A Disaggregate Data Collecting Approach for Loss-Tolerant Applications

Ziyuan Liu[1,2], Zhixiong Niu[2], Ran Shu[2], Wenxue Cheng[2], Peng Cheng[2]
Yongqiang Xiong[2], Lihua Yuan[3], Jacob Nelson[2], Dan R. K. Ports[2]

[1]Beihang University, [2]Microsoft Research, [3]Microsoft

ABSTRACT

Datacenter generates operation data at an extremely high rate, and data center operators collect and analyze them for problem diagnosis, resource utilization improvement, and performance optimization. However, existing data collection methods fail to efficiently aggregate and store data at extremely high speed and scale. In this paper, we explore a new approach that leverages programmable switches to aggregate data and directly write data to the destination storage. Our proposed data collection system, ALT, uses programmable switches to control NVMe SSDs on remote hosts without the involvement of a remote CPU. To tolerate loss, ALT uses an elegant data structure to enable efficient data recovery when retrieving the collected data. We implement our system on a Tofino-based programmable switch for a prototype. Our evaluation shows that ALT can saturate SSD's peak performance without any CPU involvement.

CCS CONCEPTS

• **Networks** → **In-network processing**; • **Hardware** → **Emerging technologies**;

KEYWORDS

Data collection, Programmable switchs, NVM Express, Remote Direct Memory Access

ACM Reference Format:
Ziyuan Liu, Zhixiong Niu, Ran Shu, Wenxue Cheng, Peng Cheng, Yongqiang Xiong, Lihua Yuan, Jacob Nelson, Dan R. K. Ports. 2022. A Disaggregate Data Collecting Approach for Loss-Tolerant Applications. In *Asia-Pacific Workshop on Networking (APNet '22), July 1–2, 2022, Fuzhou, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3542637.3542646

APNet '22, July 1–2, 2022, Fuzhou, China

ACM ISBN 978-1-4503-9748-3/22/07…$15.00
https://doi.org/10.1145/3542637.3542646

1 INTRODUCTION

Data collection is a fundamental component in various today's data center applications, and many of them have great importance and are working at a considerable scale (e.g., EverFlow [41], a passive networking monitoring system deployed in Microsoft, can operate at 380Gbps in one data center, and Scribe [24], a log service deployed in Facebook, can operate at 20Tbps globally).

Current data collection methods can be categorized based on the technology they used, including native network stack, optimized software [18, 40], and special hardware [36, 39]. Based on our analysis, we find that none of these approaches can achieve low CPU overhead, cost-efficient, and transparent to other applications simultaneously.

We achieve all the three aspects in our data collection system, ALT, showing that they indeed can be fulfilled under a programmable switch's help. Our key observation is that abundant applications which need large-scale data collection can tolerate loss to some extent because they try to mine insights from large amounts of data. It alleviates the requirement on reliability and allows us to use the resources of a programmable switch to build a data collection system by controlling remote memory and storage without the involvement of the host's CPU. However, it is not easy and there are several challenges to be addressed.

First, the current programmable switch cannot access remote storage out of the box. Moreover, modern SSDs can achieve their specification performance only when data size is a multiplier of 4KiB; otherwise, the performance will drop significantly (about 10x in our testbed). However, due to the lack of programmability of packet buffer, current programmable switches cannot batch data from MTU-level to 4KiB to meet SSD's requirement. Our insight is that by leveraging RDMA and PCIe peer-to-peer communication between RDMA NIC and SSDs, we can access storage in maximum

performance on the switch with the assistance of the host's memory while completely CPU-free in the data path.

Further, to build a post-processing-friendly data collection system, we want to aggregate data in logical flows. Though it may be trivial for a general CPU, realizing the same scheme under a programmable switch's limited resources is more challenging. Moreover, though our target applications can tolerate loss, we still need to consider loss impact carefully to avoid catastrophic cascade consequences when retrieving data back from storage. By repurposing the multi-level page table from the OS field and deliberate implementation, we provide an interface similar to the one used in Scribe. Additionally, we show that combining two simple ideas, redundancy and alignment, can significantly reduce cascade impact caused by loss.

We propose and implement a prototype of ALT in P4 using a Tofino-based programmable switch. Our evaluations show that ALT can collect data in NVMe SSD's maximum throughput with completely zero host CPU usage during data collection, and can mitigate loss impact effectively.

2 BACKGROUND AND RELATED WORK

This section provides a brief background on loss-tolerant applications and current data collection methods to motivate the need for a solution achieving low CPU overhead, cost-efficiency, and transparency simultaneously. Moreover, this section introduces programmable switch to help readers better understand its ability and constraints, which can explain some challenges we faced and our design rationales.

2.1 Loss-Tolerant Applications

Due to the tremendous computing power and storage capacity provided by large clouds, data-intensive applications have their roles in today's production environment. For example, EverFlow [41], a passive networking monitoring system deployed in Microsoft, can operate at 380Gbps in one data center, and Scribe [24], a log service deployed in Facebook, can operate at 20Tbps globally. An essential subset of these applications has loss-tolerant characteristics—some loss of input data may not change the result of the application. Intuitively, loss-tolerant is natural for the data-intensive applications that focus on mining out functional semantics from the data statistics, where the lost data is negligible compared to the whole input data volume. Two crucial instances are passive network telemetry systems and log analysis systems.

Passive network telemetry systems: Passive network telemetry systems gather data passively by listening to network traffic [33]. Packet-mirror-based monitoring systems [20, 38, 41], which leverage mirror functionality provided by commodity switches, are one typical realization of passive monitoring. Due to the hardware constraints, there is no guarantee that mirrored packets can be finally transmitted to the destination. Therefore, such packet-mirror-based telemetry systems are loss-tolerant by nature. Another typical realization of passive network telemetry systems is flow-based monitoring, which only transmits per-flow information to dedicated servers. Some of them [1, 10] rely on sampling to reduce the volume of monitored information, which will actively ignore many packets. Thus they are also loss-tolerant.

Log analysis systems: Log analysis systems are built to mine insights from amounts of log data in many scenarios, such as anomaly detection [16, 21, 23] and network debugging [11, 34]. Many such systems are loss-tolerant because of the process to generate datasets or the adopted analysis methods. On the one hand, some sampling technologies are used to reduce the amount of log to be computed [14, 27, 29], and the data discarded during sampling can be treated as losses. On the other hand, if data is missing at random, the missing process can be ignored and the validity of inference will not be influenced [31]. Therefore, these log analysis systems also have loss-tolerant characteristics.

2.2 Current Data Collection Methods

Generally speaking, the workflow of data-intensive loss-tolerant applications is similar to the one of knowledge discovery in database (KDD), which can be divided into 9 steps [17, 19]. Here we focus on the first step, the gathering part. Because collecting data does not generate any new information, it is important to achieve high performance and cost-efficiency in this step.

There are three typical data collection methods in our target loss-tolerant applications: (1) native network stack, (2) optimized software, and (3) special hardware. However, they all fall short of simultaneously achieving low CPU overhead, cost-efficiency, and transparency to other applications.

Native network stack: The native network stack is out-of-the-box in an operating system, and thus cost-efficient and transparent to other applications from nature. However, though researchers and vendors have paid great attention to the performance of the native network stack, it still induces considerable CPU overhead to saturate 100Gbps [15]. Things become worse when it comes to the packet capture scenario. For example, tcpdump, a popular and wildly used packet sniffing tool, can only capture packets to NVMe SSDs at less than 10Gbps and consume more than 4 CPU cores in our testbed when using the native network stack as backend.

Optimized software: One optimization is implementing functionalities in user space instead of kernel. For example, [18, 40] use DPDK [5] to improve their performance while reducing the CPU overhead. However, transparency to other applications is usually traded, as optimized software requires

controlling NIC directly and needs special drivers, which becomes an obstacle when deploying them in real production.

Special hardware: The other way to reduce the native kernel stack overhead is to use a special hardware appliance, such as FPGA-based packet capture systems [36, 39]. By executing almost all operations in hardware and their bump-in-the-wire installation method, these solutions achieve zero CPU usage and are transparent to existing applications. However, because the hardware is dedicated, it can only be shared among limited tasks and are often more expensive than other standard commodity components in deployment.

In conclusion, our analysis suggests that none of the current data collection methods can achieve low CPU overhead, cost-efficiency, and transparency simultaneously.

2.3 Programmable Switch

Programmable switches are proposed to provide flexibility in switches' functionalities. The behavior of such switches is usually variations of Protocol Independent Switch Architecture (PISA) which decomposes the switch data plane mainly into 2 phases: ingress and egress. In each phase, the programmability is mainly achieved by many match-action units, which can take several simple actions (e.g., addition and subtraction) based on headers and computed data. Additionally, PISA also allows some fixed-function modules such as encryption or mirroring to be added by the real implementation. Users can control the behavior of these modules by specifying the parameters in the switch program.

In the above description, a programmable switch can operate packets without persistent states: no information can be carried across multiple packets. Registers are added to empower PISA the ability to change stateful memories during processing packets.

It should be noted that in real ASIC implementation, the programmability of PISA is limited. For example, a real programmable switch can only operate on parsed, limited-length "headers". Moreover, the amounts of registers are usually small, and some ASICs require accessing them in a DAG manner. Additionally, it is hard to manipulate multiple packets, such as combining and splitting. Storing a large packet is also difficult, as the packet buffer is out of the switch program's control (though storing small packets can be achieved by registers, as we can treat the whole packets as "headers").

Though the programmability is limited, many recent works have demonstrated that under careful design, programmable switch can bring revolution in amounts of network functions [12, 13, 25, 32] and is suited for offloading partial of applications to enhance their throughput and scalability [22, 28, 30, 37]. Additionally, some companies [35] have indicated a trend that programmable switches may become part of the datacenter's infrastructure in the future.

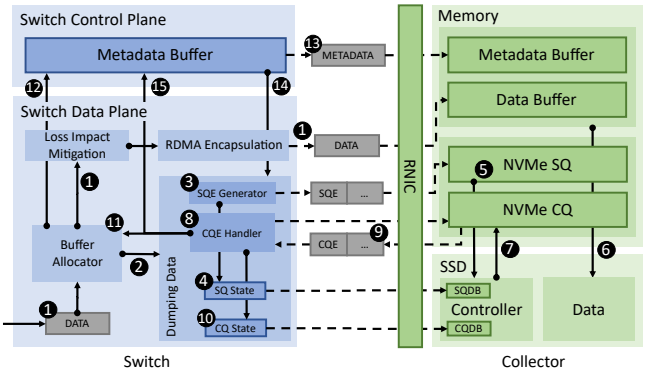


Figure 1: Summary of workflow in ALT

In this paper, we try to answer the question: is it possible to use a programmable switch to control servers' NVMe devices and further build a post-processing friendly data collection system achieving low CPU usage, cost-efficiency, and transparency simultaneously? Fortunately, the answer is our proposed system, ALT.

3 DESIGN

In this section, we will first present the workflow of ALT to give a bird-eye view (section 3.1). Then, we describe the details of ALT in the rest of this section.

3.1 ALT overview

ALT provides users a "logical stream" abstraction to collect data, which supports append and fetch operations. The data are written to an array of NVMe devices spread over multiple hosts under maximum performance of the devices, and the whole process does not involve the host CPU at all.

Figure 1 illustrates the key components and workflow of ALT. The programmable switch first adds some metadata which helps to mitigate the impact caused by packet loss to create a data record, and then writes it to the buffer block allocated by the buffer allocator through RDMA (1). When the buffer allocator detects that there is enough data in a buffer block, it locks it and notifies the dumping data module (2) to trigger the interaction with NVMe protocol (3-10, the details are explained in section 3.2). Meanwhile, it notifies the control plane to update the metadata structure (12). If all data of a buffer block has been dumped from memory to SSD, the block is able to accept new data (11). When there is enough metadata, a similar procedure applies to metadata to synchronize the state between the switch and collector's SSD (13-14, 3-10, 15).

3.2 Performant Storage Access with Zero CPU Usage

To access the storage on the remote collector with zero CPU and reach the maximum bandwidth of a single SSD, we must

design a data transmission scheme without the remote collector CPU's assistance.

NVMe-oF allows a host to access remote NVMe devices through RDMA, and with some commodity NIC's help, the execution of the protocol can be completely offloaded to hardware. At first glance, it is natural to extend TEA's [26] idea to implement the NVMe-oF with RDMA on a programmable switch. However, if the data of an NVMe command is too large and must be transmitted into multiple RDMA packets, NVMe-oF requires every data packet to be exactly RDMA MTU size, except for the last one, which is almost impossible in a programmable switch. This is mainly because the programmable switch is difficult to buffer packets to aggregate small data into MTU size. Though padding packets to MTU size is possible, it is also problematic due to the high overhead.

The above issues force us to transmit NVMe command and data in a single RDMA SEND packet to the host. However, even if we can achieve it, we still encounter performance issues. The maximum data payload supported by NVMe-oF in a single RDMA packet is strictly less than 4KiB, which causes significant degradation of SSD performance, as shown in section 5.1.

Instead of implementing NVMe over RDMA, the key idea to achieve our goal is to leverage the remote collector's memory as a pipelined buffer for batching small data into a batch and control SSD through the original NVMe protocol. When data comes to the programmable switch, the switch data plane will first use RDMA to transmit the data into the remote memory buffer and monitor the buffer's occupancy by one register. When the buffer has enough data, the switch will interact with the collector's NVMe protocol through RDMA to instruct SSD for dumping data from the memory buffer to SSD. Because we use RDMA in the whole process, the remote CPU is completely zero during data transmission and dumping.

The NVMe specification [2] defines the procedure of how to submit a command and how it should be executed. The polling version is described following: 1) host submits a command to one of SSD's IO Submission Queue; 2) host rings the Submission Queue's doorbell to notify SSD's controller of new commands; 3) SSD's controller fetches commands from the IO Submission Queue and 4) executes them; 5) after completion, SSD's controller posts corresponding completion entries into Completion Queue; 6) then host checks the content in the Completion Queue; 7) finally, host rings the Completion Queue's doorbell to notify SSD controller that corresponding completion entries can be recycled. In our design, we use a switch to replace the host in the workflow, i.e., the 1), 2), 6) and 7), and all interactions between the switch and SSD's controller are through RDMA, as shown in Figure 1, step 3-10.

3.3 Post-processing Friendly Data Structure

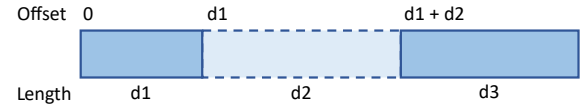
Similar to Facebook's Scribe [24], we aggregate data and provide "logical stream" abstraction to users. A logical stream is like a named pipe: each data generator of the logical stream can "append" data to it. Appended data are stored in a data structure called data entry. For consumers of a logical stream, we provide an operation called "tail", which supports a user to fetch at most N data entries from the current cursor or from the tail of a logical stream. To realize such abstraction under zero collector CPU involvement and with high space efficiency, we must design a metadata structure that is powerful but simple enough to be efficiently manipulated by switch without the remote CPU's help.

Metadata Structure We use a 2-level page table (page directory, page table, and data page), which is originally for the operating system's paging system, to describe the mapping between the logical address space of a logical stream to the physical address space of storage. By leveraging the scheme, we trade some extra space (array for the mapping) with the ability to dynamically allocate space in the granularity of pages, which can achieve high space efficiency. For convenience, we will use {size of page directory}-{size of page table page}-{size of content page} stands for a specific 2-level page table setting.

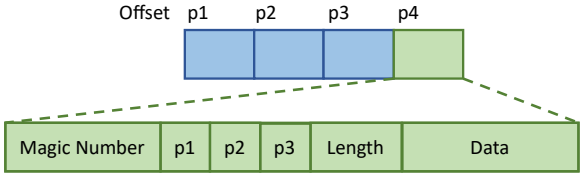
Efficient Metadata Manipulation Because metadata is important, we decide to use the switch's control plane for metadata transmission. However, simply storing the whole page table in control plane memory will cost too much memory for a logical stream and thus influence the max number of concurrent streams a switch can support. Our solution for this issue is to only keep metadata which is mandatory for append operation in switch (1 entry and its offset for each level in the 2-level page table), and store the complete 2-level page table in remote memory/storage.

3.4 Relieve Cascade Effect of Loss

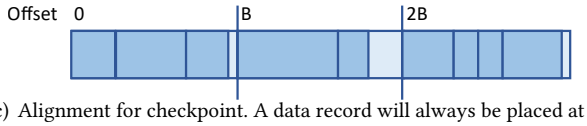
Because DCN's Ethernet is lossy, packets have no guarantee to reach their destination. For metadata, due to its importance and negligible volume compared with data, we could use SoftRoCE's RC service in the control plane for reliability, but it is hard to implement such a mechanism in the data plane. For example, buffering sent but yet acknowledged packets is required for reliability, and one can only use registers in the data plane for buffering. Due to the limited length one pipeline pass can handle, it requires multiple passes to buffer a whole packet and introduces considerable overhead. Nevertheless, some data generators may not even be able to retransmit the data, e.g., the mirror switch. Thus we choose to use RDMA UC QP to transmit data into the remote buffer. Though the data entry loss induced by DCN's Ethernet may not hurt loss-tolerant applications' performance, the key



(a) Loss impact.



(b) Reduandancy for recovering from sparse loss, assuming $R = 3$.



(c) Alignment for checkpoint. A data record will always be placed at the beginning of each block.

Figure 2: Loss impact and solutions to mitigate it

challenge here is how to mitigate the cascade effect induced by loss: a data entry loss will not only influence itself but also make the following successfully collected data cannot be retrieved. As shown in Figure 2(a), if the second data entry is lost during transmission, because we lose the length information d_2, we cannot get the address of the third entry $d_1 + d_2$ and therefore loss the access to it, and all ones following, though they have been successfully collected. We term this phenomenon the cascade impact of loss.

Our solutions for this issue are twofold: redundancy for recovering from sparse loss, and alignment for checkpoint.

Redundancy for Recovering from Sparse Loss: For sparse loss events, our observation is that the probability of losing several consecutive data entries would be low. Therefore, we could encode several previous data entries' addresses into the current data entry's header, and recover from loss if one address matches any one of the known entries, as shown in Figure 2(b). When a loss occurs, we try to search the magic number to locate a potential beginning of a data entry. To avoid that the found magic number is just part of the data, we compare the encoded address of previous data entries with known ones. If some of them match, we can be confident that the found beginning is valid and recover from the loss. With this technology, we can reduce the loss ratio exponentially.

Alignment for Checkpoint: For sparse loss events, we can recover from loss by inference from known entries and redundant information. However, when a burst loss – which losses more than R successive packets – happens, there will be no known entries for correct inference. To mitigate this issue, we divided a data page into several blocks, and assure there will always be a data entry placed at beginning of each block, as shown in Figure 2(c). It should be noted that even if the aligned data entry itself can be lost during transmission, its address information is treated as known and can be used in the recovery process described above. When consecutive losses happen, the redundancy policy fails and all following data cannot be retrieved. By providing a known entry using alignment, this technology "resets" the impact of loss, and thus reduces the loss ratio.

4 IMPLEMENTATION

We have implemented an ALT prototype in P4 using a commodity programmable switch, Arista 7170 series. For fast prototyping, we implement almost all designed data plane functionalities but use SoftRoCE in the switch's control plane to realize the NVMe workflow.

In the original workflow, the CPU needs to write some SSD's MMIO registers (SQ and CQ doorbell), but these registers are not RDMAable in the existing API. We leverage Mellanox's PeerDirect [7] technology, which is used in GPUDirect [4] to provide the ability for a Mellanox NIC to access peer devices' memory without CPU, to expose the NVMe Controller Registers BAR region as an RDMAable memory.

We leverage the switch's mirror mechanism to inject control signal packets from the data plane to trigger the multi-phased NVMe workflow. For the control signal, the data in the original packet is meaningless, therefore we truncate it to reduce the bandwidth requirement.

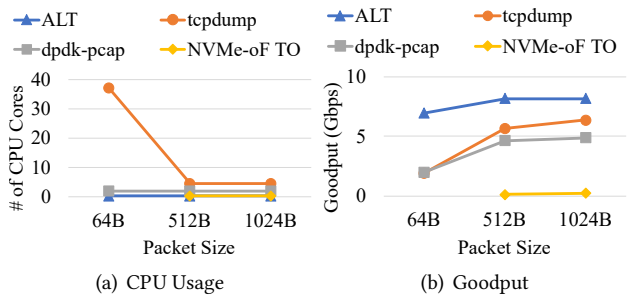
On collector host, we use an SPDK program to reserve one of SSD's SQs and CQs for ALT usage. Note that the program is only activated during the initialization and tear down phase, and isn't involved in data transmission completely.

5 EVALUATION

We evaluate ALT on a programmable switch testbed under synthetic packet traces. Our key findings are:

- For a single NVMe SSD, ALT can saturate it with completely zero CPU usage during data transmission.
- Even a low network loss ratio (0.01%) can cause considerable data integrity issue (about 12% data cannot be retrieved). With ALT's loss impact mitigation technologies, more than 98% data can be preserved under 1% network loss ratio, and more than 90% even under 10% network loss ratio. The CPU overhead induced by these technologies is close to a memory copy (2% per Gbps) when the network loss ratio is low (0.01%).

Experimental setup. Our testbed consists of a Arista 7170-64C programmable switch [3] and 2 machines with dual 10-core Intel Xeon Silver 4114 at 2.20 GHz (40 logical cores in total), 48 GB RAM, a 100 Gbps Mellanox CX-5 NIC [6] and 10 Samsung PM983 960 GB NVMe SSDs [9]. The servers run Ubuntu 18.04 with kernel version 4.15.0. All servers are



(a) CPU Usage (b) Goodput

Figure 3: Performance of ALT on Single SSD

directly connected to the switch. We use 1 server as the data generator and 1 as the collector. In all experiments, we use 4KiB-4KiB-4MiB metadata structure setting, redundancy factor $R = 3$, and 256KiB block size in alignment.

5.1 Single SSD Performance

For clearness, we use two typical packet sizes in DCN, 64B and 1024B, to compare ALT with three baselines, NVMe-oF Target Offload (hardware implementation of NVMe-oF target side, provided by some advance RDMA NIC) [8], tcpdump and DPDK testpmd + pdump. Additionally, because almost all commodity SSDs require 512B as the minimum block size, it is impossible to test the performance of NVMe-oF Target Offload under 64B packet size, thus we add 512B to meet SSDs' minimum requirement. We use fio to generate traffic for NVMe-oF Target Offload, and DPDK Pktgen for others. In both tools, we rate limit the traffic to 10Gbps, which can saturate the maximum bandwidth of the SSD model we use.

Figure 3(a) shows the collector side CPU usage. When packet size is small, tcpdump can eat up almost all CPU cores, and it is mitigated when packet size is larger. The root cause is that NIC's IRQ needs amounts of CPU cores, especially when packet size is small. DPDK testpmd + pdump always use 2 cores: 1 for testpmd and 1 for pdump. ALT and NVMe-oF Target Offload have zero CPU usage because they are hardware-based solutions, as expected.

Figure 3(b) shows the goodput achieved by different systems. NVMe-oF Target Offload demonstrates very poor performance because of the physical characteristics of SSDs. They could only reach a reasonable performance when the block size of a single write is 4KiB, and some model needs a larger block size (e.g., 128KiB) to reach their optimal write performance. The result shows the necessity of aggregating small packets into a larger one when using SSDs, as done in ALT. tcpdump and DPDK testpmd + pdump show similar behavior when packet size varies. ALT outperforms other systems. The gap between its performance and the maximum bandwidth of the SSD model is due to the data entry header and fields to mitigate data loss impact. Taking it into consideration, ALT can saturate our SSD model.

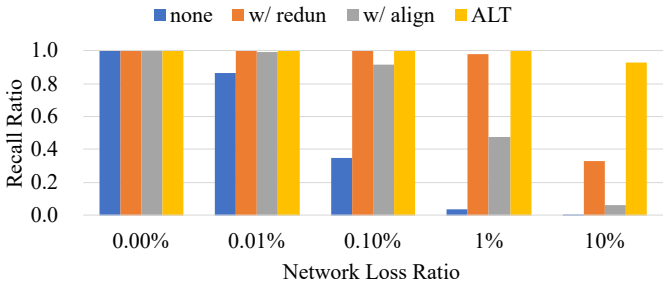


Figure 4: Performance of ALT's loss mitigation technologies

5.2 Loss Impact Mitigation

According to section 3.4, without any mitigation, all data records after the lost one on the same data page cannot be retrieved. To demonstrate the impact of loss and the effectiveness of our approaches, we measure the recall ratio of data under different network loss ratios. The recall ratio is defined as the number of retrieved data records divided by the number of received data records. Supposing the network loss ratio is l, and the recall ratio is r, the user can get $l \times r$ portion of the data records.

We evaluate the recall ratio of the baseline algorithm, which doesn't have any mitigation, and ALT's algorithm under 0.01%, 0.1%, 1% and 10% network loss ratio to show our proposals' effectiveness. We also compare redundancy only and alignment only algorithms for the breakdown analysis.

Figure 4 shows the recall ratio. We see that when the network loss ratio increases, the recall ratio for the baseline algorithm drops greatly. The ALT's solutions to mitigate loss can achieve almost 100% recall ratio and more than 90% even under 10% network loss. The breakdown analysis shows that the improvement mainly comes from the redundancy, as it can achieve almost 100% recall ratio under 0.1% loss ratio and more than 98% under 1% loss. When the network loss ratio becomes large, the combination of redundancy and alignment outperforms any single scheme.

6 CONCLUSION

Data collection is fundamental to many data center applications, however, none of the current methods can simultaneously achieve low CPU utilization, cost-efficiency, and transparency, which may become an obstacle in deployment. To address this issue, we proposed ALT, for large-scale data collection of loss-tolerant applications. ALT provides users a logical stream abstraction to collect data and uses a programmable switch to control NVMe devices on rack servers directly. Our evaluation shows that ALT can collect data without hosts' CPU involvement and mitigate cascade loss impact effectively.

REFERENCES

[1] 2004. Cisco Systems NetFlow Services Export Version 9. https://datatracker.ietf.org/doc/html/rfc3954. (2004).

[2] 2021. NVM Express Base Specification, Revision 1.4b. https://nvmexpress.org/wp-content/uploads/NVM-Express-1_4b-2020.09.21-Ratified.pdf. (2021).

[3] 2022. Arista 7170 Series - Arista. https://www.arista.com/en/products/7170-series. (2022).

[4] 2022. GPUDirect | NVIDIA Developer. https://developer.nvidia.com/gpudirect. (2022).

[5] 2022. Home - DPDK. https://www.dpdk.org/. (2022).

[6] 2022. NVIDIA Mellanox ConnectX-5 Adapters | NVIDIA. https://www.nvidia.com/en-us/networking/ethernet/connectx-5/. (2022).

[7] 2022. NVIDIA PeerDirect - MLNX_OFED v5.4-1.0.3.0 - NVIDIA Networking Docs. https://docs.nvidia.com/networking/display/MLNXOFEDv541030/NVIDIA+PeerDirect. (2022).

[8] 2022. NVME-oF - NVM Express over Fabrics - MLNX_EN v5.1-1.0.4.0 - NVIDIA Networking Docs. https://docs.nvidia.com/networking/display/MLNXENv511040/NVME-oF+-+NVM+Express+over+Fabrics. (2022).

[9] 2022. Samsung Enterprise SSD 983 DCT M.2 960GB | MZ-1LB960NE | for Business. https://www.samsung.com/us/business/computing/memory-storage/enterprise-solid-state-drives/983-dct-960gb-mz-1lb960ne/. (2022).

[10] 2022. sFlow.org - Making the Network Visible. https://sflow.org/index.php. (2022).

[11] Shridhar Allagi and Rashmi Rachh. 2019. Analysis of Network log data using Machine Learning. In *IEEE I2CT (2019)*. IEEE.

[12] Tom Barbette, Chen Tang, Haoran Yao, Dejan Kostić, Gerald Q Maguire Jr, Panagiotis Papadimitratos, and Marco Chiesa. 2020. A high-speed load-balancer design with guaranteed per-connection-consistency. In *USENIX NSDI (2020)*.

[13] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. Pint: Probabilistic in-band network telemetry. In *ACM SIGCOMM (2020)*.

[14] Gaël Bernard and Periklis Andritsos. 2021. Selecting representative sample traces from large event logs. In *IEEE ICPM (2021)*. IEEE.

[15] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. 2021. Understanding host network stack overheads. In *ACM SIGCOMM (2021)*.

[16] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *ACM CCS (2017)*.

[17] Alessandro D'Alconzo, Idilio Drago, Andrea Morichetta, Marco Mellia, and Pedro Casas. 2019. A survey on big data for network traffic monitoring and analysis. *IEEE Trans. Netw. Service Manag.* 16, 3 (2019), 800–813.

[18] Paul Emmerich, Maximilian Pudelko, Sebastian Gallenmüller, and Georg Carle. 2017. Flowscope: Efficient packet capture and storage in 100 gbit/s networks. In *IEEE IFIP Networking (2017)*. IEEE.

[19] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. 1996. From data mining to knowledge discovery in databases. *AI magazine* 17, 3 (1996), 37–37.

[20] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *USENIX NSDI (2014)*.

[21] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience report: System log analysis for anomaly detection. In *IEEE ISSRE (2016)*. IEEE.

[22] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. Netcache: Balancing key-value stores with fast in-network caching. In *ACM SOSP (2017)*.

[23] Antti Juvonen, Tuomo Sipola, and Timo Hämäläinen. 2015. Online anomaly detection using dimensionality reduction techniques for HTTP log analysis. *Computer Networks* 91 (2015), 46–56.

[24] Manolis Karpathiotakis, Dino Wernli, and Milos Stojanovic. 2019. Scribe: Transporting petabytes per hour via a distributed, buffered queueing system. https://engineering.fb.com/2019/10/07/data-infrastructure/scribe/. (2019).

[25] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. 2015. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM (2015)*.

[26] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. 2020. Tea: Enabling state-intensive network functions on programmable switches. In *ACM SIGCOMM (2020)*.

[27] Bram Knols and Jan Martijn EM van der Werf. 2019. Measuring the behavioral quality of log sampling. In *IEEE ICPM (2019)*. IEEE.

[28] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael M Swift. 2021. ATP: In-network Aggregation for Multi-tenant Learning.. In *USENIX NSDI (2021)*.

[29] Cong Liu, Yulong Pei, Long Cheng, Qingtian Zeng, and Hua Duan. 2021. Sampling business process event logs using graph-based ranking model. *Concurrency and Computation: Practice and Experience* 33, 5 (2021), e5974.

[30] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. 2019. Distcache: Provable load balancing for large-scale storage systems with distributed caching. In *USENIX FAST (2019)*.

[31] Benjamin Marlin. 2008. *Missing data problems in machine learning*. Ph.D. Dissertation.

[32] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *ACM SIGCOMM (2017)*.

[33] Venkat Mohan, YR Janardhan Reddy, and K Kalpana. 2011. Active and passive network measurements: a survey. *Int. J. Comput. Sci. Inf. Technol.* 2, 4 (2011), 1372–1385.

[34] Kazuki Otomo, Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. 2021. Latent semantics approach for network log analysis: modeling and its application. In *IFIP/IEEE IM (2021)*. IEEE.

[35] Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, et al. 2021. Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches. In *ACM SIGCOMM (2021)*.

[36] Siyi Qiao, Chen Xu, Lei Xie, Ji Yang, Chengchen Hu, Xiaohong Guan, and Jianhua Zou. 2014. Network recorder and player: FPGA-based network traffic capture and replay. In *IEEE FPT (2014)*. IEEE.

[37] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan Ports, and Peter Richtarik. 2021. Scaling Distributed Machine Learning with In-Network Aggregation. In *USENIX NSDI (2021)*.

[38] Olivier Tilmans, Tobias Bühler, Ingmar Poese, Stefano Vissicchio, and Laurent Vanbever. 2018. Stroboscope: Declarative network monitoring on a budget. In *USENIX NSDI (2018)*.

[39] Juan Camilo Vega, Marco Antonio Merlini, and Paul Chow. 2020. FFShark: a 100G FPGA implementation of BPF filtering for Wireshark. In *IEEE FCCM (2020)*. IEEE.

[40] Tianzhu Zhang, Leonardo Linguaglossa, Massimo Gallo, Paolo Giaccone, and Dario Rossi. 2018. FlowMon-DPDK: Parsimonious per-flow software monitoring at line rate. In *IEEE TMA (2018)*. IEEE.

[41] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-level telemetry in large datacenter networks. In *ACM SIGCOMM (2015)*.