of a disaggregated storage system.

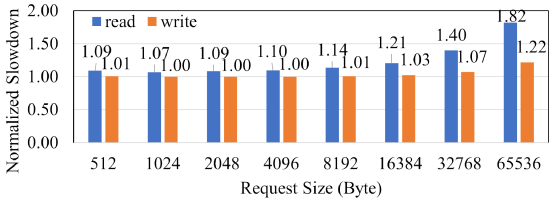
[Limitation of Existing Workloads:] Existing workload traces are collected in a proxy server [7] or a capture server using port mirroring [8], which only records the approximate arrival time to the storage system. In a disaggregated system, completing a request consists of storage processing delay and network delay. A read request should first finish its storage processing and then transfer the data through the network. In contrast, a write request needs to transfer the data through the network and then save the data in the storage. Therefore, the existing workload traces lack both the actual read arrival time to the network and the actual write arrival time to the storage. One naive approach is to estimate these arrival times by adding a constant delay for storage processing and network transfer. However, the interference between reads and writes and the randomness introduced by network congestion control and storage processing make it challenging to determine a correct estimation of such delay time.

[Performance Interference:] We conduct simulation experiments with a synthetic trace to study the performance inference between read and write requests. We first split the trace into two sequences with only read requests or write requests and then submit each of these sequences (i.e., read-only or write-only) to MQSim. The obtained results (referred to as "standalone") are used as the baseline for performance comparison. We further execute the whole trace (with both reads and writes) in MQSim that is configured with the same parameters.

Specifically, we generate synthetic traces across different inter-arrival times and request sizes and calculate the slow down (i.e., the latency normalized by the standalone one) caused by the interference of the other type of requests, as shown in Fig. 2. In Fig. 2-(a), we fix the mean request size to 4KB and vary the mean inter-arrival time. A shorter inter-arrival time indicates a heavier load to the system. As the mean inter-arrival time decreases to 10μs, we can observe that both read and write requests suffer latency degradation from the other. As the inter-arrival time further decreases to 1μs (i.e., the busiest one), the slowdown of read requests boosts more dramatically than that of write requests, indicating that



(a) Interference on intensity.



(b) Interference on volume.

Fig. 2. Latency interference between read and write requests.

write requests have more interference with read requests than vice versa. In Fig. 2-(b), we fix the mean of inter-arrival time to 100μs, and change the mean request size from 512B to 64KB. We obtain similar observations, e.g., the performance interference becomes more visible when the workload gets more intensive.

[Randomize Request Order:] Priority Flow Control (PFC) and Quantized Congestion Notification (QCN) have been widely used to mitigate network congestion. However, we also notice that pausing or slowing down the upstream sender under PFC or QCN can cause certain randomness in the request order, i.e., the actual transmitting order becomes different from the submitting order, which breaks the First In First Out (FIFO) order of network traffics. The performance of storage devices introduces randomness as well, which is affected by the read/write cache mechanism. For read requests, a cache hit in the cached mapping table can save latency. Write latency in write-back caching essentially depends on write cache capacity. At run time, the uncertainty of the cache status introduces randomness in the storage processing.

The issues above indicate that simply linking two components (i.e., storage and network) to conduct experiments cannot provide accurate performance results. This thus motivates us to develop a new storage-network simulation scheme that considers the impacts of both *performance interference* and *request order randomization* by iteratively updating actual arrival times of storage (resp. network) write (resp. read) requests to achieve convergence in performance.

III. Design of SNIS

In this section, we first introduce the storage and network simulators we deploy in our simulation, and then describe the technical design of our iterative storage-network simulator.

A. Network Simulation (NS3-RDMA)

Our network simulator is based on an existing implementation of RDMA on NS-3 simulator. NS-3 is an open-source network simulator and has been widely used in the networking research community. The NS-3 simulator with RDMA has fully implemented RoCEv2 protocol [3] and supports the

PFC [9] and ECN [10] mechanisms which are extensively exploited in popular network congestion control schemes. On the other hand, most packet scheduling-based congestion control schemes have developed their simulators, e.g., pHost's YAPS [11], NDP's NDP simulator [12], and HOMA's OMNet++ [13]. NS-3 enables PFC by default unless otherwise stated. This simulator takes network configuration and a workload trace as the input and simulates the network data transfer with a complete set of network protocols. We can configure the network topology, link capacity, number of network flows, and traffic size of each flow. Some other parameters that influence the simulation result are listed below. 1) Incast ratio, which has a great impact on the chance of the occurrence of network congestion. 2) PFC queue length threshold, which triggers the switch to send PFC PAUSE packets when the switch queue length is above that threshold.

B. Storage Simulation (MQSim)

For storage simulation, we focus on the latest NVMe SSD simulator. There are several open-sourced simulators, e.g., FEMU [14] and MQSim. FEMU is an emulator integrated with QEMU/KVM and exposed to Guest OS as an NVMe block device, while MQSim is a trace-driven discrete event simulation framework. After comparing the pros and cons between these two simulators, we choose to use MQSim as 1) it provides higher accuracy than FEMU, and 2) it is light and easy to collaborate with the network simulator. MQSim takes an SSD configuration file and a workload trace as inputs and generates performance statistics, e.g., average read/write latency, cmd queue length, and cmd waiting time.

We can launch multiple MQSim processes to emulate multiple targets (i.e., storage nodes) in the disaggregated storage system. In each MQSim, there are three sets of configurations: host configuration, device configuration, and flash configuration. We notice that the default configurations provide high latency at the scale of hundreds of microseconds. Thus, we use the micro benchmark to verify MQSim's correctness and understand the critical parameters in the MQSim configurations. We find there are a set of critical parameters in MQSim as follows. 1) *FTL* defines the Flash Translation Layer, 2) *Write Cache* defines write cache capacity, 3) *CMT (Cache mapping table)* defines read-cache capacity, 4) *Queue Fetch Size* decides the number of requests processed simultaneously, 5) *Page Capacity* specifies the unit of transactions at the device level, 6) *Read/Program/Erase Latency* defines the latency of a single internal operation, and 7) *Channel Transfer Rate* defines the transfer rate of flash channels in the SSD backend.

C. SNIS: Overall Simulation

We integrate MQSim and NS3-RDMA into a storage-network iterative simulation method (SNIS) to model disaggregated storage systems. SNIS simulates read/write requests generated by a set of initiators (users) and processed by a group of targets (storage devices). Each storage device is simulated as an individual MQSim instance with independent configurations. NS3-RDMA defines the topology of the communication network of the disaggregated storage system with a configurable network congestion control mechanism.

For the whole simulation process, the input workload is a set of I/O requests indicated as REQ with the following major attributes: 1) RequestID is the unique index of a request, 2) InitiatorID and TargetID specify the source and destination of a request, 3) ArrivalTime indicates the start time of a request, 4) Size is the amount of data in an I/O operation, and 5) IOType is either read or write. The format is almost the same as those used in the traditional storage system simulation except InitiatorID and TargetID that are specific to the disaggregated architecture. During our multiple iterations of simulations, most workload attributes are intact, and only the ArrivalTime will be updated through the process. Let us use $A(r)$ to represent the original ArrivalTime of a request r.

For both MQSim and NS3-RDMA simulators, at each iteration, they take a set of I/O requests as the input workload, and at the end of the simulation, each simulator reports the finish time or departure time of the processed requests. We use $NA_i(r)$ and $ND_i(r)$ to denote the arrival time and departure time of request r for the network simulator NS3-RDMA at iteration i. In other words, at the beginning of iteration i, request r with arrival time $NA_i(r)$ is in the workload input for NS3-RDMA, and at the end of the network simulation, $ND_i(r)$ is the time when request r finishes its network transfer. Similarly, we use $SA_i(r)$ and $SD_i(r)$ to represent the arrival time and departure time of request r for MQSim at iteration i. In addition, we use R and W to indicate the set of read requests and write requests, respectively, i.e., $REQ = R \cup W$.

Fig. 3 illustrates our iterative two-phase simulation SNIS. Our simulation process includes multiple iterative steps. The first iteration is considered as iteration 0. In each iterative step, all the requests are processed by both storage and network simulators. We always run the storage simulation first and then continue with the network simulation. At each iteration, for read requests, we assign their original arrival time as the arrival time for storage simulation, $\forall r \in R, SA_0(r) = A(r)$. Similarly, the original arrival time of write requests is assigned to their arrival time for network simulation, $\forall r \in W, NA_0(r) = A(r)$. For the other part of the requests, i.e., read requests in network simulation and write requests in storage simulation, we update their arrival time before conducting the simulation as follows:

$$NA_i(r) = SD_i(r), \quad \forall r \in R \tag{1}$$

$$SA_i(r) = ND_{i-1}(r), \quad \forall r \in W \tag{2}$$

For a read request r at iteration i, before entering the network simulation, its network arrival time is updated as the storage departure time at iteration i, see Eq. 1. The rationale here is that for a read request, data is first loaded from targets and then transferred to initiators over the network. Thus, the start time of network read flows is equal to the finish time of storage read requests. Here we use the finish

time of a read request from the storage simulation at the same iteration as an estimation. Correspondingly, for a write request r at iteration i, we update its storage arrival time as in Eq. 2, because data is first transferred from initiators to targets before written to targets. The finish time of network write flows is the start time of storage write requests. Since storage simulation is conducted first in every iteration, we update $SA_i(r)$ with the network departure time from the previous iteration. SNIS repeats interactive steps until the performance metrics converge to a stable state.

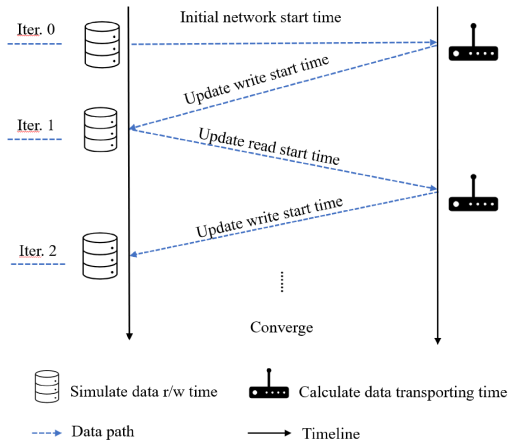


Fig. 3. Iterative two-phase simulation architecture.

Next, we present how to determine the simulation converges to a stable state. The whole process of interactive simulation involving two simulators is quite complex, and there are random factors that affect the performance. It cannot be theoretically proved that the process certainly converges. We define a stable state by considering the distribution of the key performance metrics. In particular, we use throughput as a convergence target, calculated by dividing I/O request size over the delay time. The delay time used for calculating throughput includes both storage and network delays. In this paper, we define a convergence criteria $Conv\_C$ as the weighted average of the relative distance of mean/median/90^{th} percentile of request throughput between two consecutive iterations. Let λ_i indicate the throughput distribution of all the requests at iteration i. The calculation of the convergence criteria ($Conv\_C$) at the end of iteration i is defined in Eq. 3.

$$\begin{aligned} Conv\_C_i \quad &= \alpha_1 \cdot RD(\lambda_i^{mean},\ \lambda_{i-1}^{mean}) \\ &+\alpha_2 \cdot RD(\lambda_i^{median},\ \lambda_{i-1}^{median}) \\ &+\alpha_3 \cdot RD(\lambda_i^{90_{th}},\ \lambda_{i-1}^{90_{th}})), \end{aligned} \tag{3}$$

where λ_i^{mean}, λ_i^{median}, and $\lambda_i^{90_{th}}$ represent the mean, median, and 90^{th} percentile of the throughput at iteration i. RD is the function that calculates relative distance between two values defined in the following Eq. 4.

$$RD(x, y) = \frac{abs(x - y)}{max(x, y)} \tag{4}$$

The definition of the convergence criteria $Conv\_C$ aims to capture the difference between two throughput distributions λ_i and λ_{i-1} by comparing the three characteristics of the distribution. According to the evaluation needs in different research work, it can be easily extended to include more information of the distribution. The weights α can also be adjusted to accommodate different evaluation goals.

In SNIS, we consider the iterative process converges to a stable state under one of the following two conditions: 1) $Conv\_C_i$ drops below a certain threshold (σ); and 2) $Conv\_C_i$ stays within a small range (τ) comparing to previous iteration ($Conv\_C_{i-1}$) for a sequence of n iterations.

IV. Evaluation

In this section, we present our evaluation results. We first introduce general configurations of MQSim and NS3-RDMA simulators and then thoroughly examine the iterative simulation process with various workloads. We analyze the convergence criteria and other interesting findings to validate that SNIS is an effective simulation process for evaluating disaggregated storage systems.

A. Simulation Configuration and Workload Description

We model the popular configuration in data center networks and create a CLOS network, a multistage switching architecture, containing the spine and leaf layers and multiple pods. In this network topology, servers (i.e., initiators and targets) are connected to leaf switches in the same pod, and each leaf is connected to all spines. Specifically, we have totally 4 pods and each pod consists of 2 leaf switches, 4 top-of-rack (ToR) switches, and 64 servers. We set the link capability as 10Gbps between servers and ToRs and 40Gbps between ToRs (resp. leaf switches) and leaf switches (resp. spines). The link delay is set to 1μs. We construct a homogeneous storage system such that we have the same configuration for all MQSim instances. Table I summarizes our setup of the critical parameters used in MQSim.

TABLE I
MQSim Parameter Configuration

Queue Depth	Write Cache	CMT	Queue Fetch Size	Page Capacity	Read/ Program
65534	256MB	2MB	512	16KB	2μs/100μs

We leverage the characteristics of real cloud-storage workloads (e.g., Fujitsu virtual desktop infrastructure (VDI) [15], and Tencent Cloud Block Storage (CBS) [16]) to generate a set of synthetic traces and replay these traces to evaluate the convergency of our simulator. Specifically, we construct experiments on different workloads by changing the mean of request sizes and inter-arrival times of read and write requests. We keep request size fixed and change the mean of inter-arrival time to generate workloads with different intensity levels, which produces temporal-light, -moderate and -heavy workloads (i.e., t_l, t_m and t_h). To investigate the impact of request size on convergency of SNIS, we also keep the inter-arrival time the same but change the mean of request size, which generates spatial-light, -moderate, and -heavy workloads (i.e., s_l, s_m and s_h). The network load equals to the average size divided by the average inter-arrival time. Because of the page limit, we only show experimental results of Fujitsu VDI synthetic traces.

TABLE II
SYNTHETIC TRACE CHARACTERISTICS AND CONVERGENCY

Trace Tag	Size avg.(KB)		Inter-arrival avg. (μs)		network load (Gbps)		Conv. Iter.
	Read	Write	Read	Write	Read	Write	
F_t_l	28	11	57	54	3.93	1.63	2
F_t_m	27	12	17	17	12.71	5.65	7
F_t_h	28	11	12	10	18.67	8.80	3
F_s_m	80	34	60	53	10.67	5.06	5
F_s_h	110	46	60	52	14.67	7.08	3

B. *Convergency of* SNIS

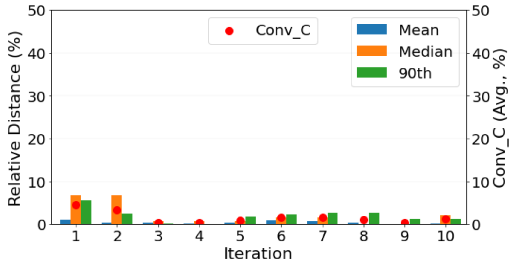
Convergence criterion is the most important measurement in our evaluation. The parameters in SNIS are set as $\sigma = 10\%$, $\tau = 2\%$, and $n = 3$. We have evaluated SNIS based on both Fujitsu VDI and Tencent CBS traces. The outcomes are quite consistent. We thus only present the results from Fujitsu VDI datasets, including detailed analysis. In each test, we repeat the simulation process for 11 iterations (i.e., 10 convergence criteria values are calculated) even if the termination conditions are met.

Fig. 4 shows the convergency metrics across iterations for Fujitsu synthetic traces with different temporal intensities. Each bar represents the relative distance of a metric (e.g., mean, median, and 90_{th}). As the index of iterations starts from 0, the relative distance of iteration 1 is the relative distance between the first and the second iterations. Each red point is the convergence criteria ($Conv\_C_i$) as defined in Eq. 3 which uses the right y-axis for a certain iteration. Fig. 4-(a) shows the convergence criteria of light Fujitsu synthetic trace. We observe that the convergence criteria are around 4.5% at the second iteration, which is considered converged, although the later iterations have fluctuated convergence criteria. This observation is consistent with the fact that the light workload is highly possible to be processed in FIFO as the randomness incurred by network and storage operations is minimized due to the long interval time between two consecutive requests.

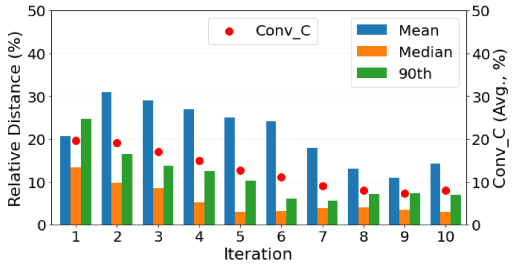
We increase the workload intensity to moderate in Fig. 4-(b) by shortening the mean of inter-arrival time. We observe that the convergence criteria drop to 10% at iteration 7 and keep stable after that. Whereas, we observe that the relative distance of mean is higher than that of median and 90_{th}. This observation also exists in Fig. 4-(c), when we further increase the workload intensity. The reason is that the long tail of the throughput distribution varies across different iterations, although the distributions of throughput are similar[1], as shown in Fig. 5, which stretches the mean of throughput of each iteration differently. In addition, we observe that the convergence criteria converge faster in Fig. 4-(c) (i.e., at iteration 3) than that in Fig. 4-(b). The reason is that a heavy workload fully utilizes storage and network resources, which yields less randomness compared to the moderate workload.

Furthermore, we investigate the real-time network conditions to confirm the existence of congestion and PAUSE packets from the PFC protocol and the effectiveness of SNIS in such a complex network setting. We measure the number of PAUSE packets received by spine switches and the total sending rate at run-time. When spine switches receive a large number of PAUSE packets, the congestion becomes severe and starts to affect the entire network. We illustrate the results from the heavy workload (F_t_h) in Fig. 6. As the network traffic for read requests in our tests is from one target to multiple initiators, there is little congestion in this direction, and the sending rate is maintained at the ToR link capacity of 10Gbps.

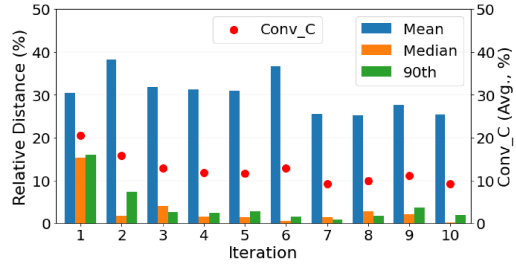
[1]In Fig. 5, we manually decrease the y-value by 0.02 for each iteration in order to clearly see all CDF curves. We remark that the original CDF curves are actually overlapping.



(a) Light workload F_t_l



(b) Moderate workload F_t_m



(c) Heavy workload F_t_h

Fig. 4. Convergence criteria of VDI traces with different inter-arrival times.

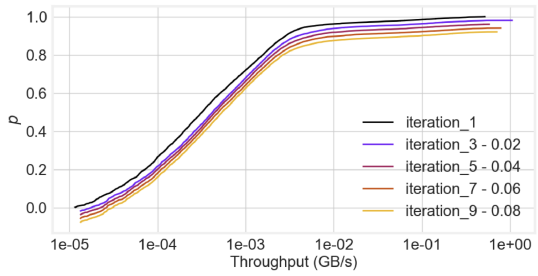


Fig. 5. Linear-log throughput CDF in each iteration (F_t_h).

However, the network encounters a high incast ratio when 10 initiators send data to one target for write requests. The link connecting the target and its ToR is congested, and the PAUSE packets are spread to upper layer switches with the PFC protocol. We observe that the trend of the sending rate for write requests matches the spikes of the number of PAUSE packets in Fig. 6. Above all, in our tests, network congestion occurs and massive PAUSE packets are generated, disordering data delivery. SNIS is still quite effective and reaches stable states under moderate and heavy workloads.

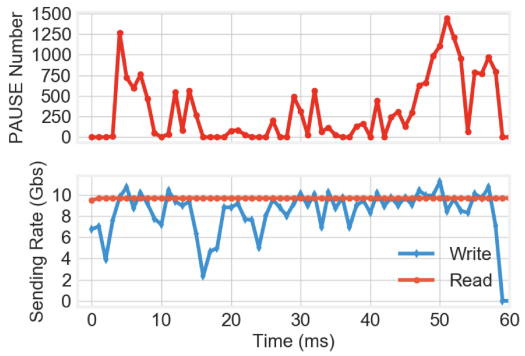
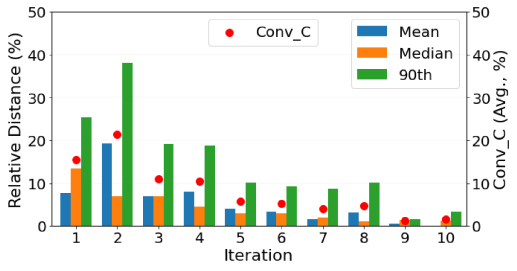
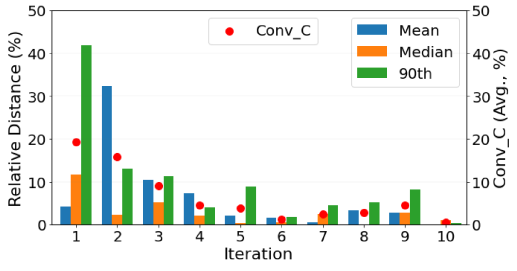


Fig. 6. Spine PAUSE packet numbers and sending rates.

Finally, we investigate the impact of spatial intensity on the effectiveness of SNIS. Fig. 7 shows the evaluation results (i.e., relative distances and convergence criteria) of the workloads F_s_m and F_s_h that have similar inter-arrival time as F_t_l (see Fig. 4-(a)) but increase the mean of request sizes by three and four times. We observe that these two traces converge at early iterations to convergence criteria below 5%, which indicates that SNIS converges faster under spacial-intensive workloads than temporal-intensive workloads.



(a) Moderate workload F_s_m



(b) Heavy workload F_s_h

Fig. 7. Convergence criteria of Fujitsu VDI traces with different request sizes.

V. Conclusion

This work presents an iterative multi-round simulation method for modeling disaggregated storage systems by simulating storage and network activities jointly. In this way, SNIS can mitigate the impacts of performance interference and request randomization and use the converged results to simulate and evaluate disaggregated storage systems accurately. We conduct the evaluation experiments of SNIS by using a variety of synthetic workloads with different request arrival rates and sizes. Our evaluation shows that after a certain number of iterations, SNIS can always reach a relatively stable state. Our evaluation also shows that different types of workloads experience different convergence speeds under SNIS. In the future, we plan to extend our work to evaluate the convergency of SNIS under various network and storage configurations. We also plan to exploit SNIS to design and evaluate new network congestion control mechanisms and storage workload managers for disaggregated storage systems.

References

[1] Y.-X. Lu, J. Chiu, S.-J. Chao, and Y.-B. Ye, "Design of instruction analyzer with semantic-based loop unrolling mechanism in the hyperscalar architecture," in *ICS*, 2018.

[2] S. Luo, G. Zhang, C. Wu, S. Khan, and K. Li, "Boafft: Distributed deduplication for big data storage in the cloud," *IEEE Transactions on Cloud Computing*, vol. 8, pp. 1199–1211, 2020.

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

[4] IBTA, "InfiniBand," https://www.mellanox.com/pdf/whitepapers/IB_Intro_WP_190.pdf.

[5] 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/

[6] 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.

[7] Y. Zhang, P. Huang, K. Zhou, H. Wang, J. Hu, Y. Ji, and B. Cheng, "Osca: An online-model based cache allocation scheme in cloud block storage systems," in *USENIX Annual Technical Conference*, 2020.

[8] C. Lee, T. Kumano, T. Matsuki, H. Endo, N. Fukumoto, and M. Sugawara, "Understanding storage traffic characteristics on enterprise virtual desktop infrastructure," *Proceedings of the 10th ACM International Systems and Storage Conference*, 2017.

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

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

[11] P. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "phost: distributed near-optimal datacenter transport over commodity network fabric," *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, 2015.

[12] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017.

[13] 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.

[14] H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, and H. S. Gunawi, "The case of femu: Cheap, accurate, scalable and extensible flash emulator," in *FAST*, 2018.

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

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