DSML: an SDN-Based Debugging Tool for Distributed Systems

Devin Ekins, Hadeel Maryoosh,Yue Fei , Eric Eide, Jacobus Van der Merwe

School of Computing, University of Utah

Abstract

(here we write the abstract)

Categories and Subject Descriptors

(If applicable)

Keyword

(any keyword in the paper)

1 Introduction

Debugging a distributed system can be a very difficult and challenging task. A distributed system consists of multiple processes, running across multiple computers, that cooperate across a network to perform some task. This debugging challenge is due to scalability of these systems which is one of their standard characteristics. An example is a web site implemented as a distributed system including a load balancer, a set of HTTP servers, and a set of database servers. If the system as a whole misbehaves, it can be tricky to figure out which of the many pieces of the system is at fault. Even monitoring the communication between the parts of the system can be a challenge.

To help the users simplifying the debugging process of these systems, many proposals and tools have been presented(see section 7 relate work). However, each of them has disadvantages and drawbacks. (we will mention the previous work and their drawbacks with citations).

We present our approach "DSML"; an SDN-Based Debugging Tool for Distributed Systems, which is a new way to observe and control the communication events between the parts of a distributed system using Software-defined networking (SDN) and state machine language. We were motivated by the need of a simple and practical debugging tool, and created our approach that meets the following characteristics: 1) It suppresses only relevant traffic to the debugger, which saves the need for monitoring the entirety of network traffic that can be expensive, huge effort required and less performance efficient 2) users can get feedbacks they need to make informed debugging corrections. (we can add more good characteristics)

Using these characteristics as our goals, we built and designed our approach " DSML", a debugging tool for distributed systems using SDN and state machine language. With DSML, we were able to (this will involve our implementation and evaluation summary with the results).

2 Solution

Our system allows users to define network protocols in terms of states of a Definite Finite Automaton

(DFA) using our Debugger State Machine Language (DSML). Because explaining network protocols often involves a graphical representation as a DFA, we felt this would be a good way to model network status from the perspective of a debugger as well.

DSML supports the definition of states as well as the transitions between them. Transitions are defined by one or more matching operations which performs deep packet inspection to match a fields contents (or a substring of its contents) against a value. Alternatively, a state transition may be defined by a timeout wherein no matching packets were encountered.

As deep packet expansion is expensive, we provide a mechanism to avoid inspecting the entirety of network traffic. To avoid performing these expensive operations on every packet sent through the network, DSML also allows for OpenFlow filters to only funnel relevant traffic to the debugger itself. This funneling is performed at line rate. For example, a user may know that this debugger only need to consider packets with destination port of 80 for HTTP traffic. Then the debugger only performs inspection on this subset of packets.

Importantly, DSML allows for the use of side-effect operations which include logging to a file or printing to a console relevant information as well as displaying its equivalent of a ?stack trace?. This stack trace is a history of network states that were traversed to arrive at the current position as well as the packets that were matched on for each transition. This allows users to get the feedback they need to make informed debugging corrections.

The DSML protocol script written by the user is read into a compiler to produce the Debugger State Machine (DSM) which then may be run on a controller with an OpenFlow-enabled switch to begin debugging network traffic. Switches need not support OpenFlow to send traffic to the DSM, but the optimization gained by the OpenFlow filters will not be present if OpenFlow is not supported on the switch.

3 Implementation

- DSML parsing in Python
- DSML parsing to Python
- Mapping between native Python and DSML
- Global Dictionary (maybe?)
- POX/RYU Controller
- Connection between switch and controller

4 Evaluation

- Compiler performance
- DSML Ease of Use
- Run-time performance
- DSM Usefulness

5 Future Work

- Syntactic Sugar
- Extend Openflow (maybe?)
- Composition of Protocols
- Parallel DSMs (if not done)

6 Related Work

The topic of debugging the distributed system has long been studied[8], but as the scale and functionalities keep growing, the components of today's distributed system are not only distributed deployed but also admixed and twisted as building blocks and may written by different languages[3]. And currently there are good examples to debug specific distributed applications, such as MPI applications[4]. But a general purpose debugging tool is still needed.

Using 'Big Data' techniques[3] to understand and modeling the behavior of distributed system(single request triggers parallel execution of multiple software components) is a good idea since with the scale grown up, there are sufficient observations for the learning procedure.

There are multiple way to observe and monitor the distributed systems or the application running on them. lprof[10] is statically listen to the binary code to infer how logs can be parsed and then the intertwined log entries can be associated to specific individual request. There are also low-cost hardware solution to it. Minerva[9] affects no execution timing and is non-intrusive. By using on-chip debug port, Minerva got full control of the system(wireless sensor network) that can stopping on one breaking point synchronously.

Other approaches including adding units either on the system or on the applications. Design a Pervasive Debugger[6] for distributed applications then execute the target in a virtual environment the debugger can get control and exam the application. Declarative Tracepoints[2] is providing a debugging system allow uses to insert action-associated checkpoints, that the applications to be debugged at runtime and require no change to source code.

Some instructive research allowed users to describe their expectations or release them from repetitious operations. Data Centric[1] allow uses to declare their expectations about the program state as a whole than a single process state, enable scalable solution to very large problems. Dataflow Language[7] is programmable or scriptable for users to define the debugging work.

7 Conclusion

- Clearly, the internet is made for pizza.
- Merits of various pizza formats
- IEEE standard for secure pizza distribution protocol

References

[1] Abramson, D., Dinh, M. N., Kurniawan, D., Moench, B., and DeRose, L. Data centric highly parallel debugging. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (2010), ACM, pp. 119–129.

[2] Cao, Q., Abdelzaher, T., Stankovic, J., Whitehouse, K., and Luo, L. Declarative tracepoints: a programmable and application independent debugging system for wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems* (2008), ACM, pp. 85–98.

[3] Chow, M., Meisner, D., Flinn, J., Peek, D., and Wenisch, T. F. The mystery machine: End-to-end performance analysis of large-scale internet services. In *Proceedings of the 11th symposium on Operating Systems Design and Implementation* (2014).

[4] Dryden, N. Pgdb: A debugger for mpi applications. In *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment* (2014), ACM, p. 44.

[5] Han, D., Anand, A., Dogar, F., Li, B., Lim, H., Machado, M., Mukundan, A., Wu, W., Akella, A., Andersen, D. G., Byers, J. W., Seshan, S., and Steenkiste, P. Xia: Efficient support for evolvable internetworking. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2012), NSDI'12, USENIX Association, pp. 23–23.

[6] Ho, A., and Hand, S. On the design of a pervasive debugger. In *Proceedings of the sixth international symposium on Automated analysis-driven debugging* (2005), ACM, pp. 117–122.

[7] Marceau, G., Cooper, G. H., Krishnamurthi, S., and Reiss, S. P. A dataflow language for scriptable debugging. In *Proceedings of the 19th IEEE international conference on Automated software engineering* (2004), IEEE Computer Society, pp. 218–227.

[8] Miller, B. P., and Choi, J.-D. Breakpoints and halting in distributed programs. In *Distributed Computing Systems, 1988., 8th International Conference on* (1988), IEEE, pp. 316–323.

[9] Sommer, P., and Kusy, B. Minerva: distributed tracing and debugging in wireless sensor networks. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems* (2013), ACM, p. 12.

[10] Zhao, X., Zhang, Y., Lion, D., Faizan, M., Luo, Y., Yuan, D., and Stumm, M. lprof: A nonintrusive request flow profiler for distributed systems. In *Proceedings of the 11th Symposium on Operating Systems Design and Implementation* (2014).