Implementation of Self-Managing Applications on Cloud Using Overlay Networks

Nasim Beigi-Mohammadi, Hamzeh Khazaei, Mark Shtern, Cornel Barna and Marin Litoiu
Adaptive Systems Research Lab, York University Toronto, Ontario, Canada
Email: {nbm, hkh, mark, cornel, mlitoiu}@yorku.ca

***Abstract*—In this paper, we present an architecture and implementation for self-managing cloud application using overlay networks and software defined networking (SDN). Through real world experiments on Amazon EC2 and Smart Applications on Virtual Infrastructure (SAVI) cloud, we demonstrate how our management mechanism autonomously maintains SLAs of application scenarios without provisioning extra resources.**

***Index Terms*—Adaptive applications, SDN, Overlay networks.**

I. Introduction

Software Defined Networks(SDN) and advancements in network virtualization create new opportunities for self-adaptive applications to take advantage of network programmability for achieving adaptation goals. In our previous work [1], we presented an adaptive service management for cloud applications using overlay networks where the application autonomic manager monitors the response time of application scenarios and then takes adaptation actions. If the response time of some scenarios is too high, it uses a hill climbing algorithm with greedy selection criteria to find scenarios for which to dynamically reduce the bandwidth. This way, autonomic manager reduces the contention for using application resources which helps to improve the response time of scenarios in need. In this paper, we present the architecture and implementation of our solution presented in [1].

II. Architecture

Figure 1 illustrates the architecture of our adaptive service management mechanism on hybrid cloud. The main components of our solution are as follows:

- Application: we consider an application that consists of different scenarios where a scenario uses a chain of services within application topology in a service-oriented architecture.
- Autonomic manager: manages the application with regards to application requirements at run-time. It follows a Monitor-Analyze-Plan-Execute(MAPE) loop to achieve application objectives. Basically, the autonomic manager first monitors the response time of application scenarios and checks if the SLA for the response time of any scenario is violated. If yes, it first tries to fix the problem by using bandwidth adaptation. To do so, using a hill climbing heuristics with a greedy criterion, it finds scenarios whose response time is below certain threshold with respect to their SLAs. Among all the candidates, it reduces the bandwidth of the flows belonging to a scenario that has the highest throughput. If no candidate scenario is found, the autonomic manager scales out the application to meet SLAs. Basically, the autonomic manager is responsible to dynamically instantiate and configure the application nodes and links automatically. It performs these tasks with the following steps:
  1) It instantiates the application nodes based on application topology;
  2) It then creates an overlay network that connects application nodes over cloud provider(s) network and configures the nodes accordingly;
  3) After application cluster is ready, autonomic manager iteratively monitors the response time of various scenarios, analyzes and plans actions when the need arises.
- SDN controller: programs network flows within the application overlay network based on policies dictated by the autonomic manager.
- Proxy: to monitor the response times, arrival rates, and throughput of application, autonomic manager makes use of a proxy node to timestamp the requests and responses. The response time will be calculated from the moment the proxy receives the request from the client until the proxy receives the response from the application. Also, in our current architecture, the autonomic manager differentiates various scenarios by using the proxy node described in the next section.

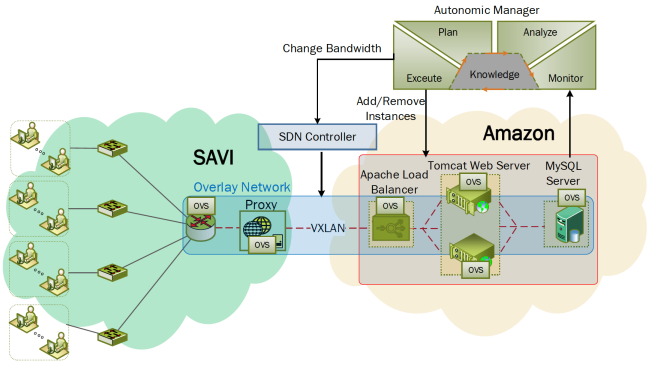


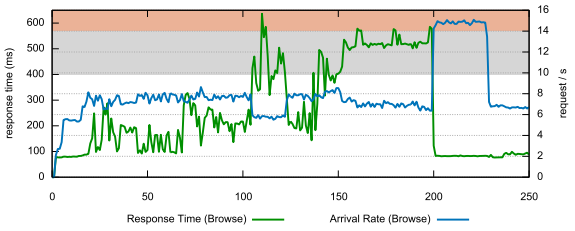
Figure 1: *An overlay network that spans over private and public cloud connects application components. The bandwidth rate of any link is programmed on the fly.*

III. Implementation

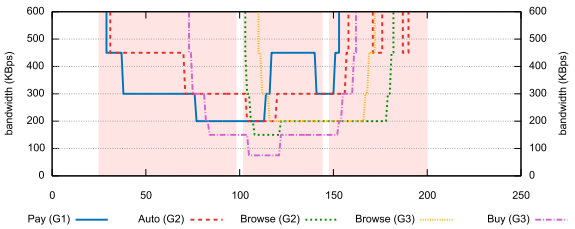
The implementation of the main components of our solution is as follows:

- Application: we use a three tier e-commerce application consisting of different scenarios including *browse*, *buy*, *pay* and *auto-bundle*. A user sends a request to use any of the offered scenarios, waits for the response, thinks for sometime and then sends the next request. User's request for using any of these scenarios uses a chain of services within application cluster.
- Autonomic manager: we have implemented an autonomic manager based on MAPE loop in Java. XML files are used to describe the application topology including the name of instance images, size, availability zones etc. The autonomic manager uses SNMP and CloudWatch probes on nodes on SAVI and Amazon respectively to monitor application specific metrics. The autonomic manger builds the application cluster dynamically using the topology XML files. Images of application nodes are Ubuntu 14.04 with Open Virtual Switch (OVS) [1] installed. Hence, once the application nodes are instantiated, the autonomic manager connects to each node and creates a virtual tunneling endpoint as well as an interface and assigns an IP to that interface. Then, based on application topology, it builds VXLAN [2] links to connect nodes to a VM that acts as a switch within the application cluster. The result will be an application cluster that is built on an overlay network on top of networks of SAVI and Amazon.
- SDN controller: we have implemented a multi-thread control application in Python on top of Ryu[3]. One thread is responsible to respond to requests coming from switches and the other continuously listens for commands arriving from the autonomic manager to apply specified bandwidth rates to interfaces. We use RabbitMQ for communication between autonomic manager and SDN controller.
- Proxy: as our solution works on dynamically adjusting bandwidth of scenario flows, we have implemented a method to distinguish flows based on scenarios and then apply the appropriate bandwidth rate on them. Using regular expressions, we have implemented a Java application on proxy that categorizes requests based on scenarios, once requests arrive (i.e., the customer-facing node is the proxy node). Then, since all application flows share common resources, we have implemented the proxy application such that each scenario request will be forwarded to a specific interface. Now that every scenario has its own interface, we can have the SDN controller control the bandwidth rate of that interface according to the adaptation goals. SDN controller uses traffic policy features of OVS to dynamically configure the bandwidth of each scenario. Such categorization and bandwidth management can happen between any two nodes of the application so that resource contention can be controlled in a fine granular manner. In [1], we present in details the features on the proxy.

[1] http://openvswitch.org
[2] https://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-00
[3] http://osrg.github.io/ryu



(a) *Arrival rate and response time of scenario Browse. The first shaded area (Orange) shows SLA violation.*



(b) *Bandwidth rates of different scenarios*

Figure 2: *The management mechanism successfully maintains application response time SLAs by adjusting the bandwidth rate of scenarios [1].*

Figure 2 shows sample results from our implementation on hybrid cloud.

Table I: *Experiment spec on SAVI and Amazon. VMs on SAVI are OpenStack small size.*

Components	# VMs	Flavor	Software
App Cluster (Amazon)	4	Load balancer (c3.xlarge), web server (m3.large), database (m3.medium)	Apache Load Balancer, eStore Tomcat Web App, MySQL
Proxy (SAVI)	1	Small	Custom Java app
Virtual Gateway (SAVI)	1	Small	OVS
SDN Controller (SAVI)	1	Small	Ryu Custom Controller App
Clients (SAVI)	4	Small	Python Scripts

IV. Conclusion

In this paper, we presented the architecture and implementation details of our work [1] on SAVI and Amazon cloud. We demonstrate that our autonomous solution helps application maintain their SLA by smartly managing bandwidth between application scenarios while avoiding provisioning extra resources for as long as possible.

References

[1] N. Beigi-Mohammadi, H. Khazaei, M. Shtern, C. Barna, and M. Litoiu, "Adaptive service management for cloud applications using overlay networks," in *To appear in 15th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2017.