Architecture of a Network Performance Monitor for Application Services on Multi-Clouds

Young-min Kim, Ki-sung Lee, Jae-cheol Uhm,
Si-chang Kim, and Chan-gun Lee
Dept. of Computer Science and Engineering
Chung-Ang University, Seoul, Korea
{remnant1120, goory00, steveum0105,
sichang0160}@gmail.com, cglee@cau.ac.kr

Minsuk Song, Honguk Woo
Software R&D Center
Samsung Electronics
Suwon, Korea
{msuk.song, honguk.woo}@samsung.com

***Abstract*—Recently, many new services are being deployed on the cloud due to its flexibility of on-demand allocation and de-allocation of computing resources. As the services require more advanced features such as fault-tolerance and high performance, much interest is focused on the multi-clouds. Hence, the needs for effectively managing the quality of services on multi-clouds emerge rapidly. Among the many factors affecting the quality of services, we focus on the network performance in this paper. We identify essential requirements of network performance monitor for multi-clouds and propose an architecture. In particular we address the necessity of supporting external agents and discuss how to integrate with them in a flexible and extensible way. In addition, the issues of timely delivery and off-line analysis of measured results are addressed.**

Keywords— cloud computing; monitor; multi-cloud; network performance; quality of service;

I. INTRODUCTION

Cloud computing has emerged as an attractive vehicle for providing virtual computing resources as a service over the internet[10]. Cloud computing renders flexible provision of services by allowing on-demand requests for allocation/de-allocation of the resources such as computing power, storage, etc.

The users of cloud applications tend to be affected by service performance fluctuations due to the nature of cloud computing, i.e. the sharing of resources. Therefore, many cloud applications rely on the quality of service management to maintain and control their service grades to the users.

Network performance impacts significantly the quality of service received by the users of network based applications. For example, the users using an IP-telephony application over the network may suffer from the intolerable quality of communication in the presence of significant network delays. This is especially true for cloud-based services consisting of a set of cloud nodes dispersed in multiple geographical locations. Even worse, the network overheads of cloud services often exhibit burst characteristics.

Hence there have been many studies and efforts on the network performance monitor on the cloud[15]. However, most of previous efforts on the monitoring tools for clouds focused on the performance and availability of intra-cloud resources such as CPU, storage, and intra-network.

We argue that the external network performance information such as latency and throughput between the end user and the VM of the cloud host should be monitored in order to understand the quality of the service experienced by the users. In addition, it is not sufficiently addressed yet how to provide the monitor with the extensibility for measuring mechanisms of the network performance.

It should be noted that the network performance measurements sampled at arbitrary times for a handful of users may not reflect the true network situation. In order to analyze and understand the network status for long duration, regular basis, and enough number of samples need to be collected. Recently, there have been a few efforts such as Dipzoom[11] and PlanetLab[2], which open the accesses to external agents with services of measuring various network performance located at diverse geographical sites.

In this paper, we propose an architecture of network performance monitor for application services on multi-clouds. We identify a set of key requirements for a network performance monitor for multi-clouds and propose an architecture of the monitor. It is designed to integrate with the external agents such as DipZoom and PlanetLab in a flexible and extensible fashion. In addition, the monitor is equipped with two separate output interfaces for real-time publish/subscribe and database management system.

The rest of this paper is organized as follows. Section 2 presents related work. In Section 3, we discuss key requirements for a network performance monitor for multi-clouds and the proposed architecture. Section 4 discusses the design of the external agents interface and the output interfaces. Section 5 introduces a prototype implementation of the architecture and initial results. Section 6 presents the future work and summary.

II. RELATED WORK

Recently extensive research related to cloud service monitoring has been conducted actively. Chaves et al.[14] addressed design and implementation of a private cloud

monitoring system while pointing out the scarceness of open source cloud management and monitoring solutions. They designed an abstract general-purpose monitoring architecture which monitors resources of cloud VM, and tried to integrate traditional monitoring tools and methods. Benedict[15] compared various tools for monitoring performance of cloud and distributed system - e.g., CloudWatch, Ganglia, Nagios, InterMapper, etc.

Ma et al.[8] presented a lightweight monitoring framework which monitors the QoS of a public cloud without modifying the implementation of the monitored object. Especially they highlighted interactive information between end users and clouds such as response time, access frequency, IP distribution, time of staying and so on. However, since their approach is based on user experiences of the cloud applications, the collected network information limited to the users access regions.

Li et al.[1] proposed a systematic comparator of the performance and cost of cloud providers. They measured the elastic computing, persistent storage, and networking services. They used two approaches for measuring network performance. One was to measure intra-cloud network performance, i.e. measuring the traffics between a VM and another VM in the cloud. The other was to measure wide-area network performance, i.e. measuring the traffics between the VMs located at different clouds. Based on the results, they provided performance and cost information in order to choose right cloud vendor for user's application.

There have been a few approaches for measuring the wide-area network performance. Rabinovich et al.[11] proposed DipZoom which is an on-demand internet measurement platform. Currently it has about 400 measurement points around the world and provides Java APIs. Custom benchmark programs can use the APIs to exploit the DipZoom measurement points. Chun et al.[2] proposed the PlanetLab which is an overlay network for testing world-wide network services. It allows researchers to access to a large set of geographically distributed hosts. Therefore it has been used to test new technologies for distributed storage, peer-to-peer systems, query processing, and so on. Using the PlanetLab, it is possible to measure various network performances with user customized tools. Both DipZoom and PlanetLab can be used to measure the network performance between world-wide agent nodes and the cloud. However, since each platform has different policies in access method, authorization, and operational method, we need to devise a design to integrate these platforms with a flexible and extensible manner.

III. Architecture of Network Performance Monitor for Multi-Clouds

We identify key requirements for a cloud network monitor as shown in the following.

Firstly, the monitor should be programmable to periodically repeat measuring both internal and external network performance of the cloud hosts. By internal network, we mean the intra network of the cloud such as traffics between a VM and another VM in the same cloud. By external network, we mean the network between a cloud host and the end-users outside of the cloud.

Secondly, the monitor should support the integration of the external agents with services of measuring various network performances which typically are not under our own management. Since extra groups of such services may appear, the monitor should provide a flexible and extensible way to integrate with them.

Thirdly, the monitor should provide a timely notification service since the condition of the environment where the cloud service operates is very dynamically changing. The notification should be delivered in a timely manner for enabling the cloud service to adapt itself to the changed network situation properly.

Lastly, the output of the distributed monitors should be aggregated into a repository providing both stability and accessibility such as database management system so that it can be archived safely and retrieved by many analysis applications.

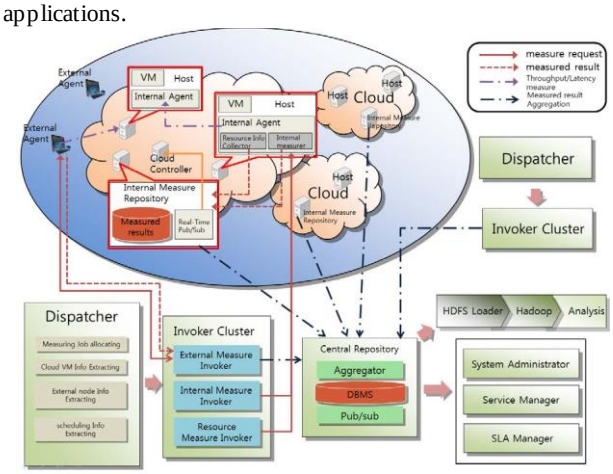

Figure 1. Architecture of Network Performance Monitoring for Multi-Cloud

Figure 1 shows our proposed architecture of network performance monitor for multi-clouds. It is assumed that each cloud includes at least one host or more hosts, an internal measure repository (IMR), and a cloud controller.

An IMR consists of a storage for measured results and real-time publish/subscribe components, which will be explained in Section 4. Typically, the cloud controller which manages the hosts in the cloud becomes a subscriber and keeps track of the status of the clouds.

A dispatcher triggers a measurement job by requesting a thread in an invoker cluster to start measuring by following a configuration containing monitoring information such as a measurement type, source nodes, target nods, and a schedule.

External agents (EA) represent the entities which are capable of measuring certain type(s) of network performance, such as latency by using ping, and they reside at diverse geographic locations typically outside of the clouds. They can be either public agents such as Dipzoom nodes or proprietary ones installed and managed for private purposes. The public agents accept the request via their own interface; hence the

consideration for this should be made in designing the architecture. We will discuss this issue in Section 4.

The actual contents of the configuration depend on the type of measurement. As an example of external measure, a periodic schedule requiring the measurement of latency per ten minutes will be passed along with information about the sources and targets such as a list of external agents and cloud hosts respectively.

An invoker cluster consists of threads of external measure invokers (EMI), internal measure invokers (IMI), and resource measure invokers (RMI).

An EMI handles the requests for external network measure sent by the dispatcher. The EMI requests the measurement to the external agents satisfying the measurement configuration information, e.g. California and Oregon states, and aggregates the results. Then the aggregated results are saved into the central repository.

An IMI handles the requests for intra network measure and contacts internal agents (IA) of the target hosts. Then each IA measures the internal latency and/or throughput. The IA also handles the resource information request for the host. After completing the measurement, the results are forwarded to the internal measure repository (IMR) which is running on the same cloud where the host containing the IA belongs to.

RMIs operate similar to the IMIs. The IA that received the request from a RMI commands the resource information collector to retrieve the corresponding information, and accumulates it into the IMR of the cloud.

The central repository (CR) gathers various measurement results and stores them into a DBMS such as Oracle. Depending on the level of timely delivery requirements, one may adopt either real-time or conventional publish/subscribe components to notify the interested subscribers of newly updated information.

Hadoop File System (HDFS) loader is attached to the Central Repository to dump the results with hadoop friendly format into HDFS for further off-line analysis.

IV. Interfacing with External Agent and Issuance of Measured Results

A. Interfacing External Measure Invoker and External Agent

An external measure invoker (EMI) sends the request of measuring network performance to the external agents (EA). It should be emphasized that the interface between the EMI and the EA should be designed for flexibility and extensibility. In designing the interface, we focused on the following three aspects.

- Allowing diverse implementations for measurement operations
- Supporting extra types of measuring operations
- Supporting external agents with heterogeneous interfaces

The first two aspects are addressed by adopting the combination of the design patterns, Strategy pattern and Adapter pattern. The third aspect is overcome by abstracting the EA.

We note that there are EAs with different types of interfaces for measuring the network performance. For example, a DipZoom server accepts region name and returns a list of available external nodes of the region. Then, by sending the request to each node, the measurement can be processed. In contrast, individual PlanetLab node can be directly accessed and requested for measuring the network performance. In fact, PlanetLab node itself runs virtual machine (VM) and the VM can be programmed for arbitrary purposes.

Figure 2 shows the class diagram for the interface between EMI and EA. Note that ExternalNodeInfo is designed to represent a general entity which is specialized into ExternalNodeRegionInfo and ExternalAddressInfo. External-NodeRegionInfo contains data structures and methods for the external nodes of which interface expects region based network performance measurement. In contrast, ExternalAddressInfo contains similar information for the external nodes of which interface expects individual based measurement. As explained above, DipZoom servers and PlanetLab nodes are modeled into instances of ExternalNodeRegionInfo and ExternalAddressInfo respectively.

ExternalTCPThroughputMeasure, ExternalLatencyMeasure, and ExternalUDPThroughputMeasur are modeled as interfaces in Java. As explained before, we adopt the Strategy pattern in the integration design of EMI and external agents. After being invoked by a dispatcher, the EMI sends a request of measurement to an instance of the class which corresponds to the external agent and implements the appropriate interface.

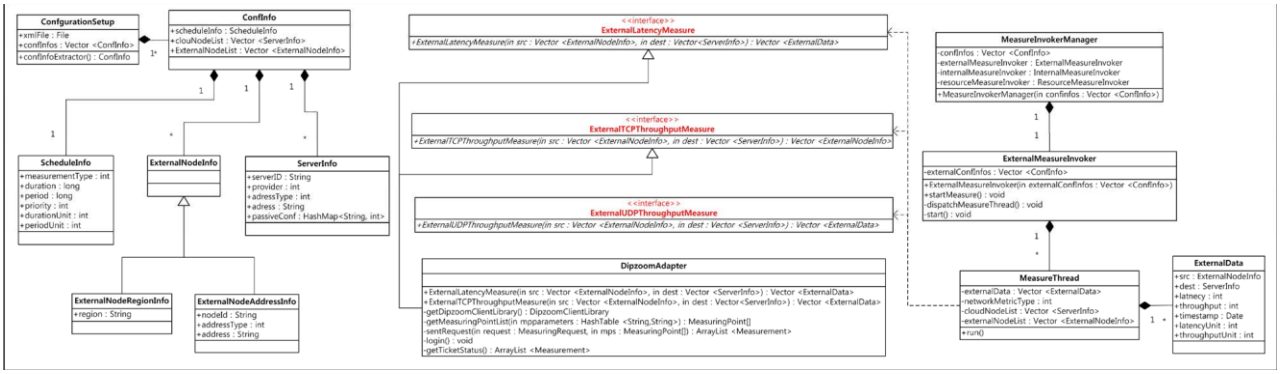

Figure 2. Interfacing between External Measure Invoker with External Agents

A typical integration is done by writing an adapter class converting each interface method into that of the corresponding proprietary external agent. In the above, the adapter class for DipZoom, i.e. DipzoomAdapter, is designed as a concrete class implementing two interfaces ExternalTCPThroughputMeasure and ExternalLatencyMeasure since DipZoom does not support UDPThroughput measurement. In addition, it implements the adaptation between the above interfaces with its own service methods.

Note that linking of the composition between DipzoomAdapter and the EMI is dynamically done. We designed that the instance of DipzoomAdapter is referenced through more general types than that of the instance such as ExternalTCPThroughputMeasure and ExternalLatencyMeasure interfaces from the EMI. Due to this dynamic composition, the modification or extension of concrete adapter or changes internal to DipZoom does not propagate to the system, which meets design goals.

Our design also provides convenient mechanism for the integration with new types of measurement tools. A typical scenario is that the generic adapter is extended by inheritance for the new type of measurement tool. Since it still supports the interfaces expected by the EMI, further modification for the extension is not needed. New measuring behaviors can be easily added too by defining a new interface for each of them. For example, ExternalLinkReliability can be added as a new interface and the EMI can be extended to use the instances of the classes implementing the interface.

B. Conventional vs. Real-time Publish/Subscribe

There are various types of measured performance results within the performance monitor. Some of them need to be delivered as soon as possible to keep their freshness due to the timing criticality to the management of the system, while other information may be tolerable to some delays.

Depending on the nature of the measured results, they need to be delivered to the interested components or users in a proper fashion. For example, every change of resource availability and intra-cloud network performance may be crucial to the cloud controller which manages a cluster of hosts. In contrast, the arrival timings of measurement results from EAs might be unpredictable. Remind that EAs are not under our control in general. Some of them may respond instantly while others may return the results with long delays.

In order to handle the above heterogeneity of the measurements, we propose two different sorts of publish/subscribe mechanisms for the network performance monitor and adopt them properly in the architecture.

One is conventional publish/subscribe model which promotes the loose coupling between publishers and subscribers but cannot guarantee the temporal predictability. The other is real-time publish/subscribe (RTPS)[18] model which was proposed by the OMG. The RTPS is a service that provides reliable publish/subscribe communications in IP network based real-time applications. The key point of the RTPS is to guarantee real-time QoS in a paradigm of "right data at the right time at the right place." It has various delivery options from "best effort" to highly reliable in-order delivery, and enables managing some real-time deadlines such as periodic publish deadline or message delivery deadline[17].

Commercial RTPS applications are found at aerospace, unmanned system, smart city, medical device, financial service, etc[13]. Recently, RTPS is adopted for cloud resource monitoring. An et al.[7] presented cloud resource monitoring mechanism that generates and delivers resource data in a RTPS based middleware.

Since we proposed the adoption of RTPS for performance monitoring of multi-clouds in the previous section, let us briefly introduce its concept and running mechanism.

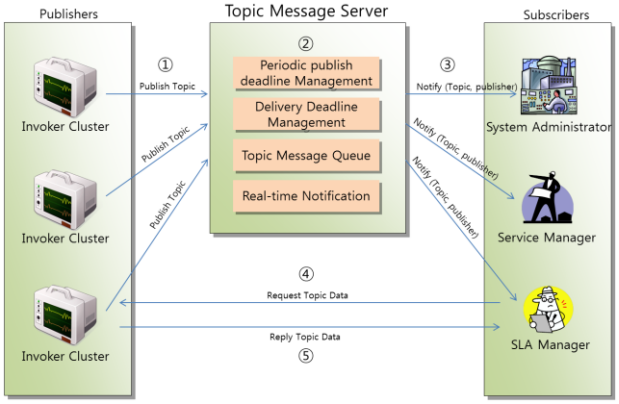

Figure 3. Real-time Publish/Subscribe Monitoring

Figure 3 shows message and control flow of real-time publish/subscribe. ①When decentralized network performance aggregators publish topics, ② central topic message server collects and categorizes topic messages, ③and notifies the messages to corresponding subscribers. The topic message server provides RTPS so that timing properties can be managed during the message delivering. In a large-scale decentralized environment, since lots of data transmission cause high network communication overhead, in order to mitigate this overhead problem, our topic is transmitted in a form of a very simple message, and is delivered to only related subscribers. ④ The subscriber who received the topic message requests actual data to the publisher directly, ⑤then the publisher, the cloud performance aggregator, gives data immediately.

From this structure, the customers of the monitor such as a system administrator, a service manager, and a SLA manager can receive an up-to-the-minute condition of the multi-cloud service. [7] demonstrated predictable performance of RTPS; in 50 publishers of cloud resource monitoring environment, the average of message latencies was 3.3ms.

In our framework, the topic is a concern for monitoring. The concern can be various factors such as target cloud server, network performance, CPU usage, memory usage, etc. So, in order to easily classify and search, we define the topic structure hierarchically. Table 1 shows a hierarchical topic example.

The proposed hierarchical topic can be expressed in a form of XQuery. Therefore a monitoring manager can subscribe by filtering his interests. The following queries show various filtering examples.

```
/CLOUD[SOURCE_ID='1'][Group_ID='2']/EXTERNAL/LATENCY
/CLOUD[Group_ID='3']/INTERNAL
//RESOURCE/CPU
```

For instance, if we want to monitor network latency from Seoul to every cloud node, we can subscribe to the topic message server using the following query.

```
/CLOUD[SOURCE_ID='SEOUL']/EXTERNAL/LATENCY
```

In this case, the topic message server selects the measured results that are in the category of network latency topic and measured by the external agents at Seoul. Then the server multicasts the topic messages to the corresponding subscribers along with the publisher information. By these topics and publisher information, the subscribers can identify which performance aggregators have necessary performance data.

TABLE I. HIERARCHICAL TOPIC DEFINITION AND SUBSCRIPTION

LEVEL 1	LEVEL 2	LEVEL 3
MONITORED CLOUD SERVER	MONITORING METHOD	PERFORMANCE METRIC
CLOUD[SOURCE_ID='*'][DEST_ID='*']	EXTERNAL	LATENCY, THROUGHPUT
	INTERNAL	LATENCY, THROUGHPUT
	RESOURCE	CPU, MEMORY, HDD

C. On-line and off-line analysis

Not only the functionality of on-line analysis for the performance data, but also off-line analysis is required for the network performance monitor. For both types of analysis, we propose to store the aggregated results in a DBMS. In addition, in order to support large distributed processing for the measure data, we propose to attach a Hadoop File System (HDFS) Loader to the DBMS. It performs bulk-load of the data set from the DBMS into the HDFS, where the Hadoop system reads the data and processes them.

The rationale behind of this design is that the DBMS provides stable storage, backup mechanisms, concurrent retrievals/updates, and convenient querying interfaces.

Figure 4 shows an example of cloud monitoring analysis using Map Reduce of the Hadoop system. Typically, the data set in the HDFS are processed in the unit of HDFS block size in each mapper. Each mapper filters the given data for its own analysis purpose and generates a set of records with pairs of a key and a value. Then each reducer in charge of handling specific keys collects the outputs from the mappers and processes further by their keys and values.

Let us briefly introduce an example of off-line analysis for extracting minimum latency for a specific region and date. As shown in Figure 4, assume that HDFS has loaded a table containing the measurement results for a cloud node for three days. Each column of the table indicates measured date, region of external nodes, external latency, and external throughput respectively. A block containing records of pairs of keys and values is given to each mapper. In this example, a key is a pair of date and region. A mapper keeps only the records of which latency is the minimum for each key. After finishing the job, the kept records are given to the reducers. Each reducer uses the pair of date and region as a key. The records with the same key will be forwarded to a unique reducer. Now, the reducer can compute the minimum latency for each pair of date and region.

By interconnecting the DBMS and HDFS loader, huge amount of results from the network monitoring can be efficiently analyzed in off-line.

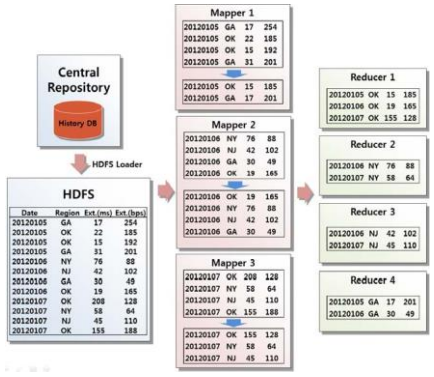

Figure 4. Batch analysis using MapReduce

V. PROTOTYPE IMPLEMENTATION

We have implemented a proof of concept based on the proposed architecture. Our current implementation adopts DipZoom and CloudCmp[1] for external and internal agents respectively. Our prototype is running on Windows 7 and SQLite with Intel i7 2.8GHZ and 4GB main memory. The program is written in Java 1.7 with Eclipse.

As explained in Section 3, DipZoom adapter converts the requests from the EMIs to the corresponding ones in DipZoom library. Figure 5 shows the corresponding code snippet and the major tasks are summarized as follows: ①the EMI instantiates a MeasureThread. Note that an instance of DipzoomAdapter which implements a singleton pattern is passed along with other measurement information to the constructor. ② Then the MeasureThread invokes an interface method which depends on networkMetricType. ③ ExternalLatencyMeasure method overriden in DipzoomAdapter measures the network latency by calling a series of DipZoom library APIs.

The internal measurement of IA is implemented by using CloudCmp, a benchmark tool designed to run diverse performance experiments.

Figure 6 shows an example of the configuration for an external latency measurement from at most five DipZoom nodes in CA region to an Amazon host. This particular configuration requires periodic measurements on every 5 minutes for 10 hours. Figure 7 shows the measured latency results that are stored in the SQLite after finishing the job.

```
public class ExternalMeasureInvoker {
    ⋮
    private void dispatchMeasureThread(){
        ⋮
①       measureThread = new MeasureThread(scheduleInfo ,networkMetricType,
                cloudNodeList, externalNodeList,
                DipzoomAdapter.getInstance(),DipzoomAdapter.getInstance(),
                DipzoomAdapter.getInstance());
        ⋮
}
```

```
public class MeasureThread extends Thread {
    ⋮
    public void run() {
        while (true) {
            try {
②               if ((networkMetricType & EXTERNAL_LATENCY) != 0)
                    externalLatencyMeasure.ExternalLatencyMeasure(
                            externalNodeList, cloudNodeList);
                ⋮
}
```

```
public class DipzoomAdapter implements ExternalLatencyMeasure,
        ExternalTCPThroughputMeasure, ExternalUDPThroughputMeasure {
    ⋮
    public Vector<ExternalData> ExternalLatencyMeasure(
            Vector<ExternalNodeInfo> insrc, Vector<ServerInfo> indest) {
        ⋮
        dipzoomClientLibrary.login();
        ⋮
③       measuringPoints = dipzoomClientLibrary.getMeasuringPointList(measuringPointParameters);
        ⋮
        pendingMeasurements = dipzoomClientLibrary.sendRequest(request, measuringPoints);
        ⋮
}
```

Figure 5. Code snippet for integrating with DipZoom

```
<?xml version="1.0" encoding="UTF-8"?>
<ConfInfos>
    <ConfInfo>
        <scheduleInfo>
            <measurementType>Latency</measurementType>
            <duration>10</duration>
            <period>5</period>
            <priority>1</priority>
            <durationUnit>H</durationUnit>
            <periodUnit>M</periodUnit>
        </scheduleInfo>
        <clouNodeList>
            <ServerInfo>
                <serverID>ec2-107-21-168-34.compute-1.amazonaws.com</serverID>
                <provider>Amazon</provider>
                <addressType>URL</addressType>
            </ServerInfo>
        </clouNodeList>
        <externalNodeList>
            <ExternalNodeRegionInfo>
                <region>CA</region>
                <maximum>5</maximum>
            </ExternalNodeRegionInfo>
        </externalNodeList>
    </ConfInfo>
</ConfInfos>
```

Figure 6. Configuration Example for External Latency Measurement

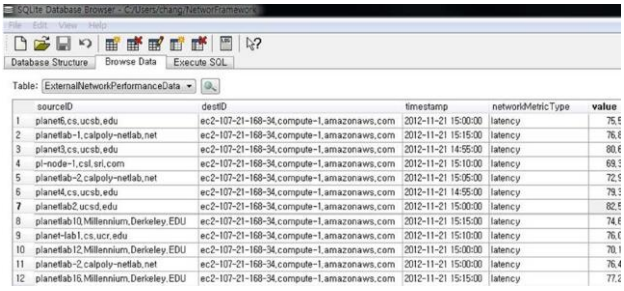

Figure 7. External Latency Measurement Results

VI. CONCLUSION

In this paper, we addressed the needs for network performance monitor for multi-clouds and their major requirements. We proposed an architecture and explained the design rationales. The key features of the architecture include flexible integration with external agents, and separate output channels such as real-time publish/subscribe and DBMS-based offline analysis friendly repository. We also reported our current implementation of a prototype for the architecture. We are planning to develop a full-fledged system based on the proposed architecture and test it on real-life scenarios.

Acknowledgements

This work was supported by the National Research Foundation of Korea Grant funded by the Korean government (NRF-2011-0013924) and a grant from Samsung Electronics.

REFERENCES

[1] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," In Proc. of Conference on Internet Measurement (IMC), pp.1-14, 2010.

[2] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman "PlanetLab: an overlay testbed for broad-coverage services," SIGCOMM Comput.Commun, vol. 33, pp. 3-12, 2003.

[3] B. König, J.M.A. Calero, and J. Kirschnick, "Elastic monitoring framework for cloud infrastructures," IDT Commun., vol. 6, pp. 1306-1315, 2012.

[4] G. Deng, M. Xiong, A. Gokhale, and G. Edwards, "Evaluating Real-time Publish/Subscribe Service Integration Approaches in QoS-enabled Component Middleware," In Proc. of IEEE ISORC, pp. 222-227, 2007.

[5] G. Wang and T. S. E. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," In Proc. of IEEE INFOCOM, pp.1-9, 2010.

[6] J. L. L. Simarro, R. M.-Vozmediano, R. S. Montero, and I. M. Llorente, "Dynamic Placement of Virtual Machines for Cost Optimization in Multi-Cloud Environments," High Performance Computing and Simulation (HPCS), pp.1-7, 2011.

[7] K. An, S. Pradhan, F. Caglar, A. Gokhale, "A Publish/Subscribe Middleware for Dependable and Real-time Resource Monitoring in the Cloud," In Proc. of SDMCMM, 2012.

[8] K. Ma, R. Sun, A. Abraham, "Toward a lightweight framework for monitoring public clouds," In Proc. of Computational Aspects of Social Networks, pp.361-365, 2012.

[9] L. S. Garfinkel,"An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS," Harvard University TR-08-07, 2007.

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia "A view of cloud computing," Communications of the ACM, vol. 53, pp.50-58, 2010.

[11] M. Rabinovich, S. Triukose, Z. Wen, and L. Wang, "DipZoom: The Internet Measurements Marketplace," In Proc. of IEEE International Conference on Computer Communications, pp.1-6, 2006.

[12] "Data Distribution Service for Real-time Systems Version 1.2," OMG, 2007.

[13] OpenSplice DDS, http://www.prismtech.com/opensplice.

[14] S. A. de Chaves, R. B. Uriarte, C. B. Westphall, "Toward an Archtecture for Monitoring Private Clouds," IEEE Communication Magazine, vol. 49, pp. 130-137, 2011.

[15] S. Benedict, "Performance issues and performance analysis tools for HPC cloud applications: a survey," Computing, vol. 95, pp. 89-108, 2018.

[16] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal Virtual Machine Placement across Multiple Cloud Providers," In Proc. of Services Computing Conference(APSCC), pp.103-110, 2009.

[17] S. Schneider and B. Farabaugh, "Is DDS for You?," A Whitepaper by Real-Time Innovations, 2009.

[18] The DDS RTPS Portal, "http://www.omg.org/spec/DDS-RTPS/2.1/