Monitoring and Measurement in Software-Defined Infrastructure

Jieyu Lin, Rajsimman Ravichandiran, Hadi Bannazadeh, Alberto Leon-Garcia
Electrical and Computer Engineering
University of Toronto
Toronto, Ontario
Email: {jieyu.lin,rajsimman.ravichandiran,hadi.bannazadeh,alberto.leongarcia}@mail.utoronto.ca

***Abstract*—Software-Defined Infrastructure (SDI) presents an approach for integrated management of virtualized heterogeneous resources. Monitoring and measurement is an essential component for effective control and management. This paper presents an architecture of a system, named MonArch, based on SDI that provides integrated monitoring and measurement functionalities. Unlike existing cloud and network monitoring systems, MonArch supports execution of user-generated monitoring tasks, offers monitoring as a service to tenants, administrators as well as management modules, and provides a framework for monitoring data analytics. We have implemented and deployed MonArch in the SAVI Testbed, and our experience shows the system is able to support a wide variety of monitoring tasks while achieving high performance and scalability.**

I. Introduction

As cloud computing and virtualization technologies evolved quickly in the past few years, it has become apparent that in certain settings cloud computing systems will utilize a variety of virtualized resources. A prime example is in the edge of the network were virtualized programmable hardware may be required to provide the required levels of performance. To meet this need, we introduced the concept of Software Defined Infrastructure (SDI) in [1]. SDI presents an architecture for integrated management of heterogeneous resources. We used the SDI concept in designing the Smart Applications on Virtual Infrastructure (SAVI) Testbed which is intended for experimentation with future applications and services. To effectively and dynamically manage resources, SDI requires monitoring and measurement of the converged heterogeneous resources.

Although there has much research and many projects on monitoring and measurement, there are still some missing pieces and challenges in cloud environment. First, due to the heterogeneous nature of cloud infrastructure, a monitoring and measurement system (MMS) should be capable of performing scalable integrated monitoring of converged heterogeneous resources. For example, in the SAVI Testbed's two-tiered cloud infrastructure, in addition to the traditional compute, storage, and network resources, we also have GPU, programmable hardware (e.g. FPGA), wireless access point, and Software Defined Radio (SDR) resources. In order to effectively manage the whole infrastructure, the MMS needs to handle all these resources and provide extensibility for new resources that may be added to the infrastructure. Second, with the large amount of monitoring data generated in the cloud environment and the diverse processing/analysis requirements, traditional MMS systems are lacking in their flexibility and capabilty to perform data processing efficiently. A new monitoring system needs to be designed to allow flexible processing of large amount of new and historical monitoring data. Third, in the age of cloud computing, many IT functionalities are provided to the users ”as a service”. We believe monitoring and measurement should be provided as a service to the users of a cloud infrastructure. More specifically, this entails providing monitoring and measurement data as well as user-specified analytics results to users on-demand, and allowing users to monitor custom metrics of resources in both the infrastructure layer and the application layer.

In this paper, we present a new monitoring and measurement architecture of a system called MonArch to tackle the challenges mentioned above. MonArch provides flexible, integrated monitoring and measurement of heterogeneous resources with data analysis functionality, and it provides monitoring and analytics results to users on-demand. A challenge in designing this system is to satisfy the requirements without sacrificing the performance and scalability.

We have implemented MonArch using open source software such as OpenStack Ceilometer [2] and Spark [3]. The system has been deployed and operational on the SAVI Testbed since May 2014. In this time, the system has demonstrated its stable performance and its ability to meet the monitoring and measurement needs in the Testbed.

For evaluation purposes, we have verified the functionalities of the system and evaluated its performance and scalability. For performance and scalability evaluation, we have focused on examining the system capability to meet demand from a large number of users and for the infrastructure to scale up. The results show that the system is able to perform the functional requirements and at the same time provides scalability.

This paper is organized as follows. Section II presents related work, and Section III details the requirements of the MonArch MMS. The system design and implementation is presented in Section IV, followed by a system evaluation (Section V) and conclusions (Section VI).

II. Related Work

The MMS in this paper is designed based on the concept of Software Defined Infrastructure (SDI). In [4] we elaborated on the SDI architecture, and presented the design and implementation of the control and management system in the SAVI Testbed.

There is a large body of research and projects focusing on various aspects of MMS. Nagios [5] and Ganglia [6] are traditional tools for IT infrastructure monitoring. They are mature and stable. However, these tools only provide basic statistics of monitoring data and can not satisfy the complex processing requirement today. The MISURE system [7] is an application-level cloud monitoring system that uses stream processing. It proposed a scalable and fault tolerant framework for monitoring applications running in a cloud environment; however, MISURE is mainly for application monitoring and therefore is not suitable for infrastructure monitoring.

Ceilometer is a telemetry component of OpenStack, an open source cloud platform. It provides mechanisms for monitoring virtual compute, network, and storage resources. Ceilometer provides very limited analytics capability for monitoring data (only average, sum, max, and min). CloudView[8] and MONaaS [9] are other systems that integrate with OpenStack, but they have the same limitation as Ceilometer. [10] and [11] presents solutions for cross layer monitoring. [11] uses Ceilometer and assumes that application layer data are available using a monitoring as service model, which raised the question of how to provide application layer monitoring data as a service. This question is addressed in this paper.

For Software Defined Networking (SDN), FlowSense [12] is a push-based SDN monitoring system for network utilization. It provides basic OpenFlow network monitoring. PayLess [13] proposed a network monitoring framework for monitoring OpenFlow networks. It uses a variable frequency flow-statistics collection algorithm to improve the monitoring overhead. OpenNetMon [14] provides per-flow metrics monitoring in an OpenFlow network: bandwidth, delay and packet loss. The SDN monitoring research work are area specific and cannot be used as a general monitoring system in a cloud environment, but it can provide network monitoring data for the MonArch system.

On the monitoring analytics side, Monalytics [15] presents a hierarchical monitoring and analytics system. It proposes integration of monitoring and online analytics, and suggests that analytic tasks should be executed locally at the monitoring data acquisition point. This analytics model suffers from lack of global view and fault tolerance.

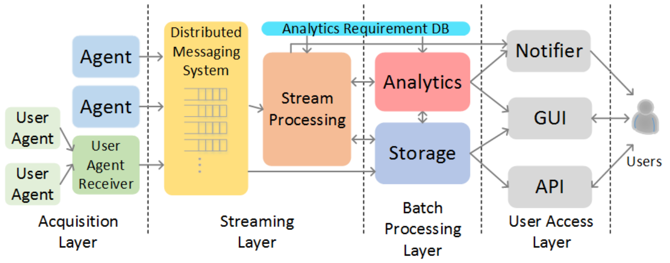

Fig. 1. Monitoring and Measurement System Architecture

III. Requirements for the Monitoring and Measurement System

To design a MMS based on the SDI concept that is suitable for the two-tiered cloud environment in the SAVI Testbed, we identified the following requirements:

Integrated and Cross-Layer Monitoring and Measurement: The system should be capable of monitoring the converged heterogeneous resources in an infrastructure as well as the applications running on the infrastructure. Heterogeneous resources include physical and virtual compute, network, storage, GPU, programmable hardware, and wireless access points.

Data On-Demand: Monitoring and measurement data should be available on-demand to the users and the management modules. In a cloud infrastructure, management modules and users obtain monitoring information or analytics results from MMS in order to make real-time decisions.

Support for Custom Metrics: Due to the nature of applications, the monitoring requirements will vary from application to application. Since it is not realistic nor efficient to monitor every metric that users might need, the system allow users to monitor user-specified custom metrics.

Others: Other requirements include 1) Storing historical monitoring data; 2) Providing (near) real-time processing capability; 3) Scalability; 4) Elasticity

IV. System Design and Implementation

We now describe MonArch which is designed based on SDI and the requirements in the last section.

A. *Design and Architecture*

The high level logical architecture of MonArch is shown in Figure 1. This architecture is divided into four vertical layers: acquisition, streaming, batch processing, and user access layer. Monitoring data generally moves from left to right. The acquisition layer obtains or generates monitoring and measurement data. These data are sent to the streaming layer for transport and stream processing. The raw data (and optionally the stream processed data) are then sent to the batch processing layer for storage and batch analytics. Users can access the monitoring data and analytics results through the access layer. This architecture allows the system to achieve monitoring data collection, processing (stream and batch), and distribution functionalities with scalability and flexibility.

1) Acquisition Layer: The acquisition layer is responsible for acquiring and obtaining monitoring data and submitting it to the streaming layer. There are three kinds of components in this layer: agent, user agent, and user agent receiver.

Agent: Agents acquire basic infrastructure level monitoring data and send it to the streaming layer. These data include virtual and physical server states, network states, and all other metrics that need to be monitored.

User Agent: A User Agent allows users to monitor custom metrics. Through the User Agent, users can specify the metrics to monitor and an optional monitoring frequency. Examples of custom metrics include web server metrics (e.g. requests per second, request latency, request location, number of connections), database throughput, number of online applications.

User Agent Receiver: The User Agent Receiver accepts monitoring data from User Agents, checks the format, and performs access control. If there is no problem with monitoring data, it sends the data to the streaming Layer. The main purpose of this component is to protect the system from user attacks through the User Agent. This component could be used in the future for billing purposes as well.

2) Streaming Layer: The streaming layer is responsible for transporting monitoring data to the storage module and performing stream processing for real-time analytics purposes. The layer includes a messaging system and a stream processing system.

Messaging system: As the monitoring data are generated and sent to the streaming layer, the messaging system buffers the data before it is processed by the Stream processing module. We emphasize that the Messaging system should be distributed and scalable.

Stream Processing: The Stream Processing module is responsible for (near) real-time processing of the incoming monitoring data. It retrieves monitoring data from the Messaging system and performs required processing tasks and actions. Examples of stream processing include detecting temporal or spatial correlation of the monitoring data across different resources, and detecting anomalies based on predefined or learned patterns. The Stream Processing module must be scalability.

3) Batch Processing Layer: This layer is responsible for storage and analytics of the stored monitoring data. It has two modules: Storage and Analytics. These modules operate cooperatively with the modules in the streaming layer.

The Storage module stores raw monitoring data as well as results from the Analytics module and the Stream Processing module. Raw monitoring data are sent to the Storage module directly from the Messaging System through a driver.

The Analytics module is similar to a batch processing system. It analyzes the monitoring data stored in the Storage module to discover patterns, trends, and useful information that could benefit infrastructure management and users of the infrastructure. Analytics results are stored in the Storage module to allow access from the Stream Processing module and the modules in the Access layer.

4) User Access Layer: The user access layer is an interface layer to users of the system. Users can access current and historical monitoring data on-demand. This layer provides capabilities for on-demand access of monitoring data. The layer has three modules: Notifier, API, and Graphical User Interface (GUI). Users pull for monitoring data through the API or GUI, and receive notifications from the Notifier module for user-specified notification conditions.

5) Analytics Requirements Database: An Analytics Requirements Database (ARD) module complements the four layers. We have mentioned that the Stream processing module and the Analytics module need to perform required monitoring tasks. These tasks and notification requirements are stored in the ARD module. A user can specify the tasks and the notification conditions through the API module or the GUI module.

V. System Evaluation

We now present the system evaluation of MonArch. For evaluation purposes, we deployed the system in virtual machines (VMs) in the SAVI Testbed and conducted a series of experiments to evaluate the system's performance and scalability. We obtained the following general results:

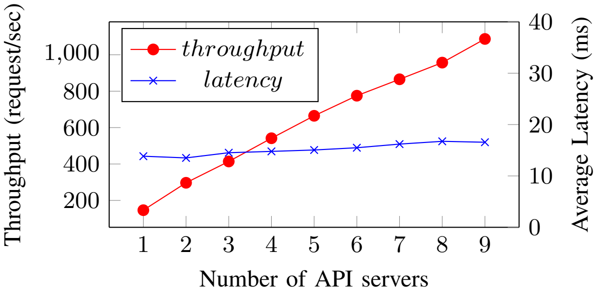

Fig. 2. System Scalability for User Requests

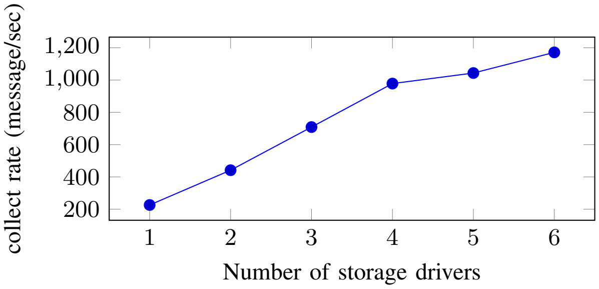

Fig. 3. Maximum average monitoring data collection rate

- MonArch is capable of handling users' requests with short delay. The system scales up closed to linearly to support increasing number of user requests.
- As the size of the infrastructure and the number of monitoring tasks increase, MonArch is able to process the large number of incoming monitoring messages and store them in the database. This part also scales close to linearly as we replicate the storage driver.

Next, we discuss the environment that the system is deployed on, and then we present the details about system performance and scalability.

A. Evaluation Environment

For evaluation, MonArch is deployed in VMs in the SAVI Testbed's Core node. The API module (modified from Ceilometer API) is deployed in an extra large VM instance (8 virtual CPU cores, 16G memory, and 160G hard disk). The Agents are running in a large VM instance (4 virtual CPU cores, 8G memory, and 80G hard disk). The storage module (Cassandra database) is deployed in 2 medium VM instances (2 virtual CPU cores, 4G memory, and 40G hard drive). The Storage drivers (for writing data from messaging system to storage module) run in the same VMs as the Cassandra database. The streaming and batch processing module (Spark clusters) run in three medium VM instances (one for the master, two for the workers). The messaging system (Kafka) runs in a medium VM instance.

B. Evaluation for Handling of User Requests

To evaluate the system's performance and scalability when handling users' requests for monitoring data, we deployed multiple API servers in the extra large VM instance, and we ran a HA Proxy to evenly distribute the requests to the API servers. The API servers are all connected to the Cassandra database (deployed in two separate VMs). Since an API server runs in a single process, by varying the number of API servers running in the VM, we can observe the scalability of the system. To run this experiment, we used the Apache Bench utility to send requests querying for the latest monitoring sample of a certain metrics (meter). Figure 2 shows the result of this experiment. We can see from the figure that the throughput (average number of requests per second) increases approximately linearly as the number of API servers increases, while delay remains low. From this result, we can easily see that by replicating the API server and the VM that hosts these API servers, we can effectively scale the system as the demand increases. Since the increase in the latency when having more API servers is quite small, scaling the system should not affect user experience.

C. Infrastructure Scaling Evaluation

Another aspect of the system is its capability to collect monitoring data and store it in the database. The amount of monitoring data generated depends on the size of the infrastructure and the number of active user-specified metrics. To evaluate the system's performance and scalability in this part, we measure the system's maximum average collection rate while having a different number of Kafka servers, storage drivers, and Cassandra database servers. For this evaluation, we ran 20 agents (10 OpenFlow agents, 5 physical server agents, and 5 user agents) that generate monitoring data every 5 seconds. In order to push the system to it's maximum performance, we also fill up the Kafka queue with about 20,000 monitoring messages. To measure the system's collection rate, we count the number of messages in the database and calculate the changes over time.

In our experiment, we found that the performance bottleneck is the storage driver, and in order to scale the system, we can run multiple storage drivers to improve the collection rate. Figure 3 shows the relationship between the system's collection rate and the number of storage drivers. As shown in the graph, the collection rate increases approximately linearly as the number of storage drivers increases. Since Kafka supports partitioning and we do not require global ordering of all the monitoring messages, replicating the Kafka server should offer scalability. On the database size, we should be able to scale up Cassandra by running it on more VMs. Therefore, the system's collection rate is scalable, which means the system is capable of handling monitoring tasks as the infrastructure scales up and the number of users increase.

VI. Conclusion

We have presented the MonArch monitoring and measurement system based on the Software-Defined Infrastructure (SDI) concept. The system offers integrated and flexible monitoring, measurement and analytics functionalities that are available to users on-demand through open APIs. We have implemented and deployed MonArch in the SAVI Testbed, and conducted system evaluations. The system meets all the functional requirements and offers high performance and scalability. For future work, we will be focusing on researching and improving the analytics part of the system to provide more advanced anomaly detection, auto diagnosis, and root cause analysis functionalities.

Acknowledgment

The work of this paper is funded by the Smart Applications on Virtual Infrastructure (SAVI) project under the National Sciences and Engineering Research Council of Canada (NSERC) Strategic Networks.

References

[1] J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "SAVI testbed: Control and management of converged virtual ICT resources," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on.* IEEE, 2013, pp. 664–667.

[2] "Ceilometer," available at http://docs.openstack.org/developer/ceilometer/, [Online; accessed 1-Febuary-2015].

[3] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[4] J.-M. Kang, T. Lin, H. Bannazadeh, and A. Leon-Garcia, "Software-Defined Infrastructure and the SAVI testbed," in *9th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TRIDENTCOM 2014)*, 2014.

[5] "Nagios," available at http://www.nagios.org/, [Online; accessed 1-Febuary-2015].

[6] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004.

[7] M. Smit, B. Simmons, and M. Litoiu, "Distributed, application-level monitoring for heterogeneous clouds using stream processing," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2103–2114, 2013.

[8] P. Sharma, S. Chatterjee, and D. Sharma, "CloudView: Enabling tenants to monitor and control their cloud instantiations," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on.* IEEE, 2013, pp. 443–449.

[9] "MONaaS," available at https://wiki.openstack.org/wiki/MONaaS, [Online; accessed 1-Febuary-2015].

[10] J. Montes, A. Sánchez, B. Memishi, M. S. Pérez, and G. Antoniu, "GMonE: A complete approach to cloud monitoring," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2026–2040, 2013.

[11] C. C. Marquezan, D. Bruneo, F. Longo, F. Wessling, A. Metzger, and A. Puliafito, "3-d cloud monitoring: Enabling effective cloud infrastructure and application management," in *Network and Service Management (CNSM), 2014 10th International Conference on.* IEEE, 2014, pp. 55–63.

[12] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: monitoring network utilization with zero measurement cost," in *Passive and Active Measurement.* Springer, 2013, pp. 31–41.

[13] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost netowrk monitoring framework for software defined networks," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2014.

[14] N. L. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in openflow Software-Defined Networks."

[15] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf, "Monalytics: online monitoring and analytics for managing large scale data centers," in *Proceedings of the 7th international conference on Autonomic computing.* ACM, 2010, pp. 141–150.