Pandit: A Service Level Distributed Proxy.

Eric Gerard Moynihan

November 2021

Outline

This project entails creating a proxy that runs on each host machine in a cluster, turning services in the cluster into widely available microservices.

This will be done by providing a translation layer between the application layer of the containers making up a service, and the clients & proxies. This translation layer will provide Protobuf [1] responses in place of the traditional container responses.

Clients running on different hosts in the cluster will make the proxy aware that they are interested in certain services. When a client sends a request to a service, the proxy will first look for cached data on the host. If it cannot find it, it will delegate the request to an authoritative container in the cluster.

These Protobuf responses can then be distributed to relevant instances of the proxy running on other hosts for caching purposes. Since the mapping between application layer and Protobuf is implementation specific, there can be side-effects pre-programmed into the resulting Protobuf, such as marking some fields to be updated in the cache before being written to the authoritative container(s).

Protobufs can be used to generate code for client applications. This allows for the implementation of a simple application programming framework to aid in the creation of clients that wish to interface with one or more Pandit services.

This approach to a distributed proxy will allow for improved read/write performance within a cluster, as well as simplify distributed application development since every dependency can be interfaced via a universal API.

First Steps

1. **November** - Implement an eBPF [2] program that uses eXpress Data Path (XDP) [3] to parse application layer payloads. This can be used as a template by the configurable userspace program.

2. **December** - Write a userspace program that can be configured by the user to load + attach an eBPF program to a container's network interface and parse the data in a user-defined manner.

3. **January - February** - Allow the program to act as a gRPC [4] server that can be interfaced with by clients, providing the proxied Protobufs as responses.

4. **February onwards** - Implement and test the distributed features of the proxy in a mock cloud setup, with multiple VMs on the same subnet.

References

[1] Profocol buffers documentation - developers.google.com.

[2] eBPF official website - eBPF.io.

[3] eXpress Data Path official website - Kernel.org.

[4] gRPC official website - grpc.io.