Palos: Fair and Flexible Flow Scheduling on RNIC

Zhenlong Ma*†, Fan Yang*, Ning Kang*, Jing Xu*†, Guojun Yuan*,
Zhan Wang*, Ninghui Sun*
*Institute of Computing Technology, Chinese Academy of Sciences
†University of Chinese Academy of Sciences

***Abstract*—In recent years, Remote Direct Memory Access (RDMA) has gained significant attraction within modern hyper-scale data centers. However, RNIC fails to provide fine-grained performance isolation among network flows with different traffic patterns which co-exist in multi-tenant data centers and typically have various bandwidth, throughput and latency requirements.**

In this paper, we reveal that the drawbacks on isolation root in the packet-level flow scheduling mechanism implemented in the RNIC hardware. To solve this problem, we introduce Palos, a fair and flexible flow-scheduling mechanism. In the hardware layer, Palos adopts a data chunk based scheduling mechanism by reconstructing communication descriptors. The data chunk based scheduling diminishes the performance interference between large flows and small flows. Palos configures the scheduler in the software layer using a hierarchical weight setting to enable customized performance policy while preventing the configuration of users from interfering each other. Our experiments demonstrate that Palos provides better performance isolation and performance control flexibility compared with the commodity RDMA NIC and existing optimization framework.

***Index Terms*—Remote Direct Memory Access, Datacenter Network, Quality of Service, Network Interface Card**

I. Introduction

Remote Direct Memory Access (RDMA), which bypasses the operating system kernel during data transmission, has been widely deployed as a novel communication technology. RDMA offloads the entire network stack to the network interface card (NIC), enabling direct data exchange between local and remote host memories without kernel involvement, and hence offers high throughput, low latency with low CPU overhead. Built on RDMA network, various communication systems have been developed to support a wide range of applications [1]–[3].

RDMA faces far more diverse traffic patterns and requirements in data centers. RDMA was originally crafted to optimize the network performance in High Performance Computing (HPC) systems, which are built for a specific set of applications, owned by a limited number of users, optimized at a full-system level and undertake predictable workloads [4], [5]. In contrast, data centers support much more variable and unpredictable workloads. For instance, of all messages in Google's datacenter, 30% are smaller than 100 bytes while about 5% are larger than 10KB [6]. Besides, each node in data centers needs to serve tens of individual tenants typically having various performance objectives [7], and each tenant also intends to customize its own performance rule for network flows [8], [9]. To meet the diverse requirements raised in such heterogeneous communication, RDMA must provide flexible and predictable network capability to communication instances. Specifically, the performance of a communication instance—a flow, a tenant, or an application—should be determined solely by the rules established by users. And it should not be affected by the communication patterns, such as message sizes, of other instances.

However, the existing implementations of RDMA cannot meet the objects above because RDMA NIC (RNIC) encounters severe unfairness and interference issues in heterogeneous communication scenarios. Our experimental findings illustrated in Fig. 2 and Fig. 3 indicate that large message flows dominate the network and hurt the throughput and latency of small message flows, so that the isolation rule is violated. Without isolation, it is impossible to achieve predictable network performance because other communication flows are unpredictable. Worse still, RDMA bypasses the operating system, which makes traditional traffic control tools like Linux tc [10] inapplicable.

Existing works tried to implement transmission control to achieve network isolation, but these works have drawbacks. Works on the software like Justitia [11], Harmonic [12] and PeRF [13] require a daemon to continuously monitor the performance of each flow and dynamically adjust the transmission rate, therefore inducing additional CPU overhead and latency which violates the intention of RDMA. Hardware optimization works [14], [15] achieve high efficiency, but they require much storage resources for each performance level.

This paper starts solving the isolation issue with experiments and analysis based on the most widely adopted RNIC, Nvidia ConnectX-5 NIC [16]. We find that the problem roots in the scheduling mechanism, which fails to protect the performance of small flows. Therefore, we claim to adopt a new scheduling mechanism in which each network flow fairly transmits a certain size of data in each scheduling iteration, and users can customize performance allocation by adjusting the scheduling weight for each flow.

Implementing such scheduling mechanism in hardware must take several key challenges into consideration. First, RNIC is a complex and tightly-coupled module and undertakes multiple offloaded functions like reliable transmission [17], address translation [18] and connection management [1], [17], [19]. To minimize the complexity of hardware modifications, the scheduling module should be loosely-coupled with the core module of the RNIC. Second, it is difficult to provide large number of performance levels by hardware scheduling due to the limited resource and inflexibility nature of the hardware.

Third, considering that all users share the same hardware and have the ability to configure the scheduler, it is crucial to prevent the performance of one user from being influenced by the configuration of others.

To address the challenges above, we propose Palos, a data chunk based scheduling mechanism in RNIC, which achieves the isolation and performance customization of network flows and tenants. On the hardware, Palos provides a standalone hardware scheduling module and does not need any modification on the RDMA processing component. The scheduling module slices and combines network messages into fixed-sized chunks. The software part calculates the chunk size for each flow based on a hierarchical weighted sharing algorithm. Our experiments show that within Palos, large flows and small flows fairly share the same RNIC, and the traffic patterns and configurations of flows and tenants do not interfere each other. Also, the performance of each flow is controlled precisely by manipulating the chunk size.

In summary, our contributions in this paper are as follows:

1) We demonstrate the isolation flaws in the RNIC and explain that the cause of the interference problem roots in the unfair scheduling mechanism in the RNIC hardware.
2) We present Palos, a flexible flow scheduler sitting alongside the RDMA processing module. It schedules flows based on data chunk, which guarantees the isolation between different types of flows.
3) We provide a simple configuration interface to achieve hierarchical weighted performance allocation through manipulating the data chunk size for each flow.
4) We implement Palos on a Gem-5 based cycle-accurate RNIC simulator and evaluate its effect. The experiment results show that Palos achieves ideal isolation and flexibility for flows and tenants.

II. Background and Motivation

A. RDMA and RNIC

Traditional TCP/IP network copies data from user space to kernel space in the sender and conversely in the receiver, because data needs to pass through the kernel protocol stack which takes at least tens of microseconds [20]. In contrast, RDMA eliminates these cumbersome data copies and context switches by offloading the entire network stack into the hardware. In RDMA, a network flow is instantiated as a Queue Pair (QP) containing descriptors called Work Queue Elements (WQEs). WQEs record the location and access permission of the message data to be transferred.

An RDMA NIC (RNIC) is logically divided into several components: an RDMA protocol engine, a context manager, an address manager, and a hardware transport layer. Figure 1 illustrates the workflow of a RDMA operation. Firstly, the user application places WQEs into the Send Queue of the QP in the host memory(①). Next, the driver writes a doorbell into a register on the RNIC to signal a new communication task(②). The RNIC then retrieves the WQEs based on the information in the doorbell(③) and DMAs the entire messages from the host memory(④). Then the Transmission Engine packetizes the messages in multiple small-sized data packets into the network. The receiver's RNIC does the opposite to store the messages into the host memory.

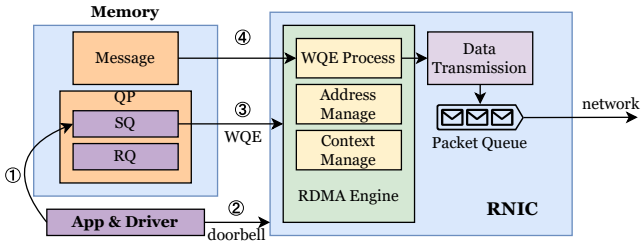

Fig. 1: RDMA NIC workflow

B. RNIC Isolation Problem

Heterogeneous environments like data center servers undertake unpredictable applications deployed by multiple individual tenants. Therefore, traffic patterns in data centers exhibit significant diversity. Message sizes across different types of applications can vary thousands or even tens of thousands times [6]. Furthermore, these tenants typically have different performance objects [7] and the applications they deploy also intend to customize performance allocation strategies to achieve optimized efficiency [8], [9]. Consequently, it is critical for datacenter network to provide predictable performance sharing with strong isolation and flexibility.

Unfortunately, RDMA cannot achieve the objects above due to significant unfairness and isolation issue. Network flows (i.e. QPs) with different message sizes interfere the performance of each other and large message flows tend to dominate the RNIC and starve the co-located flows transmitting small messages. In our experiment based on ConnectX-5 RNIC shown in Fig. 2, we found that when 16 flows transmitting 64 byte messages launches communication along with one flow transmitting 2 MB messages, the message rate of small flows decreases to 45% of the peak value while the large flow can keep 67% of bandwidth. It is unfair that the small flows overnumber the large flow but suffer much more degradation. In addition, the performance of both types of flows changes with the size of small messages, which means the performance of flows has no predictability. The same problem also happens on latency. In Fig. 3, the latency of the small flow increases with the number of co-located flows. 24 large flows can deteriorate the small flow's latency by 7.8x. Apparently, without the isolation as basic, predictability is impractical.

The root cause of this problem is that the flow scheduling mechanism implemented in the commodity RNIC does not arbitrate flows in a fair manner. The RNIC adopts the packet Round-Robin scheduling mechanism. In each schedule iteration, every flow sends the same amount of packets, regardless of the packet sizes. This mechanism is fair only in the case that all flows send messages larger than the MTU (Maximum

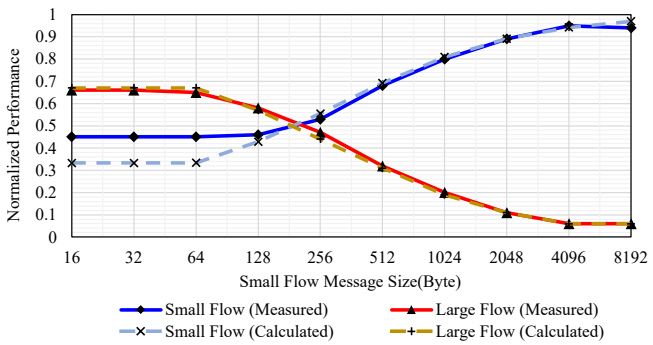

Fig. 2: Interference between one large flow and 16 small flows on ConnectX-5 NIC as small message size changes. The dashed line is our calculation results based on packet Round-Robin scheduling. We test the bandwidth and message rate of large flows and small flows respectively. The results are normalized relative to their peak values.

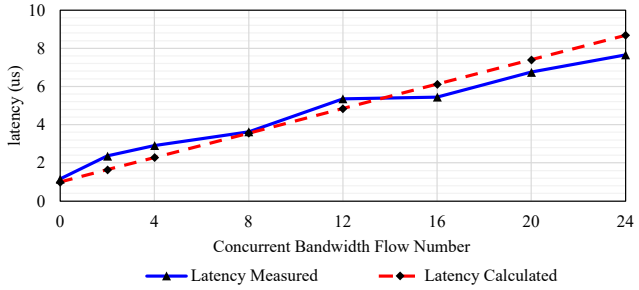

Fig. 3: Interference between one small flow and different number of concurrent large flows.

Transmission Unit), which is the size limit of one packet. In the scenario that messages are smaller than MTU, their transmission rates are determined by their message sizes. We calculated the performance results based on the packet Round-Robin scheduling and they perfectly match with our experiment results in Fig. 2 and Fig. 3, which confirms our analysis. Considering the MTU of RDMA over Converged Ethernet (RoCE) is typically 1500 bytes and 60% of messages in the Hadoop cluster of Facebook are smaller than 1 KB [21], this problem is a common case in datacenters.

III. Design Consideration

To enhance the predictability of RDMA in datacenters, it is necessary to raise a flexible and fair scheduling mechanism among different types of network flows to replace the conventional packet Round-Robin scheduling. Specifically, this scheduling mechanism should satisfy the following objects:

1) **Flexibility.** The scheduling mechanism should provide hierarchical performance control to allow both system administrators and tenants to adjust parameters according to their preferences. System administrators should be able to allocate different performance shares among tenants, and tenants should be able to distribute them among applications or flows.
2) **Isolation.** The performance of a flow or a tenant should be determined exclusively by the preset performance configuration rules and should not be influenced by traffic patterns of other flows or tenants.
3) **Low Overhead.** RDMA aims to achieve ultra-low software overhead and latency, which is the reason why RDMA gets favored. Therefore the scheduling mechanism should have minimal significant software overhead or time delay.

Reaching the objects above faces several key challenges. First, to achieve low CPU overhead, we need to implement the scheduling mechanism on the RNIC hardware. However, compared with traditional NIC which is only in charge of packet transmission, RNICs integrate more function like reliable transmission [17], address translation [18] and connection management [1], [17], [19]. The hardware complexity increases the difficulty of modification. Moreover, some vendors provide the entire RNIC as a "black-box" IP [22] which cannot be modified. Therefore, the scheduling mechanism should be a loosely-coupled component and avoid modifying the core part of the RNIC. Second, to support flexibility, the scheduling mechanism allows both system maintainers and tenants to customize performance rules for each flow. Different from software with rich host memory capacity and flexible computation logic, it is a hard job for the hardware to provide numerous diverse performance levels. For instance, existing hardware works [14], [15] use dedicated hardware queues for every scheduling instance which needs large amount of storage resource, and the algorithm used by software works [11], [23] are complex and hard to implement on hardware. Third, the scheduler hardware and the configuration is commonly shared and all users can set their own performance policy based on their preferences. In this case, it is essential to avoid the performance of tenants from being influenced by the configuration of each other or even malicious users.

IV. Palos Overview and Design

We propose Palos, an innovative RDMA connection scheduling mechanism. Through the redesigned scheduling system, we completely avoid the unfairness and interference resulting from the greedy large messages mentioned before and ensure improved throughput and reduced latency for small flows. In addition, our approach allows both system maintainers and users to regulate the data transfer rate in a fine-grained manner without hurting the performance isolation.

Within the Palos framework, communication flows are categorized into multiple distinct groups, with each group representing either a tenant in multi-tenancy systems or an application in mono-tenancy systems. Our design comprises hardware and software component. The hardware part is responsible for executing the message slicing and scheduling task, which involves segmenting sizable messages into multiple smaller data chunks within the critical data plane path to enable fine-grained scheduling. The software control path determines the message slicing size based on performance configurations specified by users, thereby achieving high flexibility.

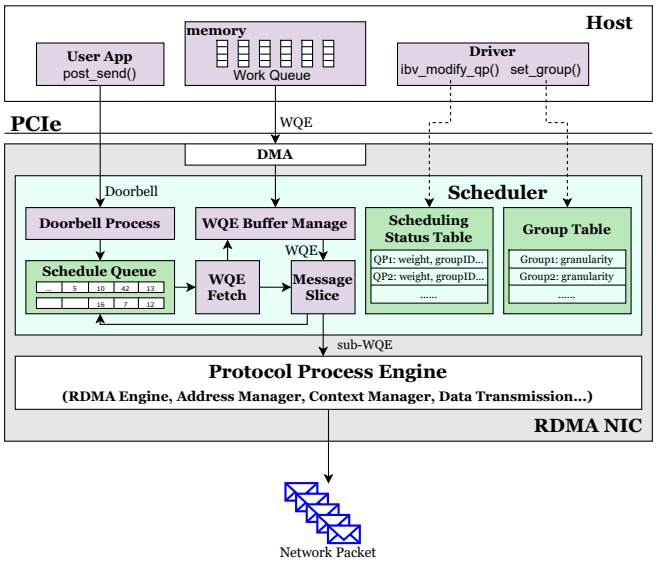

Fig. 4: Hardware architecture of Palos.

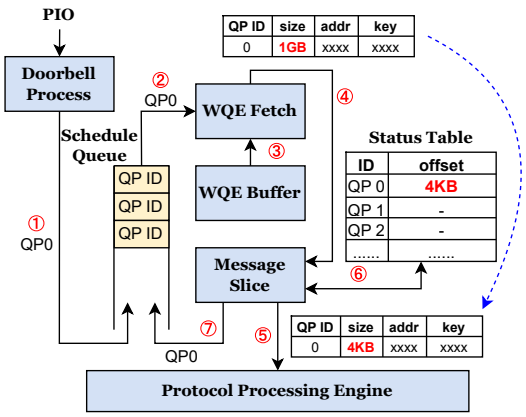

Fig. 5: Palos workflow explanation. This figure takes slicing a 1 GB message from QP 0 into 4 KB messages as an example. If the message is smaller than 4 KB, the Message Slice module launches multiple WQEs at one time.

A. Data Chunk Based Scheduling

Our main steps to diminish the interference among different kinds of flows involves enforcing that within each scheduling period, every flow transmits a determined quantity of data instead of a specified number of packets as in Round-Robin scheduling. The configuration of this predetermined quantity is proceed by users, illustrated later in Section IV-B. In our research, a scheduling period represents the time interval to loop through all active flows. Consequently, the ratio among the throughput of flows is equivalent to the ratio of the user-specified data quantity across diverse flows no matter how large the message is.

Different from the packet Round-Robin scheduling adopted by the commodity RNICs, Palos does not directly slice message data because it is a critical problem to efficiently manage the residual unsent data after slicing a large message. Storing such data on the NIC gives rise to severe scalability issues due to the increasing buffer size. To circumvent the intricate complications associated with the data buffer management, we insert a scheduling layer between RDMA Protocol Process Engine and DMA module. Within this approach, the slicing and shaping of messages is achieved through the reconstruction of descriptors (WQEs), instead of directly on transmission packets, so that we need to only consider the management of WQEs, which is far more simple.

The workflow of Palos is shown in Fig. 5. In the architectural framework of Palos, when the Doorbell Process Module receives a new doorbell for a QP from user space applications, it checks whether the head and tail pointers of the work queue of this QP are identical. If they are different, the QP is in the state of active, i.e. it has unfinished transmission tasks waiting to be processed; conversely, the QP is idle. For the latter case, the Doorbell Process Module updates the head and tail pointers and put the QP ID into the Schedule Queue (①). For the former case, the ID of this active QP already exists in the schedule queue and the Doorbell Processing Module needs only to update the pointers. WQE Fetch Module dequeues QP IDs from the schedule queue (②) continuously and subsequently retrieves a batch of pending WQEs from the WQE buffer (③) followed by delivering these WQEs to Message Slicing Module (④), which calculates the size of a data chunk for the respective QP based on a series of information such as performance weight and group granularity and regenerates new WQEs based on this data chunk size. If the message size is larger than the data chunk size, the Message Slicing Module regenerates a new WQE in data chunk size (⑤), or if the message size is smaller, the Message Slicing Module combines multiple WQEs until the total size of this batch of messages meets the data chunk size. The calculation method detail is illustrated in Section IV-B. The newly generated and grouped WQEs are then immediately sent to and processed by the Protocol Process Engine. At the mean time, the Message Slicing Module updates the header pointer and fetch offset of this QP (⑥) and checks the existence of subsequent WQEs. If such WQEs persist, the module writes the QP ID back into the tail of the scheduling queue (⑦). Because the scheduling system only modifies part of the segments of WQEs, such as size and address, and does not change the architecture of the RDMA Protocol Process Engine, the format of the original WQEs or the RDMA protocol, Palos is compatible with any RNIC IP such as ERNIC [22] .

B. Software Control Path

As illustrated previously, we accomplish message slicing by WQE reconstruction to achieve performance isolation and fairness among flows (i.e. QPs). However, although flow isolation is essential, it is not enough in multi-tenant data centers. First, the number of QPs created by different kinds of applications varies in hundreds of times [24], which means flow-level fairness does not eliminate the interference of different tenants

because one tenant may influence other tenants by establishing a huge amount of QPs. On the other hand, it is necessary to provide different performance levels, allowing users to customize the performance distribution among the QPs they create to meet their requirements.

To achieve such target, Palos adopts a hierarchical weight mechanism enabling the system maintainers to assign performance weights for each group and tenants to assign weights for each individual QP. In our design, theoretically the performance of QP_x in $Group_y$ should conform the Equation 1, in which Grp_W_y denotes the weight assigned for $Group_y$, W_{QPx} denotes the weight for QP_x, $\sum Grp\_W$ represents the sum of weights of all groups and $\sum_{Grp_y} W_{QP}$ represents the sum of weights of all QPs in $Group_y$:

$$Rate_{QPx} = \frac{Grp\_W_y}{\sum Grp\_W} \times \frac{W_{QPx}}{\sum_{Grp_y} W_{QP}} \times MaxRate \quad (1)$$

It can be inferred from the formula above that the performance distribution of QPs is decided by the chunk sizes, so the hardware needs to achieve flexible performance allocation and multi-level isolation through adjusting the chunk sizes. In summary, we need to establish the mapping relationship from user-defined weights to data chunk sizes. According to the aforementioned formula, we claim that the chunk size of each QPs should be similarly determined as the formula below, where N represents a constant proportional to the amount of QPs:

$$Chunk\_size = \frac{Grp\_W_y}{\sum Grp\_W} \times \frac{W_{QPx}}{\sum_{Grp_y} W_{QP}} \times N \quad (2)$$

Apparently, each QP has a unique chunk size. Maintaining the chunk size for each QP is another challenge. The simplest implementation is storing the chunk sizes directly in a table. However, applications and tenants may migrate their services and adjust communication requirements frequently in multi-tenant data centers with their business alteration and datacenter adjustment. Additionally we notice that the chunk size of one QP is related with the weights of all other QPs, which means that the modification of any weight leads to the update of all QPs. They together introduce operation complexity on the control path. For instance, in a node establishing 22K QPs as documented in [25] with a 100 MHz clock, updating the performance weight, creating a QP or migrating a tenant demands 220us to update the entire chunk size table. Compared with the fastest RDMA control path implementation with only several microseconds to establish a QP [26], this rough design increases the overhead by hundreds of times.

To circumvent the inefficiency induced by such cumbersome configuration method, we divide the chunk size into two parts shown in Figure 6: QP Weight and the rest part called Group Granularity. The QP Weight is stored as a segment in QP status, and the Group Granularity is stored in another standalone table. The Message Slicing Module in the hardware multiplies the two parts to get the chunk size. When a

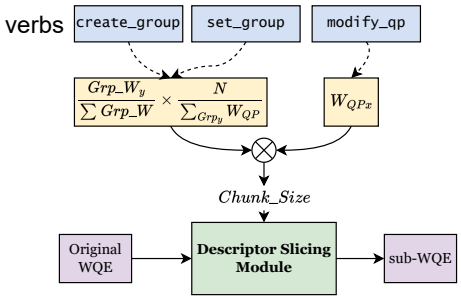

Fig. 6: Chunk size organization.

modification happens on the performance configuration, the only action the NIC needs to proceed is to update one item in the QP Weight Table and the Group Granularity Table. The amount of tenants in one host is usually far fewer than that of QPs. As a result, updating the whole Group Granularity Table, typically in several nanoseconds, does not cause remarkable time overhead mentioned above. On the software layer, we add functions `create_group` and `set_group` to enable the weight adjustment among performance groups, and we modify the function `ibv_modify_qp` in verbs API to enable tenants or applications to control the weight and traffic type of each QP.

C. Priority

From our experiment in Fig. 3 we found that the presence of large flows increases the latency of small flows. And this interference is caused principally by increasing the time of waiting to be scheduled. In addition, the message size of latency-sensitive flows is typically tens or even hundreds of times smaller than that of bandwidth-sensitive flows [27] and less frequent [11]. As a result, latency-sensitive traffic usually has a minimal impact on other flows [11].

Therefore, to guarantee the timely delivery of latency-sensitive flows such as key synchronization signals, a strict high priority is assigned to these flows. This is achieved by a dedicated Schedule Queue and a limitation of maximum message size. Latency-sensitive flows need to be labeled by user applications. WQE Fetch and Message Splitter module process these flows with strict higher priority.

V. Evaluation

To evaluate our design, we implement Palos in a cycle-accurate RNIC simulator based on Gem-5. We set PCIe bandwidth and network bandwidth to 128 Gbps and 100 Gbps respectively, same as the ConnectX-5 RNIC.

Our work focuses on traffic scheduling inside the RNIC and thus we construct our experiments on a two-node platform to avoid network congestion and jitter from influencing our result. We use 64 byte message flows as small flows and 2 MB message flows as large flows if there is no extra explanation. In our experiment, we set N as the number of QPs times one MTU, which is 1500 byte in RoCE by default.

Our evaluation results can be summarized as below:

1) The performance of both large flows and small flows can reach the expectation and have the fair competitive strength, not affected by their message sizes;
2) In the concurrent communication of large and small flows, the latency is constant with the number of large flows grows;
3) The performance of flows and groups can be controlled precisely through assigning weights by users;
4) The number or the weight assignments of flows inside one performance group has no impact on flows inside other groups.

A. Flow Isolation

Palos achieves effective isolation among different communication flows within the same group. The message size of one flow does not influence the performance of other coexisting flows, and large flows and small flows fairly share the RNIC.

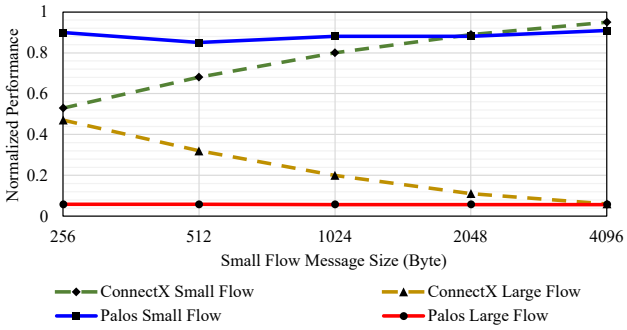

Fig. 7: Isolation between one large and 16 small message flows in the same scheduling group. The y-axis is the message rate or bandwidth normalized to its peak performance.

We conduct similar experiments as in Section II-B, which demonstrates the isolation of communication flows within the same scheduling group. In this experiment 16 small flows and one large flow transfer messages to the other node concurrently, and all flows are in the same group. In the Palos environment, the performance of both kinds of flows remains steady as the message size of small flows varies from 256 bytes to 4096 bytes, keeping at about $\frac{1}{17}$ and $\frac{16}{17}$ of the peak performance respectively. It indicates that these two kinds of flows are fairly sharing communication resources. In contrast, on the ConnectX-5 RNIC, the normalized performance of the large flow fluctuates from 47% to 0.058% as the message size of small flows changes from 256 bytes to 4096 bytes, showing interference and stronger competitiveness compared with small flows.

Besides mitigating the interference of large flows and small flows on transmission rate, Palos also diminishes the interference between large flows and latency-sensitive flows depicted in Fig. 8. We instantiate varying number of large flows sending 2 MB messages within the same scheduling group with latency-sensitive flows sending 16 B messages. On

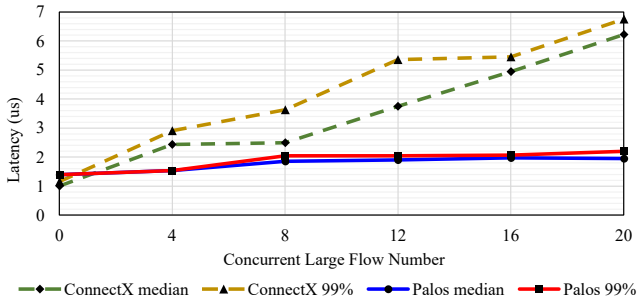

Fig. 8: Isolation between different number of large flows sending 2 MB messages and latency-sensitive flows sending 64 B messages in the same group.

ConnectX-5 RNIC, the latency of the small flow experiences a 5x increase when co-located with 16 large flows, and on Palos it modestly increases only 1.79x under the same condition. This result also proves our insight in Section IV-C that large flows introduce significant scheduling delay to small flows.

B. Group Isolation

Palos also extends its efficacy by offering optimal isolation among scheduling groups, where neither the quantity nor the weights of flows within one performance group has any impact on the performance of another group. In the practical deployment, each group can be allocated to either a tenant in multi-tenancy systems or an application in mono-tenancy systems.

We allocate 16 small flows and a variable amount of large flows into two distinct groups denoted as Group A and Group B. We compare our results with ConnectX-5 without QoS and with QoS provided by OpenSM [28]. In OpenSM QoS, we divide small and large flows into different virtual lanes. Besides, we also implement message chunking on the software similar as Flor [29] and Justitia [11], and set the chunk size to one Bandwidth-Delay-Product, which is 25 KB, as Flor's setting.

The experiment results are shown in Fig. 9. Without QoS, under the condition that Group B has eight large flows, the throughput of group A drops to 10% of its throughput with only one large flow, while the throughput of group B achieves 1.55x. The QoS and software chunking can protect the performance of small flows, but is not optimal because they do not fix the root cause mentioned in Section II-B. Palos ensures that the number of flows does not affect their performance because according to the demonstration about our data chunk size setting in Section IV-B, the performance of each group is determined only by its weight.

C. Flexibility

To meet the requirement of flexibility raised in Section III, Palos provides numerous performance levels and hierarchical performance control in which the performance of both flows and groups can be adjusted by setting their weights. We do not compare Palos with any baseline because none of

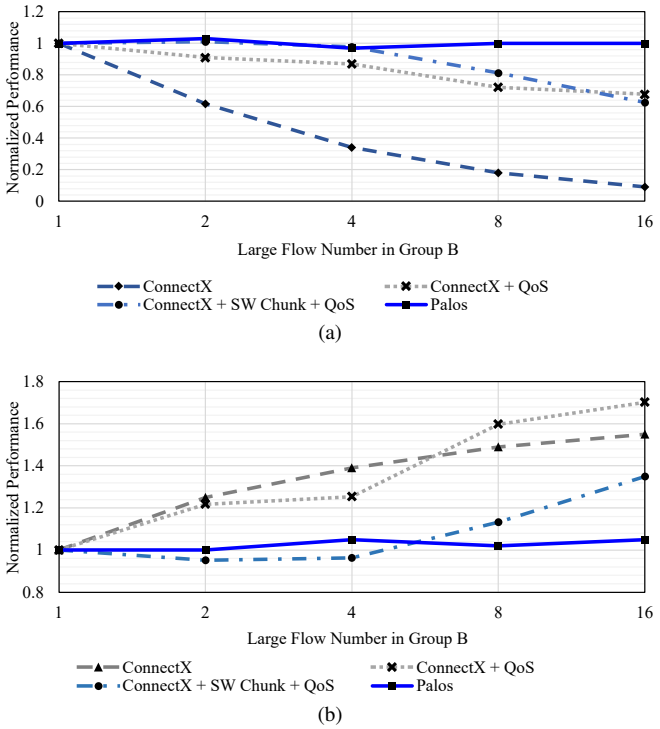

Fig. 9: Interference between varying number of large flows and 16 small flows in two separate groups. The performance are normalized with respect to the scenario where there is only one large flow. (a) Message Rate of Small Flows in Group A; (b) Bandwidth of Large Flows in Group B.

existing works supports such fine-grained performance control for RDMA.

We create 16 groups weighting from 10 to 25 linearly. Each group contains two QPs weighting 2 and 3 respectively. The experiment results in Fig. 10 demonstrate that Palos precisely controls the performance distribution of each flow and each group by dynamically manipulating the data chunk size of each individual flow. This experiment proves that Palos supports at least 32 performance levels, which is more than eight Service Levels provided by current RDMA QoS [14]. Considering creating a performance level only needs a new item in the QP Status Table and the Group Table, Palos can provide even more performance levels.

In Palos, adjusting the weights of flows does not break the isolation between different groups, i.e. change the performance of any group. In Fig. 11, we categorize six QPs into two groups assigned with the same weight. In Group A each QP has a weight of 2, while in Group B, QP 4 and QP 5 have weights 3 and 2 respectively. We varied the weight of QP 4 from 1 to 6. The test results indicate that we can precisely control the bandwidth allocation inside Group B, without affecting the performance of Group A. Both Group A and B consistently maintain a bandwidth of 50 Gbps. In summary, our approach provides flexibility while ensuring effective isolation.

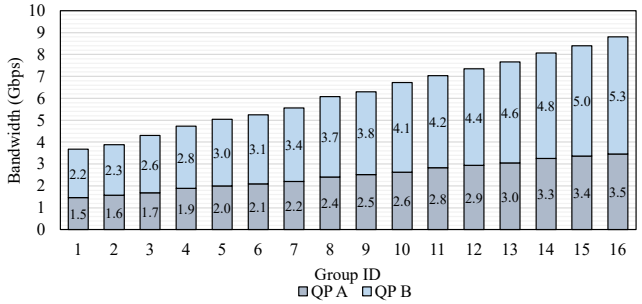

Fig. 10: Hierarchical weighted sharing of bandwidth. We create 16 groups weighted from 10 to 25. Each group contains two QPs weighted 2 and 3.

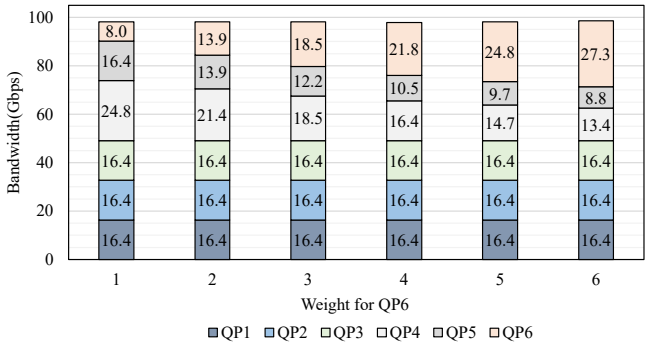

Fig. 11: Bandwidth for each connection when QP 6's weight changes. QP 1-3 are weighted 2 and are in one group. QP 4-6 are in the other group. QP 4 and QP 5 are weighted 3 and 2.

VI. Related Works

Recently, with the application of virtualization technology, there are plenty of works targeting at NIC isolation and sharing [7], [30], but most of them work on traditional Ethernet NIC and needs the involvement of either OS or smartNICs, therefore they are not applicable on RNICs.

FLOCK [19], ScaleRPC [1] and LITE [31] try to achieve sharing of RNICs by numerous amount of flows, but they target at optimizing the scalability issue caused by the hardware storage resources, instead of the interference of flows.

Justitia [11], PeRF [13] and Harmonic [12] achieve performance isolation by dynamically controlling the data transmission rate based on the performance condition on the RNIC. Their works are implemented in the software, which increase the latency and the CPU overhead.

SR-IOV [32] allows multiple virtual machines to access the same PCIe device as if they have a dedicated device. Palos can work together with it by assigning each virtual machine or function as a group and allocate diverse performance levels.

Collie [33] and husky [34] design tools to test performance anomalies on RNICs and confirm the isolation problem in RDMA network, but they do not find and fix the root causes.

RDMA QoS [14] provides performance levels by classifying flows to traffic classes. However, due to the hardware and protocol limitations, the number of traffic classes is limited, and it does not support hierarchical performance levels.

VII. Conclusion

In this paper, we make a thorough analysis about the isolation problem in current commodity RNICs, and find that the scheduling mechanism induces severe performance interference and unfairness. Guided by this insight, we introduce Palos, a novel flow scheduling system built on hardware. Palos manipulates control on the data chunk size for flows on each scheduling iteration, through which it achieves better isolation and flexibility. Consequently Palos can help RDMA to better accommodate heterogeneous traffic in data centers.

References

[1] Y. Chen *et al.*, "Scalable rdma rpc on reliable connection with efficient resource sharing," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. New York, NY, USA: Association for Computing Machinery, 2019.

[2] A. Dragojević *et al.*, "FaRM: Fast remote memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 401–414.

[3] Y. Lu *et al.*, "Octopus: an RDMA-enabled distributed persistent memory file system," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, Jul. 2017, pp. 773–785.

[4] D. G. Chester *et al.*, "Understanding communication patterns in HPCG," *Electronic Notes in Theoretical Computer Science*, vol. 340, pp. 55–65, 10 2018.

[5] T. Hoefler *et al.*, "The convergence of hyperscale data center and high-performance computing networks," *Computer*, vol. 55, no. 7, pp. 29–37, 2022.

[6] B. Montazeri *et al.*, "Homa: a receiver-driven low-latency transport protocol using network priorities," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 221–235.

[7] B. Stephens *et al.*, "Loom: Flexible and efficient NIC packet scheduling," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 33–46.

[8] M. Chowdhury and I. Stoica, "Coflow: a networking abstraction for cluster applications," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XI. New York, NY, USA: Association for Computing Machinery, 2012, p. 31–36.

[9] M. Chowdhury *et al.*, "Managing data transfers in computer clusters with orchestra," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, p. 98–109, aug 2011.

[10] M. kerrisk, "tc(8) - Linux manual page," 2001. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html

[11] Y. Zhang *et al.*, "Justitia: Software Multi-Tenancy in hardware Kernel-Bypass networks," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 1307–1326.

[12] J. Lou *et al.*, "Harmonic: Hardware-assisted RDMA performance isolation for public clouds," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA: USENIX Association, Apr. 2024, pp. 1479–1496.

[13] S. Lee *et al.*, "PeRF: Preemption-enabled RDMA framework," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 209–225. [Online]. Available: https://www.usenix.org/conference/atc24/presentation/lee

[14] Mellanox, "Deploying Quality of Service and Congestion Control in InfiniBand-based Data Center Networks," 2005. [Online]. Available: https://network.nvidia.com/pdf/whitepapers/deploying_qos_wp_10_19_2005.pdf

[15] Y. Liao *et al.*, "Optimize the tx architecture of rdma nic for performance isolation in the cloud environment," in *Proceedings of the Great Lakes Symposium on VLSI 2023*, ser. GLSVLSI '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 29–35.

[16] Mellanox, "CONNECTX-5 Datasheet," 2020. [Online]. Available: https://network.nvidia.com/files/doc-2020/pb-connectx-5-en-card.pdf

[17] Z. Wang *et al.*, "SRNIC: A scalable architecture for RDMA NICs," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 1–14.

[18] A. Psistakis *et al.*, "Optimized page fault handling during rdma," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3990–4005, dec 2022.

[19] S. K. Monga *et al.*, "Birds of a feather flock together: Scaling rdma rpcs with flock," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 212–227.

[20] J. Ousterhout, "A linux kernel implementation of the homa transport protocol," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, Jul. 2021, pp. 99–115.

[21] A. Roy *et al.*, "Inside the social network's (datacenter) network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 123–137, aug 2015.

[22] AMD and Xilinx, "AMD Adaptive Computing Documentation Portal," 2023. [Online]. Available: https://docs.xilinx.com/r/en-US/pg332-ernic/Xilinx-Embedded-RDMA-Enabled-NIC-v4.0-LogiCORE-IP-Product-Guide

[23] D. Shen *et al.*, "Distributed and optimal rdma resource scheduling in shared data center networks," 07 2020, pp. 606–615.

[24] A. Shieh *et al.*, "Sharing the data center network," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. USA: USENIX Association, 2011, p. 309–322.

[25] Y. Gao *et al.*, "When cloud storage meets RDMA," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 519–533.

[26] X. Wei *et al.*, "KRCORE: A microsecond-scale RDMA control plane for elastic computing," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 121–136.

[27] Y. Zhang *et al.*, "Aequitas: Admission control for performance-critical rpcs in datacenters," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 1–18.

[28] Nvidia, "OpenSM - NVIDIA Docs," 2023. [Online]. Available: https://docs.nvidia.com/networking/display/mlnxofedv461000/opensm

[29] Q. Li *et al.*, "Flor: An open high performance RDMA framework over heterogeneous RNICs," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 931–948.

[30] S. Grant *et al.*, "Smartnic performance isolation with fairnic: Programmable networking for the cloud," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 681–693.

[31] S.-Y. Tsai and Y. Zhang, "Lite kernel rdma support for datacenter applications," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 306–324.

[32] J. Jose *et al.*, "Sr-iov support for virtualization on infiniband clusters: Early experience," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 2013, pp. 385–392.

[33] X. Kong *et al.*, "Collie: Finding performance anomalies in RDMA subsystems," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 287–305.

[34] X. Kong *et al.*, "Understanding RDMA microarchitecture resources for performance isolation," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 31–48.