**Application Monitoring As A Network Service**

|  |  |
| --- | --- |
| Laetitia Fesselier  laetitia.fesselier@mail.mcgill.ca  School of Computer Science, McGill University  Montreal, Quebec, Canada  *Research Advisors:*  Bettina Kemme, kemme@cs.mcgill.ca  Mona ElSaadawy, mona.elsaadawy@mail.mcgill.ca | |
| **1 Introduction**  In a distributed system environment, monitoring the performance of a system is a critical and challenging task. Most traditional monitoring techniques require an integration on the application level, making the process platform dependent.  Using new techniques as Software Defined Networking (SDN) offers some promising feature, moving the monitoring logic on the network level. This approach uses a customized port sniffer for software switches which extract information from the packets flowing on the network. By doing so, a more flexible, platform independent approach is offered.  **2 Application**  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 concurently. Its architecture was reengineered with some performance improvement.  The following metric computations were integrated: average/min/max of request service times, request rates, error rates, throughputs, connection rates, 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.  This web-based application is a dashboard of application performance measurement graphs (Figure 2.2) following a distributed architecture. It is made of the following components: | ***Figure 2.1 - Front-End form options***    ***Figure 2.2 - Front-End analysis result*** |

|  |  |
| --- | --- |
| ***Figure 2.3 – Monitoring Application integration  in a Cloud system***  **Frontend** The frontend is the interface which the user interacts with, to indicate what kind of performance monitoring should be performed on the application, an optional source, and the analysis duration and computation interval, and to receive the performance results over time in near real-time. It is build over the reactive framework vue.js, particularly adapted for real-time applications, Express (Node.js) and the graph library Ricksaw.js.  **Backend** The backend, developped in rust, forwards the specific monitoring requests from the clients (WebSocket) to the relevant sniffer (TCP socket) 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 an efficient compiled language (execution time benchmarks show it performs faster than c++) and safer because of a lot of protection mechanisms on compilation time. It also comes with a web server eco-system which make it a good choice for a web backend. Finally, it can compile in web assembly, which make it also a good choice to develop frontend features. | 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 switches (Open vSwitch) of the cloud that route the cloud traffic. The sniffer use a multi-threaded architecture to listen to the network traffic, filtering packets based on their port number, and their content (only packets holding an application/HTTP headers are analyzed). 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. A lot of 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.   Mesurements were conducted with a modified benchmark tool (YCSB[3]) or a network traffic simulator (ScapyTrafficGenerator[4]).  This demonstrate how this approach can be use as an effective way to monitor any system environment, in a very flexible way, as no knowledge about the hosted application is required. Furthermore, our monitoring application provides major metrics we encounter in some commercial cloud monitoring tools[5] without the need to instrumenting 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 |