The Fourth Student RDMA Programming Competition

TensorFlow RDMA test document

Unique Studio from Huazhong University of Science and Technology

Team members:

Li Kaixi

He Yifei

Guo Yingzhong

Liu Zhongze

Li Kaijie

October, 2016 Wuhan

Contents

Introduction of RDMA

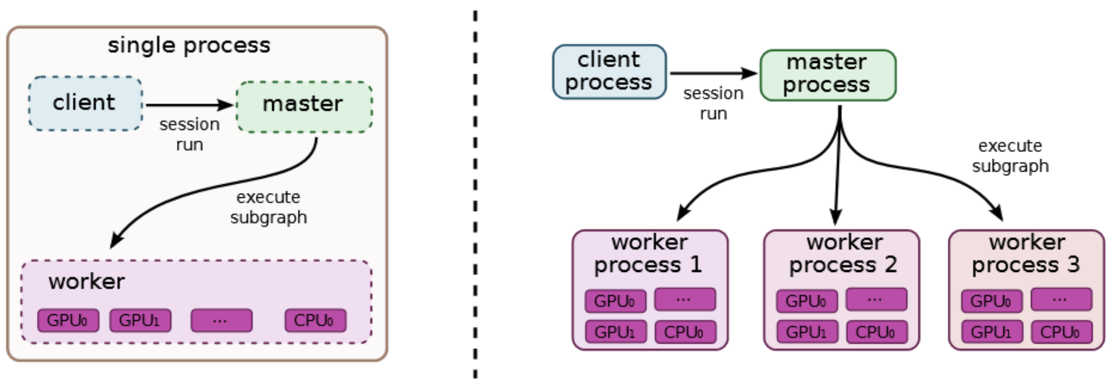
In computing, remote direct memory access (RDMA) is a direct memory access from the memory of one computer into that of another without involving either one's operating system. This permits high-throughput, low-latency networking, which is especially useful in massively parallel computer clusters.

RDMA supports zero-copy networking by enabling the network adapter to transfer data directly to or from application memory, eliminating the need to copy data between application memory and the data buffers in the operating system. Such transfers require no work to be done by CPUs, caches, or context switches, and transfers continue in parallel with other system operations. When an application performs an RDMA Read or Write request, the application data is delivered directly to the network, reducing latency and enabling fast message transfer.

Introduction of TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

TensorFlow have both local and distributed implementations of the TensorFlow interface. These two different modes are illustrated in above figure.



Single machine and distributed system structure

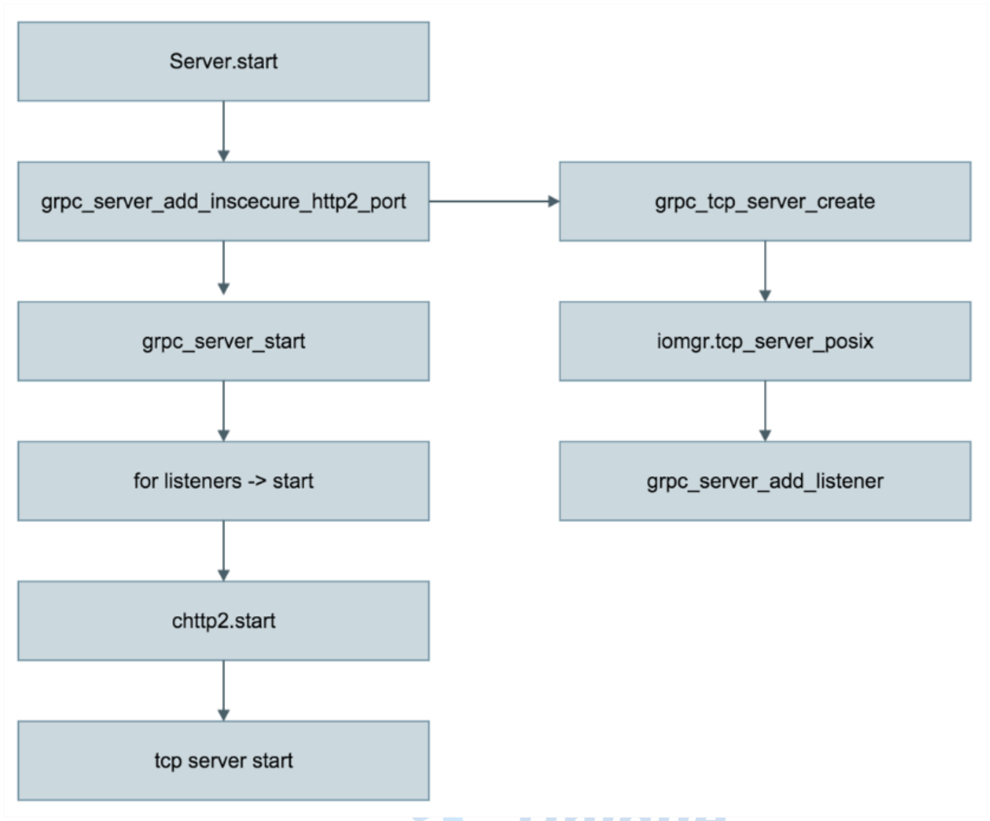
The local implementation is used when the client, the master, and the worker all run on a single machine in the context of a single operating system process (possibly with multiple devices, if for example, the machine has many GPU cards installed). The distributed implementation distributed use gRPC for inter-process communication, to share most of the code with the local implementation, but extends it with support for an environment where the client, the master, and the workers can all be in different processes on different machines. In our distributed environment, these different tasks are containers in jobs managed by a cluster scheduling system.

Introduction and analysis of communication model gRPC

gRPC is A high performance, open-source universal RPC framework. Remote Procedure Calls (RPCs) provide a useful abstraction for building distributed applications and services. It is the main component for TensorFlow distributed runtime via C++ interface. The libraries in this repository provide a concrete implementation of the gRPC protocol, layered over HTTP/2 by TCP/IP.

gRPC is based around the idea of defining a service, specifying the methods that can be called remotely with their parameters and return types. On the server side, the server implements this interface and runs a gRPC server to handle client calls. On the client side, the client has a stub (referred to as just a client in some languages) that provides the same methods as the server.

In order to port gRPC from TCP/IP to RDMA, by reading the document and analysis gRPC code, we get the call process of gRPC server start:



gRPC server start process

Call process of gRPC client start is similar to server, but the final call to gprc_client_connect in iomgr. tcp_client_posix to connect with gRPC server and return to chttp2. The most important parts in gRPC communication is grpc_client_posix and grpc_server_posix in iomgr module.

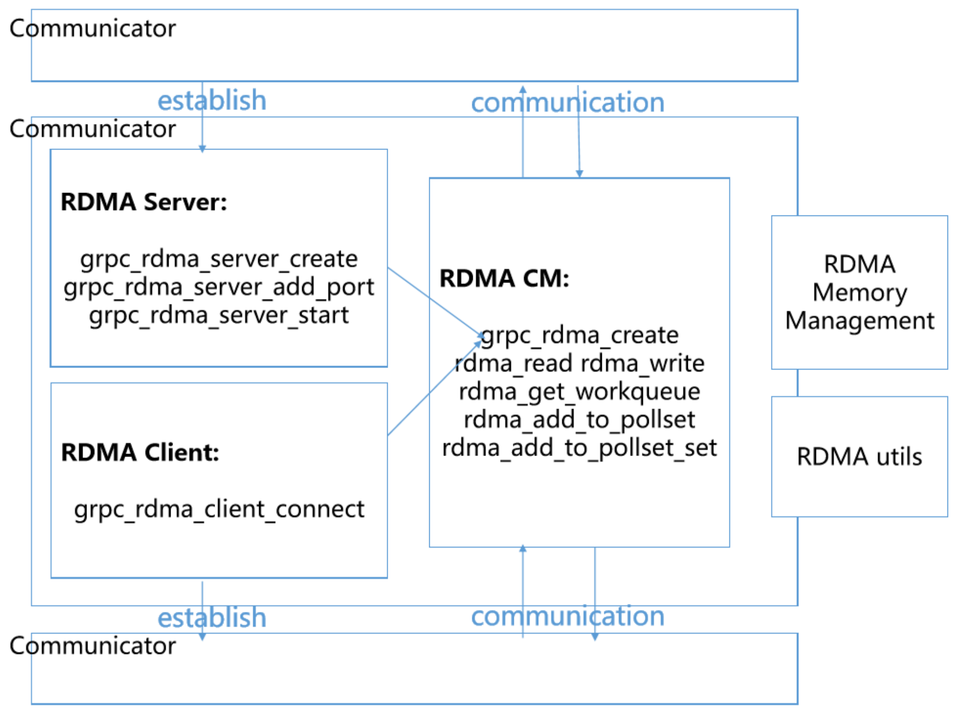
So we decide to transplant Socket/TCP parts of iomgr module directly and make less changes in other files for stability, convenient and compatibility.

How we port gRPC from TCP/IP to RDMA

The following table shows file modifications:

	Filename	Describtion
Changed	src/core/lib/security/transport/security_connector.c	replace iomgr function call to RDMA version
	src/core/ext/transport/chttp2/server/secure/server_secure_chttp2.c	
	src/core/ext/transport/chttp2/server/insecure/server_chttp2.c	
	src/core/lib/security/transport/security_connector.h	
	src/core/ext/transport/chttp2/client/insecure/channel_create.c	
	src/core/ext/transport/chttp2/client/secure/secure_channel_create.c	
	src/core/lib/http/httpcli.c	
Added	src/core/lib/iomgr/rdma_client.h	rdma version of tcp_client_posix
	src/core/lib/iomgr/rdma_client_posix.c	
	src/core/lib/iomgr/rdma_cm.c	RDMA version of tcp_posix, responsible for the communication
	src/core/lib/iomgr/rdma_cm.h	
	src/core/lib/iomgr/rdma_memory_management.c	RDMA send/receive memery management
	src/core/lib/iomgr/rdma_memory_management.h	
	src/core/lib/iomgr/rdma_server.h	RDMA version of tcp_server_posix
	src/core/lib/iomgr/rdma_server_posix.c	
	src/core/lib/iomgr/rdma_utils_posix.c	common Functions and Variables
	src/core/lib/iomgr/rdma_utils_posix.h	

The following diagram shows the function relationship:



The

The Tactics of Improving Performance

* Consistent with the gRPC original framework, the establishment and transceiver of communication is separated.
* Using RDMA send and receive with asynchronous non-blocking.
* Using grpc poll framework. As far as possible to use Kernel bypass, reducing kernel calls.
* High performance design with native InfiniBand, RoCE support at the verbs-level.
* Data package send memory zero copy.
* Data package receive memory optimization.
* RDMA layers flow control.

Applications version, configuration and running test environment.

Applications version

* **Applications RDMA version changed from**

  Tensorflow – r0.10
  Grpc – r0.14

Applications configuration

* **Applications Requirement**

  1. Bazel
  2. Python 2.7+ / 3.5+
  3. RDMA / InfiniBand or RoCE Support
  4. Cuda-7.5 / Cudnn v4 (Opitional for GPU version)

* **Install from compiled package**

```
#system require Centos 7 libc-2.1.7
$ pip install --upgrade build/tensorflow-0.10-py2-none-any.whl
```

* **Install from sources**

```
$ cd src/TensorFlow_rdma

#setting according to system configuration
$ ./configure

#build without GPU support
#$ bazel build -c opt //tensorflow/tools/pip_package:build_pip_package

# build with GPU support:
$ bazel build -c opt --config=cuda //tensorflow/tools/pip_package:build_pip_package
$ bazel-bin/tensorflow/tools/pip_package/build_pip_package /tmp/tensorflow_pkg

$ pip install --upgrade /tmp/tensorflow_pkg/tensorflow-0.10-py2-none-any.whl
```

* Run tensorflow distributed test

```
$ cd tensorflow/tensorflow/tools/dist_test/benmark_test
```

PS node
```
$ CUDA_VISIBLE_DEVICES=""    python train.py --num_workers=2 --ps_hosts=PS_node:2222 --worker_hosts=worker1_node:2223,worker2_node:2224 --job_name=ps --task_index=0
```

Worker1 node
```
$ CUDA_VISIBLE_DEVICES=""    python train.py --num_workers=2 --ps_hosts=PS_node:2222 --worker_hosts=worker1_node:2223,worker2_node:2224 --job_name=worker --task_index=0
```

Worker2 node
```
$ CUDA_VISIBLE_DEVICES=""    python train.py --num_workers=2 --ps_hosts=PS_node:2222 --worker_hosts=worker1_node:2223,worker2_node:2224 --job_name=worker --task_index=1
```

Comparison between TCP/IP and RDMA

Applications testing environment

SYSTEM:	Centos 7 3.10.0-327.22.2.el7.x86_64 libc-2.1.7
CPU:	Intel(R) Xeon(R) CPU E5-2670
GPU:	NVIDIA Corporation GK110GL [Tesla K20m]
HCA:	Mellanox Technologies MT27500 Family [ConnectX-3]
MEM:	32GB
LAN:	Gigabit Lan Ethernet

Testing Result and comparing result

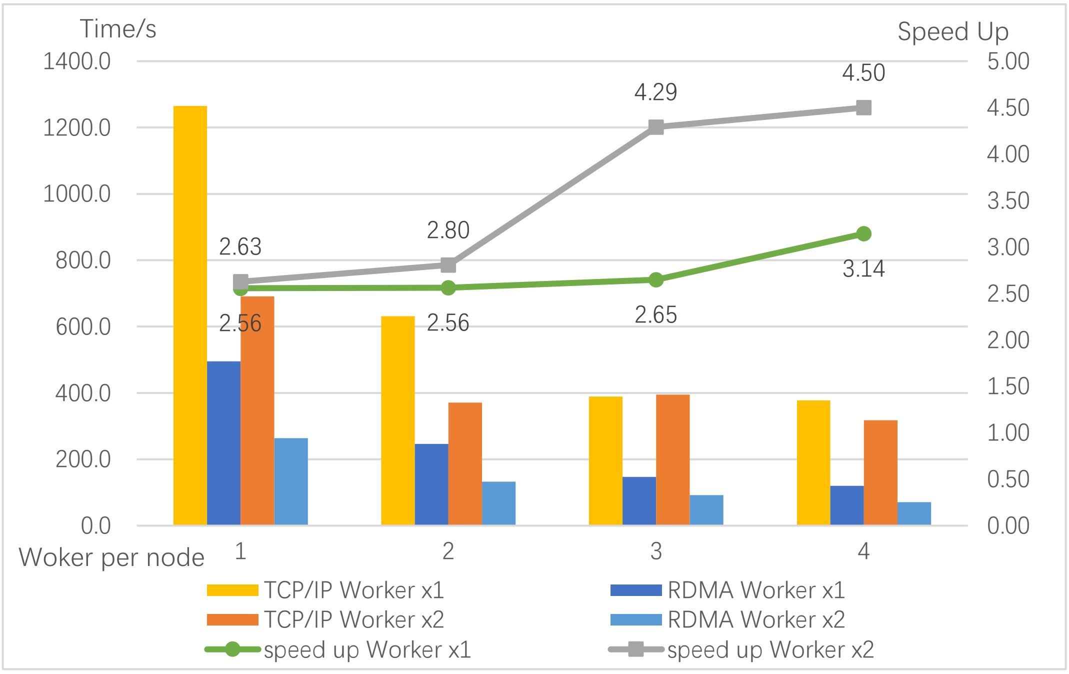
In order to compare the performance of the code between RDMA with TCP/IP, we write a testing program based on TensorFlow mnist as benchmark test. We run ps process on 1 node which is used as master node without GPU and run worker process with each worker process bind to a GPU accelerator on other 2 nodes.

We run benchmark test though different nodes and worker number in GPU environment to assess TensorFlow RDMA's performance and compare with the TCP/IP version, and here is the result*:

	worker per node	1	2	3	4
RDMA	nodes x 1	495.0	246.5	146.9	12.0
	nodes x 2	263.3	132.4	92.1	70.6
TCP/IP	nodes x 1	1265.2	631.6	389.0	377.1
	nodes x 2	691.4	371.3	395.2	317.7

Time consuming tables

*In Tensorflow RDMA version, workers has different time consumption and we use maximum value.



Time consuming and speedup

* Testing Result analysis

Compared with the TCP/IP version, Tensorflow RDMA version achieved a speedup over 2.5x in test Benchmark. With increase of worker numbers, Tensorflow RDMA version achieved more than 4 times speed up.

Compared to TCP/IP, Tensorflow RDMA Distributed’s acceleration has good linearity which can increase parallel scale and have more application scenarios. The improvement of the distributed runtime speed also enhances the performance of TensorFlow.

Appendix

references

http://www.rdmamojo.com/

https://thegeekinthecorner.wordpress.com/