RDMA Optimization on ps-lite/MXNet

Mingfan Li Ke Wen Zhiqi Lin Xiaoniu Song Yongzhou Chen

Director: Prof. Hong An han@ustc.edu.cn

University of Science and Technology of China

{mingfan, wenke51, ralzq01, cow0712, cyz7}@mail.ustc.edu.cn

Abstract

MXNet is a present open-source deep learning framework used to build, train, and deploy deep neural networks. It's computation- and memory- efficient and is able to run on various heterogeneous systems, ranging from mobile devices to distributed GPU clusters. In a distributed MXNet system, different nodes perform communication through a parameter server framework, ps-lite. Nowadays, the InfiniBand interconnect network is widely configured on high performance clusters, which features higher throughput and lower latency than the traditional TCP/IP network. However, the official MXNet has not adopt this technology.

In this paper, we present a compact design of ps-lite using Remote Direct Memory Access (RDMA) over InfiniBand. Taking advantage of RDMA technology, we improve the performance of ps-lite so as to speed up the distributed MXNet.

As MXNet is still under development, all our work following discussed was conducted on the v0.10.0 of MXNet. The performance evaluation shows that for ps-lite, our RDMA-based design obtains over 16x (even 21x at peak) speed-up, based on which the MXNet gets nearly 9x performance improvement over the original system.

Key Words: RDMA, parameter server, MXNet, optimization

1 Introduction

Among many open-source deep learning frameworks, MXNet [1] stands out for its efficiency and convenience. It represents neural networks with a mixture of declarative symbolic expression and imperative tensor computation, and provides a flexible API for developers to construct and optimize their own networks.

Ps-lite [2] is designed and implemented as an efficient parameter server framework to boost the performance of large-scale distributed optimization and inference. In ps-lite, data transfer is performed by two operations, `push` and `pull`. The former sends data entries of the shared parameter from workers to servers, and the latter is called by workers for retrieving the latest parameters from servers.

The rest of this paper is organized as follows:

In Section 2, we introduce our analysis on ps-lite and MXNet as well as some views on RDMA. Section 3 presents our contribution to ps-lite and explains how we design and implement the RDMA-based MXNet in detail. Section 4 demonstrates the evaluation of our newly designed ps-lite and MXNet. Finally, in Section 5, we conclude our work and present future directions for RDMA-based MXNet.

2 Preliminary

2.1 ps-lite

This part briefly discusses our study on the source code of ps-lite.

Ps-lite adopts a "worker-server" model, in which both data and workload are distributed into worker nodes, while server nodes maintain globally shared parameters.

Instead of directly communicating with TCP socket, ps-lite utilizes the ZMQ [3] networking library for message transmitting. ZMQ is implemented based on the message queue structure and originally designed for "zero latency" (as possible). The further improvement on ZMQ makes it so powerful that it is even evaluated as "zero cost", "zero waste" and so on.

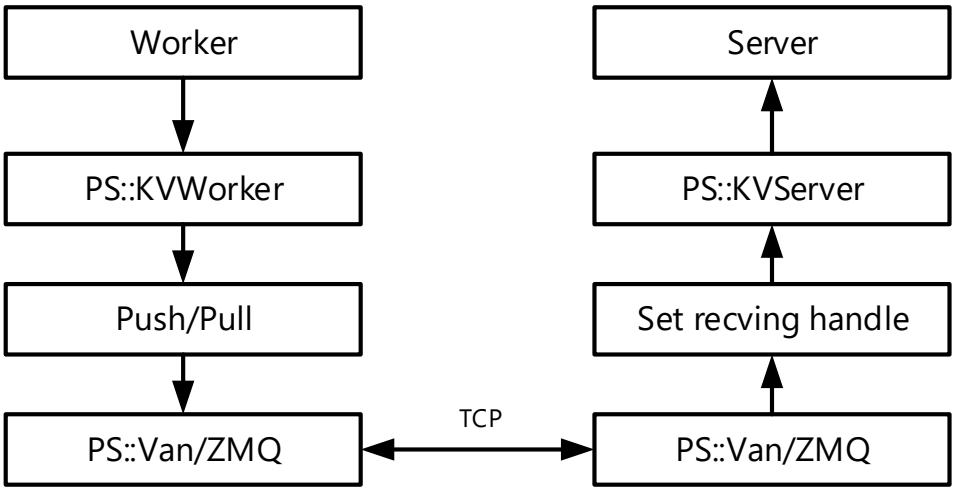


Figure 1: A brief hierarchical structure of ps-lite

In the hierarchy of ps-lite (see Figure 1), the server runs passively, waiting for worker's message; the worker sends a push request to the server to update the server's key-value pairs, or sends a pull request to the server to retrieve the server's values corresponding to some keys. Note that the server needs to be registered with a callable object (e.g. a callback function), which updates the parameters upon retrieval of workers' message.

The detailed push logic is not shown in Figure 1, and is demonstrated as follows:

- Once the worker calls the push method, the worker will first slice the key-value pairs into several segments according to the unique keys and server number. All of the segments will be distributed to the corresponding servers.
- The server will aggregate and update the local value with incoming data as the index of the global unique key as soon as it receives a push operation from any worker. After dealing with the data, the server will send an ACK message to notify the worker.
- The worker has a daemon thread which receives and confirms all the ACKs from relevant servers. In this way, the stability and accuracy of message transmitting is guaranteed.

When the worker sends a pull request, most work is similar to the push operation. The only difference is that a pull request contains only keys and no values. Thus, the server will respond with the updated values according to the keys extracted from the pull request.

In our studies mentioned above, the actual data transmission is performed by ZMQ library for both push and pull operations. Without worrying about concurrency performance, error handling and other common networking problems, the hierarchy of ps-lite is pretty clear.

Figure 2 shows the task scheduler for communication of ps-lite.

Deep into the source code of the `Van` module, the data transmission starts at `Send` operation, which is the interface exposed to upper-layer modules, such as `KVWorker`. This method is provided by the `kv_app` module, which is the basis of both `KVServer` and `KVWorker`. In fact, the `Van` module actually takes no care of TCP communication tasks, and such tasks are delegated to a module named `ZMQVan`, which performs the actual communication using ZMQ library. `ZMQVan` checks the meta of message and sends it to it's destination with APIs from ZMQ, such as `zmq_msg_send`.

From the perspective of receiving side, the workflow begins at the `ZMQVan` module where the program is blocked at `zmq_msg_recv,` in the state of waiting for remote Send operation. Once it returns with incoming data from the network, the receiving operation will assemble the data and set its ID information on the basis of front bytes. After previous steps, the raw data is parsed as a `Message` and is submitted to the upper `Van` module.

`Van` will then examine the `Control::cmd` field of the `Message`. This enumeration field can be one of `ADD_NODE`, `BARRIER`, `TERMINATE`, `EMPTY` and so on. Note that a `Message` with an `EMPTY` command is actually a data payload. Such message will be passed to the callable object we've mentioned above.

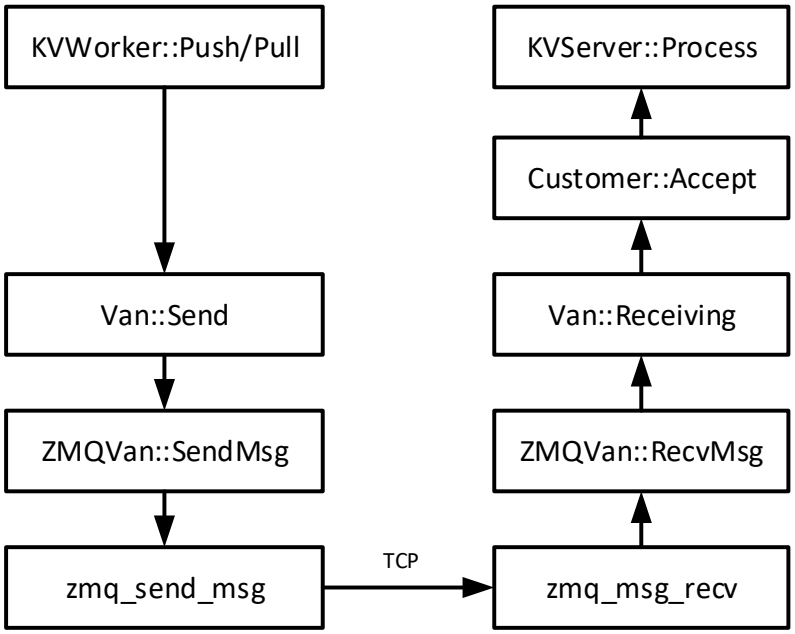


Figure 2: Workflow of ps-lite

2.2 KVStore of MXNet

In this part we briefly introduce KVStore, the communication system of MXNet [4] which is built upon ps-lite as is mentioned in Section 2.1.

With the scheduler role managing the whole cluster, MXNet consists of the other two main roles, the worker and the server. The server role is responsible for aggregating and bookkeeping (updating) the global parameters, and the worker role is responsible for retrieving and computing the parameters.

KVStore uses two kinds of messages, the control message and the data message. All control messages from workers and servers are delivered to and handled by the unique scheduler; while the data messages are passed between worker and server for actual data delivery, such as parameters' update. It's worth mention that communicating directly between workers is not available for MXNet. In other words, workers have to affect each other through the server, which is equal for servers.

In general, MXNet is commonly employed for realizing data parallelization. For that purpose, each device must be able to train the entire deep learning model, which means that any CPU or GPU devices should contain all operations of the computation graph.

KVStore is a distributed key-value store for data synchronization over multiple devices. It makes use of two primitive operations originating from ps-lite: push a key-value pair from a device to the store, and pull the value on a key from the store. In addition, a user-defined updater provides a way for merging the pushed value. The distributed model training starts at the registration of our weight updating function for the KVStore. After that, each worker repeatedly pulls the newest weight from the store and then pushes back the locally computed gradient.

The following example introduces a sample for distributed gradient descent by data parallelization.

```
1   while(1)
2   {
3       kv.pull(net.w);
4       net.foward_backward();
5       kv.push(net.g);
6   }
```

In terms of data communication, there are some differences between KVStore and original ps-lite. First, MXNet uses the engine to schedule the KVStore operations and manage the data consistency. With this strategy, it not only makes the data synchronization works seamless with computation, but also greatly simplifies the implementation. Second, the distributed MXNet cluster adopts a two-level structure. A level-1 server manages the data synchronization between the devices within a single machine, while a level-2 server manages inter-machine synchronization. With the two-level structure, outbound data can be aggregated, reducing bandwidth requirement. Figure 3 reveals the main communication for distributed MXNet.

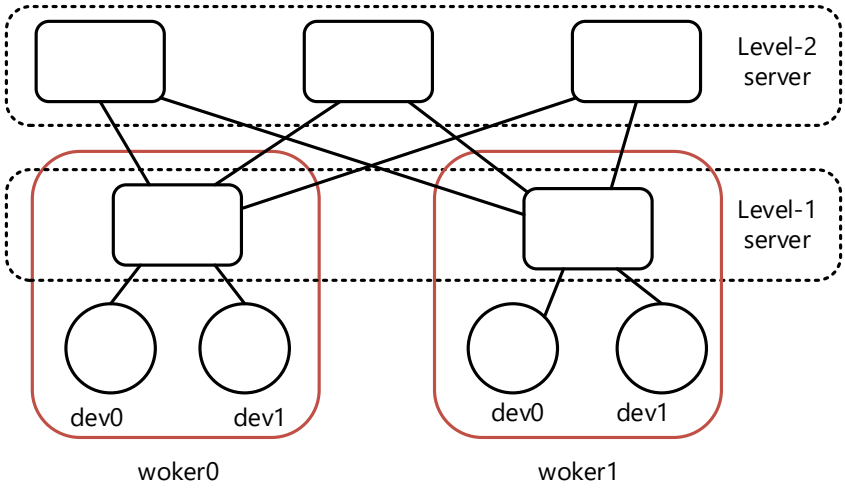


Figure 3: MXNet Data communication

2.3 InfiniBand and RDMA

InfiniBand [5] is an open standard architecture that defines a switched network fabric for interconnecting processing nodes and I/O nodes. In an InfiniBand network, I/O nodes and processing nodes are connected to the fabric by channel adapters. The InfiniBand uses a queue-based model. A queue pair (QP) consists of two queues: a send queue and a receive queue. The send queue holds instructions to transmit data, and the receive queue holds instructions that describe where received data is to be placed. Communication operations are described in work requests (WRs), and are submitted to the work queue. A completion queue (CQs) is used for reporting the completion of WRQs through a completion queue entry placed in the associated CQs. The entry can be checked by applications to confirm the state of work queue. Like traditional network, InfiniBand also supports different transport service. Our work discussed in this paper focus on the reliable connection (RC) services.

InfiniBand Architecture supports both channel and memory semantics. In the former semantic, send/receive operations are used for communication. In memory semantics, InfiniBand supports remote direct memory access (RDMA) operations, including RDMA write and RDMA read. In these operations, the sender starts RDMA by posting RDMA descriptors. A descriptor contains both the local data source addresses (multiple data segments can be specified at the source) and the remote data destination address. At the receiver side, the completion of an RDMA operation can be notified through CQs.

Compared with send/receive operations, RDMA operations are the better choice for several advantages [6]:

- RDMA operations are generally faster than send/receive operations because of the simpler design at the hardware level.
- RDMA operations do not involve managing and posting descriptors at the receiver side which would incur additional overheads and reduce the communication performance.

Therefore, we adopt RDMA operations in our work.

3 Contribution

3.1 ps-lite-rdma

Porting RDMA into ps-lite is the basis of our further works. To port RDMA, we mainly override `Van::Start`, `Van::Receiving` and `ZMQVan::SendMsg`. We came up with several models and build up several versions of ps-lite-rdma based on these models.

Recall that a worker can connect to a server (vice versa) but a worker (or server) cannot connect to another worker (or a server). The scheduler can communicate with all workers and servers. This is in fact the protocol of ps-lite. We proposed several communication models to adjust to this protocol using RDMA. Here is what we've done:

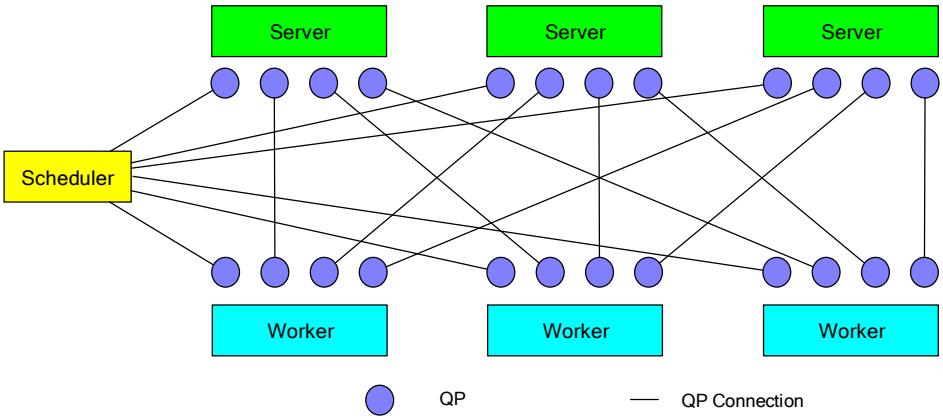


Figure 4: The basic model of ps-lite-rdma

Suppose there are n workers and m servers at run time. The basic model of ps-lite-rdma is shown in Figure 4. Every worker has m+2 QPs (m for communicating with servers, the other two for communicating with scheduler and itself), similarly every server has n+2 QPs, and the scheduler has n+m+1 QPs. Between two RDMA operations, we use RDMA write-based protocol to replace the original ZMQ transmission.

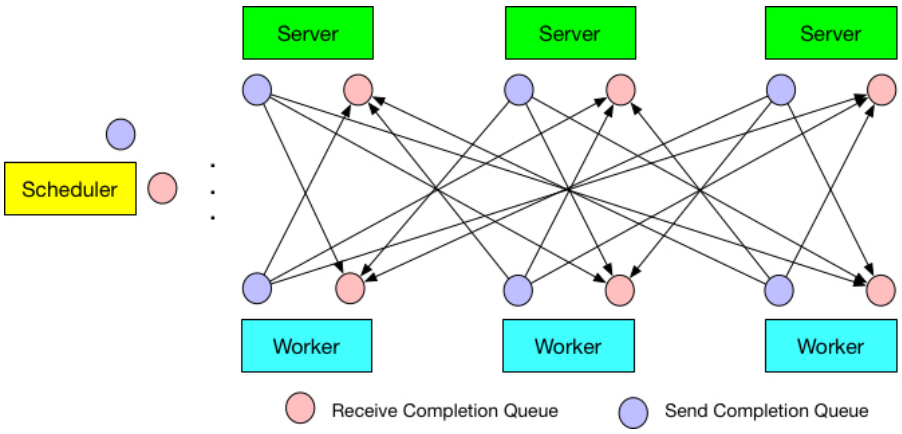


Figure 5: (Model1) The one-send-cq and one-receive-cq model

Model1: As is shown in Figure 5, in a single worker/server/scheduler, all QPs share the same Receive Completion Queue and the same Send Completion Queue. Each QP has its own registered receive buffer while all QPs share the same registered send buffer.

Model2: This model is the same as Model1 except that we add a new thread whose main task is to poll CQ and maintain the offset of receive buffer. Here, the receive buffer is maintained as a loop buffer.

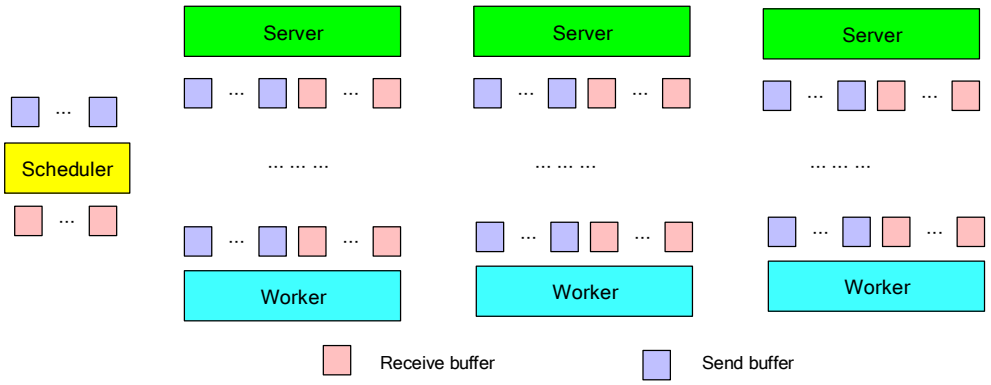


Figure 6: (Model3) The many-send-buffer and many-receive-buffer model

Model3: As is shown in Figure 6, similar to the Model2, all QPs share the same Receive CQ and Send CQ. What significantly differs from Model2 is that each QP has its own registered send buffer and receive buffer.

In our final evaluations, these models exhibit similar performance after further optimizations for ps-lite and MXNet. Recall that Model3 assigns a receive buffer and a send buffer for each QP, i.e. each receive buffer is dedicated for a single remote sender, which simplifies the synchronization on critical resources compared with using a shared buffer. Otherwise, we have to broadcast the offset of shared buffer to all senders to avoid dirty write. Thus we build our ps-lite-rdma based on Model3.

3.2 Dataflow in ps-lite-rdma

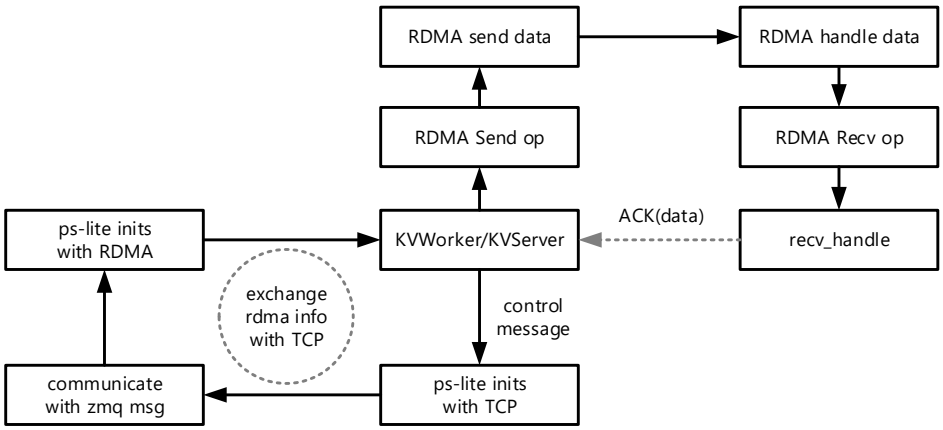


Figure 7: The dataflow of RDMA-based ps-lite

The whole dataflow of ps-lite-rdma is shown in Figure 7, the initialization of RDMA resources starts at `Van::Start` function. Before each QP connects with corresponding remote sides' QPs, RDMA uses ZMQ to exchange the required information. After RDMA connections have all been established, the ps-lite-rdma will send or receive messages via RDMA.

3.3 Optimization

For all the buffers we have mentioned above, we use the loop buffer technique to minimize the times and the amount of registering a buffer while using RDMA. Note that RDMA write-based protocol indicates that the write address is determined by the sender, which means the sender(receiver) should maintain the same offset after each operation.

We also apply other techniques (e.g. modify the `Van::PackMeta` function) to decrease the number of calls to `memcpy`. At last, `memcpy` is only called when copying `Message.data` into RDMA sending buffers before posting an RDMA send.

3.4 MXNet-rdma

The communication subsystem of MXNet totally depends on ps-lite, all we need to do for adapting our RDMA-based ps-lite for MXNet is just replacing the original ps-lite with ours. A useful tip for adaptation is that MXNet v0.10.0 employs certain version of ps-lite from github, we must work on the same version for consistency with MXNet. In fact, all MXNet uses the same version of ps-lite, which makes our RDMA-based ps-lite fit for any version MXNet (We have adapted it well for the latest version, v0.11.0).

4 Evaluation

In this section, we display the improvement of our RDMA-based ps-lite and MXNet. Compared with the TCP and IPoIB, our implementation has better performance and scalability for both ps-lite and MXNet. According to our result, the RDMA-based software stands out especially for high communication pressure, which meets our demand for large scale clusters.

The platform configuration we use for evaluation is: 9 Sugon nodes with dual Intel Xeon E5-2660 processors; 128GB RAM for each node; the nodes are connected with Both 1000M Ethernet and Mellanox 40G QDR InfiniBand.

We measure the total execution time of same tasks on 3 different versions (TCP/IP, IPoIB and RDMA), and we define the speedup ratio of Y over X to be the following equation:

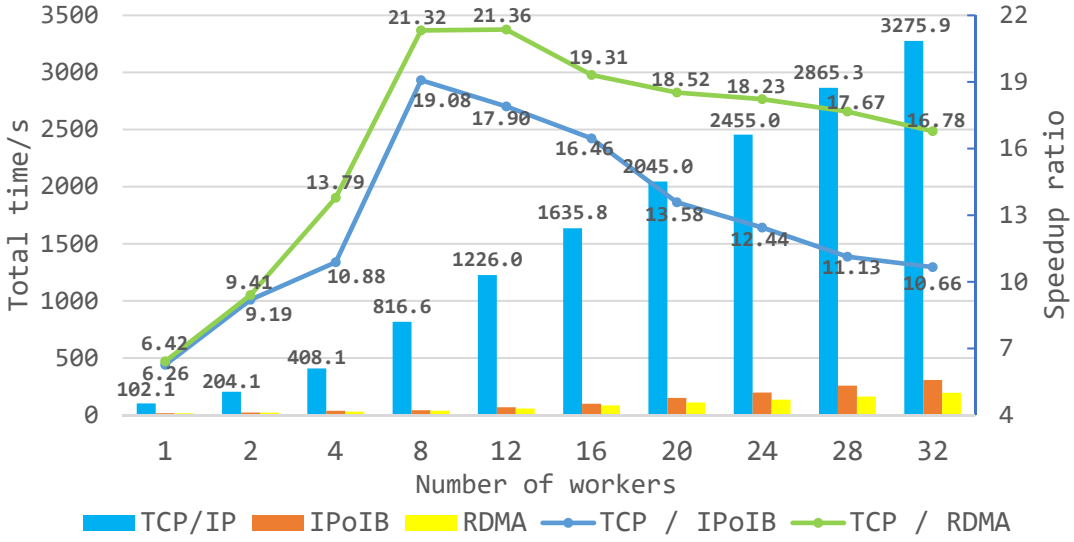
$$speedup(Y, X) = \frac{T(X)}{T(Y)}$$

where $X \in \{\text{TCP}, \text{IPoIB}\}$ and $Y \in \{\text{IPoIB}, \text{RDMA}\}$.

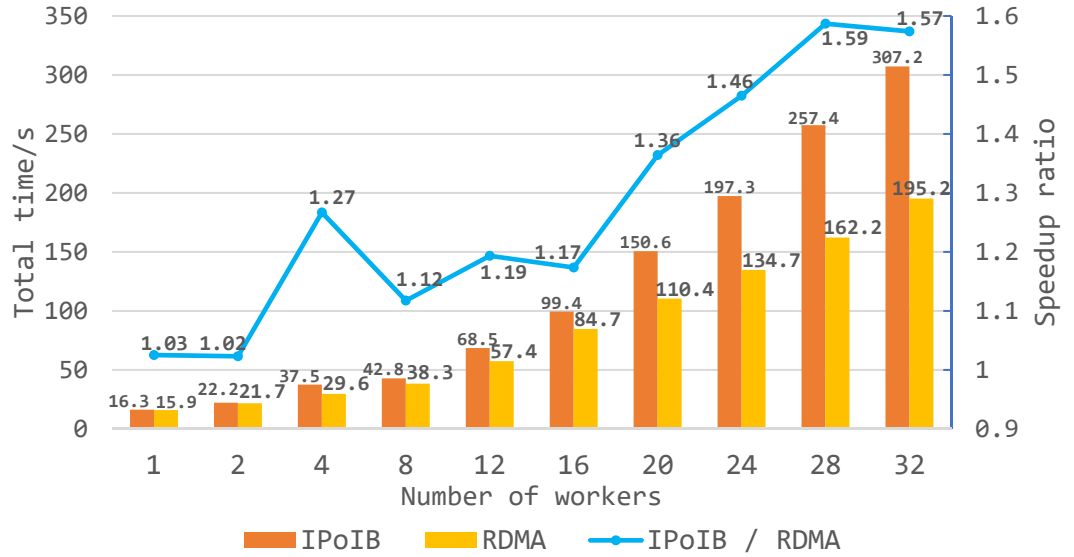
4.1 Evaluation of ps-lite

The ps-lite provides two fundamental services for distributed system, `simple_app` and `kv_app`. `Simple_app` is used for request and response between two objects, while `kv_app` completes the key-value pairs transfer. Since the effect of ps-lite mainly relies on the data transfer, we decide to benchmark our work with the testcase `test_kv_app`.

By default, `test_kv_app` sends a 10000-length KV pairs for 50 times, whose running time is less than 1s. To increase the strength of communication, we send 100000-length KV pairs for 10000 times in `test_kv_app`.



(a) The performance against TCP and IPoIB



(b) The performance against IPoIB

Figure 8: The performance of ps-lite-rdma on test_kv_app

We conclude our experiment (see Figure 8) in two aspects according to the traffic load:

- For testcases with relatively low traffic load (using ≤ 8 workers), both RDMA and IPoIB achieves significant improvement over TCP. Yet compared with IPoIB version, the improvement is small. For the testcase using 8 workers, $speedup(RDMA, TCP) = 21.32$, which mainly benefits from the InfiniBand hardware. However, $speedup(RDMA, IPoIB) = 1.12$. This is because ≤ 8 workers just don't make full use of CPU, in which case the CPU-bypass feature of RDMA is not shown.

- For testcases with relatively high traffic load (using > 8 workers), RDMA and IPoIB still beat TCP on performance. Moreover, as the scale goes large, RDMA performs significantly better than IPoIB. For the testcase using 32 workers, $speedup(RDMA, IPoIB) = 1.57$, and we believe that this ratio can get higher as the scale goes larger. In this case, the CPU-bypass feature of RDMA is fully exploited, which contributes to the scalability of our design.

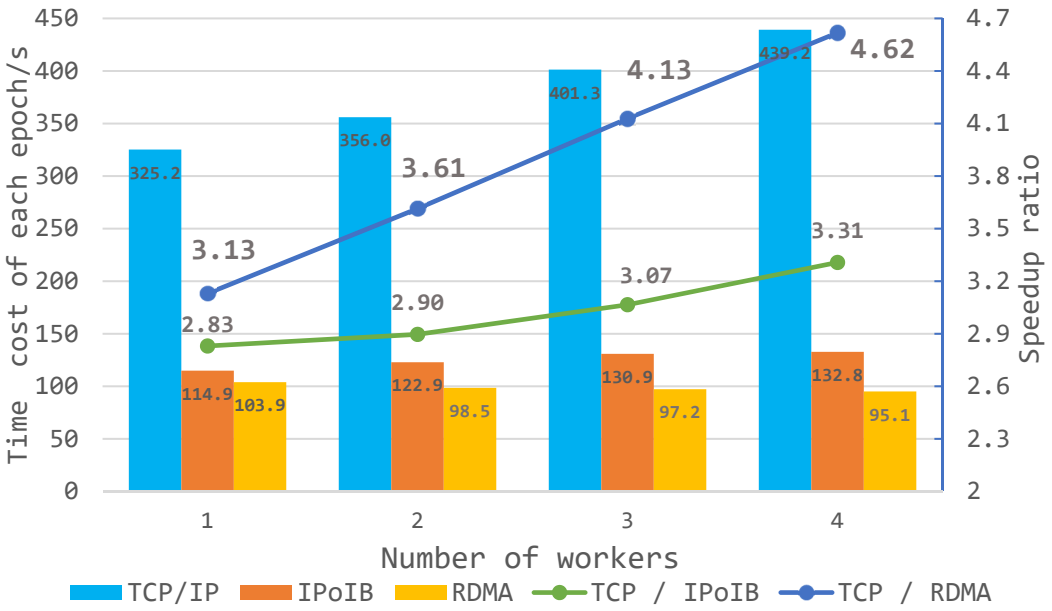
With our optimizations on ps-lite, it completes transmission with low latency and little requirements for CPU resources. Therefore, our rdma-based design makes clear progress compared with both TCP/IP and IPoIB.

4.2 Evaluation for MXNet

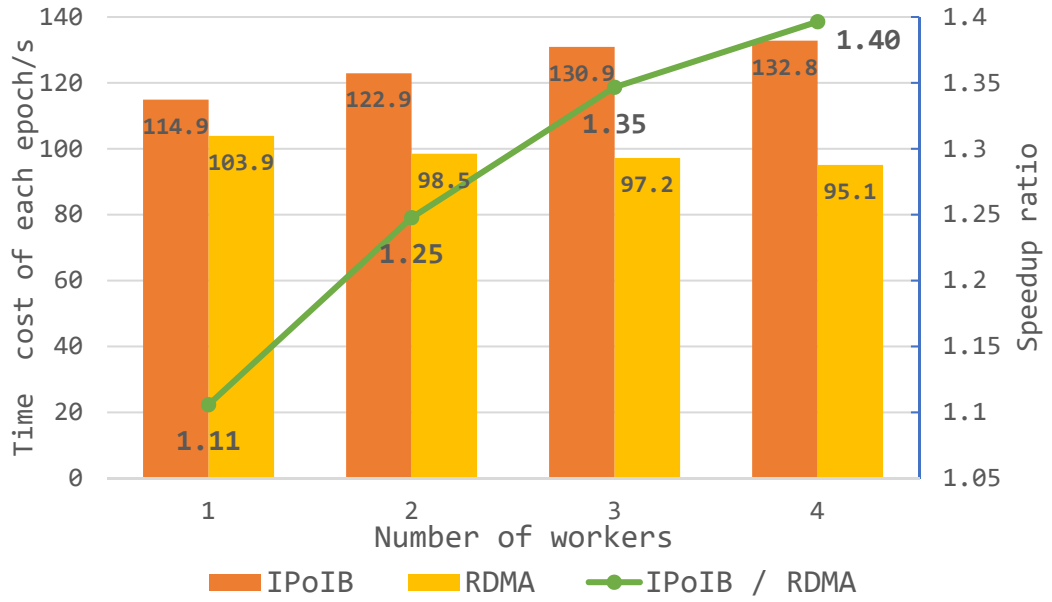
We choose to test our RDMA-based MXNet against image classification tasks with two data sets: mnist [7] and cifar10 [8].

On our early trials, the performance shows no clear improvement for either IPoIB or RDMA version. We finally blamed it on the small size of network parameters, which leads to little network payload. To emulate the heavy communication among large scale clusters, we modify the network in MXNet by increasing the number of hidden neurons for network to 2048 and 1024 (the original network just has two hidden fully connected layers with size 128 and 64).

We run tests on mnist (see Figure 9) and cifar10 (see Figure 10) data sets with our modified network. From the guide of official MXNet document, the default number of workers should be equal to the servers, so as to gain better performance for training the network. We follow this guidance in our evaluation of MXNet and, e.g. when the number of workers and servers come to 4, we used 8 nodes for the evaluation: 4 nodes are for workers and the rest are for servers.



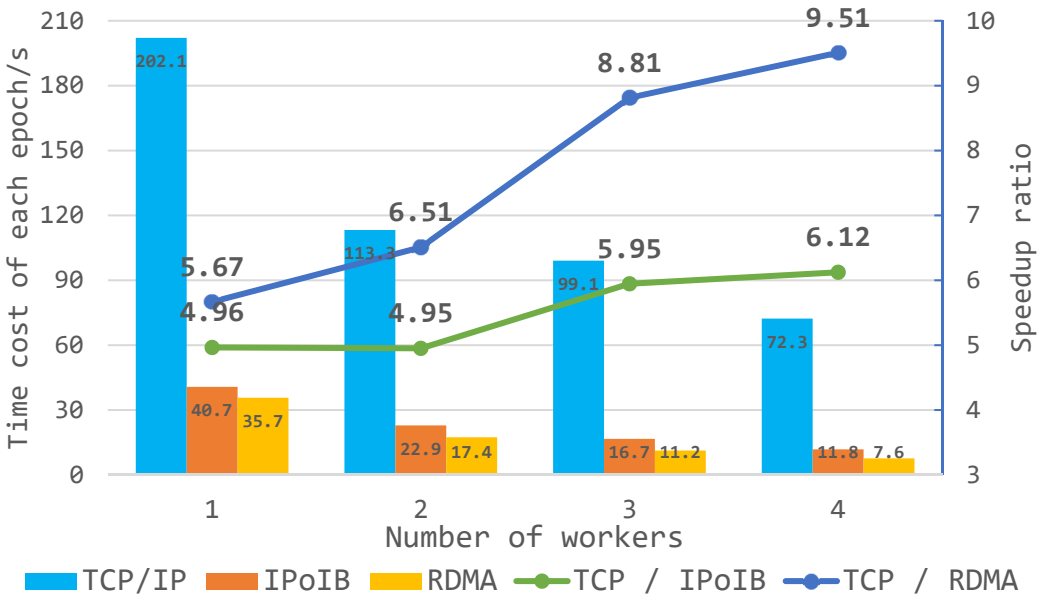
(a) The performance against TCP and IPoIB



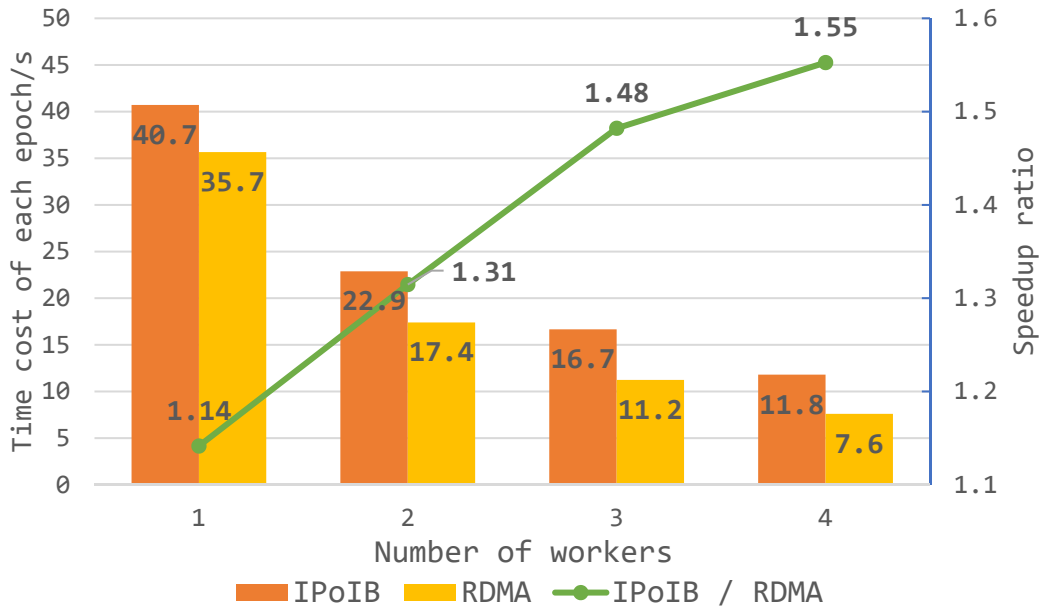
(b) The performance against IPoIB

Figure 9: The performance of mnist on ps-lite-rdma

Figure 9 shows the evaluation on mnist with RDMA-based MXNet. From TCP/IP to IPoIB, time reduction rate is about 72%. As for IPoIB and RDMA, it acts like ps-lite, too. As the communication grows, the improvement of our RDMA-based design is 10% when 1 worker and 28% when 4 workers.



(a) The performance against TCP and IPoIB



(b) The performance against IPoIB

Figure 10: The performance of cifar10 on ps-lite-rdma

Figure 10 shows the results of our evaluation on cifar10 with RDMA MXNet. From TCP/IP to IPoIB, time reduction is about 87%. As the number of worker grows, the improvement enhances: 12% when 1 worker and 35% when 4 workers.

It's clear that compared with TCP version, the performance differs between mnist and cifar10. For mnist, we just accelerate it for nearly 5 times speed-up, but for cifar10 it is more than 9 times. Similarly, compared with IPoIB version, mnist gains 28% speedup,

which is less than 35% of cifar10. According to our view, the difference lies in the proportion of traffic. The larger amount of communication, the better performance for our RDMA-based MXNet. That's what we expected for RDMA technology with InfiniBand.

5 Conclusion and Future work

In this paper, we have proposed a new design of ps-lite over InfiniBand which not only benefits ps-lite with RDMA technology, but also indirectly speeds up MXNet. In our final evaluation, the MXNet equipped with ps-lite-rdma shows a good performance over the original TCP/IP system, as well as good scalability with the increasing traffic of communication.

There are several directions for our future work. First, we would like to expand our current work by evaluating it with larger test platforms and more applications, such as text classification, speech recognition and so on. Another direction is that we attempt further optimizations with other advanced techniques, such as using shared receive queue structure provided by the IB verbs. With more techniques, we believe that we'll obtain better performance for RDMA communication.

6 Acknowledgement

Thanks to the HPC Advisory Council for giving us this chance to explore and exercise the RDMA technology.

Also thanks to the blogger of rdmamojo.com, Dotan Barak, who makes RDMA technology friendly to developers. Many of our issues have been solved thanks to his blog.

This work is supported by the Advanced Computer System Architecture Lab of University of Science and Technology of China.

References

[1] Chen T, Li M, Li Y, et al. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems[J]. arXiv preprint arXiv:1512.01274, 2015.

[2] Li M, Zhou L, Yang Z, et al. Parameter server for distributed machine learning[C]//Big Learning NIPS Workshop. 2013, 6: 2.

[3] "The ØMQ (ZeroMQ) Framework, Multithreading Magic", http://zeromq.org

[4] Li M, Andersen D G, Park J W, et al. Scaling Distributed Machine Learning with the Parameter Server[C]//OSDI. 2014, 1(10.4): 3.

[5] Pfister G F. An introduction to the infiniband architecture[J]. High Performance Mass Storage and Parallel I/O, 2001, 42: 617-632.

[6] InfiniBand whitepapers, http://www.mellanox.com/page/white_papers

[7] LeCun Y, Cortes C, Burges C J C. MNIST handwritten digit database[J]. AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist, 2010, 2.

[8] Krizhevsky A, Nair V, Hinton G. The CIFAR-10 dataset[J]. online: http://www. cs. toronto. edu/kriz/cifar. html, 2014.