Network Connection Optimization for Serverless Workloads

University of Colorado Boulder

Erika Hunhoff
erika.hunhoff@colorado.edu

Eric Rozner
eric.rozner@colorado.edu

Introduction

Serverless Background

- Application developers produce an event-driven function to the cloud provider
- The cloud provider is responsible for invocation, scaling, billing, failure, and management

Motivation

The use of short-lived, independent units of computation can lead to avoidable inefficiencies:

- Traditional applications would maintain long-held TCP connections
- Each newly instantiated serverless function must create new sockets and perform TCP slow start
- Serverless functions block for writes, while distributed services use asynchronous writes
- Caching and advanced reads can reduce time spent on repetitive, predictable operations

We first develop an OS-level shim layer to provide socket reuse between identical short-lived functions.

Architecture

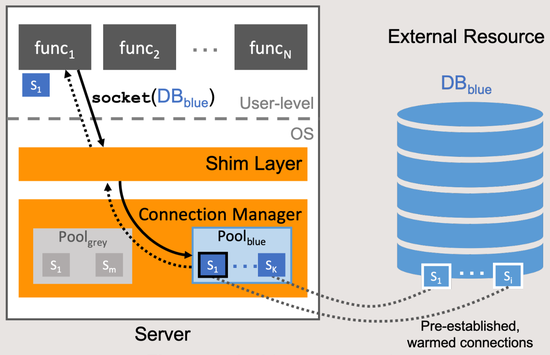

Figure 1: Architecture overview

- A serverless function running in container $func_1$ opens a socket to an external database DB_{blue}
- A shim layer intercepts socket system calls
- The pool returns pre-existing socket S_1
- S_1 is an unused warmed socket or a socket created by a previous invocation

Preliminary Work - Reuse

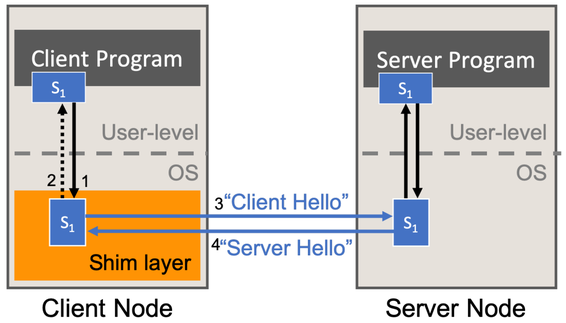

Figure 2: Experiment architecture for reuse

The effects of reuse are measured using two nodes (Figure 2):

- A node with a client program that runs twice (once to use the socket, once to reuse)
- A node running a server program representing an external resource

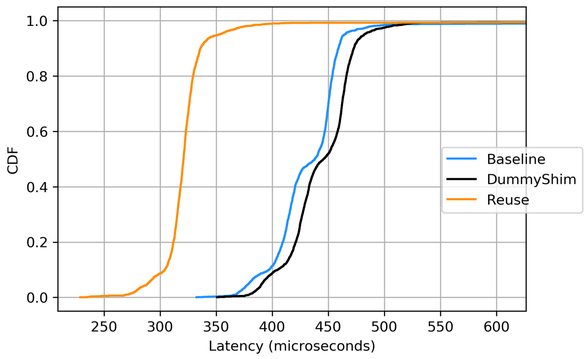

Figure 3: CDF of latencies from reuse test (2000 test runs)

Figure 3 shows latency measured from connection establishment to completion for three client configurations:

- **Baseline:** No shim layer is installed, system is unmodified (median: 446 μs)
- **DummyShim:** A shim layer intercepts system calls, but does not change socket behavior (median: 459 μs)
- **Reuse:** A shim layer contains minimal logic needed to perform one instance of socket reuse (median: 322 μs)

Experimental Setup

Two m510 nodes provisioned on CloudLab

- Eight-core Intel Xeon D-1548 at 2.0 GHz
- Dual-port Mellanox ConnectX-3 10 GB NICs
- Ubuntu 18.04.4 LTS
- Linux 5.3.0-28-generic

Shim Design

The shim is a loadable kernel module (LKM) that intercepts system calls by overwriting function addresses in the system call table (similar to the SlimSocket component in Slim [NSDI '19]).

Preliminary Work - FCT

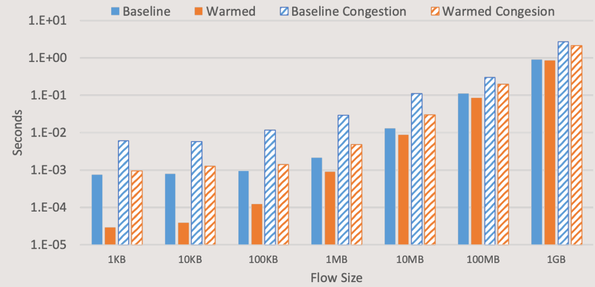

Figure 4: FCT results

Figure 4 shows **flow completion time** (FCT) for a new connection versus a **warmed** connection:

- Warmed implies a connected socket that has already converged to an appropriate TCP congestion window size
- **Baseline FCT** measures time from socket creation to flow completion
- **Warmed FCT** measures time to send the data

Warmed connections finish faster because:

- No overhead from TCP handshakes
- Larger window sizes at start of transmission
- Less round-trips to send data

Small flows (≤ 1 MB) complete 2-24 ms faster (with congestion) or 0.7-1.2 ms faster (no congestion).

Future Work

Create a pool manager with features including:

- Communication between the pool manager and the function scheduler
- Intelligent connection pool garbage collection
- Network probing to approximate appropriate congestion windows for warmed sockets

The long-term goal is to explore:

- TLS integration
- Asynchronous writes
- Caching of commonly accessed data
- Proactive retrieval of network content