Weir: Delay-based RNIC Cache Control Software Middleware for Scalable RDMA Networks

Yongchen Pan[1], Jiao Zhang[12*], Yuxuan Hu[1], Zirui Wan[1], Baohong Lin[3], Junliang Wang[3] and Huimin Luo[1]
[1]State Key Laboratory of Networking and Switching Technology, BUPT, China. Email: jiaozhang@bupt.edu.cn
[2]Purple Mountain Laboratories, China [3]China Telecom Research Institute, China

IEEE INFOCOM 2025 - IEEE Conference on Computer Communications | 979-8-3315-4305-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/INFOCOM55648.2025.11044610

***Abstract*—RDMA is widely used in distributed services in DCNs due to its high performance. As DCNs expand in scale, RDMA faces scalability issues. The reason is that the high concurrency Queue Pairs (QPs) lead to cache misses on RNIC and frequent evictions, and the behaviour of fetching the cache via PCIe leads to performance degradation of RDMA. In this paper, we model the behaviour of Work Queue Element (WQE) on RNIC as a producer-consumer model and investigate that the root cause of WQE cache misses is the mismatch between the production rate of the CPU and the consumption rate of the RNIC. We design Weir from the perspective of WQE cache control to avoid cache misses and improve throughput under high concurrent QPs. Weir determines the cache occupancy on the RNIC by monitoring the number of active QPs and the increase/decrease in the life cycle of WQEs, and calculates the production rate and pacing by credit. The implementation of Weir exhibits minimal CPU overhead. Evaluation results show that Weir can maintain 97Gbps throughput without degradation even with up to 16K concurrent QPs, and effectively reduces various observable cache misses by 5x to 10x.**

***Index Terms*—RDMA, Data Center Network, Scalable**

I. Introduction

The accelerated evolution of data centre networks (DCNs) has resulted in heightened expectations regarding transmission performance. The TCP/IP stack based on the operation system kernel is no longer capable of meeting the requirements for throughput and latency of services. In recent years, Remote Direct Memory Access (RDMA) technology has emerged as a popular high-speed network technology in data centres, with significant adoption in distributed services and cloud networks [1]–[8]. RDMA liberates the CPU by kernel bypassing, stack offloading and zero-copying, and brings the advantages of high throughput, low latency, and low CPU usage.

As data centre networks grow in size, there are tens of thousands of servers and supporting RDMA Network Interface Cards (RNICs) within a cluster. Taking distributed storage as an example, the size of the distributed interconnected RDMA network is huge due to the limited space of individual storage media. To illustrate, PanGu [6] maintains millions of storage nodes, with both block servers and chunk servers in the storage cluster use RDMA for data transfer, providing EB-level real-time data access. Nevertheless, the enormous scale of interconnectivity has brought serious scalability issues that pose a serious threat to transmission performance.

The scalability limitation of large-scale distributed services is mainly manifested in the connection scalability limitation. As the number of Queue Pairs (QPs) of RNICs increases, RDMA will experience a performance plunge, and this problem has also been mentioned in many literatures [9]–[15]. The fundamental cause of the abrupt decline in performance is the intrinsic nature of RDMA hardware offloading, which necessitates the storage and maintenance of all transfer-related context information on the RNIC hardware. The structures of these contexts include QP Context (QPC), Memory Translate Table (MTT), Memory Protect Table (MPT), Work Queue Element (WQE), where MTT and MPT maintain the memory information of QP, Complete Queue (CQ) and WQE [9] [16] [17], and the number of these data structures on the RNIC shows a linear growth with the number of QPs. The transmission of a large number of concurrent QPs simultaneously results in a considerable occupation of on-chip memory by the numerous caches. This phenomenon leads to the RNIC's cache becoming overloaded, necessitating the RNIC to retrieve the missed cache entries via the Peripheral Component Interconnect Express (PCIe). [18]. This behaviour results in a sudden decline in transmission throughput and an increase in flow completion time (FCT). Large-scale distributed services typically utilise Reliable Connection (RC) QP for communication. Each pair of processes needs at least one pair of QPs. In a full-mesh communication network comprising N servers, where each server has M processes, a RNIC is responsible for managing the number of O(M*M*N) QPs. As distributed clusters grow in size, the number of machines inevitably gives rise to more severe connectivity scalability issues. In this paper, we find that the ***WQE cache miss*** can severely limit the scalability (Details in Sec.II-B), but this perspective has been neglected by previous studies.

Current solutions for scalable RDMA can be categorised from the perspective of cache types. Most of the previous work [18]–[22] optimised from the perspective of reducing the QPC cache on the RNIC to improve scalability, the main idea being connection multiplexing to compress the total number of QP contexts, and SRD [19] re-designed the high performance protocols to decouple the connection transfer contexts to improve large scale scalability. The works for MTT/MPT [9] [13] use large pages to reduce table entries. *The above types of solutions ignore the serious negative impact of WQE cache on scalability.* Although SRNIC [9] addresses the impact of WQE cache on scalability by designing cache-free scheduling, *hardware refactoring solutions that require additional budgets are less popular with data center operators than*

*Corresponding author: Jiao Zhang

software solutions. From a deployment perspective, hardware-based solutions are difficult to deploy quickly in existing data centres and are more expensive. Software-based solutions are undoubtedly the preferred perspective for rapid problem solving, but still require attention to applications' visibility to avoid a large number of code changes. For example, some QPC-based solutions [21] [22] require application interface (API) modifications at the application layer.

In this paper, we investigated the severe performance degradation caused by WQE cache miss during connection scaling, which has been neglected by most previous studies. And we solve it from the perspective of software middleware to overcome the drawbacks of hardware deployment. We model a *producer-consumer model* of WQE behaviour to uncover the root cause of cache miss. We indirectly know the WQE cache occupancy by monitoring the WQE life cycle, globally monitor the number of active QPs, and regulate the *producer* rate based on the number of active QPs and the real-time WQE delay. We implemented our ideas based on user-space drivers for existing RNIC, presenting them as software middleware while still supporting the standard *Verbs* interface, making them transparent to the application and thus easier to deploy. **To our knowledge, our work represents the inaugural effort to optimise the WQE cache on RNIC in software form for scalable RDMA networks.**

Weir's main contributions are:

- **Investigate the causes of WQE cache misse:** We modelled the WQE cache in a *producer-consumer* pattern and investigated the root causes that limit scalability from a WQE cache perspective.
- **Delay-based WQE cache detection and control:** We propose a latency-based approach to infer cache usage and design a latency-based cache control algorithm to keep cache usage at a stable level and avoid frequent cache misses.
- **A software middleware implementation:** We designed the WQE life cycle and active QP quantity monitoring through software middleware and implemented the control algorithms based on Credit, providing an implementation approach with enhanced scalability.

In what follows, we first describe how RDMA works and the shortcomings of existing work in Section II. In Section III, we describe the challenge and our approach to addressing it. The design of Weir is presented in Sections IV, and our evaluation of Weir is presented in Section V. The related work and full paper is concluded in Sections VI and VII.

II. Background and Motivation

A. RDMA Principle

Software and hardware interaction between the user space and the RNIC provides RDMA communication. RDMA communication is based on the creation of QPs (containing a send queue and a receive queue) and describes a specific transmission task through the WQE, which contains information such as the address and length of the data in memory.

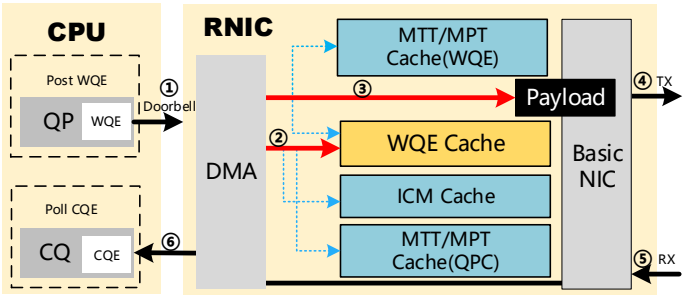


Fig. 1. RDMA principle.

QPs can be classified according to their reliability into the Reliable Connection (RC) QP and two unreliable types: the Unreliable Datagram (UD) QP and the Unreliable Connection (UC) QP provide different transmission service characteristics. Asynchronous completion is known by reading the Complete Queue Element (CQE) from the CQ. As shown in Fig. 1, RDMA has the following hardware and software interactions in the data path:

STEP① : The CPU only notifies the RNIC that a send task has been posted by constructing a WQE and a ring doorbell mechanism [23]. **STEP②** :The RNIC DMA module reads the QP context and the MTT and MPT of the QP, finds the pointer to the QP, and DMAs the WQE of the QP header to the WQE cache on the RNIC. This process requires access to the Interconnect Context Memory (ICM) cache in the RNIC and the MTT/MPT cache of the QPC, and any cache miss found by this process must be re-fetched by the PCIe. **STEP③** : The RNIC reads the memory address information carried in the WQE Cache and reads the payload via DMA. The process requires access to the WQE cache in ② and the MTT/MPT cache of the memory address specified in the WQE. Any cache miss that occurs during this process must be re-fetched by the PCIe. **STEP④**: The RNIC encapsulates the payload into packets and sends it at the RNIC's real-time rate. **STEP⑤** : For reliable QP, after the sent packet successfully arrives at the other end, the sending RNIC receives the Acknowledge (ACK), indicating that the data has arrived in order. **STEP⑥** : For reliable QPs, the RNIC will generate the corresponding CQE for the completed WQE and place it in the CQ via DMA after receiving the ACK; for unreliable QPs, the CQE will be generated once it has been successfully sent in ④. When the CQE is generated, it represents that the task has been completed and the RINC will release the task's resources in the cache. The CPU knows that the task has been completed by reading the CQE in the CQ.

B. Scalable Issue

The scalability limitations of RDMA are manifested in the following way: when the number of concurrent QPs on a RNIC exceeds a certain threshold, the throughput of the RNIC declines precipitously. The main reason is that under the large number of QPs, the total amount of on-chip cache resources occupied by the RNIC hardware is insufficient, resulting in frequent misses of various caches. The RNIC has to re-

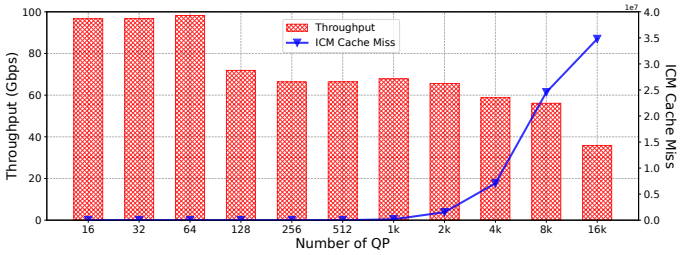


Fig. 2. Scalable issue of RNIC.

fetch the lost table entries via PCIe, and the introduction of additional latency causes a huge decline in throughput. This decline is particularly pronounced in the case of smaller messages, as the pipeline sent under large messages masks a certain amount of latency overhead. The presence of a large number of short flow in data centres [24]–[27] makes it necessary to pay more attention to this problem. Nevertheless, the extent of the scalability issue is directly proportional to the scale of QPs. The continuous addition of costly static random-access memory (SRAM) to a chip is not an optimal solution.

We verified this with bare-metal experiments and found interesting phenomena. Our testbed uses two servers with Mellanox ConnectX6 (CX6) RNICs interconnected by only one switch, and eight $perftest$ [28] processes are used on each server to send traffic to each other. The RNIC's throughput performance was evaluated at varying concurrent QP sizes, and the ICM cache miss was assessed via $neohost$ [29]. The findings are presented in Fig. 2.

We are surprised to find that when the number of QPs increases to 128, the throughput suddenly drops and is accompanied by a large number of MTT/MPT cache misses, but the ICM cache does not change at this point, and as the QP size continues to increase, the ICM only experiences cache misses when the number of QPs is greater than 2K. This makes us suspect that the throughput drop when QPs are smaller than 2K is not related to the context cache miss of QPs. Furthermore, the experiment was repeated with all QPs sharing the same memory region, yielding identical results. This eliminates the possibility of MTT/MPT cache misses caused by registering an excessive number of memory regions (MRs). [30] [17].

In accordance with the workflow of RNIC, it can be posited that the WQE cache miss on the RNIC under the concurrent QPs resulted in a precipitous decline in performance. Furthermore, the MTT/MPT cache miss may be attributed to the prefetching mechanism of RNIC, which makes the WQE and MTT/MPT bound when the WQE cache miss leads to the corresponding MTT/MPT cache miss. Cache misses also occur in QPC when QP exceeds 2K, and multiple type cache misses together lead to lower throughput. The CPU greedily posts WQEs without restriction, and if the rate of DMA WQEs to cache cannot be maintained at the same rate as the rate of cache clearing, it will definitely lead to WQE cache overflow. Therefore, we have found the root cause of WQE cache miss: *the mismatch between the generation and release rates of WQE cache table entries.*

C. Existing Limitations

The majority of existing solutions concentrate on the compressed reuse of cache resources, thereby circumventing the RNIC scalability issue. These solutions can be classified into two main categories: hardware and software.

Software-based scalability solutions. The software-based scalability solutions [6] [14] [21] [22] [31] [32]mainly improve scalability by reducing the number of connections, and reduces the cache occupancy on the RNIC by multiplexing the existing QP connections. XRC [21],DCT [22] performs context switching when multiplexing the connections, and frequent switching may lead to a further degradation of performance. And it can bring head of line blocking. The connection multiplexing based solutions also need to consider the synchronisation of multiple threads, which may bring lock contention.

Such as RoUD [20], FaSST [12], eRPC [18], these solutions use UD QP to enhance the scalability, but these solutions need to be processed at the software level for the reliability of data transmission due to the unreliability of the transmission mode itself, which will increase the host CPU overhead.

In addition to solving the problem from the connection itself, FaRM [13] reduces the number of MTTs by modifying the size of memory pages to avoid MTT cache overflow. Husky [17] performs a performance isolation of cache miss under multi-tenancy for MTT/MPT to avoid cache overflow caused by malicious registration of MRs by users.

The solutions in the above software focus on the cache miss of QPC and MTT to alleviate the scalability problem, respectively, and none of them consider the problems caused by WQE cache miss, the seriousness of which cannot be ignored.

Hardware-based scalability solutions. The main idea of hardware-based solutions is the transfer of hardware resources. In StaR [11], the connection information in the high concurrency server is stored in the low concurrency server, and when it needs to be used, it is obtained from the low concurrency server, and then the RDMA operation is performed, and the connection scalability problem is mitigated more effectively in this way. In csRNA [33] and ScalaRNIC [34], the cache management architecture on RNIC is redefined by using a non-blocking method for connection management and returning ready connections with priority. SRNIC [9] designs a cache-free WQE scheduling solution without WQE cache on RNIC, which essentially transfers WQE cache to memory and processes ready WQEs. Although reconfiguring the RNIC architecture from the perspective of hardware solutions is expected to solve the scalability problem at its root, rapid deployment is challenging due to the high cost of deployment in today's data centres.

In this paper, we have chosen to overcome the scalability problem in the form of software that is easy to deploy. And we illustrate the severe impact of WQE cache miss on connection scalability from the perspective of WQE cache miss. And we design Weir to optimise the scalability problem caused by WQE cache miss. Below we describe the challenges and opportunities we faced in designing Weir.

III. Challenges and Opportunities

A. Producer-consumer Model

We model the behaviour of the WQE cache on the RNIC as a *producer-consumer Model*. The process of DMA fetching WQEs to the RNIC after ringing doorbell is a production process. Each WQE cache is stored in a buffer on the RNIC waiting to be consumed by the consumer. When the WQE is successfully sent at the top of the RNIC scheduling queue, the RNIC releases the WQE resources and generates a CQE (for UD QP), when the ACK is received, the RNIC releases the WQE resources and generates a CQE (for RC QP), and the time of resource release represents the end of consumption. However, since the production rate grows linearly at high concurrency, it does not match the consumption rate, which eventually leads to a buffer overflow, i.e. a cache miss in the RNIC.

The Fig. 3 shows the causes of cache misses with a large number of concurrent QPs. When multiple QPs are concurrent, the number of messages for which the CPU posts WQEs and doorbell depends on the number of concurrent QPs and the *TX Depth* of each QP. *TX Depth* represents the maximum allowed number of messages that have been ringed doorbell but not yet completed, which may limit the scale of the messages to some extent. The DMA reads the WQEs to the RNIC cache based on each doorbell. In the case of the RNIC cache, the upper limit of the scale of the message entries read into the cache is related to the hardware rate of the DMA and the rate of the PCIe. Here we do not discuss the situation where the performance bottleneck occurs in the DMA and the PCIe, and assume that the scale of the messages on the RNIC cache is equal to the concurrent messages on the CPU doorbell. Before the RNIC schedules the QP to send the data referenced by the WQE, all but the WQE cache table entries that are headers are queued for processing. Thus, the real-time rate or throughput of the RNIC determines how long each table entry has to wait, and in addition the network Round-Trip Time (RTT) determines when resources are released, and together they set an upper bound on the size of the messages consumed. Since the size of the messages currently being generated under concurrency far exceeds the consumption size of the RNIC, the WQE cache that is not consumed in time may be displaced by the new WQE cache, resulting in the expulsion of the old WQE cache.

B. Challenges

We would like to be able to put a limit on the production rate of WQE on RNIC to prevent cache misses. But this is fraught with challenges:

① **Hard to get real-time usage of the RNIC cache.** The design of the RNIC is black-boxed [35] [36], and the vendor of the commercial card does not disclose the use of hardware resources to the user. It is difficult for us to determine the real-time usage of the RNIC cache. And we can't specifically distinguish which type of cache has been missed.

② **Hard to calculate the duration of each phase of WQE.** The life cycle of the WQE Cache on the RNIC consists

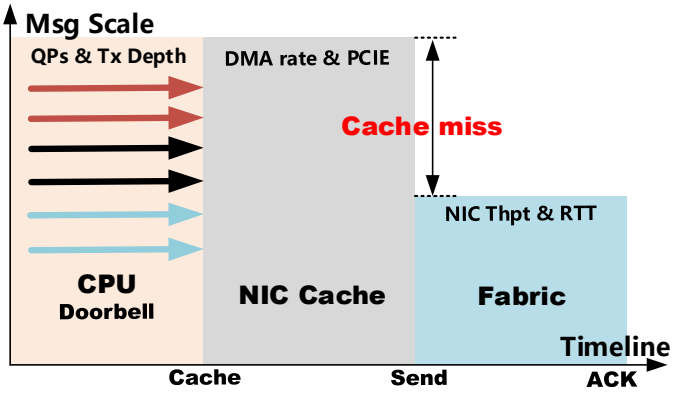


Fig. 3. Producer-consumer model of RNIC.

of multiple time periods, including: the time the RNIC waits for scheduling to send, the packet serialisation time, and the network RTT time (for RC QPs). The consumption rate is closely related to the life cycle. However, accurately obtaining the network real-time RTT is difficult due to the complexity of the network, and it is difficult to infer the specific scheduling start time from the perspective of queuing theory as we do not know the specific strategy of the RNIC in scheduling QPs and sending WQEs due to the black-boxing of the RNIC. It is impractical to obtain the length of each time slot in segments.

③ **How to monitor number of active QPs globally?** The concurrency of cache misses and throughput degradation is related to the number of QPs, and for a single RNIC we also need to monitor the number of QPs created across the card to determine the concurrency level. However, not all QPs created are active. There are QPs with states such as INIT, RESET, ERROR and Ready-to-Send (RTS) states but no data being sent. How to identify the activity of QPs and how to filter the total number of active QPs is a problem we need to consider.

④ **How to design a stable cache control algorithm?** Assuming that the real-time on-chip cache occupancy and the global number of active QPs are known, we still need to consider how to design an algorithm to ensure that the cache occupancy remains in a stable range and does not cause cache misses due to unexpected concurrency. Although each QP can limit the consumption rate by limiting the *TX Depth*, as the number of QPs increases, the message size increases linearly, which does not guarantee stability of cache usage. Therefore, an effective algorithm to adjust the production rate in real time is the challenge we have to face.

C. Opportunities

Although we cannot directly obtain the cache usage of the RINC, we can indirectly judge the trend of the cache by observing whether the production rate and consumption rate match from the perspective of the life cycle of each WQE. **If the WQE takes longer and longer to complete from the moment the doorbell rings to the end of its life cycle, this means that the cache is occupied for a longer period. If the "Ring Doorbell" is not controlled, the cache occupancy will increase.**

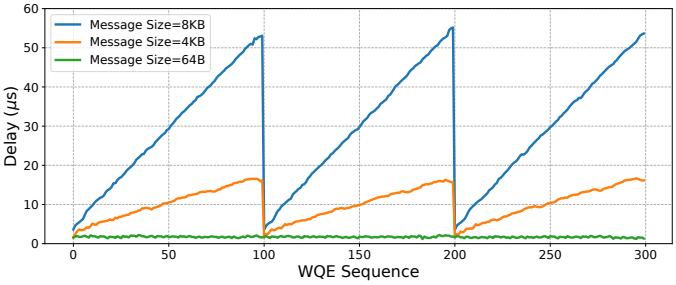


Fig. 4. Delay of different WQE size.

We want to control the frequency of the doorbell to indirectly control the change in cache occupancy represented by the delay of the WQE.

The inclusiveness of errors in caching control is far greater than one might expect. The WQE entry is 8B, and there is 2MB on-chip memory on the RNIC [9], which can hold 200K WQE table entries simultaneously. Therefore, our controllable range of WQE cache is very large, and the statistical error requirement is also lower.

We conducted an experiment to verify the above: we used a single CPU core to send WQEs of different sizes, and every 100 WQEs we paused sending for 1ms and observed the time interval between doorbell and fetching CQEs. As shown in Fig. 4, the experimental results show that the time between 8KB and 4KB grows continuously in the greedy sending of 100 batches, and since 8KB grows at a faster rate, it indicates a growing WQE cache in the RNIC, and since the serialisation time of 8KB is longer, it makes its WQE interval larger, which indirectly indicates a higher cache occupancy. However, on a single core, the 64B message size does not reach the full RNIC line rate, and any messages generated can be quickly consumed by the RNIC. Therefore, its duration remains stable, indirectly indicating that the cache will not grow.

When we pause for 1ms and resend the WQE for the batch of 100, the WQE time length increases again from the smaller value, indicating that the WQE cache of the previous batch has been fully consumed and the cache starts to accumulate again. This suggests that we can control the cache usage on the RNIC by restricting the behaviour of the doorbell, effectively avoiding cache overflow. This becomes an opportunity for us to design Weir.

IV. System Design

A. System Overview

Fig. 5 shows the main components of the Weir system. The process of generating WQEs and doorbells by the CPU and the pattern of sending and receiving data by the RNIC form a *producer-consumer model*. The WQEs generated by the *producer* are stored in a cache in the RNIC and wait for the end of the WQE life cycle to be fully consumed. Weir controls the production rate by controlling the WQE doorbell rate to avoid RNIC cache overflow.

Weir is an adaptive control middleware used to maintain RNIC cache utilization and consists of three parts: **(1) life cycle monitor**, **(2) active QP monitor** and **(3) rate control**.

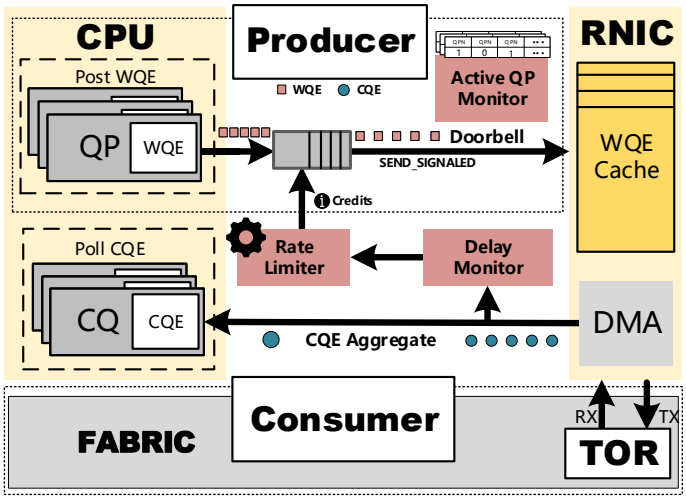


Fig. 5. System overview.

(1) Life cycle monitor determines the survival time of WQEs in the RNIC cache by fine-grained calculation of the delay from post-send to final return of the CQE for each WQE, and then evaluates the real-time cache occupancy situation for rate control decisions. **(2) The active QP monitor** counts the number of active QPs on the entire RNIC and provides parameter adjustment strategies for the rate control algorithm according to different QP connection sizes to adapt to the different pressures on the RNIC cache caused by different concurrent connections. **(3) The rate control module** includes an adaptive rate control algorithm and a credit-based speed adjustment method that determines the concurrency size by counting the number of active QPs to adjust the parameters, and counts the delay of WQEs to dynamically adjust the rate of downgoing WQEs to avoid cache overflow on the RNIC.

B. WQE Life Cycle Monitor

The life cycle of the WQE on the RNIC starts from the moment the DMA fetches the WQE to the RNIC until the end of the release of the WQE resource. And the release of WQE resources at this time is not the same in RC QP and UD QP. For RC QP, the RNIC needs to receive the ACK acknowledgement signal before generating the CQE element of the completion notification and releasing the WQE resource, while UD QP releases the WQE resource immediately after serialising the packet and only after sending it. Therefore, we treat the WQE life cycle differently on different QP types.

For the life cycle of WQE as shown in Fig. 6 contains several processes: the WQE wait scheduling time, the corresponding payload time of DMA, the packet serialisation time, and the network RTT in RC mode, while the UD mode does not contain the network RTT. Since the delay of the DMA WQE is very small compared to that of the other parts of the life cycle, we ignore it and set the life cycle start time approximated to the ring doorbell moment ($T_{doorbell}$). After the WQE resource is released, the exact completion time T_{cqe} can still be obtained, even though the CQE is fetched from the completion queue at the T_{poll} moment, because the timestamp

(T_{cqe}) can be carried in the CQE in the commercial RNICs. Therefore, even though the RC and UD undergo different processes for their life cycles, both can calculate the life cycle of the WQE by Eq.1.

$$Delay = T_{cqe} - T_{doorbell} \quad (1)$$

It is worth noting that native RDMA applications do not return CQEs for every WQE. In order to obtain the life cycle delay of every WQE for more accurate regulation, we force every WQE to generate CQEs at the end of its life cycle, which is achieved by setting the $SEND\_SIGNALED$ flag bit of the WQE. When CQEs are obtained, we filter the extra CQEs generated by Weir and aggregate them according to the requirements of native RDMA applications, making the extra processing at the bottom layer transparent to the upper layer applications.

When each WQE ring doorbell, a software timestamp is recorded on the host side, which is determined by the frequency and cycles of CPU. When the CQE is obtained from CQ, the timestamp carried in the CQE is parsed, and since the timestamp is the hardware timestamp of the RNIC, it is not possible to directly subtract the two timestamps to get the delay. We need to do additional hardware and software timestamp conversion. The simplest way is to record a software timestamp and a hardware timestamp at the same time, and synchronise the two timestamps by the difference of the two timestamps. However, since an OS context switch occurs when actively requesting a hardware timestamp, the overhead is huge, so it is not desirable to record both timestamps at the same time for each post WQE in the data path. In our implementation, we set the interval of timestamp synchronisation to 1000 WQE to reduce the overhead caused by hardware timestamps. In addition, synchronising timestamps at intervals reduces the software timestamping error caused by CPU frequency jitter and results in more accurate timestamps.

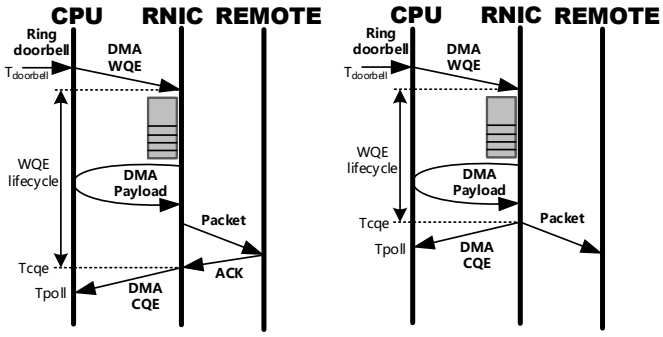
When the life cycle delay of the current WQE is obtained through the timestamp, in order to avoid the interference caused by noise that leads to inaccurate delay, we filtered the calculated delay to remove the noise through Exponential Weighted Moving Average (EWMA).

$$curr\_delay = W * prev\_delay + (1 - W) * curr\_delay \quad (2)$$

C. Active QP Count Monitor

The on-chip resource utilisation of the RNIC is closely related to the number of concurrent QPs in the same instant, and the number of active QPs represents the resource utilisation pressure on the RNIC.

Not only do the contexts of multiple QPs occupy the RNIC cache, but each QP posts a large number of WQEs, and each WQE occupies a table entry in the cache until the end of its life cycle. In order to know the real-time concurrent pressure on the RNIC, we keep statistics on the number of active QPs on the RNIC and dynamically adjust the parameters of the rate control algorithm according to the real-time changes in the number of QPs to adapt to different degrees of resource



(a) RC life cycle (b) UD life cycle

Fig. 6. WQE life cycle of different QPs.

occupancy pressure. We create a bitmap for each RNIC to store the active status of all QPs created on the RNIC.

The bitmap is built based on non-repeating QPNs as indexes to achieve O(1) efficient lookup complexity, and the active state of each QP is stored in the corresponding bit. The Weir implementation uses a 64-bit integer array to store the entire bitmap.

The maintenance of active QPs requires the participation of both the control and data paths of RDMA *Verbs*. On the control path, the maintenance of the QP state table starts when the application first calls ibv_create_qp, and the active state of the corresponding QP is cancelled when calls $ibv\_destroy\_qp$. On the data path, when posting a WQE, look up the position of the corresponding QPN and mark the corresponding bit position as **1** to mark it as active. When receiving a CQE, it determines whether there are any outstanding tasks on the QP, i.e., whether there are inflight messages. If all messages have been completed, the QP is considered to be inactive, and is only enabled when a new WQE is posted.

After obtaining the real-time QP active state management, the two parameters of the rate control algorithm must be dynamically adjusted to the threshold based on the real-time number of active QPs. These two parameters represent the interval of relative pressure that the RNIC cache can withstand. The adjustment methods are shown in Eq.3 and Eq.4: The meaning of *N* means that the alpha and beta parameters need to be changed every *N* active QPs.

$$\alpha = \lceil ActiveQPcnt/N \rceil * init\_\alpha \quad (3)$$

$$\beta = \alpha + init\_\alpha - init\_\beta \quad (4)$$

D. Rate Limiter

1) Rate Limit Algorithms: The rate control module contains a rate calculation algorithm and a rate limiter. The pseudo-code of the algorithm for rate calculation is as follows:

When the RNIC starts to send WQEs from idle, we set the initial rate of the doorbell to the RNIC's linerate to ensure that the maximum transmission capacity of the RNIC can be used in the initial phase.

We set a base delay $base\_delay$, and interval factor α and β to adjust the rate, where α and β need to be calculated based on the number of active QPs, and the algorithmic

Algorithm 1 Algorithm of Weir

On Post WQE

$CalCreditGap()$
$Active$ = $bitmap[QPN/64][QPN\%64]$
if $Active = 0$ **then**
  $ActiveQPcnt \leftarrow ActiveQPcnt + 1$
end
$Active \leftarrow 1$
while $QP.tail - QP.head \neq 0$ **do**
  if $Credit$ **then**
    $RingDoorbell$
  else
    $Waiting\ for\ Credit$
  end
end

On Poll CQE

$curr\_delay \leftarrow W$ * $prev\_delay$ + $(1 - W)$ * $curr\_delay$
if $QP.tail - QP.head = 0$ **then**
  $ActiveQPcnt \leftarrow ActiveQPcnt - 1$
  $Active \leftarrow 0$
end
$\alpha \leftarrow \lceil ActiveQPcnt/N \rceil * init\_\alpha$
$\beta \leftarrow \alpha + init\_\alpha - init\_\beta$
$delay\_ratio \leftarrow curr\_delay/base\_delay - 1$
if $delay\_ratio \leq \alpha$ **then**
  $rate \leftarrow rate + inc$
else
  if $delay\_ratio \geq \beta$ **then**
    $rate \leftarrow rate - dec$
  end
end

adjustment principle adopts the Additive Increase Additive Decrease (AIAD) way to regulate the rate of the doorbell. When the tested WQE delay satisfies $\alpha * base\_delay < delay < \beta * base\_delay$, it is considered that the current RNIC cache occupancy is at the expected level, and we do not do any regulation. When delay is lower than $\alpha * base\_delay$, it is considered that the queue of WQEs in the cache is short and there is a risk of RNIC starvation, so we need to perform speedup. Conversely, when delay is higher than $\beta * base\_delay$, rate reduction is performed to reduce the Cache usage. The RNIC cache can be maintained at a stable level by the regulation of the rate algorithm.

2) Credit Generation: After calculating the *producer* rate, we need to make adjustments to the WQE's post behaviour according to the target rate, which is driven by credits. We calculate the time of the next credit generation by the current rate and the data size specified by the WQE at the head of the queue to be sent $time\_deadline = now + size/rate$, and the doorbell operation is performed on the WQE at the head of the queue only when the credit is obtained. In order to avoid some large WQEs, the time interval between credit generation is too long, resulting in traffic bursts, we also additionally do the large WQE cut processing, in the form of small WQEs for fine-grained doorbell. This also has the advantage of providing more and finer delay information, which allows for more precise control. The WQE slicing here does not involve copying each other's memory, but is only achieved by multiplexing the WQEs and changing the pointers to the data addresses. The WQE split here also needs to aggregate CQE.

V. Evaluation

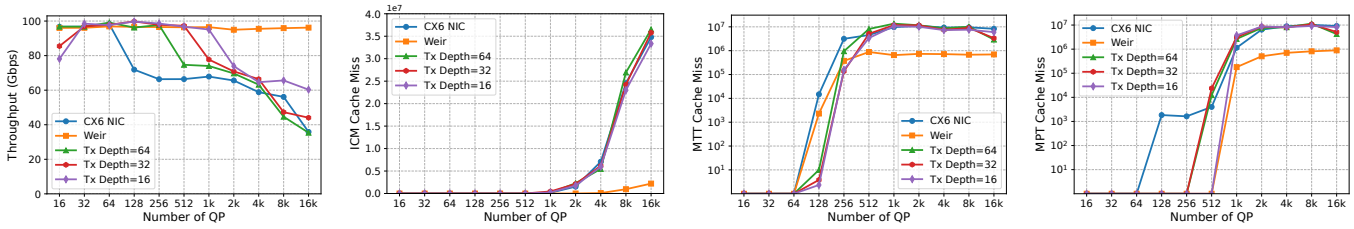
A. Evaluation Setup

We built a testbed consisting of two bare metal servers (Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz with 256G Memory) and a Ethernet switch (Mellanox SN2100), with a Mellanox ConnectX6 (CX6) RNIC on each server and connected to the switch via two 100Gbps Active Optical Cables (AOCs). We used $perftest$ [28] as a traffic generation tool and started eight Send/Receive processes on each server to increase concurrency. In our evaluation, we gradually increased the number of QPs in each process exponentially, and then recorded and analysed the network performance under different scenarios. We use this experimental environment to test and analyse the changes in throughput and cache misses as the number of QPs increases for Weir, CX6 RNICs and different *TX Depth*. The overhead of the Weir was also tested and analysed and compared to the CX6 RNIC to ensure the feasibility of Weir. We test the throughput of the RNIC using $mlnx\_perf$ and get the number of ICM cache misses and MTT/MPT cache misses using $neohost$ [29]. There are two levels of caches: L0 and L1 in MTT/MPT, we only show the cache miss data that occurs in L1.

B. Throughput and Cache Miss

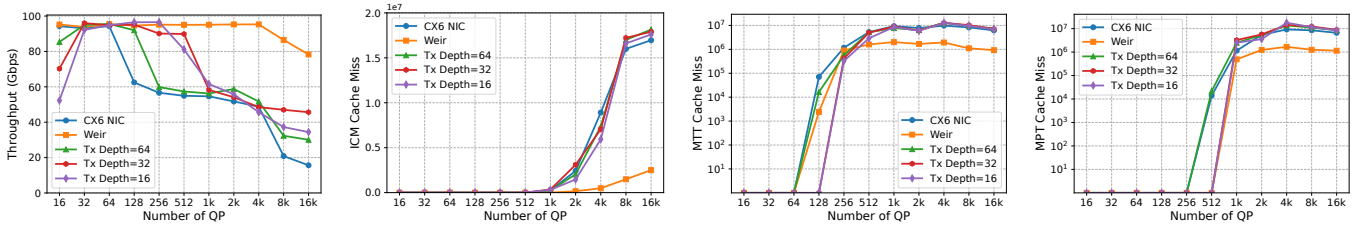
We compared five different scenarios of Weir (*TX Depth=128*), CX6 RNIC (*TX Depth=128*), *TX Depth=64*, *TX Depth=32*, and *TX Depth=16*, the network throughput as the number of QPs increases, and the cache misses of ICM, MTT, and MPT in the NICs, and we also evaluated the network performance with different message sizes, which were chosen as 512B, 256B, and 128B, respectively. The results are shown in Fig. 7, Fig. 8 and Fig. 9.

Weir's throughput is basically the same or decreases less as the number of QPs increases. From the Fig. 7, Fig. 8 and Fig. 9, it can be seen that when the number of QPs is small, the throughput of Weir and CX6 RNIC is basically the same, and when the number of QPs increases, the throughput of CX6 RNIC decreases significantly, and the throughput of Weir remains basically unchanged when the message size is 512B, indicating that Weir effectively reduces the occurrence of WQE cache misses by limiting the rate of the doorbell. When the message size is 256B and 128B, the throughput of Weir basically remains at the highest level. In the 8K∼16K interval, the throughput decreases due to the drastic increase in QPC, but by avoiding WQE cache miss, the throughput is increased by 1.5x∼2x. The three cases of *TX Depth=64*, *TX Depth=32* and *TX Depth=16* limit the in-flight size of each QP, so the throughput is lower when the number of QPs is small. The throughput increases as the number of QPs



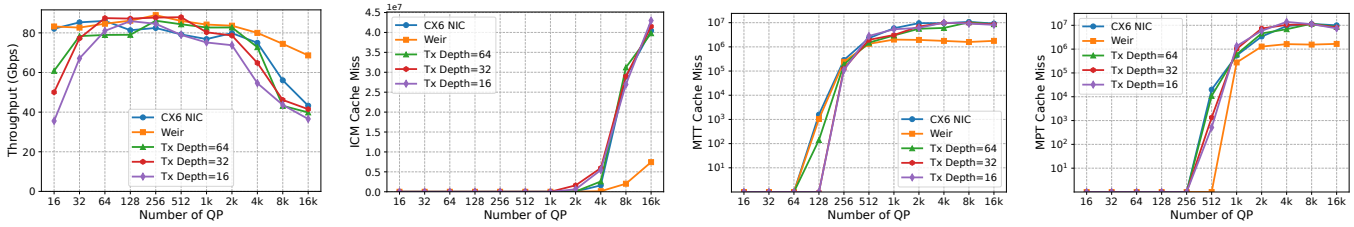
(a) Throughput (b) ICM Cache Miss (c) L1 MTT Cache Miss (d) L1 MPT Cache Miss

Fig. 7. Throughput and Cache Miss of 512B WQE.



(a) Throughput (b) ICM Cache Miss (c) L1 MTT Cache Miss (d) L1 MPT Cache Miss

Fig. 8. Throughput and Cache Miss of 256B WQE.



(a) Throughput (b) ICM Cache Miss (c) L1 MTT Cache Miss (d) L1 MPT Cache Miss

Fig. 9. Throughput and Cache Miss of 128B WQE.

increases and then drops abruptly when concurrency causes a WQE cache miss. This suggests that a smaller *TX Depth* can improve scalability to some extent, but there is a low throughput when there are fewer QPs, and it is still not possible to avoid WQE cache misses when there are a large number of QPs in concurrency.

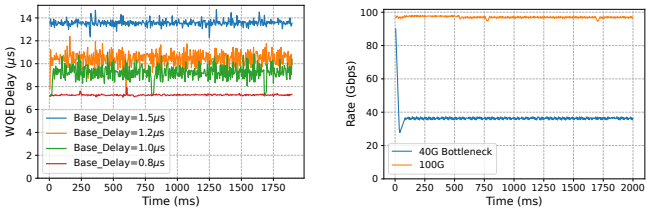
Weir has a lower ICM cache miss. The ICM cache miss of the CX6 NIC, *TX Depth=64*, *TX Depth=32* and *TX Depth=16* have the same trend with increasing QPs, and the overall trend is that it tends to be zero when the number of QPs is small, and the ICM cache miss grows rapidly when the number of QPs is greater than 2K. When the number of QPs is greater than 2K, the ICM cache miss grows rapidly, and the growth rate decreases slightly when the number of QPs is greater than 8K. Weir's ICM cache miss tends to be **0** when the number of QPs is small, but as the number of QPs continues to increase, Weir's corresponding ICM cache miss increases very little, and its total cache miss is about an order of magnitude smaller than that of the other comparison cases. As Weir reduces the frequency of the doorbell, the frequency of the DMA lookup of QPC is relatively lower, thus avoiding the ICM cache miss when reading QPC.

Weir has lower MTT and MPT cache miss. The trend of MTT and MPT cache miss for CX6 RNIC, *TX Depth=64*, *TX Depth=32* and *TX Depth=16* is also basically the same, as the number of QPs increases, the cache miss first grows rapidly and then tends to a more stable value. tends to a more stable and smoother value. Although the trend of Weir's MTT and MPT cache miss is basically the same as that of several other comparison cases, when Weir's cache miss tends to a smooth value, it is about an order of magnitude smaller than that of the other cases. Since there is a pre-fetching mechanism for MTT/MPT during DMA WQE cache, we suspect that a WQE miss will cause the corresponding associated MTT/MPT to be lost together, and Weir reduces the frequency of MTT/MPT cache miss by reducing the WQE cache.

C. Stability Evaluation

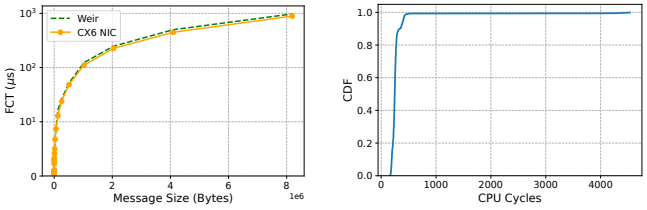
We have modified various $base\_delay$ to illustrate the effect of this parameter on the stabilisation point. We represent the cache occupancy by the WQE delay. The experimental results are shown in Fig. 10(a). The larger the pair of $base\_delay$, the higher the stability point of cache occupancy, the higher the occupied cache, and the overall smoothness. In practice, the $base\_delay$ can be adjusted according to the different on-chip resource capacities of the RNICs to make the cache converge to different occupancies, so that Weir can easily cope with the demand of different cache capacities.

In addition, we observed the convergence speed and stability of Weir's algorithm in terms of the rate of the overall



(a) Stabilisation point (b) Convergence of Weir

Fig. 10. Stability Evaluation of Weir.



(a) FCT of different WQE size (b) Overhead of CPU cycles

Fig. 11. Overhead of Weir.

aggregated doorbell. We tested the stability under 100Gbps network and the convergence and stability under the 40Gbps network bottleneck, respectively. As shown in Fig 10(b). Since the initial rate of Weir is set to 100Gbps, Weir's doorbell rate can be directly stabilised at a level close to 100Gbps, and dynamically adjusts the rate with the constant feedback of WQE delay. We set the switch port to 40Gbps to create a bottleneck link, and the experimental result shows Weir can quickly converge to the level of 40Gbps thus matching the consumption rate, and fluctuates slightly as the weir algorithm adjusts.

D. Overhead

We tested the impact of Weir's FCT under different message sizes. The results in Fig. 11(a) show that its performance is essentially the same as that of the CX6 RNIC. Weir essentially does not affect the overall flow completion time by moving WQEs queued for consumption in the RNIC to the user space. As a result, Weir is able to significantly improve throughput while keeping the FCT essentially unchanged, effectively solving the connectivity scalability problem.

We also measured the additional CPU cycle overhead of the various measurement monitoring components used by Weir, as shown in Fig. 11(b). Most of the CPU cycles are within 1000, with only one in a thousand falling around 4000 cycles due to the fact that Weir only synchronises software and hardware clocks every 1000 WQEs. The results of the CPU overhead tests are consistent with our design and the impact on performance is negligible.

VI. Related Work

The connectivity scalability of RDMA networks is still the focus of research. In the following, we summarise some representative connectivity scalability solutions for RDMA networks, classified by the type of optimised data structures.

QP-based multiplexing reduces the pressure of QP contexts on the RNIC cache, thus improving connection scalability. Software-based solutions that address scalability from this perspective include RoUD [20], eRPC [18] and ScaleRPC [14]. These solutions are based on software communication libraries or RPC libraries as an entry point, and reuse existing connections according to certain strategies. XRC [21] and DCT [22] provide different call interfaces from the perspective of *Verbs* to support more scalable connection types.

Hardware-based solutions for QP connection state resources are StaR [11], csRNA [33], ScalaRNIC [34]. StaR creatively proposes the idea of stateless servers, which divides the servers into stateful servers with low concurrency and stateless servers with high concurrency, and then get the relevant information from the stateful server when RDMA operation is needed.

FaRM [13] provides a large page memory approach to reduce the number of MTTs, thus avoiding cache miss. Husky [35] restricts the MR of malicious tenants from the perspective of tenant segregation, avoids frequent MTT/MPT Misses, avoids more on-chip resource contention, and also improves scalability under multi-tenant concurrent QP. It is worth noting that these optimisations from different data structures are orthogonal to Weir, and it is worth investigating how to avoid cache miss for multiple cached data structures.

SRNIC [9] minimises the on-chip data structures and their memory requirements in the RNIC through careful co-design of protocol and architecture. Connection scalability is improved by implementing a QP scheduler design without WQE cache, and FPGA prototypes are designed. **We propose Weir to manage the WQE cache occupancy on RNIC from a software perspective. This represents the inaugural software optimisation solution for WQE cache.**

VII. Conclusion

In this paper, we propose a delay-based software middleware Weir for controlling the RNIC cache utilisation rate to overcome the performance degradation caused by cache miss when the number of concurrent QPs is too high, in order to improve the scalability of large-scale RDMA networks, in response to the WQE cache miss problem in RNICs. In this paper, we consider the WQE cache in RNIC as a *producer-consumer model*, and ensure that the cache utilisation rate is kept at a stable level to avoid WQE cache miss by monitoring the WQE life cycle and the number of globally active QPs in real time, and adjusting the WQE production rate according to the adaptive algorithm, and limiting the doorbell rate by credit. Prototype evaluations show that Weir maintains the highest throughput and reduces cache misses of other resources, is able to run stably, and incurs negligible CPU overhead.

VIII. Acknowledgment

This work is partly supported by the National Natural Science Foundation of China (NSFC) under Grant No.62132022 and Program for Youth Innovative Research Team of BUPT NO.2024YOTD02.

REFERENCES

[1] D. Kim, T. Yu, H. H. Liu, Y. Zhu, J. Padhye, S. Raindel, C. Guo, V. Sekar, and S. Seshan, "{FreeFlow}: Software-based virtual {RDMA} networking for containerized clouds," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 113–126.

[2] Z. Wang, T. Ma, L. Kong, Z. Wen, J. Li, Z. Song, Y. Lu, G. Chen, and W. Cao, "Zero overhead monitoring for cloud-native infrastructure using {RDMA}," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 639–654.

[3] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using rdma efficiently for key-value services," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, 2014, pp. 295–306.

[4] J. Shi, Y. Yao, R. Chen, H. Chen, and F. Li, "Fast and concurrent {RDF} queries with {RDMA-Based} distributed graph exploration," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 317–332.

[5] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed {DNN} training in heterogeneous {GPU/CPU} clusters," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 463–479.

[6] Y. Gao, Q. Li, L. Tang, Y. Xi, P. Zhang, W. Peng, B. Li, Y. Wu, S. Liu, L. Yan *et al.*, "When cloud storage meets {RDMA}," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 519–533.

[7] R. Miao, L. Zhu, S. Ma, K. Qian, S. Zhuang, B. Li, S. Cheng, J. Gao, Y. Zhuang, P. Zhang *et al.*, "From luna to solar: the evolutions of the compute-to-storage networks in alibaba cloud," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 753–766.

[8] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong *et al.*, "{MegaScale}: Scaling large language model training to more than 10,000 {GPUs}," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 745–760.

[9] Z. Wang, L. Luo, Q. Ning, C. Zeng, W. Li, X. Wan, P. Xie, T. Feng, K. Cheng, X. Geng *et al.*, "{SRNIC}: A scalable architecture for {RDMA}{NICs}," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1–14.

[10] J. Tang, X. Wang, and H. Dai, "Scalable rdma transport with efficient connection sharing," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[11] X. Wang, G. Chen, X. Yin, H. Dai, B. Li, B. Fu, and K. Tan, "Star: Breaking the scalability limit for rdma," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 2021, pp. 1–11.

[12] A. Kalia, M. Kaminsky, and D. G. Andersen, "{FaSST}: Fast, scalable and simple distributed transactions with {Two-Sided}({{{{{RDMA}}}}}) datagram {RPCs}," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 185–201.

[13] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "{FaRM}: Fast remote memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 401–414.

[14] Y. Chen, Y. Lu, and J. Shu, "Scalable rdma rpc on reliable connection with efficient resource sharing," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–14.

[15] A. Singhvi, A. Akella, D. Gibson, T. F. Wenisch, M. Wong-Chan, S. Clark, M. M. Martin, M. McLaren, P. Chandra, R. Cauble *et al.*, "1rma: Re-envisioning remote memory access for multi-tenant datacenters," 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*, 2020, pp. 708–721.

[16] X. Ma, F. Yang, Z. Wang, N. Kang, and X. An, "Understanding the scalability problem of rnic cache at the micro-architecture level," in *ICC 2023-IEEE International Conference on Communications*. IEEE, 2023, pp. 6059–6065.

[17] X. Kong, J. Chen, W. Bai, Y. Xu, M. Elhaddad, S. Raindel, J. Padhye, A. R. Lebeck, and D. Zhuo, "Understanding {RDMA} microarchitecture resources for performance isolation," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 31–48.

[18] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter {RPCs} can be general and fast," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 1–16.

[19] L. Shalev, H. Ayoub, N. Bshara, and E. Sabbag, "A cloud-optimized transport protocol for elastic and scalable hpc," *IEEE micro*, vol. 40, no. 6, pp. 67–73, 2020.

[20] Z. He, Y. Chen, and B. Hua, "Roud: Scalable rdma over ud in lossy data center networks," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2023, pp. 36–46.

[21] M. J. Koop, J. K. Sridhar, and D. K. Panda, "Scalable mpi design over infiniband using extended reliable connection," in *2008 IEEE International Conference on Cluster Computing*. IEEE, 2008, pp. 203–212.

[22] H. Subramoni, K. Hamidouche, A. Venkatesh, S. Chakraborty, and D. K. Panda, "Designing mpi library with dynamic connected transport (dct) of infiniband: early experiences," in *International Supercomputing Conference*. Springer, 2014, pp. 278–295.

[23] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance {RDMA} systems," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, 2016, pp. 437–450.

[24] G. Kumar, N. Dukkipati, K. Jang, H. M. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan *et al.*, "Swift: Delay is simple and effective for congestion control in the datacenter," 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*, 2020, pp. 514–528.

[25] W. Cheng, K. Qian, W. Jiang, T. Zhang, and F. Ren, "Re-architecting congestion management in lossless ethernet," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 19–36.

[26] P. Goyal, P. Shah, N. K. Sharma, M. Alizadeh, and T. E. Anderson, "Backpressure flow control," in *Proceedings of the 2019 Workshop on Buffer Sizing*, 2019, pp. 1–3.

[27] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "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*, 2018, pp. 221–235.

[28] (2021) OFED Perftest. [Online]. Available: https://github.com/linux-rdma/perftest/

[29] (2022) Mellanox NEO-Host. [Online]. Available: https://support.mellanox.com/s/productdetails/a2v50000000N2OlAAK/mellanox-neohost

[30] J. Xue, T. Vijaykumar, and M. Thottethodi, "Network interface architecture for remote indirect memory access (rima) in datacenters," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 17, no. 2, pp. 1–22, 2020.

[31] T. Ma, T. Ma, Z. Song, J. Li, H. Chang, K. Chen, H. Jiang, and Y. Wu, "X-rdma: Effective rdma middleware in large-scale production environments," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2019, pp. 1–12.

[32] D. Shen, J. Luo, F. Dong, X. Guo, K. Wang, and J. C. Lui, "Distributed and optimal rdma resource scheduling in shared data center networks," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 606–615.

[33] N. Kang, Z. Wang, F. Yang, X. Ma, Z. Ma, G. Yuan, and G. Tan, "csrna: Connection-scalable rdma nic architecture in datacenter environment," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 2022, pp. 398–406.

[34] X. Ma, F. Yang, Z. Wang, N. Kang, G. Yuan, and X. An, "A scalable rdma network interface card with efficient cache management," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–5.

[35] X. Kong, Y. Zhu, H. Zhou, Z. Jiang, J. Ye, C. Guo, and D. Zhuo, "Collie: Finding performance anomalies in {RDMA} subsystems," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 287–305.

[36] S.-Y. Tsai, M. Payer, and Y. Zhang, "Pythia: remote oracles for the masses," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 693–710.