Implementing Replication for Predictability within Apache Thrift

Jianwei Tu
The Ohio State University
tu.118@osu.edu

ABSTRACT
Interactive applications, such as search, social networking and retail, hosted in cloud data centers generate large quantities of small workloads that require extremely low median and tail latency in order to provide soft real-time performance to users. These small workloads are known as short TCP flows. However, these short TCP flows experience long latencies due to the heavy congestion and network traffic volume. TCP is the dominating transport protocol used in data centers. However, the performance for short flows in TCP is very poor: although in theory they can be finished in 10-20 microseconds with 1G or 10G interconnects, the actual flow completion time (FCT) is as high as tens of milliseconds. This is due to heavy congestion. State of the art forwarding in enterprise and data center environments uses ECMP to statically direct flows across available paths using flow hashing. This doesn’t account for either current network utilization or flow size, and may direct many long flows to the same path causing flash congestion and long-tailed FCT even when the network traffic is lightly loaded.

We propose a transport mechanism using replication for predictability to achieve low FCT for short flows. For each short TCP flow, we replicate it and send identical packets for both flows by creating two connections to the receiver. The application uses the first flow that finishes the transfer. Our results show that by using replication for predictability for short TCP flows, we can reduce median FCT by 15% and tail FCT by 20%. When integrated with Cassandra, we can also improve the performance of Read operation with flow replication.

Keywords
Replication for Predictability; Apache Thrift; Interactive Applications; Flow Completion Time; Tail Latency

1. INTRODUCTION
A variety of interactive applications, such as search, social network content composition, and advertisement selection, are hosted in cloud data centers nowadays. In order to provide high availability to the users, these interactive applications have very strict requirements on the completion time. If these requirements are violated, the services become less available. Less available Internet services cause loss of customers and reduce companies’ revenue. According to the paper [9], delays exceeding 100ms decrease total revenue by 1%.

Interactive applications often follow a Partition/Aggregate design pattern [3]. The incoming requests to the higher layer aggregator of the application are partitioned into several pieces and sent out to the workers in the lower layer. After some computation, the responses of the workers are sent back to the higher layer aggregator and aggregated into the final result. During Partition/Aggregate processes, a large amount of short TCP query and response flows are generated. These short TCP flows need to be completed with the latency boundary to meet strict service level objectives (SLOs).

The network traffic in data centers is a mix of short and long flows, and flow statistics exhibit strong heavy-tail behaviors which mean that most flows (short flows) only have a number of packets, while a very few flows (long flows) have a large number of packets. A study indicated that about 0.02% of all flows contributed more than 59.3% of the total traffic volume [1]. TCP is the dominating transport protocol used in data center. However, the performance for short flows in TCP is very poor: although in theory they can be finished in 10-20 microseconds with 1G or 10G interconnects, the actual flow completion time (FCT) is as high as tens of milliseconds [2]. This is due to heavy congestion. State of the art forwarding in enterprise and data center environments uses ECMP to statically direct flows across available paths using flow hashing. This doesn’t account for either current network utilization or flow size, and may direct many long flows to the same path causing flash congestion and long-tailed FCT even when the network traffic is lightly loaded [2].

We propose a transport mechanism using replication for predictability [2] [9] [10] to achieve low FCT for short flows. For each short TCP flow, we replicate it and send the identical packets for both flows by creating two connections to the receiver. The application uses the first flow that finishes the transfer. We observe that the congestion levels of different paths in data center networks are statistically independent. The original flow and replicated flow are highly likely to traverse different paths, reducing the probability that both paths experience long queuing delay.

We implement flow replication in Apache Thrift [11] transport layer. Apache Thrift is a RPC framework that provides scalable cross-language services implementation. It can be used as a middleware at the application layer that means these is no need to modify the switches and operating systems. We conduct the experiments on our private cloud and Amazon EC2 data center. The latest EC2 data center is known to have multiple equal cost paths between two virtual machines. Our experiment results show that replication for predictability can reduce the Flow Completion Time of short TCP flows over 20%. When integrated with Cassandra, we can also improve the performance of Read operation with flow replication.

2. BACKGROUND
In this section, we introduce the background information of this project.
A. Replication for Predictability
In recent years, replication for predictability has gained increasing attention in both academia and industry. It can be used to provide high throughput, low latency and high availability. In [9], Zoolander scales out using replication for
predictability and uses redundant accesses to mask outlier response times. Google uses request replications to achieve rapid response times in their large online services [14].

B. ECMP Routing

Equal-cost multi-path (ECMP) routing is widely used in today’s datacenters. It’s a mechanism to distribute traffic across multiple paths of the same network cost and perform flow-level load balancing. For a switch that uses ECMP as the routing mechanism, it routes all packets of the same flow based on the hash value of the five-tuple in each packet header (source/destination IP address, source/destination port number, and transport protocol) to the same path. Due to its randomness, it’s highly possible that short TCP flows and large TCP flows are routed to the same path that causes heavy tail latency for short TCP flows.

C. Apache Thrift

Apache Thrift [11] is a popular RPC framework developed by Facebook. It provides scalable cross-language services implementation. For example, if a client that is written in Python needs to communicate with a server that is written in Java, Apache Thrift can be used to provide the interfaces for the cross-language communication. Apache Thrift provides a full software stack which includes Server Layer, Processor Layer, Protocol Layer and Transport Layer. It’s also used as middleware in Apache Cassandra, Apache HBase and JBoss to provide cross-language communication.

D. Apache Cassandra

Apache Cassandra [15] is a distributed storage system developed by Facebook. It manages very large amounts of structured data that spread out across many commodity servers and provides highly available service with no single point of failure. Apache Thrift is used in Cassandra to allow portable (across programming languages) access to Cassandra database.

3. DESIGN

In this section, we first discuss the default design of Apache Thrift and its whole software stack. Then we propose our design for flow replication in Apache Thrift that takes advantage of multi-path diversity that exists due to randomized load balancing in data center networks. We discuss the key components of this design and describe the related challenges and our approach to find the solution.

Because we only replicate short TCP flows ($\leq 100$KB) to achieve better latency, so flow replication is a good idea for latency-sensitive applications that generate a lot of short TCP flows. Here, we keep the existing design and add flow replication as a configurable option with a selection parameter, flow replication enabled. When this parameter is set to true, the application can enable flow replication feature and vice versa.

3.1 Existing Design

Apache Thrift [11] provides a software stack that aims at simplifying the development of cross-language applications. Figure 1 [12] shows the software stack of Apache Thrift.

![Figure 1. Apache Thrift Software Stack](image)

**Transport Layer:** The Transport Layer provides the ability to read from and write to the network. It decouples the underlying transport from other components in the rest of the system. The methods exposed by the Transport interface are open, close, read, write and flush. In addition to the Transport interface, Thrift also uses a ServerTransport interface that is used mainly on the server side to create new Transport objects for incoming connections [17].

**Protocol Layer:**

The Protocol Layer maps in-memory data structures to a wire-format. A protocol defines the way of data types using the underlying Transport Layer to encode and decode themselves. It’s also responsible for serialization and deserialization. Some examples of protocols include JSON, XML, plain text, compact binary etc. [17].

**Processor Layer:** A Processor provides the ability to read data from input streams and write data to output streams. It uses Protocol objects to represent the input and output streams. The compiler of Thrift generates service-specific processor implementations [17].

**Server:** A server is used to create a transport, create input/output protocols for the transport, create a processor based on the input/output protocols and wait for incoming connections and hand them off to the processor [17].

3.2 Proposed Design

Flow replication happens in the Transport Layer that is on top of TCP/IP, so there is no need for us to modify the transport protocol or operating system. We can use this as a middleware for any application to invoke. Before we build the design, we need to make several design choices. First, which short flow should be replicated? We consider flows that are less than or equal to 100KB as short flow and thus can be replicated to reduce the flow completion time. We get this threshold value from [2]. Second, when to replicate the short flows? If we only replicate the short flows when they are suffering long queuing delays, it definitely decreases the performance due to the short duration of these short flows. So in our design, we replicate each short flow from the very beginning to get the best performance. Because short flows
only account for a small fraction of the total bytes in the network, the overhead of replication is negligible.

The major components that we have added in Server side are:

**MultiListener**: The Server initiates a MultiListener during its startup. Instead of listening on just one port, MultiListener is listening to two different ports on the same IP address. MultiListener waits for incoming connection requests from the Client side, and adds the connection to a pre-established queue.

**ConnectionQueue**: ConnectionQueue is used to hold all the connections between Server and Client. Once a connection is established, MultiListener adds the connection into this queue.

**MultiReceiver**: Each MultiReceiver is responsible for two connections: one is the connection for the original flow; the other is the connection for the replicated flow. We allocate an appropriate buffer for each MultiReceiver. For a read operation from Server, once there are some bytes available in one connection, MultiReceiver gets the bytes out of the connection. We use a COUNTER variable to track the number of bytes we have read so far. So if we find out that the bytes we just read are old data, we just throw them away saliently; otherwise we put these new bytes into the buffer and deliver them to the upper level. In this way, we can make sure that we always use the result that arrives first. For a write operation from Client, MultiConnect writes the same data to both of the connections. It’s very similar to the functionality of MultiReceiver.

4. PERFORMANCE EVALUATION

A. Evaluation on Private Cloud

First, we evaluate our implementation on our private cloud. For each pair of virtual machines, there is only one path between them. So it’s obvious that our design can’t benefit from multi-path diversity in this private cloud. However, we can examine the replication overhead here. Because replication of short TCP flows generate more traffic that goes into the network, we want to know if the overhead is negligible or not.

We use a simple add program to do the evaluation. The Server is running on one virtual node and providing an add function which can be called by a remote Client. The remote Client is running on another virtual node. It performs a RPC operation every 1 second and sends out two parameters to the remote Server. The remove Server calculates the result and sends it back to the Client. We measure the latency of this RPC operation.

**Figure 3. FCT for short TCP flows on private cloud**

Figure 3 shows the results. As we can see in (a) and (b), the FCT for short TCP flows with/without replication are almost the same. It proves that the overhead of flow replication can be negligible. This can be explained by the fact that short TCP flows only account for a small fraction of the total bytes in the network.

B. Evaluation on Amazon EC2

To fully exploit the benefits of our implementation, we need to evaluate it in data centers with multiple equal cost paths between many pairs of virtual machines. We evaluate the performance of our implementation in Amazon EC2. Amazon EC2 data center is a large-scale production center that allows us to conduct real-world experiments. The latest EC2 data centers (us-east-1c and us-east-1d) [13] are known to have topology structures that provide many alternative equal cost paths between certain pairs of virtual machines. The typical switches in EC2 data centers run a variant of ECMP routing that perform hash-based flow-level load balancing by hashing flows to equal cost paths across the network topology.
However, we don’t have the knowledge of the exact network topology of the EC2 US East data centers, and we can only infer the network topology. This brings a lot of complexity to our evaluation. Even worse, each instance is a virtual machine that shares the physical hardware with other virtual machines, and it’s difficult for us to decide if two virtual machines are on the same physical hardware or not. The background workload traffic levels in the network are also unknown to us, and can change between experiments.

In order to utilize the multi-path property of the environment, we ran our experiments on 10 EC2 instances (1 server, 9 clients) for 24 hours. The test program used here is the same as we used in our private cloud. We measured the flow completion time between each pair of hosts.

C. Integration with Apache Cassandra

Cassandra uses Thrift to allow portable (across programming languages) access to Cassandra database. We integrate our Thrift libraries into Cassandra. Thrift server in Cassandra is modified to use our libraries. We conduct our experiments on Amazon EC2. We set up a single node Cassandra on an EC2 instance and four Thrift clients on other four EC2 instances.

5. RELATED WORKS

Researchers have proposed many solutions to reduce the FCT of short TCP flows. In [16], they use redundancy to achieve significant mean and tail latency reduction in real-world systems including DNS queries, database servers, and packet forwarding within networks. In [3], they leverage Explicit Congestion Notification (ECN) in the network to provide multi-bit feedback to the end hosts. Preemptive Distributed Quick (PDQ) flow scheduling protocol is designed to complete flow quickly and meet flow deadlines [4]. Deadline-aware scheduling is used in [5]. Beyond short flows, the general problem of handling performance anomalies has also been well studied. These solutions are effective; however, they need to modify switches and operating systems, which make them difficult to be deployed in a large-scale data center. Shen et al [6] used reference executions to detect and debug problems, but their granularity was too large for TCP. Stewart et al [7] and Attariyan et al [8] studied the effects of poor configurations. These approaches detect and fix anomalies online but require running times that are much larger than short flows.

6. CONCLUSION

In this paper, we address the problem of how to use replication for predictability to reduce the Flow Completion Time of short TCP flows. We design and implement replication for predictability within Apache Thrift, which is a popular RPC framework used by companies such as Facebook. Our design replicates each short TCP flow to exploit the multi-path diversity that exists in modern data centers and needs no modifications to transport protocol or operating systems. We conduct the experiments in Amazon EC2 data center, and the results show that our design can achieve better median and tail Flow Completion Time compared to default single flow design.
7. ACKNOWLEDGMENTS
I want to thank my advisor Dr. Christopher Stewart for his support and help. He gave me a lot of insightful advices for this whole project.

8. REFERENCES