X-RDMA: Effective RDMA Middleware in Large-scale Production Environments

Teng Ma[*,†], Tao Ma[†], Zhuo Song[†], Jingxuan Li[†], Huaixin Chang[†], Kang Chen[*], Hai Jiang[‡], and Yongwei Wu[*]
[*]Tsinghua University, *mt16@mails.tsinghua.edu.cn*, {*chenkang, wuyw*}*@tsinghua.edu.cn*
[†]Alibaba, {*boyu.mt, songzhuo.sz, jingxuan.ljx, huaixin.chx*}*@alibaba-inc.com*
[‡]Arkansas State University, *hjiang@astate.edu*

***Abstract*—X-RDMA is a communication middleware deployed and heavily used in Alibaba's large-scale cluster hosting cloud storage and database systems. Unlike recent research projects which purely focus on squeezing out the raw hardware performance, it puts emphasis on robustness, scalability and maintainability of large-scale production clusters. X-RDMA integrates necessary features, not available in current RDMA ecosystem, to release the developers from complex and imperfect details. X-RDMA simplifies the programming model, extends RDMA protocols for application awareness, and proposes mechanisms for resource management with thousands of connections per machine. It also reduces the work for administration and performance tuning with built-in tracing, tuning and monitoring tools.**

X-RDMA has been deployed in several large-scale clusters with over 4000 servers in Alibaba cloud since 2016. It can save at least 70% development and maintenance time over RDMA, effectively improve performance and reduce network jitter especially when production servers are under pressure. It also helped locate over 30 issues in different layers of productions with over 5000 connections for each server on average.

***Index Terms*—RDMA, middleware, large-scale deployment**

I. Introduction

RDMA network is more and more popular in modern data centers due to the dramatic improvement of performance. The newest generation RDMA NIC (RNIC), ConnectX-6 Infiniband NIC [1], can support 200Gb/s bandwidth with an ultra-low latency (nearly 0.6 μs). There are plenty of recent works focusing on how to squeeze out the raw performance of RDMA in several specific use cases, including key-value stores [2]–[5], file systems [6], [7], graph computing [8], virtual machine migration [9], etc. For instance, based on hardware features, Kalia *et al* [4] proposed several optimization methods to gain a $2\times$ throughput improvement. However, raw performance gains of RDMA are far from being enough when faced the concerns in large-scale production environments.

The complexity of native RDMA library is the main hindrance for developers to build efficient RDMA based applications. They need to be familiar with the concepts of remote memory accesses, queue pairs, as well as many other essentials on RDMA to better construct their applications. One important semantic gap between RDMA and socket programming is that the remote read/write is quite different from the connection based programming paradigm. Sender side has no awareness of the receiver side applications' processing progress and aliveness. This makes it impossible to port existing commercial applications directly to RDMA-enabled clusters. For debugging and tuning, with flexible and complex access patterns provided by the underlying hardware and programming interface, it is actually very difficult for developers to find the fundamental causes of performance bottleneck. Sometimes, it cannot be easily identified whether the performance problem is related to RDMA or not. In addition, large-scale production systems have to face the distinct performance jitter [10] and congestion [11], [12]. These issues rarely show up in small-scale and controllable experimental settings. Such problems can easily hurt the overall performance of applications running on over hundreds of machines with thousands of connections for each machine. Large messages are yet another issue as they increase the probability of incast congestion.

In this paper, we share our experiences on large-scale RDMA deployment, and how they inspire our designs of an RDMA based communication middleware, called X-RDMA. X-RDMA is heavily motivated by the commercial production requirements: robustness, efficient resource management, and convenient tools for debugging and performance tuning. It extends the current protocol by adopting seq-ack window and keepAlive option. The extended protocol can improve the system robustness by enabling RDMA on traditional applications. X-RDMA uses hybrid polling and a "run-to-complete" thread model which reduces memory footprint by using per-thread cache for connections and RDMA-enabled memory. It also adopts an RDMA Send/Write/Read mixed message strategy to balance performance and memory utilization. Furthermore, extra flow control strategies are integrated into X-RDMA to remedy DCQCN [11] for smoothing network. All these techniques are used not only for efficient resource management but also for alleviating network jitter and congestion. X-RDMA has its own built-in analysis framework tracing, tuning and monitoring. They can be effectively used for understanding the current system status, performance tuning, as well as locating and then fixing bugs.

X-RDMA has been deployed and heavily used in Alibaba for nearly two years. Almost all RDMA based platforms and applications including production cloud databases and storage systems are using X-RDMA instead due to its various benefits. X-RDMA can improve the total network bandwidth usage by 24%, lower the average latency by 5% (5.60 μs compare to 5.87 μs in `ucx-am-rc` of UCX). Besides, it could effectively mitigate network jitter and improve 24% throughput by using extra flow control and resource management when congested. With X-RDMA, developers can save at least 70% development and maintenance time over RDMA. At the same time, developers have found 20 potential issues via its analysis frame-

978-1-7281-4734-5/19/$31.00 ©2019 IEEE

work before application deployment and located 10 issues in different layers of production application after deployment. X-RDMA has helped the overall production platform to smoothly support annual sales event with extreme large traffic. Overall, this paper makes the following contributions:

I. We study and share our experiences and identify issues of deploying large scale production RDMA network.

II. We analyze issues, abstract design principles, identify production requirements, and propose our solutions.

III. We implement X-RDMA, a user-space middleware to fulfill these requirements. It is light-weight but fully functional, offering some components urgently needed by the industry.

IV. We have conducted several stress tests, micro-benchmarks and three real-world applications in the Alibaba Cloud to evaluate the performance of X-RDMA.

II. Background

A. RDMA Programming Model

Our practices at Alibaba show that native Ethernet cannot satisfy the extreme performance requirements from the cloud database/storage systems. Beyond ultra-low latency (usually around 2 μs) and high throughput, RDMA also supports zero copy and kernel bypass, and hence can reduce the overhead associated with traditional network protocol stacks including context switch, protocol processing and data copying [13].

RDMA supports two commonly used modes: reliable connection (RC) and unreliable datagram (UD). In production environments requiring high reliability, applications use RC mode to guarantee hardware-layer reliability. RDMA has two communication paradigms: RDMA Write/Read/Atomic which is memory semantics and RDMA Send/Recv similar to traditional Ethernet (**two-sided**). RDMA Write/Read is known as **one-sided** operation which can access remote memory without peer side CPU involvement.

RDMA programming model has quite a few abstracts and complex structures [14]. Queue Pair (QP) is a crucial building block that represents a pair of completion queues (CQ): Send Queue (SQ) and Receive Queue (RQ). To establish a connection, each side should create and initialize a QP. Before connection establishment, a node should create a protected domain (PD) and then use this PD to register one or multiple memory regions (MR) with a unique remote key (rkey). Access to memory protected by MR is allowed only if rkey is correct. Each RDMA operation requires to post a work request (WR) to the corresponding CQ. Notice that each CQ has a **depth**: the number of queuing WRs cannot exceed. In the two-sided mode, the receiver should pre-post a WR into its RQ, then the sender posts a send request and indicates the buffer address of the payload. Finally, the receiver should poll this RQ to ensure the arrival of the packet. After completing a RDMA operation, in most cases, a completion queue entry (CQE) will be generated. In the one-sided mode, only the sender side needs to post a WR. One-sided RDMA operations always have better performance than two-sided RDMA operations [3], [15], [16]. Additionally, both RDMA Write and Send support an extra **immediate data** which can notify receiver side immediately with an `uint_32` data.

Different from traditional socket programming, RDMA developers should indicate the source address, destination address, and transmission data size for communication. These memory regions have to be registered as RDMA-enabled memory. Both sides need to ensure RDMA-enabled memory is available until completion of the transmission.

In conclusion, RDMA programs require a complex ritual: initializing context, registering memory, establishing connections, exchanging meta-data, creating QP, modifying QP to "ready to send/receive" and finally posting/polling WR/CQE.

B. Network Deployment at Alibaba

Alibaba's data center network is an Ethernet-based clos network [17]. We call this architecture as HAIL (**H**igh **A**vailability, **I**ntelligence and **L**ow latency). Distinct from previous network stack techniques, HAIL implements both active-active and stackless Top of Rack (ToR) simultaneously with a new feature in kernel and switch hardware.

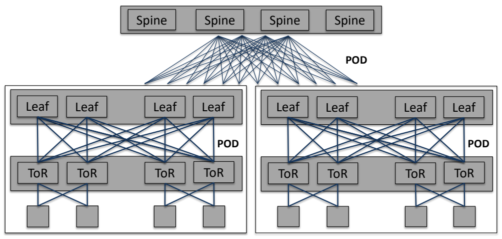


Fig. 1: RDMA Network at Alibaba.

As shown in Figure 1, a typical data center at Alibaba has three layers: the spine layer, the leaf layer, and the ToR layer. In a common configuration, forty nodes connect to a ToR switch. ToR switches connect to leaf switches and leaf switches connect to spine switches, as usual in general clos networks. Each machine is equipped with a dual-port RNIC.

C. RDMA Use Cases at Alibaba

RDMA has three major implementations: Infiniband [18], RoCE/RoCEv2 [19], [20] (RDMA over Converged Ethernet), and iWARP [21] (Internet Wide-Area RDMA Protocol). Besides, other manufacturers such as Cray, Fujitsu, and Intel (Omni-Path [22]) have their own solutions which are similar to RDMA. Though Infiniband protocol is the only one which has native support, it needs Infiniband and special switches as the infrastructure. Both RoCE and iWARP are based on RDMA-enabled NIC (RNIC) with IP/TCP/UDP protocols in the network layer to allow RDMA operations over Ethernet. Especially, RoCEv2 relies on Priority based Flow Control (PFC) [23] to enable a drop-free network. Alibaba uses RoCEv2 to guarantee compatibility with TCP/IP and adopts fine-tuned DCQCN [11], [24], which is an end-to-end congestion control protocol.

We will introduce three representative applications in detail to illustrate the requirements in Alibaba's data center: Enhanced Solid State Drives (ESSD), X-DB and PolarDB [25].

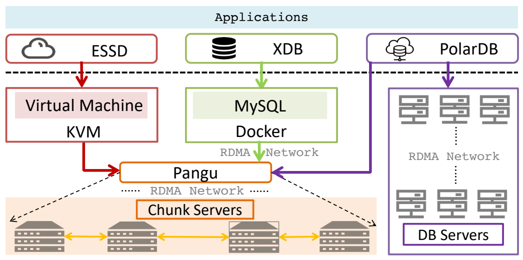


Fig. 2: The usage of RDMA in the productions.

As shown in Figure 2, Pangu is a high-reliability, high-availability, and high-performance distributed file system developed by Alibaba Cloud. Similar to Ceph [26], there are two key components (block server and chunk server) on each server in Pangu. Each block server receives data from front-end (e.g., virtual machines in ESSD) and distributes two or three copies to chunk servers on different machines via full-mesh RDMA communications. Currently, two core products at Alibaba use Pangu as distributed storage: ESSD and X-DB. The ESSD's half of I/O path is from virtual machine with QEMU/KVM [27] virtualization to Pangu. X-DB is a distributed database which provides high availability, strong ACID, and horizontal scalability for transaction systems (e.g., an online shopping website). The front-end of X-DB is a MySQL instance in Docker [28] and uses RDMA to connect Pangu. PolarDB is rather different since there are two modes in its implementation: one is for its own back-end and the other is for Pangu. Both modes use RDMA.

We can conclude that Alibaba's applications running over RDMA are crucial and complex.

III. Large-scale Production Issues

We now address the issues related to large-scale deployment, along with their feasible solutions.

Complex Programming Abstraction. The native RDMA library (i.e., libverbs) is more complex than traditional socket programming. As mentioned in Section II-A, the developer has to consider various parameters, corner cases, performance tuning [29] and hidden costs [30] carefully, especially in complex production environment with different applications. Meanwhile, many key innovations such as ODP [31] are only offered in some specific implementations [32]. Take a simple ping-pong program as an example. Using libverbs the program requires at least 200 lines of code, while 50 lines of code is enough for socket programming. In our practice, since most developers have insufficient experiences in RDMA programming, appropriate simplifications and classifications of RDMA library APIs are desired.

Scalability Challenges. In the production environment of Alibaba, there are usually over hundreds of machines and thousands of connections per machine. Three challenges can be concluded when scaling RDMA.

Issue 1: *RDMA resource footprint will increase rapidly as the cluster scale.* One of the biggest advantages of RDMA is zero-copy, but the one-sided mode needs on-demand memory allocation, i.e., reserves more memory even before any transmission starts for each connection. Instead, in TCP/IP, the kernel can manage the buffer automatically with fewer reservations. For instance, each block server and chunk server in Pangu runs N and M threads separately, and each chunk server thread needs to establish connection with all block server threads. Such full-mesh connectivity consumes a significant amount of memory (around $N * M * blockserver\_number * depth * message\_size$), and hence incurs a high memory footprint.

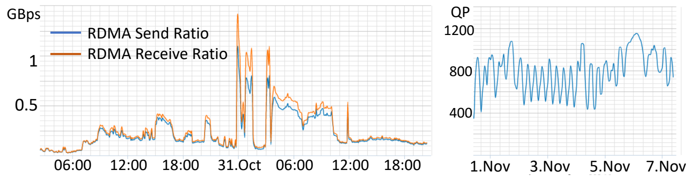


Fig. 3: Per-machine online monitoring of PolarDB.

Issue 2: *Congestion and heavy incast exist commonly in large-scale RDMA network* [33]. A typical scenario in our production servers is distributed storage such as Pangu or ESSD and always follows incast traffic pattern [34]. Besides, similar to the deployment in Facebook [35], the network workload always switches between saturated and unsaturated as shown in Figure 3, and it can easily overload the RNIC with congestion. As another aspect, congestion can also result in jitter. Despite the fine-tuned DCQCN, because the *large-size messages* block the RNIC processing [36], some serious jitter cases have been observed in larger clusters. In production environment, serious jitter can incur 70% throughput degradation (from 3.4 GBps to 1.1 GBps) and $2\times\sim15\times$ higher latency.

Issue 3: *Slow connection establishment can delay the recovery and increase the time for the cluster return to steady-state.* The connection establishment time for RDMA using `RDMA_CM` [37] is about 4 milliseconds compared with almost 100 microseconds in TCP. This doesn't attract enough attention in the previous steady-state cluster. However, in the production environment, the condition is more complex, some applications may be restarted and machines can be added or removed elastically in large-scale production environment. Additionally, the cluster may sometimes be scaled or upgraded. This situation gets worse as the scale extends and easily causes the network jitter and long tails. While establishing connections in a 64 machines cluster, the throughput of ESSD will be nearly 65% lower than the steady-state.

Lower Robustness. There are mainly two issues in using RDMA based on its original software/hardware stack.

Issue 1: *It is hard for the sender side to be aware of the receiver side applications' processing progress via RDMA one-sided operations.* Even though RNIC provides acknowledgment in the hardware layer, the sender side cannot determine if the packet has already been perceived by the actual application. The buffer is not freed until the receiver (i.e., the application) finishes processing the data. In this case, more and more data is received, yet the sender does not know the progress and keeps transmitting continuously. A receiver-not-ready (RNR) error will be raised when there is no buffer available. This problem becomes more difficult to fix while using one-sided operations. The RNR error can increase the retransmission ratio and hence wastes both network bandwidth and CPU cycles. In worse cases, it can result in jitter.

Issue 2: *Native RDMA library and RNIC cannot ensure the peer is active all the time*. For example, while using the native RDMA library, if one machine crashes, there is no notification to the peer side. Consequently, the QPs and various resources are held until future communication happens again and then an error is generated. Especially, in long time running, even when the chunk servers in Pangu has already been disconnected, some will still occupy connection resources and memory consumption could be nearly at GB scale. This kind of resource-leak has been resolved well by TCP's keep-alive option [38]. Unfortunately, there is not an alternative solution in lower-level implementation of RDMA.

Bugs and Performance Interferences. In production environments, the communication patterns are complex and rapidly-changing so as to trigger bugs easily. Locating potential bugs and performance bottlenecks quickly is beneficial to quality improvement and performance of operations as well as maintenance in the long run. We still need some complementary gadgets in six aspects: 1) The original RDMA stack can not supply `netstat` [39] as analysis tool to help find connection level information; 2) There is no effective `pingmesh` [40] tool; 3) Utility is lacking for complex performance assessment and stress test; 4) There must be some measures to simulate network exceptions, e.g., message drops, but unfortunately, Linux `netfilter` [41] does not work on RDMA data-plane; 5) There should be some mechanisms to help distinguish issues at the first place quickly; 6) Dynamic setting adjustment is helpful for tuning different applications.

Conditional Performance Maximization. How to fulfill the above desires and at the same time maintain raw performance is the major consideration in our RDMA based design.

These issues motivate that X-RDMA adopts several design principles to achieve the desirable goals:

I. Abstract high-level data structures and interfaces to simplify the complex programming model.

II. Provide light-weight and efficient protocol extensions to improve robustness.

III. Add resource management system and internal flow control to maintain performance in bad cases with large scale datasets.

IV. Integrate utilities and schemes to fulfill the requirements for debugging, performance bottleneck detection and tuning.

V. Explore effective thread model, message model and work flow as well as simplifies functional implementation to play RDMA's raw power.

IV. X-RDMA Design

A. Overall Architecture

X-RDMA can be abstracted to three layers and 16 primary components as shown in Figure 4.

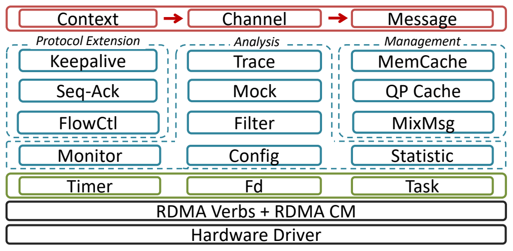


Fig. 4: Overall Architecture.

In the upper layer, X-RDMA provides three highly abstracted data structures and eight major APIs to simplify RDMA primitives. The native RDMA verbs library has nearly 30 data structures (`ibv_pd/mr/cq/wc/qp/wr`, etc). X-RDMA, on the other hand, offers three simple but crucial data structures: `context`, `channel`, and `msg`. As shown in Table I, X-RDMA focuses on providing a minimal set of APIs based on combined requirements for better performance and usability.

TABLE I: Major APIs of X-RDMA.

APIs (xrdma_)	Descriptions
send_msg	common routine of sending message to remote
polling	polling the context to check events/messages.
get_event_fd	get the xrdma_fd to do select/poll/epoll
(de)reg_mem	register/deregister RDMA-enabled memory
set_flag	dynamic changing configurations
process_event	handle event notified by fd
trace_request	trace information of the request message

In the middle layer, X-RDMA implements useful components for reliable protocol extension, resource management, flow control, and performance analysis. Firstly, this layer extends the traditional RDMA protocol stack by *Seq-Ack* and *KeepAlive*. *FlowCtl* is offered as another protocol extension in the same layer. Secondly, resource management is handled by *MixMsg*, *MemCache* and *QP Cache*. The first two components are specific to RDMA-enabled memory, while the last one is related to QP. Furthermore, the analysis system consists of six components including *Trace*, *Statistic*, *Config*, *Filter*, *Mock*, and *Monitor*. Lastly, there are five associative utilities: *XR-ping*, *XR-perf*, *XR-stat*, *XR-server* and *XR-adm*, which are convenient for debugging and administration. Some basic data structures, such as `timer`, `task`, `fd`, construct the bottom layer of X-RDMA.

Overall, these components are classified into four separate systems and bring convenience to various developers. X-RDMA's APIs are designed for large-scale production environments and sacrifice infrequently used functionalities to avoid the complex programming abstraction trap.

B. Thread Model

Since the latency using modern RNIC can be as low as nanoseconds, all operations on data-plane could be completed in constant time ($O(1)$) even in the worst cases to match the lower network latency. Thus, X-RDMA adopts *lock-free*, *atomic-free* and *no-syscall* strategies to reduce the overheads in bus locking and context switching between user space and kernel space [42]. It avoids using any lock and only allows atomic operations and `syscall` on non-critical paths.

The thread model design principle of X-RDMA is *run-to-complete* generally. High-level resources, such as *context*, *channel*, *memCache*, *QP cache*, etc, all operate on a *per-thread* level to avoid synchronization between threads. These resources will be initialized in each `context` only once.

X-RDMA uses a **hybrid** polling approach where a thread uses *epoll* first and then switches to *busy polling* when a message or timer event is triggered, similar to NAPI [43] in the Linux kernel. X-RDMA registers some events including *keepAlive*, *statistic*, etc, to the per-thread `timer`. During the idle time, the polling mode is configurable according to the scheme of applications.

C. Message Model

At Alibaba, various applications (e.g., Pangu) are based on RPC in which the request-response mode is typically used for the communication between multiple processes. X-RDMA is implemented in essentially the same manner.

Since the buffer pre-allocation phase will introduces a significant overhead [30], to better balance performance and resource utilization, X-RDMA adopts a **mixed** messaging strategy including two modes: *small message* mode for maximizing performance and *large message* mode for reducing memory footprint. When the payload size is less than a threshold of S bytes, it should be treated as a small message, and vice versa. This value is set to 4 KB by default. To some extent, this mechanism is similar to the eager and rendezvous protocols of MPI [44].

Small Messages v.s. Large Messages: For small messages, the sender side directly sends data to the receive side and hence triggers a receiving WR. Each data transmission only needs *one RDMA operation*. However, the receiver side needs to pre-allocate enough buffers. Thus, the payload size of small message cannot be large, or it will consume more buffers and hence incur high memory consumption. With large message, the sender side will first post an RDMA Send WR to wake up the receive side. The receiver will prepare the RDMA-enabled buffers on demand. We call this phase as *buffer-preparation* phase. After that, the actual data transmission relies on RDMA write/read from the sender side. In this mode, each data transmission needs at least *two RDMA operations*. In production environments, small messages are more sensitive to high latency than large messages which usually have a long transmission time. To some extent, large messages can tolerate a little downgrade of latency through prepared buffers.

Read Replace Write: X-RDMA supports built-in RPC. In most RPC implementations, the receiver sends back response to the sender. Large size responses are prohibitive to handle. Since the sender side doesn't know the response's payload size, the sender side should reserve "over-size" buffer for the receiver to write the response back via RDMA Write. Even worse, in the RPC scenario, RDMA *out-bound* operation (Write) is always slower than *in-bound* operation (Read) on the receiver side [15], [45]. To remedy this, X-RDMA allows the sender to handle the response directly. The receivers should let the sender know the response's size and address, and the sender will passively fetch the response via RDMA Read.

D. Work Flow

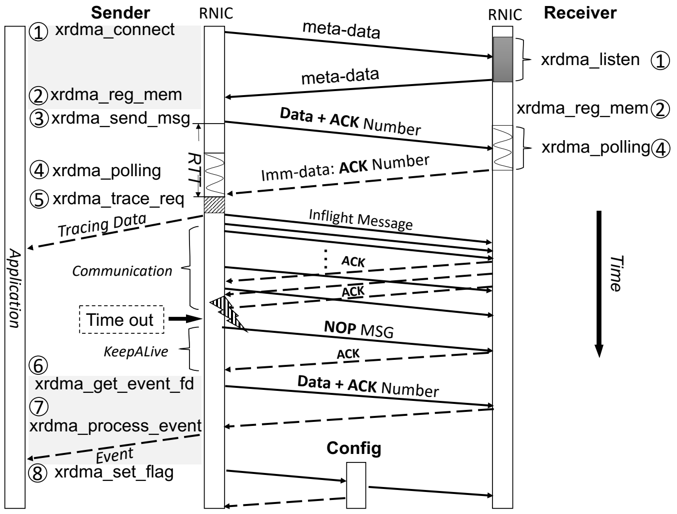


Fig. 5: Per thread workflow.

The workflow for each thread is shown in Figure 5. To establish the connection (`xrdma_channel`), both sides start out in listening or connecting state at first (①). After that, the developer should request RDMA-enabled memory from the `memory cache`, as the described in Section IV-E, or register RDMA-enabled memory manually (②). For an instance of data transmission with **polling** mode, the receiver will poll for the completion (④) of the message sent from the sender (③). The receiver will send back the response with an ACK number attached. If the sender is in the debug mode, the header of the response payload (⑤) is teared off to trace this request. More details will be described in Section VI-A. During the communication, messages will not block the next message even if it is unfinished. Unacknowledged messages are called *inflight messages*. However, X-RDMA limits the number of inflight messages as $depth$ which is less than the CQ depth. Other than polling mode, X-RDMA also provides **event** mode for receiving messages. The sender should get the event fd (⑥) first and then handle the event notifications (⑦). If a time-out event occurs during communication, some extended operations such as heartbeat messages (*keepAlive*) will be generated immediately. Additionally, X-RDMA can change configuration dynamically (⑧) to adjust running state.

E. Resource Management

To improve performance, reduce memory footprint, and shorten establishment time, we manage per-thread resources with *memory cache* and *QP cache*.

Memory Cache: To adjust the capacity of RDMA-enabled memory, X-RDMA manages per-context RDMA-enabled memory as the `memory cache` which contains several MRs of identical size. If the capacity is insufficient, it will register for a new MR. Otherwise, if the resource utilization becomes lower, it will shrink its capacity by reclaiming idle MRs. Additionally, previous research with LITE [46] has pointed out that the performance will downgrade as the number of MRs increase. LITE adopts 4KB size for each registration and faces pressure when the number of MRs is larger than one thousand. We set each MR to 4MB in order to avoid performance downgrading. In our implementation, we can extend the memory cache's capacity automatically or manually (for finer-grained tuning). We will further describe how to improve memory cache for convenient debugging in Section VI-C.

QP Cache: Since slow connection establishment will result in jitter (as described in Section III), we manage connections as a per-thread resource cache. X-RDMA accomplishes the QP destroying phase directly by setting its status to `IBV_QPS_RESET`, and releasing it to the QP cache. X-RDMA will reuse this QP from the `QP cache` to accelerate connection establishment.

V. Protocol Extensions

A. KeepAlive

In the traditional TCP/IP protocol, the keepAlive mechanism can check whether the connection is still alive or not through sending heartbeat messages [47]. From our experience with RDMA deployment in production environments, we observe that the connection leak might happen inevitably due to various conditions such as timeout, network fault, machine crashing, etc. Distinct from the case in TCP/IP protocol, the node is prohibitive to know that the peer side is unreachable.

Based on the native reliability (reliable delivery and in-order arrival) of RC mode, X-RDMA employs RDMA Write to implement KeepAlive since its kernel bypass feature will not awake the peer side's application and consume CPU resources. In our implementation, a probe request, i.e., RDMA Write, will be triggered if either side fails to communicate with peer side more than S ms. The RNIC will automatically respond to an acknowledgment to declare its aliveness. To reduce any negative impact on overall performance and avoid any demand of RDMA-enabled memory, the payload size of this probe request is **zero** [48]. If the connection is broken, corresponding resource (e.g., QP) should be released immediately to avoid connection leaks.

B. Seq-Ack Window

X-RDMA requires the seq-ack mechanism in two aspects: 1) As we mentioned in Section III, the acknowledgment about data arrival from RNIC (i.e., CQE) cannot become an evidence to ensure that *receiver side*'s application has perceived this data. 2) With small messages, the sender side needs the receiver side to reserve buffers for the incoming data. With massive messages, the receiving buffers may be not enough. As a result, the RNIC on the receiver side will throw an RNR exception and then disconnect immediately.

Accordingly, based on messages, not the commonly-used *bi-directional byte stream* [49], X-RDMA offers an application-layer seq-ack window mechanism and hence guarantees *RNR-free*. In our implementation, each side has a window to buffer in-flight requests, and the windows adopt a **ring buffer** style whose ring length is the in-flight message depth. Either side will record its current seq-ack number and send it to the other via RDMA Write/Send. This seq-ack number is attached to the **immediate data** to reduce DMA trips and notifies the peer side immediately [6], and each transmission will carry the current *ACK* number. After receiving N messages successfully but without any *ACK*, a standalone *ACK* message will be triggered to acknowledge arrival.

Algorithm 1 shows the send and receive operations on both sides with seq-ack window mechanism. *ACK/ACKED* and *SEQ/WTA* are the left and right edges of the sender/receiver's window respectively. On the sender side, the SEQ will be incremented by one for each send request. The receiver side should also increment WTA upon receiving a request. After that, it decides whether the sender side fetches the data via RDMA Read or not based on request's status. Then, the receiver calls `rdma_read_done` and determines if the ACK in this message equals RTA. If true, the receiver will update RTA until RTA equals WTA or any un-completion message is missed. When sending back a message to the sender side, it will update $ACKED$ as RTA, and attach $ACKED$ to the next message as the newest ACK number.

Avoid Deadlock: If both sides of the connection try to post WR to each other simultaneously before ensuring the previous ACK, they cannot post WR successfully until one side is aware of the previous ACK, because none of the sides have available window for acknowledgement and hence trap into deadlock. Such deadlock is different from the TCP deadlock [49] whose root cause is the limited size of the buffers. To avoid the scenario, we add a NOP message mechanism to trigger completion proactively. A NOP message is preserved in the ring buffer. X-RDMA exploits the per-context `timer` to detect deadlocks for all connections in one context to reduce overhead. If it exists, the sender side will use the NOP message to notify the receiver side to break the deadlock.

C. Flow Control

As a reactive congestion control method, we observed that DCQCN [11] fails to perform very effectively in the large-scale cluster with heavy incast, i.e., massive inbound connections. We add built-in flow control based for the following reasons: 1) DCQCN is a passive control in incast scenario, but it may incur harmful effects on the application before the reaction works; 2) According to our statistics in large-scale clusters, much more CNP (Congestion Notification Packets) and PFC pause frames are generated due to the heavy incast, which will downgrade performance and robustness of the whole network. X-RDMA adopts *fragmentation* and *queuing* to coordinate with DCQCN and hence avoid network congestion:

Fragmentation: Large size request will usually block the RNIC processing since RNIC should ensure the completion of this request. Thus, congestion is inevitable. X-RDMA exploit fragmentation to preemptively schedule multiple RDMA WRs

Algorithm 1 Seq-Ack Mechanism.

```
1: Initialize timer per xrdma_context
Sender
2: procedure SEND_MESSAGE(msg)
3:     $QP.tx.seq++$
4: procedure RECV_MESSAGE(msg)
5:     for $i$ in $range(QP.tx.acked$ to $msg.ack)$ do
6:         call_on_acked(messages[i])
7:     $QP.tx.acked = msg.ack$
8: procedure TIME_OUT(timer)
9:     if deadlock ocurred then
10:        send_message(NOP_MSG)
Receiver
11: procedure SEND_MESSAGE(msg)
12:     $QP.rx.wta++$
13:     if $need\_rdma\_read(msg)$ then
14:         do_rdma_read(msg)
15:     else
16:         $msg.recved = true$;
17: procedure RECV_MESSAGE(msg)
18:     $QP.rx.acked = QP.rx.rta$
19:     $msg.acked = QP.rx.acked$
20: procedure RDMA_READ_DONE(msg)
21:     $msg.recved = true$
22:     if $msg.id == QP.rx.rta$ then
23:         $QP.rx.rta++$
24:         while $QP.rx.rta < QP.rx.wta$ & $msgs[QP.rx.rta].recved$ do
25:             $QP.rx.rta++$
```

- ACK - current received sequence number;
- SEQ - current sending sequence number;
- $ACKED$ - current acknowledgment number sending to receiver;
- RTA - current acknowledgment number which is ready to ack;
- WTA - current acknowledgment number which is wait to ack;

to reduce incast. For a large size RDMA WR, its payload will be broken into multiple fixed moderate size fragments with its original destination in order. However, if the fragment size is small, too many fragments will saturate the RNIC. Moderate size fragments can benefit the RNIC and at the same time do not block other requests. In practice, X-RDMA sets the fregment size as 64KB.

Queuing: To remedy the congestion, X-RDMA limits the number of outstanding RDMA WRs to N and hence employs a *queue* to buffer extra requests. Before posting a new RDMA WR, X-RDMA determines whether the number of current outstanding RDMA WRs n exceeds the threshold N or not. The RDMA WR can only post into the SQ if n is less than N. Otherwise, it will be pushed to the queue first and then posted to the SQ under satisfied conditions (the SQ is not full).

These two flow control algorithms are constructed based on native RDMA library without specific hardware or software constraints.

VI. Analysis Framework

In large-scale production environment, bugs such as jitter, time-out, performance downgrade, and glitch may appear at different stages. X-RDMA is responsible for tracing these issues, regardless of whether they are from either the upper layer applications or the lower hardware layer. How to locate bug as soon as possible is a foremost demand in production environments. As the Figure 6 and Table II shows, X-RDMA provides several mechanisms and tools for covering various types of bugs.

TABLE II: Classification of bugs.

Bug Type	Tracking Method
heavy Incast	tracing, XR-Stat
broken network	keepAlive, XR-Ping
jitter	tracing, XR-perf
long tail	tracing, XR-perf
bugs hard to reproduce	filter
memory leak or crash	isolated memory cache

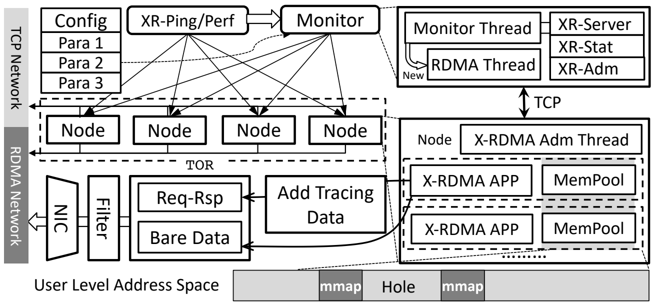


Fig. 6: Analysis framework during running time.

A. Tracing

To satisfy the requirements of tracing issues and latency map, X-RDMA implements an **internal** *req-res* mode distinct from the native *bare-data* mode. In the *req-res* mode, X-RDMA reconstructs the original payload, and each message contains a *header* inside the raw payload. We use this header to attach necessary information for tracing and trouble shooting. X-RDMA sets *bare-data* as its default mode to push for extreme performance. If it requires tracing, X-RDMA will switch to *req-rsp* mode where the tracing data can be attached to the header as show in Figure 6.

Previous work has only focused on measuring RTT and ignored the fine-grained decomposition of each RDMA request. Actually, some of these phases could be the actual causes of poor performance. X-RDMA implements the following three case-by-case methods to detect the root cause of long latency:

I. Generally, reasons causing long request latency include packet loss, long pausing time, and over-saturated requests in the network L1. X-RDMA implements a discovery mechanism to measure overhead under L1. Both sides synchronize their clocks and calculate the time offset as T_{off}. Then the sender side will attach a time-stamp to the *header* with local time T_1, and the receiver will also keep the received time-stamp as T_2. The real request time can be estimated as $T_2 - T_1 - T_{off}$. A prerequisite for this method is that the sender and receiver should have well-synchronized clocks to avoid bias. For this, X-RDMA provides a clock synchronization service [50].

II. Working threads of the application may abort time-consuming operations, and there is no timely polling. Consequently, it will result in slow requests and long tail. X-RDMA counts the time interval between two polling operations. Thus, it can detect performance degrades caused by polling.

III. X-RDMA inserts time measurement instructions into different critical code segments. If the execution time of a specific code segment exceeds a threshold, X-RDMA will record its location in a **log**. These logs can be collected by the monitor as shown in Figure 6. Developers can then use these logs to detect performance bottlenecks.

B. Monitoring

In the X-RDMA monitoring system, we integrate three new utilities necessary in production environments.

XR-Stat: Based on the *channel* abstraction, X-RDMA can provide per connection statistics similar to those of the `netstat` tool. These statistics can also include other resources such as `memory cache`. While coordinating with the monitoring system, it provides the raw data for both troubleshooting and performance analysis. Despite this, the RDMA network is sensitive to packet loss and high latency. Meanwhile the whole system will also monitor *PFC status*, *queue drop counter* and *buffer utilization* as crucial indexes.

XR-Ping: Since the original `ping` cannot emulate the real RDMA network and the RDMA-based `rping` is too simple and buggy, X-RDMA designs an RDMA-friendly ping tool able to show a network connection matrix, i.e., the full-mesh connection status, through exploiting the centralized monitor. As shown in Figure 6, *XR-ping* will ping all machines in the ToR layer, and then aggregate the results to the connection matrix in the monitor.

XR-Perf: Besides benchmark and stress test, we need a more flexible method to customize flow models, e.g., elephant and mice flows [51]. *XR-Perf* can fulfill these requirements. By integrating it into the monitoring system, we can collect information, analyze complicated scenarios and understand performance for RDMA and X-RDMA from a higher perspective.

C. Extra Schemes

Memory Cache Isolation: Raw RDMA libraries lack detection mechanism to that warn developers of memory access bugs, especially those caused by out-of-bound access to RDMA-enabled memory. As a result, it is inconvenient for the developer to find the root causes. In X-RDMA's implementation, the memory cache will be assigned to a higher address space near the stack space via `mmap`. Besides, memory cache addresses are marked to avoid conflict with other threads' addresses.

Emulate Fault: To emulate fault cases, X-RDMA implements a simple error injection module named *Filter*, for fault cases such as dropped messages, slow messages, etc. The developer can enable or disable filter online via the tuning system.

Switch between RDMA and TCP: To handle some rare RDMA network anomaly scenarios such as heavy congestion, high-degree incast or protocol stack collapse, X-RDMA provides a *Mock* mechanism to temporarily switch to TCP network as shown in Figure 6.

D. Tuning

Different applications may have distinctive models and requirements. As a middleware, X-RDMA supplies a mechanism to export useful X-RDMA-internal and lower-level RDMA tokens to those applications. As shown in Figure 6, while running an X-RDMA application, there is a specific idle thread that administers local configurations. An admin tool *XR-adm* is responsible for distributing the configurations to these control threads from the running X-RDMA applications. Table III gives several crucial configuration parameters in the production environment. X-RDMA subdivides them into two types: "online" which can change values dynamically, and "offline", which remain the same at runtime.

TABLE III: Configurations.

Name	**Description**
Online	
keepalive_intv_ms	keepAlive probe interval
slow_threshold	decide whether to record log about slow operation
polling_warn_cycle	threshold (between 2 polling) to detect slow poll
trace_sample_mask	indicate whether to trace a message
Offline	
use_srq	use SRQ or not
cq/srq_size	maximum SRQ wqes
fork_safe	support fork or not
ibqp_alloc_type	QP buffer type (Huge-Page/Anony-Page/Malloc)
small_msg_size	less than this threshold, use RDMA Send

VII. Evaluation and Experience

X-RDMA has been fully implemented and used in Alibaba for nearly two years. Many business systems benefit from X-RDMA by enhancing the robustness of data transfer and simplifying debugging procedures. With X-RDMA, the various services at Alibaba can survive heavy load situations, such as the moment when the peak throughput reaches 35.78 million requests per second during the shopping spree. The large-scale deployment of X-RDMA also exhibits strong robustness without any bugs within one year.

Deployment at Alibaba: At Alibaba, over 4000 servers are deployed with X-RDMA using RoCEv2 protocol. The largest RDMA based cluster contains 4 sub-clusters, each of them having 256 nodes. Nodes are typically equipped with a dual-port 25Gbps (50Gbps in total) Mellanox ConnectX4-Lx RNIC. Almost all business scenarios are deployed with X-RDMA as the basic component. These RDMA based productions include X-DB, ESSD, Pangu, and PolarDB [25], which require robustness, maintainability, and high performance. Applications built with X-RDMA form the infrastructure used for web shopping services at Alibaba. These services have been running under real-world workloads during shopping transactions.

A. Performance Benchmarks

Comparison to other middleware. We use X-RDMA to implement the ping-pong tests under *req-res* mode and *bare-data* mode respectively, and compare bare-data mode with `ibv_rc_pingpong` which is part of the native RDMA library. `ibv_rc_pingpong` can be seen as an ideal baseline compared to other test tools, since it has no extra overhead other than the primitive RDMA operations. Because X-RDMA adopts mixed messaging strategy, as shown in Figure 7, `XR-perf` achieves similar performance as `ibv_rc_pingpong` with at most 10% degradation on average in rare cases. We also compare X-RDMA with state-of-the-art RDMA libraries: UCX [52] and Libfabric [53]. In some cases, X-RDMA performs with 5%/10% lower latency (5.60μs) than the better one `ucx-am-rc` of UCX (5.87μs) and Libfabric (6.20μs).

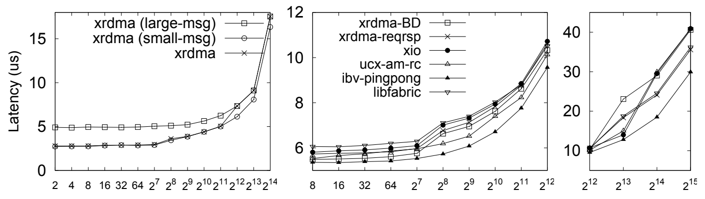


Fig. 7: The comparison between X-RDMA (bare-data/req-res), xio, ucx, and ibv_rc_pingpong under different data sizes.

With mixed messages, the larger ones need an extra RDMA operation for buffer-preparation. As shown in Figure 7, the latency of large messages is about 40% higher than small ones (with size under 128 Bytes). Beyond 128 Bytes, the difference is acceptable (at most 1.4μs which is less than 10% latency increment). In fact, the large size can benefit from buffer-preparation with lower memory footprint, which is only 1%∼10% of small size depending on CQ depth.

To evaluate the overhead brought in by tracing, we make a comparison of X-RDMA under *bare-data* mode and *req-res* mode. The results show the ping-pong latency only increases by 2∼4% for around 200ns extra time.

B. Programming Simplification

Compared to native RDMA: X-RDMA provides almost all required network functionalities for large-scale production. For instance, without X-RDMA, to implement data plane and protocols in Pangu, 2000 LOC native RDMA code is needed. In comparison, only about 40 LOC of X-RDMA APIs is required without any other network code.

Engineering Period: According to the statistics in a typical project named ERPC which is a `protobuf` RPC framework with RDMA support at Alibaba, compared with the original RDMA stack, X-RDMA has helped save at least 70% of man-month from development to maintenance for a team.

C. Robustness Enhancement

Establishment Time: Once QP cache is applied, the connection establishment time will decrease from 3946μs to 2451μs to save 38% of time. This comes from the reduction of QP creation time by the reuse of reclaimed QP. Besides, the connection establishment phase with 4096 connections only costs around 3s whereas 10s is required by using `rdma_cm` [37]. Figure 8 presents ESSD can switch to steady-state rapidly within less than 2s and reach 6 KOPS (around 24 Gbps).

RNR Free: Figure 9 shows that X-RDMA can ensure applications are RNR free with seq-ack mechanism, compared to the primitive RNR error number which is 0.91 on average.

Flow Control: We emulate the incast scenario for one node with 6144 connections and all out-bound RDMA Read/Write

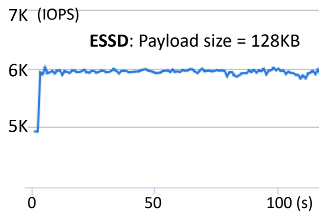


Fig. 8: Aggregate IOPS.

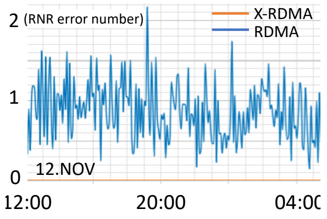


Fig. 9: RNR counter (Pangu).

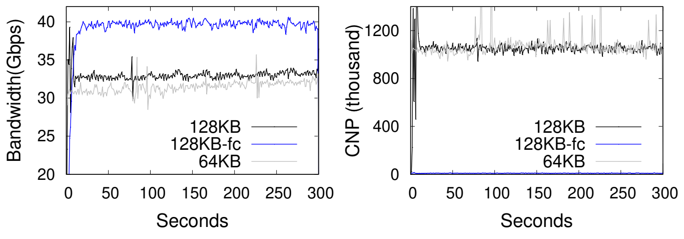


Fig. 10: The comparison among 64KB, 128KB and 128KB payload sizes with flow control (fc).

operations. Figure 10 shows the effects of flow control. CNP (Congestion Notification Packets) and $TX\ Pause$ are cruial indexes in DCQCN and PFC respectively. A higher value means the network suffers heavier congestion. After applying flow control, the bandwidth can be improved by around 24%. Besides, the average CNP number is reduced to 1∼2% and the $TX\ pause$ is directly minimized to nearly zero.

D. Tracking Case Study

To show the effects of X-RDMA's analysis framework, we select two cases for discussion.

Application Issue: we found that Pangu has occasional I/O jitter. With the self-adaptive logging in X-RDMA, the monitor system can gather these logs along those critical paths automatically. We have noticed that the polling phase of several threads sometimes has a higher overhead simultaneously. According to its location, we find the **allocator lock** in applications on the top of Pangu is the primary cause of these performance issues.

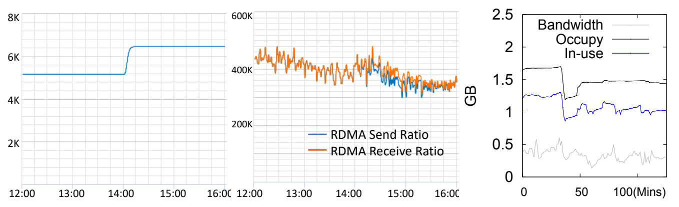
Network Issue: As some slow I/O situations only existed in specific servers, we tuned X-RDMA into `rep-res` on these servers. According to the monitor system, we found that the time spent was mostly on the network. After checking network status by some messages with recorded IP addresses and timestamps, we realized that out-of-sequence event was happening.

We have received several anecdotal stories about using analysis framework to successfully detect performance issues. Totally, before large-scale deployment of X-RDMA based applications at Alibaba, we has found out about 20 potential issues via analysis framework of X-RDMA. After production deployment, it again helped locate more than 10 issues in different layers of production application systems.

E. Production Evaluation

We have deployed about 1200 back-end servers with X-RDMA and 20K virtual machines (i.e., front-end servers) in

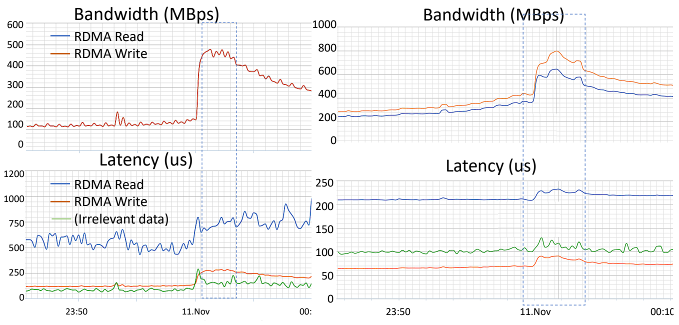
ESSD. In X-DB, another 1200 back-end servers and 10K dockers with mysql are running X-RDMA. Over 5000 connections exist on each server on average. As shown in Figure 11a and 11b, the online upgrading will increase the QP number rapidly but will not harm the performance or result in jitter. Figure 11c demonstrates that the memory caches operate smoothly with the changing bandwidth.



(a) Pangu QP Num (b) Pangu IOPS (c) Mem Usage

Fig. 11: Online QP number and Memory Usage (20.OCT).

Figure 12 shows the online monitoring about latency and throughput of two X-RDMA applications: ESSD and X-DB. Jitter is mitigated effectively under pressure. In the dotted box at Figure 12a, the throughput of ESSD is increased by nearly 300%. However, thanks to anti-jitter strategies (protocol extension and resource management), the latency has no significant increment during this period. Similar to ESSD, X-DB also has better performance in jitter mitigation and latency stabilization as shown in Figure 12b.



(a) ESSD (b) X-DB

Fig. 12: The online application anti-jitter (11.NOV).

Totally, during the shopping spree in 2018, ESSD handled massive requests at a peak of 35.78 MOPS. The peak write bandwidth is at thousands of Gigabytes per second scale in total with a 267.07μs end-to-end latency on average. As we expected, X-RDMA can handle it without any exception.

F. Experiences

Influence of RNIC cache is limited: Poor performance of RDMA for large-scale applications has been pointed out in some recent works [36]. This is due to the limited capacity of SRAM in RNIC [46]. According to our evaluation upon ConnectX-4 RNIC, cache influence on performance is almost below 10% even when the number of QP grows up to 60K. It should not be a major issue about scalability.

Pay attention to SRQ: With shared receive queue (SRQ), multiple QP's can bind its RQ to the same one [54]. SRQ can effectively reduce memory usage. However, it violates our RNR-free design principle which means SRQ can potentially cause network jitter. In X-RDMA, SRQ is supported although disabled by default. We suggest not to enable SRQ when the number of QPs for each node is under 10K.

Avoid to use continuous physical memory: According to our statistics, memory fragmentation on massive servers is inevitable. Using continuous physical memory can be cache-friendly [5], but this will cause out-of-memory issue and trigger memory recycling in kernel to slow down the whole system in some cases. We evaluate three modes (i.e., non-continuous, continuous and hugepage) and the results show that the non-continuous mode has comparable performance and less fragmentations.

VIII. Related Works

Rsocket [55], accelio [56] and UCX [52] are currently the three most commonly-used RDMA middleware. Rsocket is a simple wrapper of RDMA APIs. Accelio is an early implementation with complex abstractions to support both RDMA and TCP. UCX is the most recent, state-of-the-art implementation, but it is hard to maintain due to its diverse features. X-RDMA keeps the middleware compact and small. LITE [46] is an academic representative work, implemented within the Linux kernel to provide easy-to-use interfaces and share resources safely. As an industrial-grade project, Libfabric [53] provides abundant libraries aligned with application requirements and has good impedance match with multiple fabric hardware (InfiniBand, iWarp, RoCE, raw Ethernet, UDP offload, Omni-Path, GNI, etc). In contrast, at Alibaba, we tend to simplify and move data-plane from kernel to user-space completely to increase clarity and eliminate complex interference within the kernel as much as possible.

Beyond general purpose middlewares, MVAPICH2 [57] and [58] provides several MPI interfaces for HPC environments. Similarity, OpenSHMEM [59] provides one-sided RDMA and parallel-processing interfaces for distributed memory access. Remote Regions [32] argues that file system interfaces will be better while using RDMA. Several research works [3], [29] focus on how to improve the usage of RDMA and increase raw performance. Lu *et al* [60] exploit multiple RNIC ports and design a mechanism to avoid out-of-order packets and reach a nearly linear improvement of bandwidth with an increasing number of ports. Current Big Data softwares can further be optimized with the features of RDMA such as Spark [61] and Hyper-V [62]. From another perspective, we share X-RDMA experiences about commercial requirements such as robustness, efficient resource management, convenient debugging tools, etc in large-scale production systems, and how to resolve them with less impact on the raw performance.

Before the wide deployment of RDMA, several technical reports such as Portals [63], [64] propose a set of one-sided memory semantics APIs to support zero copy and kernel bypass within the protocol layer.

IX. Discussion

Towards X-RDMA, we would like to discuss the future development road-map of RDMA. Currently, data centers still have to face some challenges when using RDMA even after 10 years of deployment. Our experiences may help predict the possible trends.

Connection Establishment: X-RDMA uses QP cache to reduce the connection establishment time. However, we realize there is still huge room for improvement for hardware vendors. The connection overhead of modifying QP is mostly due to the synchronization of hardware resources. We expect to see improvement of RDMA hardware in the future. This improvement will help to eliminate I/O jitter in millisecond level and shrink recovery time dramatically.

Massive Connections in RC: We are evaluating DCT [65] from different aspects, and the recent test result shows DCT can benefit massive connections to some extent but DCT is not mature and stable enough in our tests.

Hardware-based Filter: X-RDMA offers a software-layer filter to test the network's robustness in production environments. However, direct RNIC support is more desirable.

Transparent to Software Developers: Native RDMA lacks enough support for performance analysis. We hope more hardware logic and counters in RNIC could be integrated into a unique framework like the Linux perf-tool [66].

Eradicate PFC: In some cases, PFC storm [10] happens with a higher possibility due to the blocking in the switch's MMU. PFC storm can easily incur network congestion, deadlock, performance downgrade and even unavailability [67] of the whole clusters. We think the trend to enhance flow control is to discard PFC [68] and focus on the lossy network [36].

X. Conclusion

In this paper, we have presented the issues and experiences about large-scale deployment of RDMA at Alibaba. Accordingly, we implement a middle-ware, X-RDMA, and fulfill the desires from practical applications in the modern data center.

X-RDMA has been successfully used in various applications at Alibaba for nearly two years. With X-RDMA, Alibaba's key businesses have been running in the production environments smoothly. X-RDMA helps achieve effective trade-offs between availability and performance, and also it is necessary complement to the native RDMA stack. Through X-RDMA, we have demonstrated to developers which features are the most desired ones in the production environments.

Acknowledgment

We would like to extend our sincere thanks to the entire X-RDMA team members and our colleagues for their enthusiastic assistance. At the same time, we thank *IEEE Cluster* anonymous reviewers for their helpful feedbacks. Teng Ma, Kang Chen and Yongwei Wu are with the Department of Computer Science and Technology, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, China and supported by National Key Research & Development Program of China (2016YFB1000504), Natural Science Foundation of China (61433008, 61373145, 61572280), China Postdoctoral Science Foundation (2018M630162). Corresponding Author: Kang Chen and Yongwei Wu.

References

[1] Mellanox, "Connectx-6 vpi card product brief," http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-6_VPI_Card.pdf, 2019.

[2] 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*, ser. EuroSys '19. New York, NY, USA: ACM, 2019, pp. 19:1–19:14.

[3] C. Mitchell, Y. Geng, and J. Li, "Using one-sided rdma reads to build a fast, cpu-efficient key-value store," in *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, 2013, pp. 103–114.

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

[5] 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.

[6] Y. Lu, J. Shu, Y. Chen, and T. Li, "Octopus: an rdma-enabled distributed persistent memory file system," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 773–785.

[7] N. S. Islam, M. W. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda, "High performance rdma-based design of hdfs over infiniband," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 35.

[8] M. Wu, F. Yang, J. Xue, W. Xiao, Y. Miao, L. Wei, H. Lin, Y. Dai, and L. Zhou, "Gram: scaling graph computation to the trillions," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 2015, pp. 408–421.

[9] W. Huang, Q. Gao, J. Liu, and D. K. Panda, "High performance virtual machine migration with rdma over modern interconnects," in *2007 IEEE International Conference on Cluster Computing*. IEEE, 2007, pp. 11–20.

[10] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016, pp. 202–215.

[11] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 523–536.

[12] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "Ictcp: Incast congestion control for tcp in data-center networks," *IEEE/ACM Transactions on Networking (ToN)*, vol. 21, no. 2, pp. 345–358, 2013.

[13] P. Balaji, H. V. Shah, and D. K. Panda, "Sockets vs rdma interface over 10-gigabit networks: An in-depth analysis of the memory traffic bottleneck," in *In RAIT workshop*, vol. 4, 2004, p. 2004.

[14] P. MacArthur and R. D. Russell, "A performance study to guide rdma programming decisions," in *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS)*. IEEE, 2012, pp. 778–785.

[15] M. Su, M. Zhang, K. Chen, Z. Guo, and Y. Wu, "Rfp: When rpc is faster than server-bypass with rdma," in *Proceedings of the Twelfth European Conference on Computer Systems*. ACM, 2017, pp. 1–15.

[16] Y. Chen, X. Wei, J. Shi, R. Chen, and H. Chen, "Fast and general distributed transactions using rdma and htm," in *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016, p. 26.

[17] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[18] G. F. Pfister, "An introduction to the infiniband architecture," 2001.

[19] Mellanox, "Roce in the datacenter," http://www.mellanox.com/related-docs/whitepapers/roce_in_the_data_center.pdf, 2014.

[20] Infiniband Trade Association, "Rocev2," 2014.

[21] RDMA consortium, "Architectural specifications for rdma over tcp/ip," 2009.

[22] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, and R. C. Zak, "Intel® omni-path architecture: Enabling scalable, high performance fabrics," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. IEEE, 2015, pp. 1–9.

[23] IEEE. 802.11Qbb, "Priority based flow control," 2011.

[24] Y. Zhu, M. Ghobadi, V. Misra, and J. Padhye, "Ecn or delay: Lessons learnt from analysis of dcqcn and timely," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM, 2016, pp. 313–327.

[25] W. Cao, Z. Liu, P. Wang, S. Chen, C. Zhu, S. Zheng, Y. Wang, and G. Ma, "Polarfs: an ultra-low latency and failure resilient distributed file system for shared storage cloud database," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1849–1862, 2018.

[26] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. USENIX Association, 2006, pp. 307–320.

[27] I. Habib, "Virtualization with kvm," *Linux Journal*, vol. 2008, no. 166, p. 8, 2008.

[28] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[29] 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*. ACM, 2019, p. 19.

[30] P. W. Frey and G. Alonso, "Minimizing the hidden cost of rdma," in *2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE, 2009, pp. 553–560.

[31] M. Li, K. Hamidouche, X. Lu, H. Subramoni, J. Zhang, and D. K. Panda, "Designing mpi library with on-demand paging (odp) of infiniband: challenges and benefits," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 433–443.

[32] M. K. Aguilera, N. Amit, I. Calciu, X. Deguillard, J. Gandhi, S. Novaković, A. Ramanathan, P. Subrahmanyam, L. Suresh, K. Tati, R. Venkatasubramanian, and M. Wei, "Remote regions: a simple abstraction for remote memory," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, 2018, pp. 775–787.

[33] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, "Revisiting network support for rdma," *Acm Special Interest Group on Data Communication*, pp. 313–326, 2018.

[34] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding tcp incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 73–82.

[35] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *Acm Special Interest Group on Data Communication*, vol. 45, no. 4, pp. 123–137, 2015.

[36] 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.

[37] RDMA_CM, "Rdma_cm website," https://github.com/ofiwg/librdmacm, 2018.

[38] L. Eggert and F. Gont, "Tcp user timeout option," 2009.

[39] G. Vigna and R. A. Kemmerer, "Netstat: A network-based intrusion detection approach," in *14th Annual Computer Security Applications Conference*. IEEE, 1998, pp. 25–34.

[40] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 139–152.

[41] H. Welte, "The netfilter framework in linux 2.4," in *Proceedings of Linux Kongress*, 2000.

[42] C. Li, C. Ding, and K. Shen, "Quantifying the cost of context switch," in *Proceedings of the 2007 workshop on Experimental computer science*. ACM, 2007, p. 2.

[43] P. Emmerich, D. Raumer, A. Beifuß, L. Erlacher, F. Wohlfart, T. M. Runge, S. Gallenmüller, and G. Carle, "Optimizing latency and cpu load in packet processing systems," in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*. IEEE, 2015, pp. 1–8.

[44] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda, "High performance rdma-based mpi implementation over infiniband," *International Conference on Supercomputing*, vol. 32, no. 3, pp. 295–304, 2003.

[45] Y. Wu, T. Ma, M. Su, M. Zhang, C. Kang, and Z. Guo, "Rf-rpc: Remote fetching rpc paradigm for rdma-enabled network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 7, pp. 1657–1671, 2019.

[46] S.-Y. Tsai and Y. Zhang, "Lite kernel rdma support for datacenter applications," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 306–324.

[47] K. R. Fall and W. R. Stevens, *TCP/IP illustrated, volume 1: The protocols*. Addison-Wesley, 2011.

[48] RDMAmojo, "Zero byte messages," https://www.rdmamojo.com/2013/09/20/zero-byte-messages/, 2015.

[49] J. Postel, "Transmission control protocol," Tech. Rep., 1981.

[50] D. Mills, *Computer Network Time Synchronization: The Network Time Protocol*. CRC Press, 2006. [Online]. Available: https://books.google.com.hk/books?id=pdTcJBfnbq8C

[51] Y. Zhang, J. Gu, Y. Lee, M. Chowdhury, and K. G. Shin, "Performance isolation anomalies in rdma," in *Proceedings of the Workshop on Kernel-Bypass Networks*. ACM, 2017, pp. 43–48.

[52] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss *et al.*, "Ucx: an open source framework for hpc network apis and beyond," in *IEEE 23rd Annual Symposium on High-Performance Interconnects (HOTI)*. IEEE, 2015, pp. 40–43.

[53] OpenFabrics, "Libfabric," https://ofiwg.github.io/libfabric/, 2019.

[54] S. Sur, L. Chai, H.-W. Jin, and D. K. Panda, "Shared receive queue based scalable mpi design for infiniband clusters," in *20th International Parallel and Distributed Processing Symposium*. IEEE, 2006, pp. 10–pp.

[55] S. Hefty, "Rsockets," in *2012 OpenFabris International Workshop, Monterey, CA, USA*, 2012.

[56] Accelio., "accelio," http://www.accelio.org",, 2013.

[57] W. Huang, G. Santhanaraman, H. Jin, Q. Gao, and D. K. Panda, "Design of high performance mvapich2: Mpi2 over infiniband," in *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, vol. 1, 2006, pp. 43–48.

[58] T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, "High performance rdma protocols in hpc," *Lecture Notes in Computer Science*, pp. 76–85, 2006.

[59] P. Shamis, M. G. Venkata, S. Poole, A. Welch, and T. Curtis, "Designing a high performance openshmem implementation using universal common communication substrate as a communication middleware," in *Workshop on OpenSHMEM and Related Technologies*. Springer, 2014, pp. 1–13.

[60] Y. Lu, G. Chen, B. Li, K. Tan, Y. Xiong, P. Cheng, J. Zhang, E. Chen, and T. Moscibroda, "Multi-path transport for rdma in datacenters," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 357–371.

[61] X. Lu, M. W. U. Rahman, N. Islam, D. Shankar, and D. K. Panda, "Accelerating spark with rdma for big data processing: Early experiences," in *2014 IEEE 22nd Annual Symposium on High-Performance Interconnects*. IEEE, 2014, pp. 9–16.

[62] Mellanox, "Applications acceleration with mellanox rdma enabled networking solutions," http://www.mellanox.com/related-docs/solutions/RoCE_MSFT_StorageSpaces_SB.PDF, 2019.

[63] R. B. Brightwell, T. B. Hudson, R. E. Riesen, and A. B. Maccabe, "The portals 3.0 message passing interface revision 1.0," Sandia National Labs., Albuquerque, NM (US), Tech. Rep., 1999.

[64] B. W. Barrett, R. Brightwell, S. Hemmert, K. Pedretti, K. Wheeler, K. D. Underwood, R. Reisen, A. B. Maccabe, and T. Hudson, "The portals 4.0 network programming interface," *Sandia National Laboratories, November 2012, Technical Report SAND2012-10087*, 2012.

[65] R. Graham, "Dynamically-connected transport," https://www.openfabrics.org/images/eventpresos/workshops2014/DevWorkshop/presos/Monday/pdf/05_DC_Verbs.pdf, Annual OFA Workshop, 2014.

[66] A. C. De Melo, "The new linux'perf'tools," 2010.

[67] Y. Gao, Y. Yang, T. Chen, J. Zheng, B. Mao, and G. Chen, "Dcqcn+: Taming large-scale incast congestion in rdma over ethernet networks," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 2018, pp. 110–120.

[68] Y. Le, B. Stephens, A. Singhvi, A. Akella, and M. M. Swift, "Rogue: Rdma over generic unconverged ethernet," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2018, pp. 225–236.