Application Monitoring As A Network Service

Laetitia Fesselier, laetitia.fesselier@mail.mcgill.ca
School of Computer Science, McGill University, Montreal, Canada

Research Advisors:
Bettina Kemme, kemme@cs.mcgill.ca
Mona ElSaadawy, mona.elsaadawy@mail.mcgill.ca

1 Introduction

In today's service-oriented platforms, application execution is distributed across many components. Monitoring the performance of such a system is a critical and challenging task. Most traditional monitoring techniques require an integration on the application level, making the process platform dependent.

With the global rise of server virtualization, traditional network switches are being gradually replaced with software-defined network controllers (SDN switches) that rely on code as opposed to network bridges and hardware.

The approach used in our research uses a customized port sniffer for software switches, moving the monitoring logic on the network level by extracting useful information from messages exchanged between application components. By doing so, a more flexible, platform independent approach is offered.

2 Monitoring Service

To demonstrate the benefits of this approach, an application monitoring prototype, restricted to http traffic as a first stage, was developed using an existing http sniffer[1] software which was extended to behave as a server service to perform different analysis tasks concurrently. Its architecture was re engineered with some performance improvement.

The service was enhanced to allow for the following computations: average/min/max of request service time, request rate, error rate, throughput, connection rate, and cumulative distribution of clients, request paths, request methods, request type and response status. They are computed using information extracted from the http packet headers using the libpcap library.

Users who want to monitor the performance of an application (in Figure 2.3 depicted with a web-server/ database / memcache architecture) receive these performance measurements through a dashboard (Figure 2.2). The monitoring service itself follows a distributed architecture (Figure 2.3).

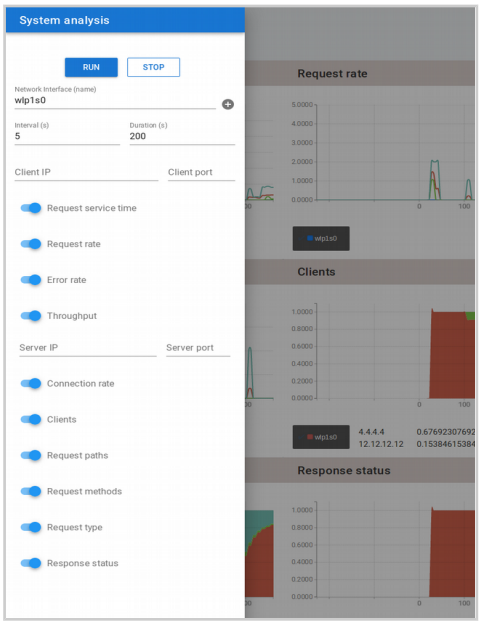

Figure 2.1 - Front-End form options

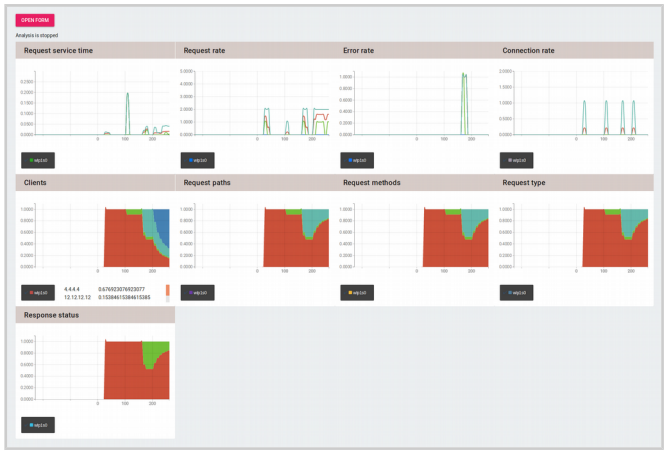

Figure 2.2 - Front-End analysis result

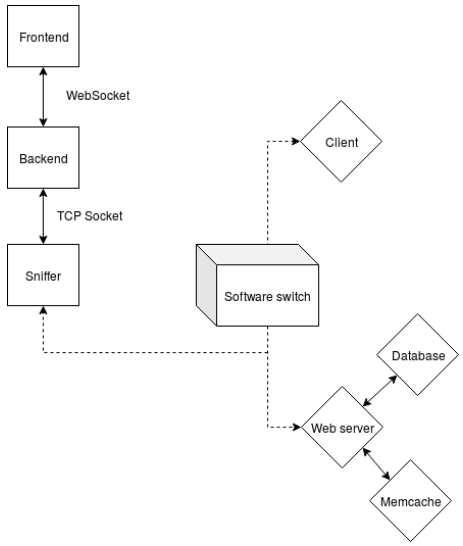

Figure 2.3 – Monitoring Application integration in a Cloud system

2.1 Architecture

Front-end The front-end is the interface which the user interacts with. It is build over the reactive framework vue.js, particularly adapted for real-time applications, the server Express (Node.js), and the graph library Ricksaw.js.

Back-end The back-end, developed in Rust, forwards the specific monitoring requests from the clients to the relevant sniffer and receives the performance measurement results from the sniffers to be forwarded to the clients. It might also perform some analysis tasks itself to minimize the computation done by the sniffers on the network level. Rust is a highly efficient compiled language which comes with a web server ecosystem, making it a good choice for a web back-end.

For better performance, the messages sent over TCP are compressed using Google Protocol Buffers[2].

Sniffer The sniffer, written in c, is attached to the software switch (Open vSwitch) that routes the cloud traffic. It uses a multi-threaded architecture to listen to the network traffic, filtering packets based on their port number. Useful information is then extracted and stored in a collection of hash tables, and metrics are computed in given time intervals.

3 Conclusion, related and future work

The preliminary results of this first attempt demonstrate some great potential. Optimizations on the original sniffer code allowed us to achieve pretty good performance :

- low to no drop of packets under relatively high traffic;
- obtained metrics agree with the theoretical values.

Measurements were conducted with a modified benchmark tool (YCSB[3]) and a network traffic simulator (ScapyTrafficGenerator[4]).

This demonstrates how this approach can be used as an effective way to monitor any cloud system environment, in a very flexible way, as no knowledge about the hosted application is required. Furthermore, our monitoring application provides major metrics encountered in some commercial cloud monitoring tools[5] without the need to instrument the application.

Future works will extend this prototype capabilities to other defined protocols (mysql, memcache), as well as a system to integrate user-defined packet parsers and metrics. This later feature will bring this project to a next level, giving the user the ability to extend this application to any kind of protocol.

[1] github.com/caesar0301/http-sniffer
[2] developers.google.com/protocol-buffers/
[3] research.yahoo.com/news/yahoo-cloud-serving-benchmark/
[4] pypi.org/project/ScapyTrafficGenerator
[5] cloud.google.com/monitoring/api/metrics