TIGA: Towards Efficient Near Data Processing in SmartNICs-based Disaggregated Memory Systems

Zhuohui Duan, Zelin Yu, Haikun Liu†, Xiaofei Liao, Hai Jin, Shijie Zheng, Sihan Wu
National Engineering Research Center for Big Data Technology and System/Services Computing Technology and System Lab/Cluster and Grid Computing Lab, Huazhong University of Science and Technology, Wuhan, 430074, China
{zhduan,yzljungle,hkliu,xfliao,hjin,zhengsj,wush}@hust.edu.cn

ABSTRACT

Memory disaggregation, facilitated by *Smart Network Interface Cards* (SmartNICs), has emerged as a cost-effective approach for sharing memory resources in data centers. However, current SoC-based SmartNICs face several challenges for supporting *near-data processing* (NDP) in disaggregated memory (DM) systems effectively, such as inefficient resource allocation for SmartNICs employed in NDP, and the lack of collaboration between SmartNICs on data nodes and CPUs on compute nodes. To address these issues, we propose TIGA, an efficient NDP framework for SmartNICs-based disaggregated memory systems. We propose an adaptive resource allocator to fully utilize the SoC cores among NDP engines automatically, and a SmartNIC-CPU cooperative computing mechanism to schedule NDP tasks among CPUs and SmartNICs. We prototype TIGA with FPGAs and evaluate it with several typical workloads. Experimental results show that TIGA significantly improves the efficiency of NDP tasks in DM systems compared with state-of-the-art SmartNIC-based co-processing schemes.

ACM Reference Format:
Zhuohui Duan, Zelin Yu, Haikun Liu†, Xiaofei Liao, Hai Jin, Shijie Zheng, Sihan Wu. 2024. TIGA: Towards Efficient Near Data Processing in SmartNICs-based Disaggregated Memory Systems. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3656244

1 INTRODUCTION

Modern data centers are constrained by monolithic server architectures and are still struggling in memory resource scaling for many memory-hungry applications. By disaggregating computing and memory resources, disaggregated memory systems provide a high degree of flexibility to scale computing and memory resources and significantly improve the performance of big data applications, and therefore has attracted increasing attention recently [6, 7, 15, 17]. Emerging SmartNICs, which integrate a large amount of on-board memory and computing units, can offer cost-effective and near-network computing power [7, 10, 15]. As a result, they are widely used as remote memory management units in DM systems [6, 7].

There has been a growing interest in using SmartNICs for NDP within DM systems. Many recent studies [6–8, 15] have demonstrated that SmartNICs enable a promising approach to NDP in DM systems. The SmartNIC-based NDP offers a shorter data path for memory accesses in DM systems, effectively reduces the number of network round trips (RTTs) and network bandwidth consumption, and also frees up the CPU resource on compute nodes (*CNs*). However, the utilization of SoC-based SmartNICs for NDP within DM systems still presents a multitude of challenges. The conventional time-sharing manner proves inadequate for SmartNICs engaged in NDP. Moreover, existing exclusive core partition mechanisms [5] exhibit a lack of meticulous consideration to application and traffic characteristics, which leads to substantial resource wastage. Lastly, notwithstanding the potential of NDP on SmartNICs to augment overall system performance, its efficiency may be diminished by the processing capacity constraints of the SmartNICs, thereby limiting its viability in managing high workload intensities.

To overcome the drawbacks of existing NDP schemes in SmartNIC-based DM systems, we propose TIGA, a high-performance NDP framework designed specifically for SmartNIC-based DM systems. We propose an adaptive resource allocator to best utilize the SoC cores among NDP tasks. By monitoring and analyzing the characteristics of NDP workloads, TIGA can automatically adjust the allocation of SoC cores for NDP tasks to improve the performance gain of NDP. We also propose a SmartNIC-CPU cooperative scheduler to dispatch NDP tasks among compute nodes' CPUs and remote SmartNICs, and further improve the performance of NDP tasks. We build a SmartNIC-based disaggregated memory system using FPGAs and prototype our TIGA framework on it. To validate the effectiveness and efficiency of our designs, we conduct extensive experiments to evaluate the performance of TIGA. Furthermore, we compare TIGA with state-of-the-art SmartNIC-based offloading schemes such as Floem [12], iPipe [10], and FairNIC [5]. Our experimental results demonstrate that TIGA can improve the performance of NDP on SmartNICs by up to 90%. Moreover, for overloaded scenarios, TIGA can improve the maximum throughput by 1.6× to 3.3× through SmartNIC-CPU co-processing.

2 BACKGROUND

The SmartNIC-based memory disaggregation approach deploys SmartNICs on the data nodes (*DNs*) and utilizes them for both memory management (including address translation [6], memory allocation [15], and page fault handling [6]) and network transmission operations (such as metadata messaging [7]). By organizing

†Haikun Liu is the corresponding author. This work is supported jointly by National Key Research and Development Program of China under grant No.2022YFB4500303, National Natural Science Foundation of China (NSFC) under grants No.62302178, 62332011, and China Postdoctoral Science Foundation under grants No.2023M731195, BX20230134.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3656244

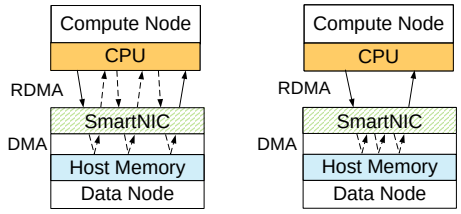

Figure 1: Different processing dataflows in SmartNIC-based disaggregated memory system

and managing memory on the SmartNIC, this solution provides an autonomous disaggregated memory system for the *CNs*, and thus simplifies the design of distributed computing systems, effectively reduces the *total cost of ownership* (TCO), and gives the computing power that is not typically available in traditional DM systems, which offers new opportunities for NDP in *DNs* [6, 7].

As shown in Figure 1(a), in a processing flow without NDP, the application on the *CNs* needs to fetch the required data from the *DNs* via network transfer, which incurs additional network round-trip latency. Naturally, NDP, designed to minimize redundant data movement, proves effective in reducing expensive network transmission overhead in SmartNIC-based DM systems. As shown in Figure 1(b), the processing task is posted by the *CNs* on the SmartNIC of the *DNs*, and the SmartNIC acquires all the necessary data for the current processing task through local DMA. Subsequently, the processing result is returned to the *CNs* by the SmartNIC. Utilizing SmartNIC for NDP also yields an effective reduction in network bandwidth consumption, thereby alleviating network congestion. Moreover, by relieving the CPU from the burden of complex control during remote data access, local computing tasks can optimally utilize the CPU resources on the *CNs*. These advantages have sparked a growing interest in adopting SmartNIC for NDP in DM systems.

Many researchers have recognized the potential of using SmartNICs on the *DNs* for NDP. For instance, LineFS [7] compresses network transmission data on the *DNs*' SmartNICs to reduce the network bandwidth consumption. Similarly, Clio [6] offloads a DataFrame-like data processing application to SmartNICs, and thus minimizes network round trips. Furthermore, Farview [8] offloads database query operations, such as selection, projection, aggregation, and regular expression matching to SmartNICs, and efficiently reduces the data transmission cost in database queries.

3 MOTIVATION

To evaluate the performance of NDP with SoC-based SmartNICs in DM systems, we conduct extensive experiments using the Mellanox BlueField-2 DPU as our testbed. This SmartNIC has been widely used by several resource disaggregation systems for task offloading [7, 10]. We measure the capability of SmartNICs in a client-server mode. The client is equipped with an InfiniBand RNIC Adapter (i.e., 100 GbE Mellanox ConnectX-4), and the host server is equipped with a NVIDIA BlueField-2 DPU using PCIe 3.0 ×16. Both the client and the server are equipped with a 2.2 GHz Intel Xeon(R) Gold 5117 CPU and 512 GB DDR4 DRAM.

Current SoC-based SmartNICs are typically equipped with multiple cores and are capable of concurrently executing various NDP

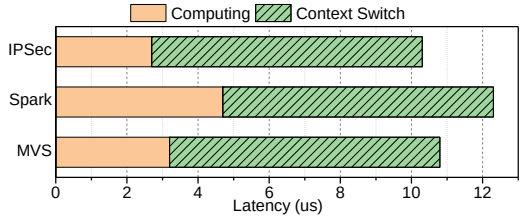

Figure 2: The latency of SoC computing and context switches for processing individual requests

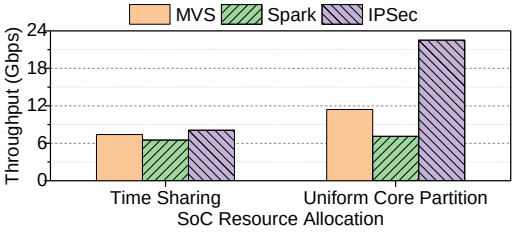

Figure 3: The throughput of different NDP applications under time-sharing scheme and uniform core partition

programs. Consequently, the SmartNIC's resource allocation mechanism plays a pivotal role in shaping the overall NDP performance. While the traditional approach of time-sharing cores across applications might enhance resource utilization, its context switches may incur substantial delays in packet processing.

To assess the contextual impact on different NDP tasks, we conduct a comprehensive analysis of processing latency breakdowns across three intricate NDP applications. The first application, known as *Multi-Version Storage* (MVS), mandates the *DNs* to commence with a CRC consistency check before proceeding to serialize the data during an MVS query. The second application involves *Spark-based data analysis* (Spark) [13], wherein the *DNs* undertakes data analysis prior to serialization to ensure uniform transmission and storage. The third application, termed IP Security (IPSec), entails the *DNs* utilizing SHA and AES engines for traffic verification and decryption [4].

Figure 2 presents a comprehensive breakdown of the processing latency about the previously mentioned applications. For MVS, Spark, and IPSec, the SoC cores are actively involved in factual computational operations for a mere 30%, 38%, and 26% of the overall latency, correspondingly. The predominant origin of latency in the execution of these applications emanates from the recurrent incidence of context switches, which account for a substantial portion of up to 70% of the total latency. Concurrently, Figure 3 offers an exhaustive visualization of NDP performance encompassing the two distinct resource allocation strategies: uniform core partition and time-sharing. In contrast to the time-sharing methodology, the uniform core partition scheme results in noteworthy improvements in the performance of MVS, Spark, and IPSec, leading to enhancements of 54%, 9%, and 177%, respectively.

Our investigation yields the following noteworthy observations: **The traditional time-sharing scheme is not suitable for SmartNICs to perform near-data processing.** *The core partition scheme demonstrates greater suitability for SmartNIC-based NDP compared to the time-sharing scheme. Our experimental results reveal that the time-sharing approach provided by traditional operating system*

introduces significant context switch overhead, rendering it less adaptable to NDP scenarios involving frequent processing of packet-level workloads. This overhead is exacerbated when multiple NDP engines operate concurrently on the SmartNIC. Conversely, the core partition scheme eliminates context switch overhead, making it a more appropriate choice for NDP tasks.

4 DESIGN

The operating system approach to computing resource management of existing SmartNICs significantly diminishes their potential benefits for NDP in disaggregated memory systems. In response to this challenge, we introduce TIGA, an efficient NDP framework for SmartNIC-based disaggregated memory systems. We design an adaptive core allocation mechanism and a co-processing mechanism, and utilize FPGAs to build a computational resource management infrastructure to implement the TIGA framework.

4.1 Adaptive SoC Resource Allocator

Many NDPs in DM systems require intricate processing, such as data analysis operations that consist of multiple components (e.g., checksum, security validation, and data processing [9]). It is common practice to use multiple independent processing engines organized in a pipeline to implement such services [1, 9].

The allocation strategy of cores among distinct NDP engines significantly influences the holistic performance of the NDP pipeline within the core-partition scheme. During the initial deployment of NDP pipelines, SmartNICs operate without foreknowledge of workload characteristics. Therefore, TIGA refrains from pursuing intricate static allocation strategies in its preliminary state. Instead, it advocates the equitable distribution of cores among NDP engines.

However, the uniform allocation method is insufficient to fulfill the efficiency requirements for different NDPs. TIGA strives to enhance the SoC utilization within SmartNICs by implementing an adaptive core reallocation mechanism. Nevertheless, the efficiency of data processing on SmartNICs is notably influenced by the distinct characteristics of applications and network traffic. This complexity hampers the formulation of a core reallocation scheme solely based on an analysis of these performance-affecting factors.

The efficiency of data processing on SmartNICs can be appraised using a straightforward metric, namely, the discrepancy between outbound and inbound throughputs. In case the outbound throughput is lower than the inbound throughput, it indicates that the sustained processing capacity of the SmartNIC is not able to meet the resource prerequisites of the workloads. In such instances, TIGA dynamically adapts the allocation of SoC resources to enhance the throughput of SmartNICs, ensuring efficient data processing.

Figure 4 depicts the framework underpinning our task scheduling and SoC resource allocation within the TIGA system. To facilitate the management of SoC resources, we abstract the program, computing resources, and runtime information of a data processing task as a *Processing Agent* (PA) and use it for dynamic SoC resource allocation. A PA includes multiple SoC cores to execute a single *processing engine* (PE), such as network functions and microservices. Moreover, it contains a packet dispatcher to store incoming packets in a buffer queue and then dispatch them to each SoC core, as well as a set of performance counters.

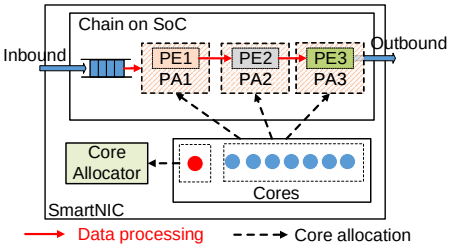

Figure 4: The core allocation framework in TIGA. PA and PE denote processing agent and processing engine, respectively

The performance bottleneck within a processing chain is typically attributed to the slowest PA, thereby delineating the lowest throughput [18]. The optimal performance of a processing chain is inherently achieved when all PAs exhibit uniform throughput. Consequently, the endeavor of core reallocation should primarily aim to equalize the throughput across all PAs. Let C_{agent_i} denote the number of SoC cores allocated to PA_i, C_{total} denote the total number of cores in the SoC, and P_{agent_i} denote the processing capacity (or throughput) of a single core in PA_i. P_{agent_i} represents the total network traffic processed by a single core in a period of CPU time, and can be calculated by referring to the performance counters in each PA. Thus, the throughput of PA_i (i.e., T_{agent_i}) can be calculated by Equation 1.

$$T_{agent_i} = C_{agent_i} * P_{agent_i} \tag{1}$$

Our goal is to balance the throughput of each PA in the pipeline, then for any PA_i and PA_j, we have Equation 2.

$$C_{agent_i} * P_{agent_i} = ... = C_{agent_j} * P_{agent_j} \tag{2}$$

With Equation 1 and Equation 2, we can derive the allocation proportion of SoC cores for different PAs. By summing all cores allocated to each PA, we have Equation 3,

$$\begin{aligned} C_{total} &= C_{agent_1} + ... + C_{agent_n} \\ &= \frac{C_{agent_i} * P_{agent_i}}{P_{agent_1}} + ... + \frac{C_{agent_i} * P_{agent_i}}{P_{agent_n}} \\ &= C_{agent_i} * P_{agent_i} * (\frac{1}{P_{agent_1}} + ... + \frac{1}{P_{agent_n}}) \end{aligned} \tag{3}$$

Finally, we can derive Equation 4, which calculates the allocation ratio of SoC cores (C_{agent_i}) for PA_i.

$$C_{agent_i} = \frac{C_{total}}{P_{agent_i} * (\frac{1}{P_{agent_1}} + ... + \frac{1}{P_{agent_n}})} \tag{4}$$

We use the ratio of the total amount of data processed by a single core from the start of the last core reallocation to the present and the total CPU time spent by the core to process this data, to calculate the SoC processing power of the PA. When reallocating SoC cores, the core allocator first estimates the processing capacity of each SoC core for each PA based on the performance counters, and then calculates the allocation ratio of SoC cores. The allocator always selects an over-allocated PA, and reallocates the remaining SoC cores to other PAs. The scheduler blocks the inflow of the pipeline for these cores and unpins them from the PA upon completion of the current task. Finally, the scheduler sequentially allocates the remaining cores to PAs that require more cores. The scheduler forks the processes on the original core and pins them to the new core,

which does not block the processing of other PAs and minimizes the number of context switches.

However, when a significant change in payload size occurs, the computational capacity of the SmartNIC is insufficient. This consequence gives rise to congestion within the network's reception queue, leading to a mismatch between the outbound and inbound bandwidths. To address the potential problem of excessive reallocations due to dynamically shifting workloads, TIGA undertakes regular monitoring of the SmartNIC's counters to ascertain the occurrence of bandwidth mismatches.

The reallocation period constitutes a critical factor influencing the efficacy of this optimization strategy. As a resource reallocation approach, a shorter detection period can promptly identify minor fluctuations in the load, ensuring the strategy's responsiveness to rapid load changes. Nevertheless, TIGA's SoC core reallocation and periodic detection sampling introduce a substantial performance overhead, and the frequent triggering of SoC core reallocation may nullify or even surpass its performance benefits. To evaluate the overall overhead imposed by detection and reallocation, we measure the overhead of a single detection operation and reallocation, and based on this we calculate the overhead at different detection frequencies under the different load change intervals.

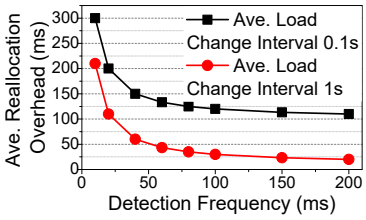

Figure 5: Ave. overhead of reallocation for different load variations and detection frequencies

When the detection period is 0.01s, the average overhead per second is up to 0.21s, which wipes out the performance gains from core reallocation. On the contrary, excessively reducing the detection frequency not only fails to reduce the overall overhead, but also prevents the SmartNIC from sensing the workload changes in time. Hence, in consideration of achieving a balance between timely response to reallocation requirements and maintaining a manageable monitoring overhead, we establish the default detection period at 0.1s. To facilitate users' convenience and operational flexibility, we provide preparatory scripts designed for setup and profiling. These tools empower developers to customize the detection period in alignment with the specific contextual needs of their respective scenarios.

4.2 SmartNIC-CPU Cooperative Processing

The acceleration gain of NDP in a SmartNIC-based DM system is affected by the relationship between the processing power of the SmartNIC and the allocated NDP pressure. To address this issue, TIGA optimizes processing performance by adjusting the allocation of SoC cores when SmartNIC performance is insufficient to meet the current demand for near-data processing. However, when the NDP workloads of SmartNIC far exceed its computational capacity limit, TIGA frequently performs ineffective SoC reallocation due to its inability to match outbound throughput with inbound throughput through the SoC reallocation policy. We propose an active workload shifting approach, i.e., SmartNIC should forward the workload that exceeds its processing capacity to the CPU on the *CNs* for execution to address the over-computation allocation caused by the existing NDP logic. To engage this mechanism, users are required to furnish TIGA with applications that can execute on *CNs*.

Upon the identification of three consecutive instances characterized by mismatches between inbound and outbound bandwidths, coupled with three successive and identical reallocations (i.e., the same reallocation ratio), the core allocator within TIGA draws the inference that the extant performance requisites of the NDP cannot be fulfilled through core reallocation alone. TIGA adjusts the ratio of requests to be executed on the SmartNIC and CPU by forwarding a certain ratio of requests to the CN-side CPU for execution, and thus utilizing the CN-side CPU to further increase throughput. In the subsequent cycle, TIGA initiates the transmission of a certain proportion of requests to the *CNs* for processing. Specifically, TIGA embarks by redirecting 10% of the incoming requests to the *CNs*, manifesting as a forwarding operation every ten received requests. Should the SmartNIC persistently encounter bandwidth disparities after a single cycle, TIGA incrementally escalates the forwarding ratio by 10%. This escalation continues iteratively until an equilibrium is achieved, where inbound and outbound bandwidths are matched. How much the forwarding ratio is escalated each time depends on the user's trade-off between utilizing CNs to reach the optimum throughput quickly and taking more advantage of the SmartNIC's NDP to reduce the computational pressure. The ratio can also be customized by the user in the profile, and we set the ratio to 10% by default.

TIGA endeavors to diminish the forwarding ratio once this ratio stabilizes and the SmartNIC experiences no inbound/outbound bandwidth discrepancies. This is pursued to fully harness the SmartNIC's potential for NDP. Should the SmartNIC refrain from instigating any reallocation across three consecutive detection cycles, it signals that the performance enhancement achieved through reallocation has sufficed to address the current workload. TIGA believes that there is still spare capacity within the SmartNIC to accommodate additional requests, prompting an attempt in the subsequent cycle to curtail the forwarding ratio by 10%. Should the following cycle similarly exhibit no bandwidth disparities, TIGA persists in diminishing the forwarding ratio by 10%, continuing this trend until either all NDP requests are executed on the SmartNIC or a bandwidth discrepancy emerges. Subsequent to these adjustments, TIGA recommences the identification of successive uniform reallocations, ensuring prompt activation of the *CN* co-processing mechanism in instances of inadequate performance.

4.3 TIGA Implementation

We build a software-hardware co-designed resource management architecture using FPGAs and implement TIGA on top of it. There are two benefits to using hardware rather than OS for the implementation of a core partition resource management scheme. Primarily, the further introduction of core partition encapsulation on OS-level may inadvertently incur additional overheads. Secondly, the existing SoC-based SmartNICs lack the necessary programmable packet scheduling mechanisms. Resorting to software-based packet scheduling engrosses a significant portion of the SoC resources. The implications of these two challenges notably impact the effectiveness of the core partition scheme.

We implement 8 RISC-V cores on FPGAs and package them with caches and performance counters to implement our resource management units (i.e., PA). The core we use has a five-stage pipeline with a single level of cache, 250 MHz frequency, and 512-bits memory data width. For inter-core communication, we leverage the interconnect approach of PsPIN [3], using the AXI interconnect, where each core is both the master and slave port of the transmission. We interconnect the on-board memory, the SoC cores, and the DMA engine. The data source of DMA request is specified in the command issued by the handlers. The data is moved by DMA engine that translates the virtual address specified in the handler command into a physical address, and drives the DMA access.

To efficiently schedule packets between cores, we implement a single-stage Match+Action (RMT) pipeline [2] on the FPGA with an additional management module for user registration and configuration of their NDP applications. The RMT pipeline maintains a flow table in which each flow is assigned an offloading chain. The frequency of the RMT pipe is 250 MHz. In order to streamline the process of both developing and utilizing NDP applications, TIGA employs data labeling techniques commonly utilized in SmartNIC development [6, 9, 12]. TIGA mandates developers to register the NDP application deployed on the SmartNIC to the RMT pipeline before leveraging SmartNIC for NDP. The *CN* side application invokes the requisite NDP application by explicitly calling the TIGA API and providing the assigned NDP type flag. Additionally, the NDP coordinator situated on the *CN* side acknowledges forwarded requests through the corresponding API.

5 EVALUATION

We implement and evaluate TIGA with FPGAs [16], and compare it with state-of-the-art SmartNIC processing frameworks.

5.1 Testbed and Methodology

Experimental setup. We use two physical machines as the *CN* and *DN*. Each node is equipped with two Intel Xeon Gold 5117 28-core CPUs and 512GB memory. We implement our TIGA prototype using Xilinx Alveo U250 data center accelerators on the *DN*, and develop several NDP applications on it.

Systems for comparison. We compare TIGA with several state-of-the-art SmartNIC processing schemes, such as Floem [12], iPipe [10], and FairNIC [5]. Floem is a programming system that aims to simplify the SmartNIC offloading programming process. It runs stationary programs on the SmartNIC in a time-shared manner. iPipe executes processing engines using an actor model and migrates the slowest actor back to the host when the SmartNIC cannot achieve the highest throughput. FairNIC statically and evenly allocates all SoC cores among different processing engines. To ensure consistency in hardware architecture, we implement these SmartNIC processing schemes on our FPGA. For the time-sharing scheme, we implement automatic context switch in FCFS and DRR modes on the FPGA's RISC-V cores. To avoid the unfairness of CPU co-processing, we implement a dedicated system called TIGA-without CPU, which disables the CPU co-processing policy of TIGA. This allows us to evaluate the performance of TIGA based solely on SmartNIC, while still leveraging the core reallocation mechanism.

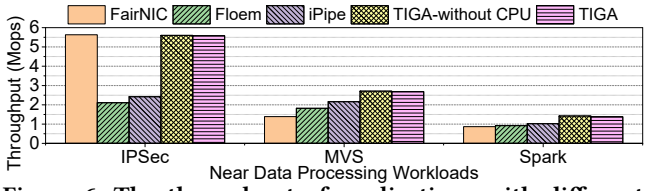

Figure 6: The throughput of applications with different SmartNIC processing schemes

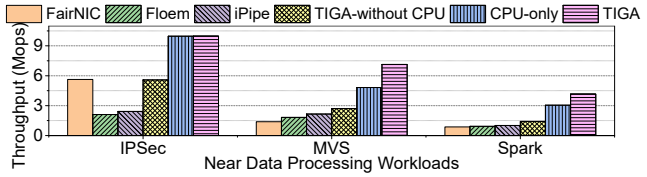

Figure 7: The throughput of applications under high load

Benchmarks. We evaluate TIGA with the IPSec, MVS, and Spark NDP applications mentioned in Section 3. Without any special emphasis, we evaluate the performance of these NDP applications using a default payload size of 1024B.

5.2 Overall Performance

Figure 6 illustrates the performance of three NDP applications using different SmartNIC processing schemes. TIGA demonstrates a significant improvement in the performance of MVS, achieving gains of 89%, 44%, and 21% compared to FairNIC, Floem, and iPipe, respectively. Specifically, the CRC checksum engine and SER engine have significantly different performance when executed on a single SoC core. In this case, FairNIC's evenly allocation strategy leads to significant resource wastage. Floem and iPipe improve the utilization of computational resources by using cores in a time-sharing manner, but they also impose the overhead of context switch. In comparison, iPipe has a more reasonable scheduling mechanism and therefore has relatively better performance. The SoC reallocation mechanism of TIGA achieves more efficient allocation of cores between different processing engines, eliminates the cost of context switch, and therefore achieves the best performance. With the same benefit of reallocation mechanisms, TIGA demonstrates an up to 58% improvement in the performance of Spark and an up to 164% improvement in the performance of IPSec, respectively. In particular, the IPSec performance of TIGA is slightly degraded compared to FairNIC. This is because the optimal core allocation ratio for IPSec is exactly 1:1, and TIGA has an extra detection overhead.

We also evaluate the performance of TIGA by disabling the CPU collaboration mechanism. The performance of TIGA slightly improves when disabling the mechanism. When the system is under a moderate load, both the CPU and SmartNIC can handle all data processing requests. At this point, the CPU collaboration mechanism cannot increase throughput by processing requests in parallel. However, the migration mechanism still needs to periodically check the overall throughput and determine whether to make proportional adjustments, which increases the overhead slightly.

5.3 Stress Testing

When the load intensity greatly exceeds the SmartNIC's capacity, utilizing the SmartNIC for fixed NDP will not continue to improve

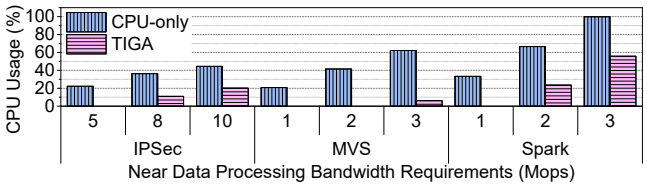

Figure 8: The CPU usage for different throughputs

performance. To assess the benefits of SmartNIC co-processing under pressure, we conduct tests using 10 Mops request traffic (in our measurements, the maximum throughput achievable by SmartNIC remains below 10 Mops). We also evaluate a CPU-only solution, which employs the CPU on *CN* to handle all data processing requests without involving any NDP application on the SmartNIC.

Figure 7 illustrates the performance of three NDP applications using different SmartNIC processing schemes under pressure. While the use of SmartNIC for NDP does provide some benefits, its inadequate processing power still limits its performance under stress scenarios. As a result, the CPU-only solution yields a significant performance advantage over the SmartNIC-only solution, with an improvement of up to 255%. TIGA addresses this issue through the SmartNIC-CPU collaboration mechanism. Compared to the CPU-only solution, TIGA further enhances the performance of MVS and Spark by 48% and 36%, respectively. Since the CPU is sufficient to handle 10Mops of IPSec traffic, TIGA's collaboration mechanism does not result in an increase in throughput. Moreover, compared to the TIGA-without CPU solution, TIGA enhances the overall performance by anywhere from 1.6x to 3.3x. These outcomes demonstrate that TIGA's SmartNIC-CPU collaboration mechanism is effective in ensuring the overall system performance under stress load.

5.4 CPU Usage

In overloaded conditions, TIGA optimizes the utilization of CPU and SmartNIC resources for data processing. However, data processing pressure may exceed the maximum processing capacity of SmartNIC while still remaining within the processing capacity of the entire system. In such cases, SmartNICs should handle as much load as possible without compromising performance and, therefore, reduce CPU overhead. TIGA forwards excess requests from the SmartNIC to the CPU, which can fully utilize the performance gains provided by NDP while ensuring the overall system throughput. Figure 8 depicts the CPU usage of TIGA for different throughputs. Compared to the CPU-only, TIGA reduces CPU overhead by 22% to 56% for the same throughput. In particular, while using only CPUs can balance IPSec throughput well, TIGA effectively reduces the CPU usage of CN by 25%. The above experiments demonstrate that the SmartNIC-CPU collaborative mechanism fully utilizes SmartNIC's NDP capabilities under various load intensities.

6 RELATED WORK

SmartNIC SoC resource allocation. It is essential to carefully think about how to fully utilize the relatively rich on-chip resources on SmartNIC. Some researches sidestep this issue by assigning tasks to CPUs [11]. In comparison, the goal of TIGA is to appropriately allocate cores to multiple stages (i.e., multiple processing agents) of a complex application. Within each processing agent, TIGA employs mechanisms proposed in these works to improve performance.

Collaboration between SmartNICs and CPUs. Some studies have explored the distribution of applications between SmartNIC and CPU. SmartChain [1] uses 0-1 integer linear programming model to direct the offloading. Clara [14] exploits a machine learning approach to offer offloading strategies. These studies have explored which applications are suitable for offloading to SmartNIC for execution. Based on their findings, developers can targetedly use TIGA framework for NDP.

7 CONCLUSION

In this paper, we propose TIGA, a novel and efficient NDP framework designed for SmartNIC-based disaggregated memory systems. Several innovative designs are integrated into TIGA to fully exploit the SmartNIC's potential for NDP. Our evaluation demonstrates that TIGA achieves significant improvements in data processing performance over existing SmartNIC co-processing schemes.

REFERENCES

[1] Gaurang Bansal, Amit Dua, Gagangeet Singh Aujla, Maninderpal Singh, and Neeraj Kumar. 2019. SmartChain: A Smart and Scalable Blockchain Consortium for Smart Grid Systems. In *Proceedings of ICC Workshops' 19.* 1–6.
[2] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 99–110.
[3] Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beránek, Luca Benini, and Torsten Hoefler. 2021. A RISC-V In-Network Accelerator for Flexible High-Performance Low-Power Packet Processing. In *Proceedings of ISCA' 21.* 958–971.
[4] Naganand Doraswamy and Dan Harkins. 2003. *IPSec: the New Security Standard for the Internet, Intranets, and Virtual Private Networks.* Prentice Hall Professional.
[5] Stewart Grant, Anil Yelam, Maxwell Bland, and Alex C Snoeren. 2020. SmartNIC Performance Isolation with FairNIC: Programmable Networking for the Cloud. In *Proceedings of SIGCOMM' 20.* 681–693.
[6] Zhiyuan Guo, Yizhou Shan, Xuhao Luo, Yutong Huang, and Yiying Zhang. 2022. Clio: A Hardware-software Co-designed Disaggregated Memory System. In *Proceedings of ASPLOS' 22.* 417–433.
[7] Jongyul Kim, Insu Jang, Waleed Reda, Jaeseong Im, Marco Canini, Dejan Kostić, Youngjin Kwon, Simon Peter, and Emmett Witchel. 2021. LineFS: Efficient SmartNIC Offload of a Distributed File System with Pipeline Parallelism. In *Proceedings of SOSP' 21.* 756–771.
[8] Dario Korolija, Dimitrios Koutsoukos, Kimberly Keeton, Konstantin Taranov, Dejan Miloji, and Gustavo Alonso. 2021. Farview: Disaggregated Memory with Operator Off-loading for Database Engines. *arXiv:2106.07102* (2021).
[9] Jiaxin Lin, Kiran Patel, Brent E. Stephens, Anirudh Sivaraman, and Aditya Akella. 2020. PANIC: A High-Performance Programmable NIC for Multi-tenant Networks. In *Proceedings of OSDI' 20.* 243–259.
[10] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. 2019. Offloading Distributed Applications onto SmartNICs Using iPipe. In *Proceedings of SIGCOMM' 19.* 318–333.
[11] Sarah McClure, Amy Ousterhout, Scott Shenker, and Sylvia Ratnasamy. 2022. Efficient Scheduling Policies for Microsecond-Scale Tasks. In *Proceedings of NSDI' 22.* 1–18.
[12] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. 2018. Floem: A Programming System for NIC-Accelerated Network Applications. In *Proceedings of OSDI' 18.* 663–679.
[13] Baiyou Qiao, Bing Hu, Junhai Zhu, Gang Wu, Christophe Giraud-Carrier, and Guoren Wang. 2020. A Top-K Spatial Join Querying Processing Algorithm Based on Spark. *Information Systems* 87 (2020), 101419.
[14] Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, and Ang Chen. 2021. Automated SmartNIC Offloading Insights for Network Functions. In *Proceedings of SOSP' 21.* 772–787.
[15] David Sidler, Zeke Wang, Monica Chiosa, Amit Kulkarni, and Gustavo Alonso. 2020. StRoM: Smart Remote Memory. In *Proceedings of EuroSys' 20.* 1–16.
[16] TIGA. 2024. https://github.com/CGCL-codes/TIGA. (2024).
[17] Shin-Yeh Tsai, Yizhou Shan, and Yiying Zhang. 2020. Disaggregating Persistent Memory and Controlling Them Remotely: An Exploration of Passive Disaggregated Key-Value Stores. In *Proceedings of ATC' 20.* 33–48.
[18] Seongdae Yu, Seongbeom Park, and Woongki Baek. 2017. Design and Implementation of Bandwidth-Aware Memory Placement and Migration Policies for Heterogeneous Memory Systems. In *Proceedings of SC'17.* 1–10.