EasyChair Preprint

№ 1302

HiPower: A High-performance RDMA Acceleration Solution for Distributed Transaction Processing

Runhua Zhang, Yang Cheng, Jinkun Geng, Shuai Wang,
Kaihui Gao and Guowei Shen

July 18, 2019

HiPower: A High-performance RDMA Acceleration Solution for Distributed Transaction Processing

Runhua Zhang[1,2,3], Yang Cheng[2], Jinkun Geng[2], Shuai Wang[2], Kaihui Gao[2], and Guowei Shen[1,3]

[1] Dept. of Computer Science and Technology, Guizhou University, Guizhou, China
[2] Dept. of Computer Science and Technology, Tsinghua University, Beijing, China
[3] CETC Big Data Research Institute Co. Ltd., Chengdu, China
{zhangrh18, cheng-y16}@mails.tsinghua.edu.cn, steam1994@163.com
{s-wang17, gkh18}@mails.tsinghua.edu.cn, gwshen@gzu.edu.cn

Abstract. The increasing complex tasks and growing size of data have necessitated the application of distributed transaction processing (DTP), which decouples tasks and data among multiple nodes for jointly processing. However, compared with the revolutionary development of computation power, the network capability falls relatively behind, leaving communication as a more distinct bottleneck. This paper focuses on the recent emerging RDMA technology, which can greatly improve communication performance but cannot be well exploited in many cases due to improper interactive design between the requester and responder. Our research finds that the typical implementation of confirming per work request (CPWR) triggers considerable CPU involvement, which further degrades the overall performance of RDMA communication. Targeting at this, we propose HiPower, which leverages a batched confirmation scheme with lower CPU utilization, to improve high-frequency communication efficiency. Our experiments show that, compared with CPWR, HiPower can improve the communication efficiency by up to 75% and reduce CPU cost by up to 79%, which speeds up the overall FCT (Flow Completion Time) by up to 14% on real workflow(Resnet-152).

Keywords: RDMA · Distributed Transaction Processing · Batched confirmation · One-by-one confirmation.

1 Introduction

Distributed transaction processing (DTP) has become a practical problem with extensive study in system building. Generally speaking, performance optimization for DTP systems can be summarized as computation acceleration and communication acceleration. In the past decades, the computation power has been greatly improved due to the rapid development of hardware accelerators, such as GPUs and TPUs [16]. Whereas the communication capability, although also making some progress, cannot match the speed of computation enhancement and is left as the major bottleneck in DTP. A series of related works have shown that network performance now has a substantial impact on the efficiency of DTP [2,

15, 16]. Considering this, the recent emerging RDMA technology is widely concerned and is believed to remedy the communication deficiency in DTP.

There are two types of RDMA operations (i.e., one-sided RDMA and two-sided RDMA) widely used today. Between them, the former one is more pursued in DTP scenarios, because it directly accesses the memory of remote server without involving kernel and remote CPUs, which has been widely implemented in some applications like the Key-Value system [17, 5], etc. However, pure one-sided RDMA is not suitable for the distributed applications that the receiver needs to perceive data. Some current RDMA designs use the native `RDMA_WRITE_WITH_IMM` operation to ensure that the receiver can perceive data, which also have some distinct drawbacks, and we argue as follows:

First, one-by-one message confirmation triggers much CPU overhead. Concretely speaking, the `RDMA_WRITE_WITH_IMM` operation first writes user-data to the remote memory in a context-oblivious way and then uses the immediate value to notify the receiver. The receiver can get the address of user-data written before from the immediate value. It provides higher communication performance than two-sided RDMA and TCP/IP. However, CPU is involved in the confirmation of each message, which incurs much overhead and affects the overall communication performance, especially when CPU resource is scarce or burnt for other processing logic. The performance drawbacks can become more distinct in many-to-one primitives like *gather* or high-frequency one-to-one transmission scenarios.

Second, one-by-one recycling reusable memory is an inefficient way with high-concurrent communication flows. In order to support concurrent operations and avoid the expensive cost of temporary registration [4], both sender and receiver register multiple MRs for data transmission. The receiver sends feedback to the sender after getting the user-data, which means the memory can be reused. Usually, the feedback can be achieved by two-sided RDMA. However, too many send/receive operations cause unacceptable communication overhead on both sides.

Targeting at the drawbacks, we design HiPower, which uses a batched confirmation mechanism to improve communication performance and reduce CPU utilization. Compared with existing works, HiPower enjoys three main advantages to achieve faster communication.

(1) **More efficient message confirmation**. HiPower allows the receiver to perceive messages through a well-designed bitmap and reduce the number of RECV operations, thereby reducing CPU involvement and improving communication efficiency.

(2) **More economic reusability and better quality of service (QoS)**. HiPower also uses the bitmap flag to represent batched reusable MRs, which can effectively improve the memory utilization, as well as improve the concurrency. In addition, HiPower guarantees better quality of service through pre-registered reusable memory pools.

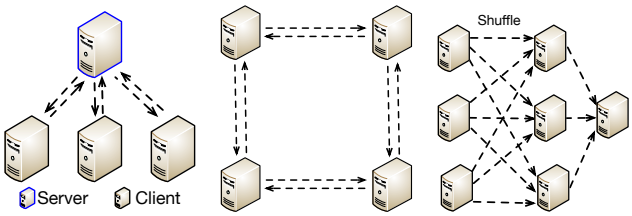
[4] The cost includes the time to register MRs and exchange necessary information of MRs

(3) **Strong compatibility and usability**. HiPower is implemented as a middle layer between distributed applications and RDMA communication libraries, thus it keeps transparency to the upper layer and provides performance boosts for various DTP applications.

The rest of this paper is organized as follows: Section 2 briefly introduces the background and motivation of our work. Section 3 describes the design details of HiPower. Section 4 shows the results of the experiment and proves the outperformance of HiPower. Section 5 includes the related work and, Section 6 concludes the paper.

2 Background and Motivation

2.1 Typical Communication Pattern in DTP



(a) PS (b) Ring (c) Map/Reduce

Fig. 1: Distributed transaction processing architecture

DTP partitions a task among multiple servers for data processing and synchronization. Figure 1 illustrates the typical DTP architectures in practice, including Parameter Server (PS)-based, Ring-based, Map/Reduce-based, etc. Recent studies have shown that the performance of a single GPU has improved by 35× compared to that in previous years [16], but communication can hardly match the speed. Among the typical architectures for DTP tasks, the communication bottleneck is becoming more distinct. In order to mitigate the communication bottleneck, high-performance communication methods represented by RDMA are gaining more attention in the field of DTP. However, RDMA suffers from a couple of essential drawbacks, and requires us to carefully design to well embrace communication capabilities [11, 3, 10, 20].

2.2 Background on RDMA

RDMA is well known for its zero-copy and kernel-bypass features, when compared with the TCP counterpart. Traditional TCP cannot serve high-speed dataflow well since it involves complicated kernel processing and at least two copies. In contrast, RDMA sinks the protocol stack to hardware and thus avoids the overhead of context switching. RDMA supports both one-sided and two-sided communication operations. As for one-sided RDMA, user-level applications can directly access the memory of a remote node. Note that the remote node is unaware and do not need CPU to involve. `RDMA_WRITE` and `RDMA_READ` are two typical one-sided operations. As for two-sided RDMA, operations must appear

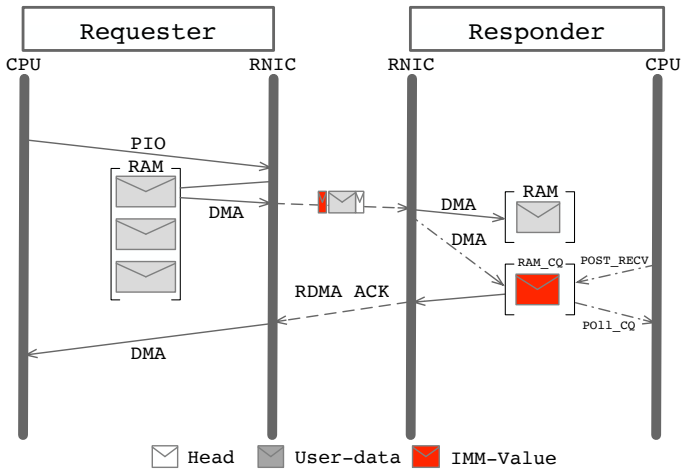


Fig. 2: The whole process of `RDMA_WRITE_WITH_IMM` operation.

in pairs. More specifically, the `RDMA_RECV` operation should be prepared before launching an `RDMA_SEND` operation. One-sided operations achieve higher performance than two-sided operations[11].

Our research observes that one-sided RDMA operations can effectively solve the communication bottleneck of practical applications like key-value store systems. For example, a client submits data to the server, which can be achieved by `RDMA_WRITE` with a hash algorithm. The server can be seen as a memory-based database and don't need to perceive data. However, pure one-sided RDMA is not suitable for some DTP applications like Distributed machine learning (DML) or Distributed data processing because the receiver needs to perceive data. Most of current designs use `RDMA_WRITE_WITH_IMM` operations for data transfer and notification. The `RDMA_WRITE_WITH_IMM` operation first writes user-data to the remote memory in an unaware way and then uses the immediate value to notify the receiver. The receiver can obtain the storage address of user-data from the immediate value. It provides higher communication performance than two-sided RDMA and TCP/IP. Figure 2 presents the workflow of this operation.

However, in order to receive the immediate value, the receiver needs to prepare the `RDMA_RECV` event in advance as the two-sided RDMA. It means that the CPU needs to participate in the confirmation of each message, which triggers much CPU overhead and affects the overall communication performance to some extent. It becomes more serious in many-to-one primitives (e.g. Gather) or high-frequency one-to-one transmission scenarios. High CPU overhead is unacceptable to us for the reason that a large number of distributed applications are deployed in the cloud, where strong hardware is provided to speed up computation. Recent studies have shown that the CPU is a precious commodity in cloud service [13]. It's necessary to reduce CPU participation. In addition, in order to support concurrent operations and avoid the expensive cost of temporary registration, both sender and receiver register multi-MRs for data transmission. The receiver sends feedback to the sender after getting the user-data, which means the memory can be reused. Usually, the feedback can be achieved by two-sided RDMA. However,

too many send/receive operations incur significant communication overhead on both sides.

3 Design for HiPower

In HiPower, we focus on improving the communication performance in DTP based on one-sided `RDMA_WRITE` operation, which has been proved higher efficiency than the two-sided `RDMA_WRITE` operation in previous works [18, 11, 5].

3.1 HiPower Overview

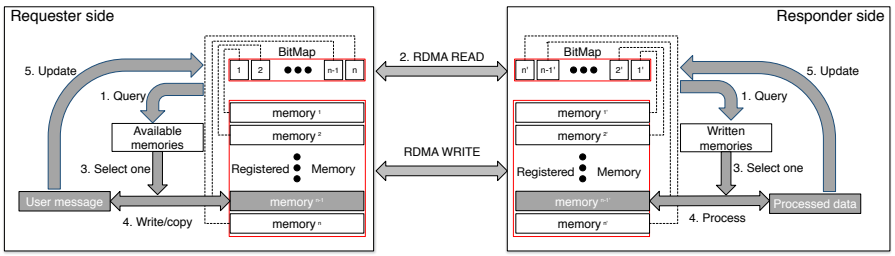


Fig. 3: The architecture of HiPower

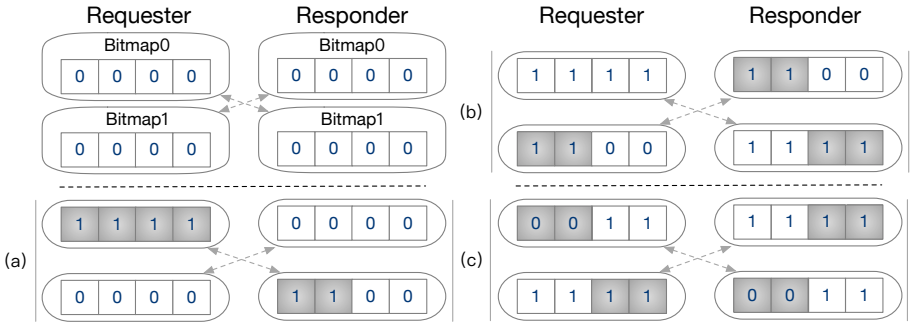


Fig. 4: The design of bitmap.

There are two roles involved in HiPower: the requester and the responder. The requester acts as a data generator, which will transport user-data to the responder. The responder acts as a data consumer and will process the user-data it received. The architecture of HiPower is shown in Figure 3.

Both the requester and the responder first perform initialization operations and then take further actions according to their business logic. At the initialization stage, both requester and responder register a number of Memory Regions(MR). Every MR in each side is associated with a mirrored one in the peer's side with a unique id, respectively. These MRs can be registered in the CPU or GPU memory, which we call MRs-Pool. As shown in Figure 4, we use an extra MR called bitmap0 on each side to record the transmission state of local MRs. Meanwhile, we use an extra MR called bitmap1 on each side to record the transmission state of the peer's MRs. Each item in the bitmap is associated with one local memory. There are two different states for each item. The state of each item will be converted once the related memory is changed.

The requester launches the `RDMA_READ` operation to query the state of remote bitmap0 and saves it to local bitmap1 and then compares local bitmap0 with bitmap1 to collect all available MRs. For the requester, the MR is available only if the state of local bitmap0's item is same as the state of local bitmap1's item. To limit the frequency of querying event and better overlap computation and communication, we put the reusable MRs of the responder into a pool and perform the `RDMA_READ` operation again when the number of reusable MRs in the pool below a certain threshold (e.g., 40% of the total MRs). The requester can select one or more available MRs and then posts the `RDMA_WRITE` operation. RNIC writes user-data to the related MR of the responder directly and updates the state of local bitmap0's item. Last, the requester sends an end-flag to the responder after all the transfers are completed.

The responder queries the bitmap0 of the requester by launching a `RDMA_READ` operation and saves it to local bitmap1 and then compares local bitmap0 with bitmap1 and finds all MRs that have stored user-data. Different from the requester, the MR carries user-data only if the state of requester bitmap1's item is different from the state of local bitmap0's item. The Responder also puts the MRs into a pool and then processes them according to its business logic.

3.2 Remarkable Advantages of HiPower

The bitmap design in HiPower enjoys three remarkable advantages compared to the baseline solutions, which we summarize as follows.

Batched Confirmation and Recycle Mechanism In order to perceive the user-data, RNIC needs to confirm each packet and Requester needs to recycle reusable memory, which incurs a large amount of CPU utilization and significantly affects the communication efficiency. HiPower mitigates this problem with batched confirmation and recycles mechanism, which can perceive multiple data-MRs and obtain multiple reusable MRs with the bidirectional `RDMA_READ` operation. The reusable MRs will be put into a pool, which can be directly obtained from the pool when the next `RDMA_WRITE` operation is performed. The Responder also puts the MRs into a pool and then processes them according to its business logic.

Strong Quality of Service Native RDMA lacks quality of service (QoS). For example, as for many-to-one communication under the PS architecture, both parties need to temporarily register memory and exchange memory information for further communication. At this point, the parameter server needs to temporarily register lots of MRs for multiple senders. If the available physical memory is insufficient, registration operations will fail. The application will not be aware of the physical memory state in time when the memory resource is released. HiPower guarantees QoS of the entire system by pre-registering memory and reusing the MRs more efficiently. As for larger DTP applications, the amount of MRs can be allocated according to the size of actual physical memory, business requirements and the ratio of sender/receiver.

Low Consumption for Memory Bitmap0 of requester and responder will be modified and read at the same time. Our design avoids mutex locks and further reduced system overhead. To further mitigate the effects of query overhead, each

bitmap occupies only one MR, and the size of each item is only 1 bit. Therefore, the time complexity of single processing is O(C) and the total time complexity is $O(Cn)$ [5], which is acceptable for most practical DTP cases.

4 Implementation and Evaluation

4.1 Experiment Setting

We deployed the comparative experiment on 5 servers. Each server is equipped with a Mellanox ConnectX-3 40Gbps NIC, two Intel Xeon E5 CPUs (each CPU has 16 physical cores) and 64GB DRAM. We implement both HiPower and Vanilla with 4500 lines of C++ codes and run the prototypes in Ubuntu 16.04. More specifically, as for Vanilla, we use `RDMA_WRITE_WITH_IMM` to transfer user-data and the responder confirms them in turn. Then, we launch two-sided RDMA to feedback the message of reusable memory. As for HiPower, we use `RDMA_WRITE` to transfer user-data. Next, we use `RDMA_READ` with the bitmap to confirm user-data and recycle reusable memory. For fairness, we use the same size memory pool for HiPower and Vanilla. The threshold is set to 50% of the total MRs.

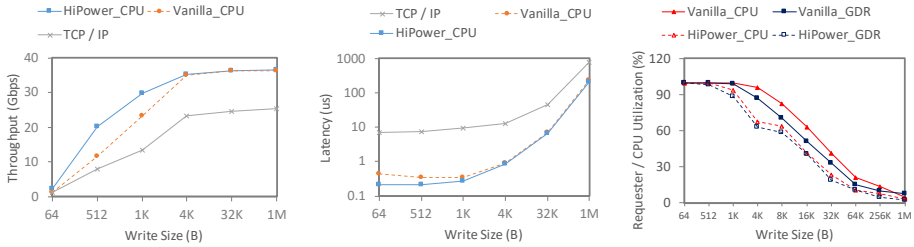
We conduct our experiments with three commonly-used types of communication primitives: one-to-one, one-to-many(broadcast), many-to-one(gather). As for broadcast and gather primitives, we choose one server as the master and the other four servers as workers. We evaluate the performance of the two prototypes using CPU-based and GPUDirect RDMA-based memory transport and take throughput, latency and CPU utilization as our metrics to evaluate the performance of both prototypes. We also use a practical DML application (i.e. ResNet-152 model training) as the benchmark to further compare the performance between HiPower and baseline solutions.

4.2 Experiment Result and Analysis

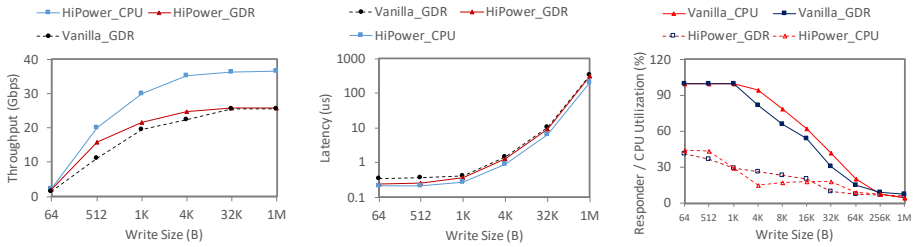
One-to-One Communication Pattern One-to-One communication plays a vital role in ring-based architecture. In other words, each pair of adjacent servers in a ring-based architecture can be considered as one-to-one communication.

The comparison of CPU-based and GDR-based throughput performance can be illustrated in Figure 5(a) and Figure 5(d). The experimental results show that our batched confirmation strategy makes HiPower achieve a higher throughput performance than Vanilla, especially for data sizes below 4KB. We calculate the average throughput of HiPower and Vanilla in one million iterations. In CPU-based transmission scenario, as for 512B and 1KB packets, vanilla's throughput is 11.5Gbps and 23.3Gbps respectively, while HiPower's throughput is 20.1Gbps and 29.9Gbps respectively. Test performance based on GDR transmission scenarios is slightly inferior to the former, which improve throughput performance by 11.2% and 43% respectively. As for data sizes above 4KB, the throughput performance of HiPower and Vanilla is basically the same. However, in most cases, HiPower's requester and responder CPU utilization are significantly lower than that of vanilla. We will discuss it in detail later. In addition, we use qperf

[5] C is a constant, which denotes the number of MRs pre-registered. n denotes the number of executions of `RDMA_READ`.



(a) Throughput comparison results base on RDMA (b) Latency comparison results base on RDMA (c) Requester CPU utilization comparison results



(d) Throughput comparison results base on GDR (e) Latency comparison results base on GDR (f) Responder CPU utilization comparison results

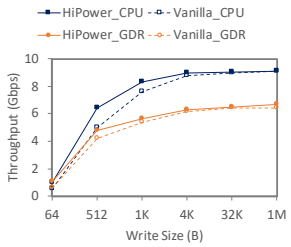
Fig. 5: One-to-one performance comparison.

[1] to measure the performance of TCP/IP on RNIC/40Gbps. The throughput performance of TCP/IP is mostly lower than HiPower and Vanilla.

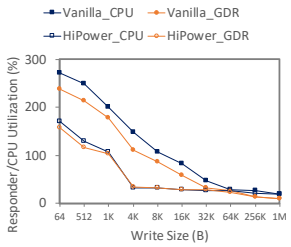
The comparison of CPU-based and GDR-based latency performance can be illustrated in Figure 5(b) and Figure 5(e). TCP/IP's latency is always higher than both HiPower and Vanilla. As for both CPU-based and GPU-based transmission scenario, HiPower has lower latency for packets below 4KB than Vanilla. As the packet size grows, HiPower and Vanilla's latency continues to shrink, but HiPower's latency is never higher than vanilla.

Figure 5(c) and Figure 5(f) take a closer look at the requester/CPU utilization comparison and responder/CPU utilization comparison of HiPower and Vanilla. Both of them contain two transmission scenarios. As for requester, HiPower has lower CPU utilization than Vanilla. Among them, Vanilla's CPU utilization includes `RDMA_WRITE_WITH_IMM` operation and `RDMA_RECV` operation. HiPower's CPU utilization includes `RDMA_WRITE` and `RDMA_READ` operations. As for data sizes below 4KB, HiPower launches more WRITE operations than Vanilla, but the CPU utilization is lower because Vanilla needs to continuously perform `RDMA_RECV` operations to obtain reusable memory messages. On the contrary, HiPower only needs one `RDMA_READ` operation to get multiple reusable memory, which greatly reduces CPU overhead. As for the data size between 4KB and 256KB, it can be further verified. The throughput performance of HiPower and Vanilla are basically the same, and the extra CPU utilization comes from in-

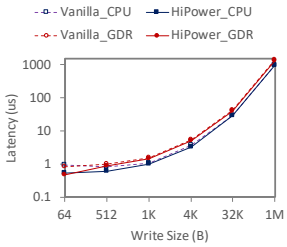
efficient recycling scheme. HiPower can reduce the frequency of `RDMA_READ` operations so that each reclaim can acquire multiple reusable memories while ensuring that the sender has enough reusable memory. As for the data size above 256KB, Vanilla's `RDMA_RECV` operation slows down, so their CPU utilization is basically the same. As for responder, Vanilla's CPU utilization includes `RDMA_RECV` operation and `RDMA_SEND` operation while HiPower only has the `RDMA_READ` operation, HiPower saves half of the CPU utilization compared to Vanilla in CPU-based and GDR-based transmission scenarios. The main reason is that HiPower avoids lots of `RDMA_SEND` and `RDMA_RECV`(Used to receive imm_value) operations.



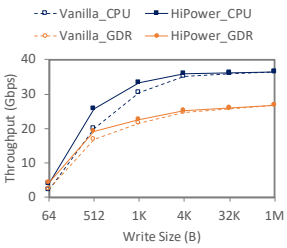
(a) Gather throughput comparison results



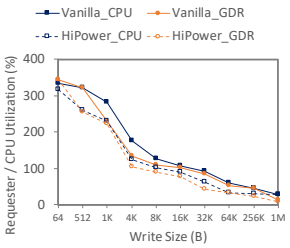
(b) Gather CPU utilization comparison results



(c) Latency comparison results



(d) Broadcast throughput comparison results



(e) Broadcast CPU utilization comparison results

Fig. 6: Broadcast and gather performance comparison.

Incast Communication Pattern Incast communication mainly includes broadcast and gather. Map operations in the Map/reduce architecture and parameter update operations in the PS architecture can be viewed as broadcast [7]. Parameter syncing operations in the PS architecture can be viewed as gather.

We further use the incast pattern to compare the communication performance of the two prototypes, and the throughput performance comparison is illustrated in Figure 6(a) and 6(d) respectively. In the experiment, we found that the throughput of each worker in the gather scenario is close to 1/n of the master server's total throughput in the broadcast scenario (where n represents the number of workers) and never exceeds the total throughput. The total throughput in the broadcast scenario is slightly higher than the one-to-one scenario. As for broadcast and gather, we try to send packets in a faster way,

which trigger the PFC pause frame[6] and the throughput is no longer increasing. One major reason is that the Receiver queue buffer has reached the upper limit. This phenomenon is unavoidable when the rate of packet delivery is too fast. In this paper, we are more concerned about the maximum performance that two prototypes can achieve in the incast pattern. Experimental results show that the batched confirmation scheme can detect user-data and reusable memory more quickly. As for broadcast, HiPower improves the throughput performance of 8.9%∼70% and 5.9%∼75% compared to Vanilla in CPU-based and GDR-based scenarios respectively. As for gather, HiPower improves the throughput performance of 7.8%∼68% and 5.2%∼72% compared to Vanilla in CPU-based and GDR-based scenarios respectively.

The latency comparison is illustrated in Figure 6(c). In our experiments, we found that the latency for each stream (four in total) in the gather scenario is higher than the latency of each worker in the broadcast scenario (four in total). When transmitting packets over 4KB, the latency is greatly increased, and each stream's latency is about four times of one-to-one. Similar to one-to-one communication Pattern, HiPower has lower latency for packets below 4KB. As the packet size grows, the latency gap between HiPower and Vanilla shrinks, but HiPower's latency is never higher than vanilla.

For incast communication pattern, we pay more attention to the CPU utilization of the master server in both the gather and broadcast scenario, which are illustrated in Figure 6(b) and Figure 6(e). As for the gather server, HiPower can save up to 79% CPU utilization than Vanilla in both the CPU-based and GPU-based scenarios. When transferring large size packets, HiPower's CPU utilization is comparable to that of Vanilla. As for the broadcast server, compared with Vanilla, HiPower can save 20% and 15% CPU utilization in CPU-based and GPU-based scenarios respectively.

Case Study on ResNet-152 We conduct another experiment with a DML application (i.e. Resnet-152) to evaluate how HiPower performs on real workflow. We transport traffic flow of training ResNet[9] among the distributed cluster.

ResNet is a classic deep learning model and has won the champaign of ILSVRC2016, it inspires the model designer exploring deeper neural network for higher accuracy. How to design a DL model for high accuracy is beyond our discussing in this scope and e only focus on how to accelerate the training phase among the distributed cluster.

We choose ResNet-152 as our benchmark application, which contains 152 layers and 60 million parameters (organized in 514 blocks) in all. Most of these blocks are less than 4KB and the maximum one is no more than 8 MB. To remove the bottleneck of the network, the 60 million parameters (i.e. 240MB) must be synchronized within 0.15s each iteration[7]. Bandwidth consumption will

[6] PFC (Priority-based Flow Control), priority-based flow control. The upstream device is notified to suspend the delivery by sending a Pause frame to prevent the buffer from overflowing.

[7] it is derived from NVcaffe rc 0.17 with GPU supported. The input size is 224 * 224 * 224 * 3 and the batch size is 16. It is trained by P100. The forward stage cost 0.1s, and backward cost about 0.15s

increase when training the ResNet-152 model with more nodes or by the smaller batch size, which brings a heavier burden to the network.

We compare the performance of our proposed prototype with the Vanilla and find that the FCT (Flow completion time) of HiPower is about 64 ms, FCT of Vanilla is about 73 ms. The HiPower shows 14% gains.

5 Related Work

Optimizations of RDMA Transmission. RDMA is a popular hardware-based solution. It aims to provide high-performance transport services but requires careful configuring and deep understanding. Recent study[12, 4] gives a comprehensive analysis of RDMA verbs from a low-level perspective (e.g. PCIe, NIC and etc.) and offers guidelines on how to use RDMA verbs efficiently. Frey, et.al.[6] have also explored the hidden cost in using the RDMA verbs from the hardware-resources view. Taking task's workflow and RDMA verbs into consideration has been proved to be effective in many scenarios(e.g. Key-Value Stores, File System and etc.). Pilaf [17] optimizes the `get` operation using multiple `RDMA_READ` commands at the client side. Recently, RDMA has also been used in distributed machine learning. Yi, et.al.[2] focus on improving transport performance in distributed machine learning. They implemented a zero-copy RDMA-based transport service. Many works are explored to find the hidden obstacles in the application of RDMA such as PFC issues[8] etc. RDMA optimizations have brought benefits to computer systems, and this motivates us to start rethinking the design with RDMA in building high-performance transport services.

Mitigation of Communication Overheads in DTP. Distributed transaction processing has become the standard practice and there have been extensive studies focusing on mitigating the communication overheads in DTP. Communication compression technique can be well incorporated in the DTP process. Since most of the data transmitted by DTP are compressible numbers, such as zero, small integers and 32-bit foats with high precision, the communication costs can be reduced by using compression algorithms. Li et al. [14] compress the sparse matrix by eliminating most zeros values and Wei et al. [19] used a 16-bit float to replace 32-bit float value to improve the utilization of bandwidth. The effectiveness of these compression-based solutions are also demonstrated in the recent works for mitigation of communication overheads in DTP.

6 Conclusion

This paper proposes HiPower, which is a novel RDMA-accelerated solution for distributed transaction processing (DTP). Compared with existing works, HiPower leverages an elaborate bitmap design to execute batched transmission and confirmation, thus can efficiently improve communication efficiency and reduce CPU utilization for DTP tasks. HiPower can adjust dynamically to fit more complicated scenarios such as CPU-based computing and GPU-based computing. Our evaluations prove the effectiveness of DTP and the experimental results show that HiPower can achieve higher throughput performance and lower latency while consuming lower CPU utilization. Besides, HiPower can also reduce 14% FCT in ResNet-152 training compared to existing works(CPWR), which implies great potential performance gains to more practical applications.

Acknowledgement. This work is supported by the National Natural Science Foundation of China (No. 61802081), the Guizhou Provincial Natural Science Foundation(No. 20161052,No.20183001)

References

1. qperf - measure rdma and ip performance. Tech. rep., Johann George (2009), refer: https://linux.die.net/man/1/qperf
2. How to compile, use and configure rdma-enabled tensorflow. Tech. rep., HKUST and Tensorflow community (2018), refer: https://github.com/ tensorflow/tensorflow/tree/master/tensorflow/contrib/verbs
3. Chen, H., Chen, R., Wei, X., Shi, J., Chen, Y., Wang, Z., Zang, B., Guan, H.: Fast in-memory transaction processing using rdma and htm. TOCS'17
4. Dragojevic, A., Narayanan, D., Castro, M.: Rdma reads: To use or not to use? 2017 IEEE Data Eng. Bull.
5. Dragojević, A., Narayanan, D., Hodson, O., et al.: Farm: Fast remote memory. In: NSDI'14
6. Frey, P.W., Alonso, G.: Minimizing the hidden cost of rdma. In: 2009 29th IEEE International Conference on Distributed Computing Systems
7. Geng, J.: CODE: Incorporating correlation and dependency for task scheduling in data center. In: ISPA'17
8. Guo, C., Wu, H., Deng, Z., Soni, G., Ye, J., Padhye, J., Lipshteyn, M.: Rdma over commodity ethernet at scale. In: SIGCOMM'16
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR'16
10. Kalia, A., Kaminsky, M., Andersen, D.G.: Fasst: Fast, scalable and simple distributed transactions with two-sided (rdma) datagram rpcs. In: OSDI'16
11. Kalia, A., Kaminsky, M., Andersen, D.G.: Using rdma efficiently for key-value services. SIGCOMM'15
12. Kaminsky, A.K.M., Andersen, D.G.: Design guidelines for high performance rdma systems. In: ATC'16
13. Kim, D., Memaripour, A., Badam, A., Zhu, Y., Liu, H.H., Padhye, J., Raindel, S., Swanson, S., Sekar, V., Seshan, S.: Hyperloop: group-based nic-offloading to accelerate replicated transactions in multi-tenant storage systems. In: SIGCOMM'18
14. Li, M., Andersen, D.G., Smola, A.J., Yu, K.: Communication efficient distributed machine learning with the parameter server. In: Advances in Neural Information Processing Systems. pp. 19–27 (2014)
15. Lu, X., Rahman, M.W.U., Islam, N., Shankar, D., Panda, D.K.: Accelerating spark with rdma for big data processing: Early experiences. In: Hot Interconnects'14
16. Luo, L., Nelson, J., Ceze, L., Phanishayee, A., Krishnamurthy, A.: Parameter hub: a rack-scale parameter server for distributed deep neural network training. In: SOCC'18
17. Mitchell, C., Geng, Y., Li, J.: Using one-sided {RDMA} reads to build a fast, cpu-efficient key-value store. In: ATC'13
18. Mitchell, C., Geng, Y., Li, J.: Using one-sided RDMA reads to build a fast, cpu-efficient key-value store. In: ATC'13
19. Wei, J., Dai, W., Qiao, A., Ho, Q., Cui, H., Ganger, G.R., Gibbons, P.B., Gibson, G.A., Xing, E.P.: Managed communication and consistency for fast data-parallel iterative analytics. In: Proceedings of the Sixth ACM Symposium on Cloud Computing
20. Wei, X., Dong, Z., Chen, R., Chen, H.: Deconstructing rdma-enabled distributed transactions: Hybrid is better! In: OSDI'18