Towards Zero Copy Dataflows using RDMA

Bairen Yi
HKUST
byi@connect.ust.hk

Jiacheng Xia
HKUST
jxiaab@connect.ust.hk

Li Chen
HKUST
lichenad@connect.ust.hk

Kai Chen
HKUST
kaichen@cse.ust.hk

ABSTRACT

Remote Direct Memory Access (RDMA) offers ultra-low latency and CPU bypass networking to application programmers. Existing applications are often designed around socket based software stack that manages application buffers separately from networking buffers and do memory copies between them when sending/receiving data. With large sized (up to hundreds MB) application buffers, the cost of such copies adds non trivial overhead to the end-to-end communication pipeline. In this work, we made an attempt to design a zero copy transport for distribute dataflow frameworks that unifies application and networking buffer management and completely eliminates unnecessary memory copies. Our prototype on top of TensorFlow shows 2.43x performance improvement over gRPC based transport and 1.21x performance improvement over an alternative RDMA transport with private buffers and memory copies.

CCS CONCEPTS

• **Networks** → **Application layer protocols**; • **Computer systems organization** → *Distributed architectures*; • **Information systems** → *Record and buffer management*; Data exchange;

KEYWORDS

Kernel bypass networking, Memory management

ACM Reference Format:
Bairen Yi, Jiacheng Xia, Li Chen, and Kai Chen. 2017. Towards Zero Copy Dataflows using RDMA. In *Proceedings of SIGCOMM Posters and Demos '17, Los Angeles, CA, USA, August 22–24, 2017,* 3 pages.
https://doi.org/10.1145/3123878.3131975

1 INTRODUCTION

High speed network fabrics built with commodity Ethernet switches and network interface cards (NICs) is the key component of data centers. Recently, RDMA over Converged Ethernet (RoCE) [2] has received wide attention to achieve high throughput, low latency inter-connect for servers in 40/100 GbE data center network fabrics. While application traffic is moved swiftly from/to the NIC and within the network, real-world applications are often designed bearing the in-kernel TCP/IP stack and the burden of software infrastructures built on top of the decades-old Berkeley sockets API, for example, Remote Procedure Call (RPC) or Actor architecture. To achieve good software compatibility, existing application is often migrated to RDMA using protocols (e.g. rsocket or SDP) that are designed to emulate socket API, bringing back the copy overhead from kernel space to user space, resulting in unnecessarily high messaging latency.

Dataflow is a popular architectural pattern among data analytics frameworks such as Spark, Naiad, and TensorFlow. A dataflow is a directed acyclic graph such that each node represents a predefined operation that has a number of inputs and outputs. For distributed dataflow, usually the graph is partitioned among all the workers such that each edge being split represents a stage of communication, where the output of last stage is transferred across the communication channel to feed the input of next stage. We observed that, typically those output and input are large and immutable data buffers, and in current application design, they are usually copied multiple times even in user space, not to mention the copying overhead in kernel TCP/IP stack. Such data buffers could consist of tens or even hundreds of megabytes in a single unit. Copying large data buffers adds non-trivial overhead to the pipeline of end-to-end cross-node dataflow data transfer, in which context RDMA zero-copy and kernel bypass are rendered less meaningful.

In this work, we show that the it is possible to achieve end-to-end zero-copy for transport in dataflow applications using a customized memory allocator that collects buffer registration information from the computation graph. We have demonstrated that our integrated design on top of TensorFlow, which completely eliminates unnecessary memory copies. Our implementation achieves 2.43x end-to-end speedup over its TCP/IP based gRPC transport, and 1.21x speedup over an alternative RDMA transport that does memory copying.

2 DESIGN

Existing dataflow applications uses either RPC or file block transfers for their communication pipeline. For example, both open sourced version of TensorFlow and early version of Apache Spark use RPC for cross-node data transfer; later version of Spark and Hadoop MapReduce stores the output to local file system and shuffles the file blocks to downstream. RPC system is a poor fit for transferring large blocks of data; the RPC request and response message will go through not only copying but also encoding and decoding, which severely congests CPU and hurts the throughput of end-to-end data transfer pipeline. File blocks are often stored first into an external file system, and it also requires several rounds of copies even for memory based file system (`ramfs`).

SIGCOMM Posters and Demos '17, August 22–24, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5057-0/17/08.
https://doi.org/10.1145/3123878.3131975

A key observation is that moving buffers around the memory space in applications shares the same design rationale of the socket API: each of the computation and communication subsystems manages its own memory without sharing. This poses a fundamental difficulty to adapt this design with end-to-end zero copy; no matter how fine grained buffer management is done in each subsystem, once the data transfer pipeline goes through both of them, at least one memory copy is required. For example, the TCP/IP stack in Linux kernel uses the complicated data structure `sk_buff` for packet abstraction, and careful design has been carried out such that moving `sk_buff` only requires moving a linked list pointer but not the payload. However, it always requires at least one copy to move data from user space to kernel space. Similar design choice has been made for various RPC and messaging libraries.

For dataflow applications deployed in high speed DC fabric, there are two conditions that make copying memory the bottleneck: (1) application buffers are too large to fit it L1/L2/L3 cache; (2) a single application is unlikely to congest the network. Note that moving buffers around is not necessarily the bottleneck of end-to-end data transfer pipeline if either condition fails to hold. For the first condition, the distinction of copying a large buffer and copying many small buffers is that the latter usually reside in cache after being dynamically computed, while the former will cause cache-misses and eventually fetched from memory. In some RDMA aware applications like FaSST [3], small buffers are often copied from or to page-locked memory that is managed solely by the communication subsystem, to avoid the overhead of dynamic pinning of virtual memory pages to physical memory. For the second condition, latency sensitive applications such as KV store usually do not fully utilize network bandwidth, i.e. they are bound by number of concurrent request, while most keys and values are of small sizes (tens of bytes to several kilobytes). The throughput for copying small sized buffers could be 10 times (20-30 GB/ with buffer sizes around 4KB) more than copying large buffers of tens to hundreds megabytes (2-4 GB/s with buffer sizes larger than 4MB) that cannot fit into the cache. It is thus essential to break the wall of memory sharing between computation and communication subsystem, and incorporate an integrated zero-copy design such that the computation tasks read/write to communication buffers directly.

Our solution to this problem is a unified memory allocator that manage computation and communication buffers in the same system. The allocator has two parts: (1) implementations of different memory allocation specification, e.g. whether the allocated buffer should be pinned for RDMA or device DMA; (2) information collector that parse the dataflow graphs and determine the specification of buffer management for each step, based on the source and sink of an edge in dataflow graph. The buffer allocated with RDMA or device DMA specification will be shared between computation and communication subsystems, and the buffer is freed when it is no longer being used by both of two subsystems.

3 IMPLEMENTATION

We have implemented[1] zero copy dataflow on top of TensorFlow [1], as it allows drop-in replacement of its memory allocation policy using a customized memory allocator. In our memory allocator, we collect information from TensorFlow's computational graph and distributed graph partition, so only tensors that is either source or sink of a cross-node send/receive operation will be registered as RDMA capable buffer. We intercept memory registration and deregistration logic, right after memory allocation and before memory free. In its open sourced version, TensorFlow uses HTTP/2 based gRPC for its tensor transport, and we have modified it to bypass the RPC system and transfer tensors using out-of-band RDMA transport instead. To support transfers with GPU as either source or target, we manage to use GPU direct RDMA whenever the PCI-e topology permits, i.e. NIC could access GPU memory using the same PCI-e host bridge. We have observed bad performance when GPU direct RDMA occurs on path that traverses a CPU-socket level link, i.e. different NUMA nodes, so those tensors are transferred to the main memory before they are sent to NIC or GPU. RDMA capable tensor buffers are also allocated in the same NUMA node as of NIC. To simplify our implementation, we only uses one-way RDMA read with invalidate, as for minimal sized buffers it only requires single round-trip to complete the tensor transfer.

[1]Special thanks to Paul Tucker in Google for helping us to merge this patch into the TensorFlow official repository; see https://github.com/tensorflow/tensorflow/pull/11392

4 EVALUATION

We evaluate our implementation on an testbed consists 4 servers that connect to a Mellanox MSN2100-BB2F 40GbE RoCE Switch. Each server is installed with a Mellanox MT27500 40GbE NIC, dual 6-core Intel Xeon E5-2603v4 CPU, 4 NVidia Tesla K40m GPUs, and 256 GB DDR4-2400MHz memory. The switch is configured with priority-base flow control (PFC) for a lossless fabric.

We trained a distributed version of the VGG16 [4] convolutional neural network model, provided in Google's benchmark suite. The total size of the model parameters is 528 MB. The model is trained in synchronous mode as opposed to asynchronous mode, so each worker will compute the exact same number of iterations to avoid random fluctuations. We use the same number of parameter servers as of workers, i.e. one per node. Workers use both GPU and CPU for computation, while parameter servers only use CPU with the host memory to aggregate model parameters.

The result shows that our implementation of RDMA based zero copy dataflow could achieve overall 2.43x performance improvement in job completion time (measured by training throughput in number of images) over the original gRPC (TensorFlow v1.2.1) based tensor transport, and 1.21x performance improvement over Yahoo's RDMA implementation [5] using communication-private buffers with memory copying. With respect to computation scaling, it scales to 16 GPUs in total with 13.8x scale factor compared to single GPU. As to our RDMA capable memory allocator, the fraction of memory registration overhead in the whole tensor transfer pipeline on average is only 2.7%, and 82.6% of RDMA transfers do not incur extra memory registration overhead. No memory copies within host memory are performed in the pipeline.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA.*

[2] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over commodity ethernet at scale. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 202–215.
[3] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2016. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs.. In *OSDI*. 185–201.
[4] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[5] Lee Yang, Jun Shi, Bobbie Chern, and Andy Feng. 2017. TensorFlow on Spark. https://github.com/yahoo/TensorFlowOnSpark. (2017).